Práctica 2
Deep Learning

Hugo Gil Parente
Jorge Moreno Fernández

Máster Universitario en Visión Artificial
Universidad Rey Juan Carlos

Repositorio GitHub

Introducción

En esta memoría se presentan diferentes arquitecturas para un problema de clasificación de imágenes. Cada arquitectura contará con una pequeña introducción a la misma y se mostrarán imágenes con los resultados de entrenamiento y test (en caso de que haya datos de test).

Para la realización de la práctica, se han realizado varios scripts de Python que permiten entrenar y probar cada red de manera individual. En cuanto al preprocesamiento, se ha utilizado OpenCV para la carga y la normalización de las imágenes. Para realizar un aumentado de datos, se han utilizado las transformaciones proporcionadas por *PyTorch*. En concreto, se ha utilizado la siguiente lista de operaciones:

- *RandomHorizontalFlip*
- *RandomRotation*
- *Normalize*

Pruebas realizadas

A la hora de abordar la práctica, hemos realizado pruebas sobre diferentes modelos de redes neuronales:

CNN

El código original de esta red se obtuvo de las sesiones de prácticas de la asignatura de Fundamentos Matemáticos. Inicialmente la red contaba solamente con dos capas convolucionales (backbone) y una capa densa (cabezal), y al realizar las primeras pruebas, los resultados no eran prometedores.

Se decidió probar a extender la red, añadiendo nuevas capas para ver cómo evoluciona la clasificación, observando mejoras tras el cambio. La versión actual tiene una estructura de cuatro capas convolucionales y dos capas densas con una activación *softmax* para llevar a cabo la clasificación.

En la Figura 1 se pueden ver los resultados del entrenamiento de la última versión de la CNN, los parámetros de este entrenamiento se corresponden con la primera entrega de la competición (entrada CNN1 de la tabla de entrenamiento).

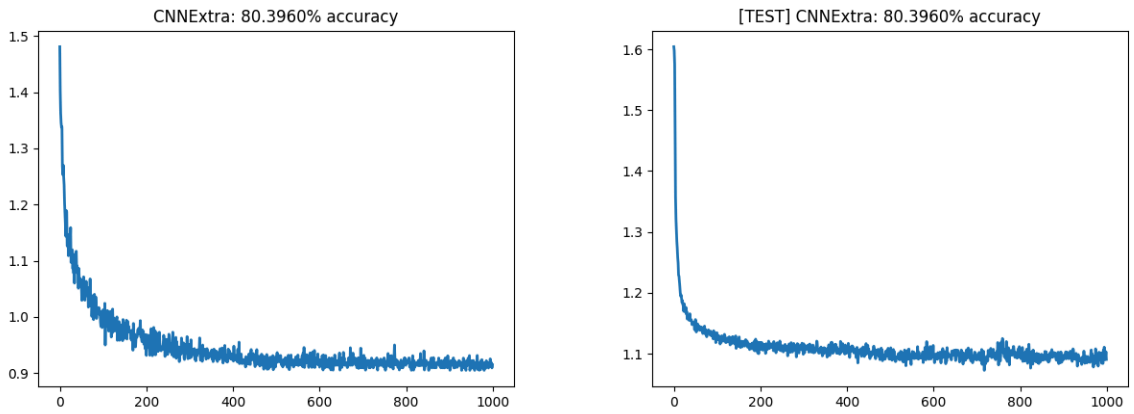


Figura 1: Entrenamiento correspondiente con la entrada CNN1.

AlexNet

Paper

AlexNet se compone de múltiples capas convolucionales y de pooling, seguidas por capas densas. Introduce el uso de funciones de activación no lineales, como la activación ReLU, que ayudan a mitigar el problema de la desaparición del gradiente y aceleran el entrenamiento de la red. Además, incorpora capas de dropout para reducir el *overfitting*.

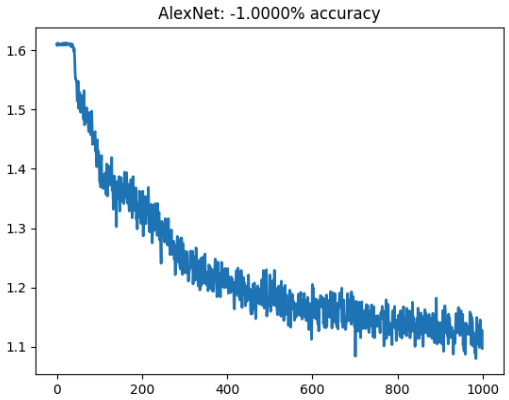


Figura 2: Entrenamiento correspondiente con la entrada Alex1.

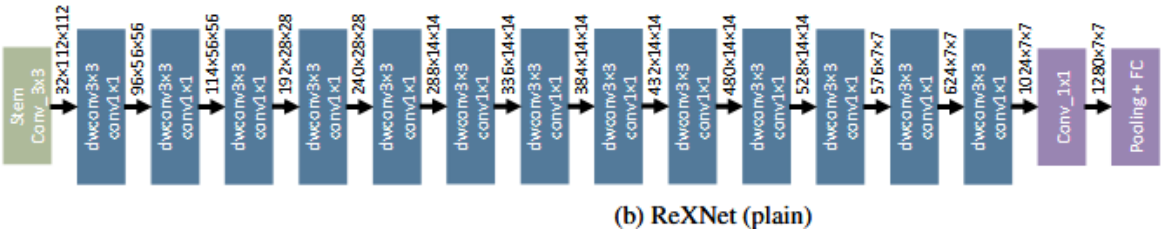
Como se puede observar en la Figura 2, la pérdida del entrenamiento disminuye de forma consistente, finalizando con poca variabilidad. Si se entrenase más épocas apenas habría variabilidad en la pérdida, pudiendo surgir *overfitting*.

ResNet

Se realizaron varias pruebas iniciales con la arquitectura **ResNet**, pero se descartó su uso por su lento aprendizaje y porqué no conseguimos muy buenos resultados.

ReXNet

Paper



(b) ReXNet (plain)

Figura 3: Arquitectura de la ReXNet.

ReXNet (Figura 3) es una arquitectura que se centra en reducir el coste computacional, para ello estudia el rango de la salida de cada capa. Como se puede observar en la Figura 4, la arquitectura **RexNet** es más eficiente que la arquitectura **EfficientNet**.

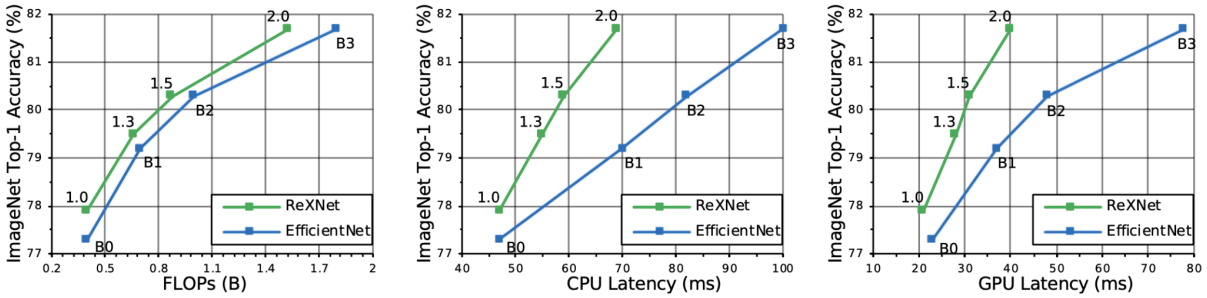


Figura 4: Comparación coste computacional entre ReXNet y EfficientNet

En la Figura 5 se puede observar el resultado de la pérdida durante el entrenamiento de la época 150 a la época 200. Los valores de la época (eje x) están mal, ya que esta red la entrenamos por partes. Este entrenamiento es el correspondiente a la segunda entrega de la competición.

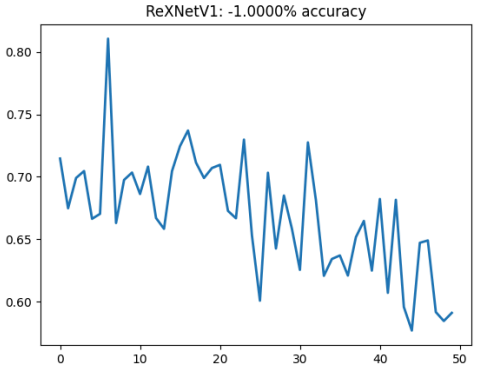


Figura 5: Entrenamiento correspondiente con la entrada Rex1.

CoAtNet

Paper

CoAtNet (Figura 6) es una familia de redes neuronales que utilizan módulos *transformadores* para la implementación de mecanismos de *atención*. Uno de los puntos

importantes de CoAtNet es que consigue una buena generalización en bases de datos más pequeñas, mejorando los resultados de otras arquitecturas de *Vision Transformers*.

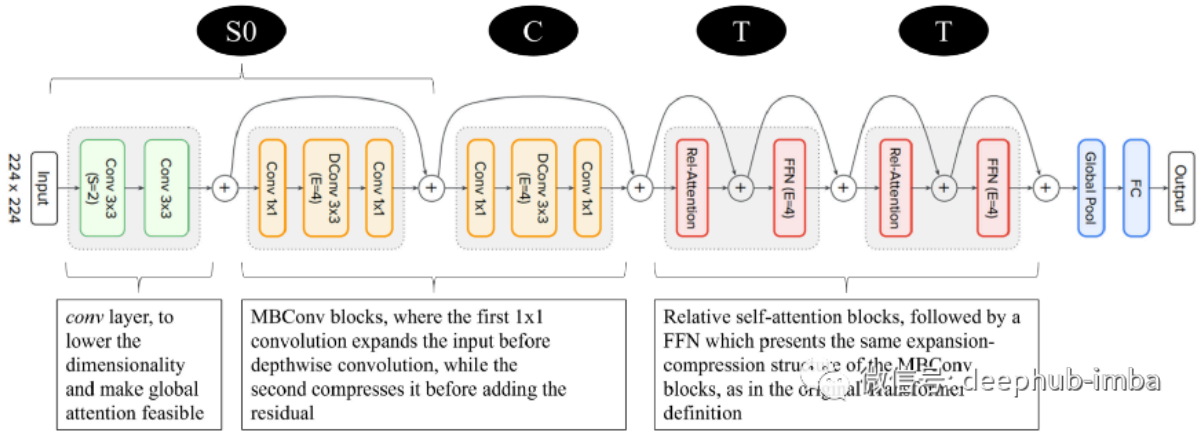


Figura 6: Arquitectura de la red CoAtNet.

Hemos entrenado la red utilizando la versión CoAtNet 0, los resultados obtenidos se muestran en la Figura 7. Para el entrenamiento mostrado en la Figura 7, se ha entrenado con los parámetros mostrados en la tabla de entrenamientos (CAN1).

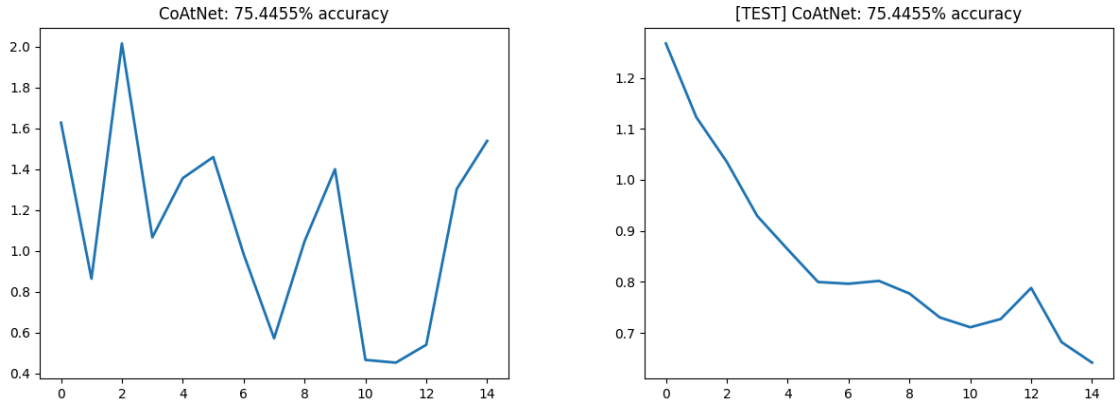


Figura 7: Entrenamiento correspondiente con la entrada CAN1.

Como se puede observar, la perdida de entrenamiento parece un poco inestable, esto se podría corregir reduciendo el *learning rate*, aun así, la pérdida de test tiene una tendencia decreciente sin haberse llegado a estabilizar, probablemente realizando más épocas se podría mejorar el rendimiento de la red.

Tabla de entrenamientos

En la Tabla 1 se muestran algunos de los entrenamientos realizados a lo largo de la práctica.

En algunos casos el Test Accuracy se muestra vacío, eso significa que se ha entrenado el modelo sin realizar el split entre datos de entrenamiento y datos de test, haciendo que haya más datos de entrenamiento. Estos entrenamientos se han realizado para realizar las entregas de la competición.

	Arquitectura	Epochs	Batch Size	Test Accuracy
CNN1	CNN	1000	256	80.39%
Alex1	Alexnet	1000	700	-
Rex1	ReXNet	50	512	-
CAN1	CoAtNet	20	16	75.44%

Tabla 1: Tabla de parámetros de entrenamiento.