**INFORME DE MODELO**

**INTRODUCCIÓN:**

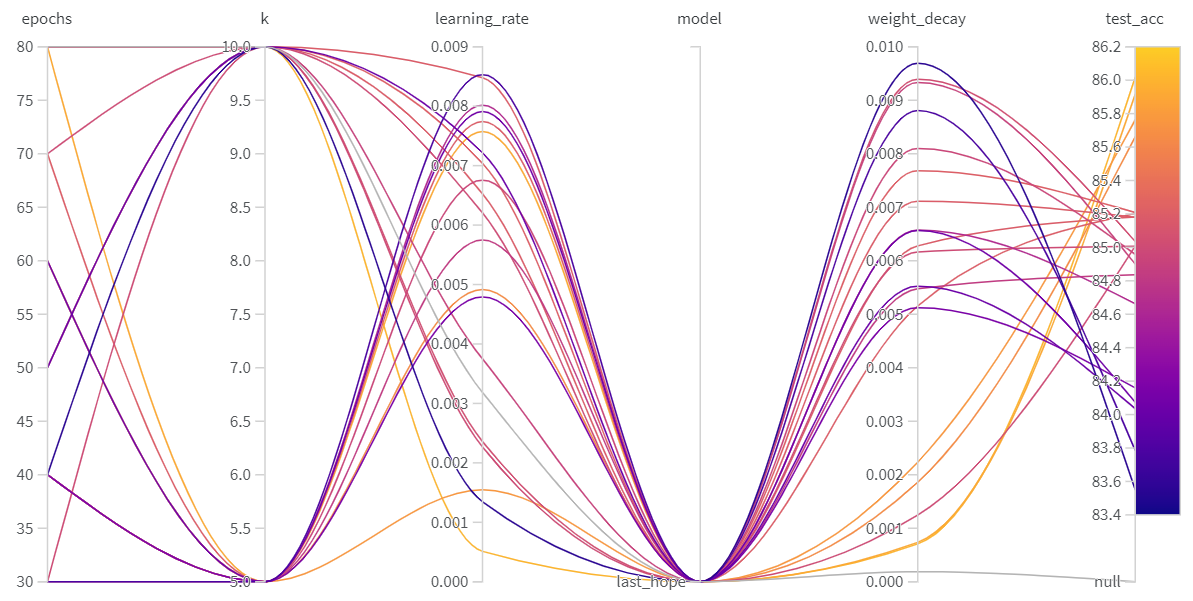
Durante el entrenamiento de las redes neuronales se han creado múltiples modelos con distintas arquitecturas y profundidad. La mayoría de modelos nos daban resultados similares, asique se ha escogido como modelo final el que hace uso de redes residuales.

Escogimos este porque nos parece una arquitectura interesante, y aunque no esté pensada para ser aplicada a redes lineales, queríamos mostrar nuestra implementación.

**ENTRENAMIENTO:**

Para el proceso de entrenamiento, hemos realizado una búsqueda bayesiana implementando la librería wandb de Weights & Biases. De esta forma podemos ver el entrenamiento en tiempo real, y realizar un seguimiento automatizado para posteriormente escoger los hiperparámetros que queramos.

Podemos encontrar los resultados de la búsqueda [aquí](https://wandb.ai/kat-lon/DDoS_sweeps/sweeps/jret03s3?nw=nwusernunezlouresiago), y a continuación tenemos una muestra de los resultados de la búsquedas.



Hemos usado un k-fold estratificado para el entrenamiento, regularización l2, y una proporción entre train\_set y test\_set de 85-15.

**OPTIMIZER:**

Como optimizar hemos escogido Adam por su facilidad de uso, ya que no requiere optimizar tantos parámetros para obtener un buen resultado. SGD podría haber funcionado mejor, pero requiere un mayor ajuste.

Además, Adam suele converger más rápido, y ajusta la tasa de aprendizaje para cada parámetro individualmente, dando resultado a un entrenamiento más eficiente. También suele ser más estable durante el entreno.

**FUNCIÓN DE PÉRDIDA:**

Por tratarse de un problema de clasificación multiclase, hemos escogido **CrossEntropyLoss** como nuestra función de pérdida. Este esta ya pensado para clasificaciones de este tipo, y penaliza fuertemente las predicciones incorrectas, especialmente si el modelo está muy seguro de su predicción. También proporciona gradientes más informativos, agilizando así la convergencia del modelo.

**ARQUITECTURA: Diagrama

Descripción generada automáticamente**

Diseñamos una red siguiendo la arquitectura de redes residuales. Como podemos ver, está constituida por las capas de entrada y salida, y 5 bloques residuales como capas ocultas.

Empleamos ReLU como función de activación, ya que es simple, computacionalmente eficiente, y ayuda a mitigar problemas de desvanecimiento del gradiente.

El bloque residual duplica la entrada, y a una de las copias la pasa por dos capas lineales, mientras que la otra pasa por la skip connection que básicamente aplica una multiplicación por el tensor identidad correspondiente. En el caso de que sean diferentes las capas de entrada y de salida, debe cambiarse el tamaño del tensor. Lo pasaremos por una capa sin BIAS y sin función de activación para lograrlo. Al final, se agregan las salidas de la capa 2 y de la skip connection, se les aplica la función de activación, y se retorna.

Hemos diseñado la red para ser modular, por si se quisiese hacer una búsqueda de hiperparámetros sobre el modelo. Podremos cambiar el tamaño de las capas y el número de las mismas, facilitando así la búsqueda de un modelo eficiente.