TRANSFORMER-6

Jorge Sillero
dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
jorsilman@alum.us.es jorgesilleromanchon@gmail.com

Alberto Gallego
dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
albgalhue@alum.us.es albertogalleg365@gmail.com

***Resumen*— En este trabajo vamos a realizar la tarea de clasificación de imágenes usando redes neuronales. Como veremos a continuación, las redes transformer son las más indicadas para esta tarea. Cabe indicar que la hemos comparado con una red convolucional.**

Palabras Clave—Inteligencia Artificial, Encoder, Decoder, Transformer, Multi-Head Attention, Dropout, Cabezas.

I. INTRODUCCIÓN

En este trabajo hemos realizado la tarea de clasificación de imágenes usando redes neuronales, realizando previamente un entrenamiento de estas, para después clasificar las imágenes de test que teníamos asignadas. Para ello hemos usado dos tipos distintos de redes, una red transformer y una red convolucional. Ambas son redes que se usan con este propósito, las convolucionales se llevan usando desde hace bastante tiempo, pero las transformer están causando una revolución, ya que presentan mejores resultados, como veremos a continuación.

PRELIMINARES

Las técnicas empleadas han sido una red convolucional y una red transformer.

A. Métodos empleados

Red Convolucional

Una red convolucional es un tipo de red neuronal artificial que es muy efectiva para la realización de tareas de visión artificial, como en la clasificación y segmentación de imágenes entre otras aplicaciones.

Estas redes constan de varias capas de filtros convolucionales de una o más dimensiones. Después de cada capa, por lo general, se le aplica una función de activación. Al final de la red, se encuentran otras neuronas más sencillas para la clasificación final de las imágenes sobre las características extraídas.
Para que estas redes clasifiquen las imágenes correctamente es necesario un entrenamiento previo de la misma, con una cantidad importante de imágenes.

Red Transformer

Una red transformer es una arquitectura de redes neuronales, que resuelve tareas Seq2Seq sin los problemas de dependencias largas que presentan las LSTM(Memoria a corto plazo) o RNN(Redes Neuronales Recurrentes).

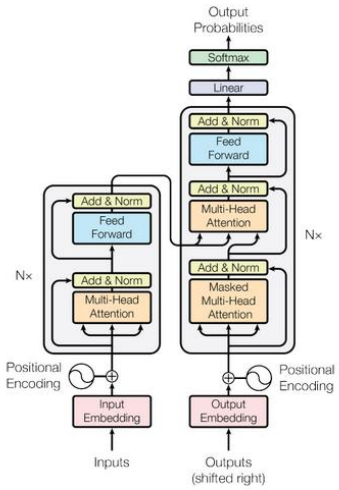

Fig. 1. Arquitectura Transformer.

El esquema básico de la arquitectura transformer se basa en dos elementos principales, un Encoder y un Decoder. El encoder se encarga de analizar el contexto de la secuencia de entrada y el decoder es el encargado de generar la secuencia de salida a partir de este contexto.

En nuestro caso, usamos la arquitectura de Visual Transformer, que es la necesaria para la clasificación de imágenes. Esta consiste únicamente de un encoder.

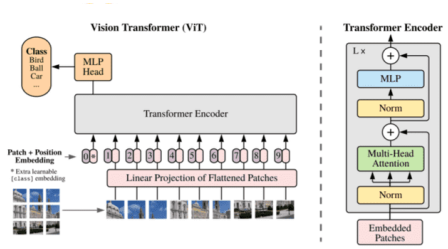

Fig. 2. Arquitectura Visual Transformer.

La primera capa Embedded Patches, se encarga de dividir la imagen en trozos de imágenes más pequeños, para así asignarles un valor numérico y retener la información posicional. La secuencia resultante será la entrada para el encoder.

El encoder consiste en capas alternas de autoatención por multicabezas y bloques MLP. La capa de normalización se aplica antes de cada bloque.

El algoritmo de autoatención permite al modelo integrar información de la imagen incluso a las capas más bajas. Es un mecanismo que permite al modelo saber con qué otro elemento de la secuencia está relacionada la información que está procesando. Este mecanismo toma tres valores de entrada:

- Q: se trata de la consulta (query) que representa el vector de entrada.
- K: las keys, que son todos los demás vectores de la secuencia.
- V: el valor vectorial de la información que se procesa en ese momento.

El algoritmo que nos devuelve la importancia de la información que se está procesando es el siguiente:

$$Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

La versión del algoritmo que usa transformer es una proyección de Q, K, V en h (cabezas) espacios lineales. Esto permite que cada cabeza se enfoque en diferentes características, para después poder concatenar los resultados. La arquitectura multi-cabeza hace posible aprender dependencias más complejas sin añadir tiempo de entrenamiento, ya que son operaciones que se ejecutan en paralelo.

El bloque MLP contiene dos capas con una función GELU no lineal.

B. Trabajo Relacionado

- https://github.com/JACantoral/DL_fundamentals
- https://github.com/lucidrains/vit-pytorch/tree/main/vit_pytorch

II. Metodología

i. Carga de datos

Hemos utilizado los datasets de Pytorch, concretamente la función ImageFolder:

También a cada una de las imágenes le hemos aplicado una transformación para redimensionarla a 64x64 y pasarla a Tensor.

Finalmente hemos dividido el conjunto de imágenes en conjunto de entreno y conjunto de prueba (usa un 10% de las imágenes). He de mencionar también que hemos usado un tamaño de batch de 200.

ii. Red Convolucional

Nuestro modelo está compuesto por dos capas convolucionales y una capa lineal de salida.

La primera capa convolucional recibe como canales de entrada 3, ya que la imagen es a color (RGB) y la salida es de 16. El kernel size es de 3x3 y el padding de 1, esto hace que las dimensiones de la imagen se conserven. La función de activación de esta capa es una relu.

Nuestra segunda capa convolucional recibe como valores de entrada 16, que es la salida de la capa anterior, y la salida es de 32 neuronas. El kernel y el padding son el mismo que en la capa anterior, al igual que la función de activación.

A continuación, aplicamos una capa de Max Poolling, que nos va a dividir la imagen de entrada a la mitad. Después de esto aplicamos la función flatten, que convierte la imagen de dos dimensiones en un array continuo de una dimensión.

Por último, una capa lineal, que tiene como salida los 400 tipos de clases de pájaros que disponemos.

iii. Red Transformer

El modelo consta, para empezar de un embedding, que se encargar de dividir la imagen en otras más pequeñas según el tamaño que se les haya indicado. Después de esto, llegamos al

bloque del decoder, este está formado por la capa del algoritmo de Multi-Head Attention y el bloque MLP.

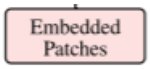

Fig. 3. Capa Embedded Patches

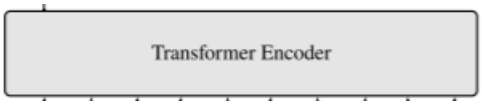

Fig. 4. Capa Encoder.

Sobre el bloque de Multi-Head Attention, podemos decir que lo primero que realiza es obtener los valores de q, k, v usando la función to_qkv, la cual aplica una transformación lineal que toma como tamaño de entrada el parámetro dimensión y como tamaño de salida la dimensión interna por 3. La función chunk divide el tensor resultante en las partes indicadas como parámetro.

Posteriormente reordena los componentes de los tensores de las componentes para que sean más "manejables".

A continuación, multiplica Q por K traspuesta, cambiando la primera y la segunda fila por la última y la penúltima, respectivamente.

Después aplica la función softmax y tras esto dropout, para evitar el overfitting.

Finalmente multiplica el resultado por V y el resultado de esto vuelve a la ordenación inicial de los tensores.

Por último, si project_out no es "false" aplica una transformación lineal con tamaño de entrada de la dimensión interna y con tamaño de salida la dimensión. Si esto no se cumple se usa la función identidad.

Fig. 5. Capa Multi-Head Attention.

Por otra parte, el bloque MLP, consta de dos capas, en este caso lineales y una función GELU. Las funciones dropout sirven para evitar el overfitting, añaden ceros de forma aleatoria con la probabilidad que le indiquemos.

Fig. 6. Capa MLP.

Y, por último, disponemos de una función de normalización, está va antes de los bloques MLP y Multi-Head Attention. Consiste en una capa de normalización, a la que se le aplica la función de activación que se le indique.

Fig. 7. Capa de Normalización.

Cabe añadir, que el número de capas que se van a usar viene dado por el parámetro depth, que nos indica la profundidad de la red.

iv. Entrenamiento de la red

Para el entrenamiento de la red hemos usado como función de pérdida o coste CrossEntropyLoss, y como optimizador Adam, con un learning rate de 1e-3.

Con respecto al test o validación, iteramos sobre el dataloader de test que habíamos creado, calculamos la predicción sobre cada imagen y la comparábamos con su label correspondiente para ver si coincidían o no.

Para la red convolucional hemos usado cinco épocas (epochs), ya que añadir más era innecesario, debido a que el porcentaje de precisión no aumentaba. Sin embargo, para la red transformer, hemos usado quince épocas, debido a que seguía aumentando la precisión en cada época.

No hemos querido añadir demasiadas épocas debido a que queríamos evitar el overfitting, es decir, que se especialice en el conjunto de entrenamiento y baje la precisión a la hora de hacer el test.

v. Creación del CSV

Para la creación del CSV, primero, guardamos el estado del modelo, a continuación, volvíamos a cargar el modelo para asegurarnos que el test se hacía con los datos de entrenamiento guardados.

Para el dataset del test, tuvimos que crear uno personalizado, ya que necesitábamos obtener el id de la imagen. Esto no nos dió muchos problemas, ya que una vez entendido cómo funcionaba fue sencillo. Cabe añadir que el dataloader usa un tamaño de batch de 1.

En cuanto a la creación del CSV en sí, primero añadimos la cabecera (Id, Category) y finalmente iteramos sobre las imágenes, calculando su predicción y añadiéndola al CSV junto con su id.

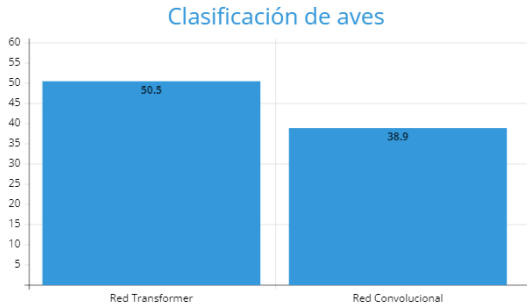

Fig. 8. Resultados obtenidos.

III. Resultados

Los experimentos que hemos realizado han sido:

- Primero hemos cargado el dataloader de entrenamiento con todas las imágenes de entrenamiento de las que disponíamos.
- Luego, hemos entrenado los modelos:
    - La red convolucional, con número de canales de entrada en la primera capa de 3, debido a que la imagen es a color, y de salida 16. La segunda capa, el valor de entrada ha sido de 16, ya que se debe corresponder con la salida de la capa anterior y como número de canales de salida de 32.
    - La red transformer, con tamaño de imagen de 64, el tamaño de patch de 8, número de clases 400, la dimensión de salida después de la transformación lineal ha sido de 256, una profundidad de 2, el número de cabezas para el algoritmo ha sido de 12, como dimensión de la capa de salida del bloque MLP 512, y, por último, ambos dropout a 0.1.
- Una vez entrenados, el convolucional con 5 épocas y el transformer con 15, los hemos guardado y hemos generado los csv correspondientes.
- Para finalizar, hemos subido los resultados a la plataforma correspondiente y hemos obtenido los siguientes resultados.

Observando estos resultados, podemos concluir que, aunque las redes convolucionales tienen un porcentaje aceptable de aciertos, las redes transformer lo superan notablemente, lo que nos indica que son más adecuada para este tipo de tareas.

IV. Conclusiones

En este trabajo hemos aprendido a hacer uso del framework de pytorch, creando redes neuronales, entrenándolas y clasificando imágenes.

Con respecto a las dos redes que hemos implementado, hemos llegado a la conclusión de que las redes convolucionales han sido una buena herramienta para la tarea de clasificación de imágenes durante muchos años, pero basándonos en los resultados que hemos obtenido, así como la información que hemos leído, podemos decir que los transformers mejoran bastantes las prestaciones de estas, por lo que el uso de las redes convolucionales se está viendo reducido.

Referencias

[1] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, 3rd ed, Pearson, 2010.

[2] Y. LeCun, Y. Bengio, G. Hinton. “Deep Learning”, Nature, vol. 521, 2015, pp. 436-444.

[3] Página web del curso IA de Ingeniería del Software. https://www.cs.us.es/cursos/iais. Consultada el 24/03/2018.

[4] Página web del tutorial de Pythorch https://pytorch.org/tutorials/beginner/basics/intro.html consultada el 20/06/2022

[5] Página web de de pytorch ejemplo red convolucional https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html consultada el 19/06/2022

[6] Blog modelos de secuencia https://www.themachinelearners.com/modelos-secuencia/ cunsultada el 23/06/2022

[7] Blog transformer https://www.themachinelearners.com/transformer/ consultada el 23/06/2022

[8] Página web con información sobre las rede convolucionales https://es.wikipedia.org/wiki/Red_neuronal_convolucional consultada el 18/06/2022 consultada el 20/06/2022

[9] Repositorio de github sobre transformer https://github.com/lucidrains/vit-pytorch/tree/main/vit_pytorch consultada el 24/06/2022