## Presentación de Diseño de modelo

### Objetivo

Evaluar la variable objetivo que mejor pueda estimar el nivel de engagement de diferentes puntos de interés (POIs). Para ello, se definió una métrica llamada puntuación de interacción (*Interaction Score*), calculada como la proporción de interacciones positivas (la suma de *Likes* y *Bookmarks*) respecto al total de interacciones.

Dado que el valor a estimar se encuentra en un rango entre 0 y 1, se adopta un enfoque de análisis de clases para el modelo de *deep learning*. Este enfoque utiliza probabilidades, donde de los 10 posibles valores clasificados se selecciona el más alto como la predicción final.

### Análisis y exploración de los datos. (app_features.py)

Su objetivo es analizar los datos tabulares disponibles para el estudio, identificando posibles problemas y características clave. Durante este análisis, se examina la presencia de valores nulos, la distribución de los datos y otros patrones relevantes.

Los hallazgos principales incluyen:

- El dataset no contiene valores nulos, lo que facilita su procesamiento.
- Las distribuciones de las variables **Likes**, **Dislikes** y **Bookmarks** muestran una tendencia sesgada con colas largas.

Dado este comportamiento, se recomienda aplicar transformaciones adecuadas para minimizar el impacto de los valores extremos. Una opción viable es el escalado logarítmico, que puede ayudar a normalizar las distribuciones y mejorar el rendimiento de los modelos posteriores.

Entre las variables del dataset se encuentran **locationLon** y **locationLat**, que representan coordenadas geográficas en formato numérico. Para darles un mayor significado en el contexto del análisis y facilitar la interpretación de los datos en el modelo, se decidió crear una nueva columna llamada **geo_cluster**.

Esta columna se genera aplicando el algoritmo de agrupamiento **K-means**, cuyo objetivo es identificar grupos de ubicaciones geográficas con patrones similares de interacción. De esta manera, se busca asociar patrones de *engagement* con ubicaciones específicas, lo que podría proporcionar insights clave para el análisis y la optimización del modelo.

Para determinar el número óptimo de clústeres (**n_clusters**), se emplea el **método del codo** (*Elbow Method*). Este método evalúa la suma de las distancias cuadradas dentro de los clústeres (*inertia*) en función del número de clústeres, seleccionando el punto donde la tasa de disminución comienza a estabilizarse, es decir, el "codo". Este enfoque asegura que el número de clústeres elegido equilibre la complejidad del modelo y la representatividad de los datos.

Se decidió excluir del estudio las siguientes características: 'tags', 'main_image_path', 'id', y 'shortDescription. Estas variables no aportan valor significativo al análisis o no son relevantes para los objetivos del modelo, ya sea por su naturaleza no estructurada o por su irrelevancia directa en la estimación del engagement.

En cuanto a la columna categories, que consiste en listas de categorías asociadas a cada observación, se aplicó un análisis más detallado. Primero, se creó un índice único para cada categoría presente en las listas, asignando un identificador numérico a cada categoría distinta. Este índice se utilizó para representar las categorías de manera consistente y estructurada.

A partir de estos índices, se generaron representaciones vectoriales mediante la técnica de embeddings. Este enfoque permite capturar relaciones latentes entre las categorías, asignando vectores densos en un espacio multidimensional. Los embeddings no solo reducen la dimensionalidad del problema, sino que también permiten que el modelo capture similitudes semánticas entre categorías, lo cual es crucial para mejorar la capacidad predictiva del modelo

**Desarrollo de Clase ProcessDataset y DataBuilder (app_dataset.py)**

Se diseñó una arquitectura basada en clases para estructurar y procesar los datos de manera eficiente. Esta arquitectura incluye una clase principal denominada **ProcessDataset**, encargada del preprocesamiento de los datos. Su propósito principal es preparar las variables **features** y **labels**, siguiendo las definiciones establecidas en el estudio.

Detalles de la implementación:

1. Variable features:
    - Se genera eliminando columnas irrelevantes.
    - Se aplica escalado a las características dependiendo de su tipo:
        - Se construyen dos listas: log_columns y minmax_columns, que agrupan las columnas que requieren transformaciones logarítmicas y escalado Min-Max, respectivamente.
        - Para el escalado logarítmico, se utiliza np.log1p, lo que asegura estabilidad frente a valores pequeños o nulos.
        - Para el escalado Min-Max, se implementa un modelo de la clase MinMaxScaler.
2. Variable labels:
    - Se construye según la definición del problema, asegurando consistencia con las especificaciones del modelo.
3. Método preprocess:
    - Este método realiza el procesamiento adaptado al tipo de conjunto de datos:

        - Si el conjunto es de entrenamiento (training):
            - Se aplica el algoritmo K-means para generar clústeres en la columna geo_cluster.
            - Se realiza el escalado de las características utilizando las listas definidas (log_columns y minmax_columns).
        - Si no es de entrenamiento (testing o validación):
            - Los modelos previamente entrenados (como K-means y MinMaxScaler) se cargan directamente para garantizar consistencia en el procesamiento.
4. Método __getitem__:
    - Este método permite acceder a un índice específico del conjunto de datos.
    - Devuelve una tupla que contiene:
        - La instancia de la imagen asociada al índice.
        - Las características tabulares preprocesadas (*features*).
        - La etiqueta correspondiente (*label*).

Por otra parte, contamos con la clase **DataBuilder** que controla el flujo de la clase **ProcessDataset** con la incorporación de varios métodos que se encarga de retornar los dataloader.

**Hiperparámetro Optimización (app_hyperparam.py)**

Se implementa un marco automatizado para la optimización de hiperparámetros de un modelo de aprendizaje profundo utilizando Optuna. Su propósito es encontrar la mejor combinación de hiperparámetros que minimice la pérdida de validación, garantizando un rendimiento óptimo del modelo.

**Training (app_training.py)**

Diseñada para gestionar tanto datos tabulares como imágenes, realizando el entrenamiento, validación, y guardado del modelo con el mejor rendimiento. Su método principal es run donde se recoge las siguientes:

- Implementa el bucle de entrenamiento y validación durante un número definido de épocas (num_epoch).
- Entrenamiento por época:
    - Activa el modo de entrenamiento del modelo (self.model.train()).

    - Realiza pasos de optimización iterando sobre el conjunto de datos de entrenamiento.
    - Calcula la pérdida promedio y la precisión en entrenamiento.

- Validación por época:
    - Cambia al modo de evaluación (self.model.eval()).
    - Calcula la pérdida promedio y la precisión en el conjunto de validación, sin realizar ajustes en los pesos.
- Almacenamiento de métricas:
    - Registra las pérdidas y precisiones de entrenamiento y validación para cada época.
- Mejor modelo:
    - Guarda el estado del modelo con la menor pérdida de validación alcanzada durante el entrenamiento.

Guardado del modelo:

- Al finalizar el entrenamiento, guarda el mejor modelo en un archivo con el formato best_model_<val_loss>_<type_model>.pth. Esto permite recuperar el modelo más óptimo en futuras sesiones.

Progreso en tiempo real:

- Utiliza la barra de progreso tqdm para mostrar métricas de entrenamiento y validación en tiempo real, proporcionando una interfaz visual clara del estado del entrenamiento.

**Modelo (app_model.py)**

Clase NetModel diseñada para integrar el análisis de imágenes y datos tabulares. La arquitectura se compone de las siguientes partes principales:

1. Análisis de imágenes:
    - Incluye tres capas convolucionales construidas con:
        - Conv2d: Para realizar convoluciones y extraer características espaciales.
        - BatchNorm2d: Para normalizar los valores intermedios y acelerar la convergencia.
        - MaxPool2d: Para reducir las dimensiones espaciales y evitar el sobreajuste.
        - Dropout2d: Para regularizar y prevenir el sobreajuste.

        - Funciones de activación: Definidas dinámicamente a través de parámetros, permitiendo flexibilidad en la configuración del modelo.

2. Análisis de datos tabulares:
    - Una capa completamente conectada (fully connected) diseñada específicamente para procesar las características tabulares.
3. Capa de combinación:
    - Combina las salidas de las capas convolucionales de imágenes con la salida de la capa tabular, integrando ambos tipos de datos para un análisis conjunto.
4. Capas finales:
    - Dos capas completamente conectadas que procesan la salida combinada.
    - La última capa contiene 10 neuronas, correspondientes al número de clases en el problema de clasificación.

La clase TransferLearning está diseñada para extraer características de las imágenes utilizando transferencia de conocimiento y ajuste fino (*fine-tuning*) basado en el modelo EfficientNet B0. Su implementación permite adaptar un modelo preentrenado a los datos específicos del conjunto actual, siguiendo estos pasos principales:

1. Transferencia de aprendizaje:
    - Se utiliza EfficientNet B0, un modelo preentrenado en grandes conjuntos de datos, como ImageNet.
    - Las últimas 5 capas del modelo se congelan para preservar las características aprendidas durante el entrenamiento original. Esto permite que el modelo procese el nuevo conjunto de datos sin alterar completamente su estructura preentrenada.
2. Extracción de características:
    - Se incluye un extractor de características (feature_extractor) que aplana la salida del modelo utilizando nn.Flatten(), transformando las características convolucionales en un vector de características para su integración con los datos tabulares.
3. Procesamiento de datos tabulares:
    - Se añaden dos capas completamente conectadas (fully connected) diseñadas para procesar las características tabulares.
4. Capa de combinación:
    - Combina la salida del extractor de características de imágenes con la salida de las capas de datos tabulares, integrando ambos tipos de información para un análisis conjunto.
5. Capa de salida:

- Una capa final compuesta por 10 neuronas, correspondiente al número de clases en el problema de clasificación

**Entrenamiento y Optimización (app_main.py)**

Se implementó la sintaxis necesaria para cargar el conjunto de datos de entrenamiento y obtener su correspondiente instancia de *loader*. A partir de esto, se creó una instancia de la clase HyperparameterOptimization para optimizar el valor de la pérdida de validación. Durante la ejecución, se observó que la pérdida disminuía progresivamente, alcanzando su mejor valor en el ensayo 14, con una pérdida de validación de 0.57149.

Sin embargo, al intentar replicar el entrenamiento, los resultados obtenidos no fueron consistentes. Esto llevó a implementar una semilla fija para garantizar la replicabilidad de los resultados, no solo en la partición de los conjuntos de datos (ya configurada previamente), sino de forma general en todos los aspectos del entrenamiento, incluyendo las transformaciones aleatorias aplicadas a las imágenes. Esta medida permitió tratar la aleatoriedad de manera determinista.

Durante el proceso de entrenamiento, se realizaron ajustes manuales en hiperparámetros clave, como el dropout, el número de filtros en las capas convolucionales, y el número de neuronas en las capas tabulares, con el objetivo de mitigar el sobreajuste detectado. A pesar de los esfuerzos, los resultados no alcanzaron los niveles esperados.

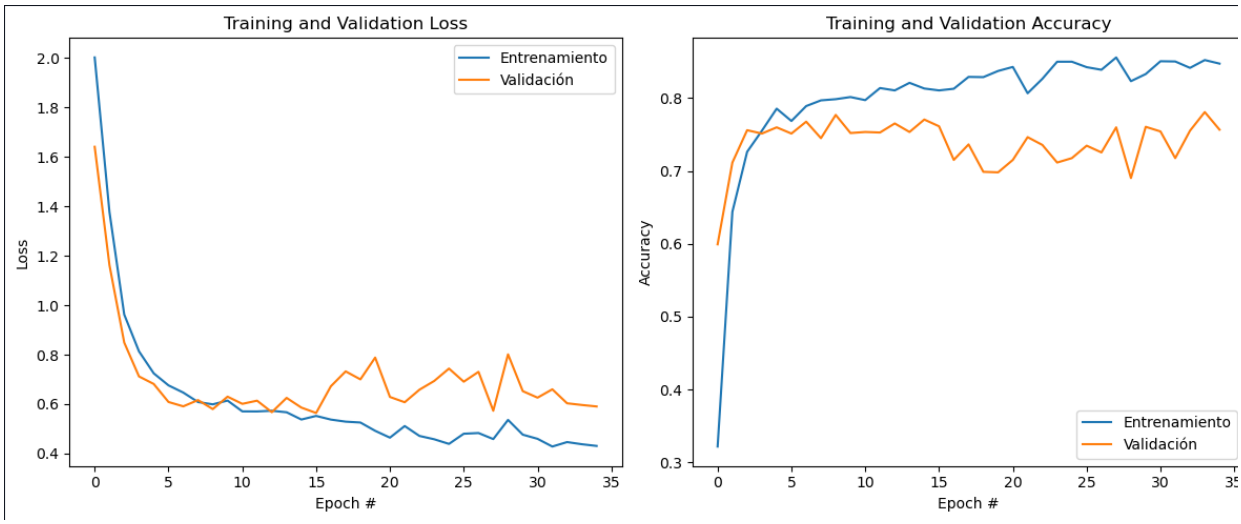
Dado este escenario, se decidió emplear la clase TransferLearning, aprovechando su capacidad para extraer características relevantes a partir de modelos preentrenados. Este enfoque prometía potenciar significativamente el rendimiento del modelo, abordando las limitaciones detectadas en las iteraciones previas.

**Nota:** Las imágenes de entrada tienen un tamaño inicial de 128 x 128, pero dado que el modelo EfficientNet fue preentrenado con imágenes de 224 x 224, se decidió aplicar un resize a las imágenes para adaptarlas al tamaño esperado por el modelo.

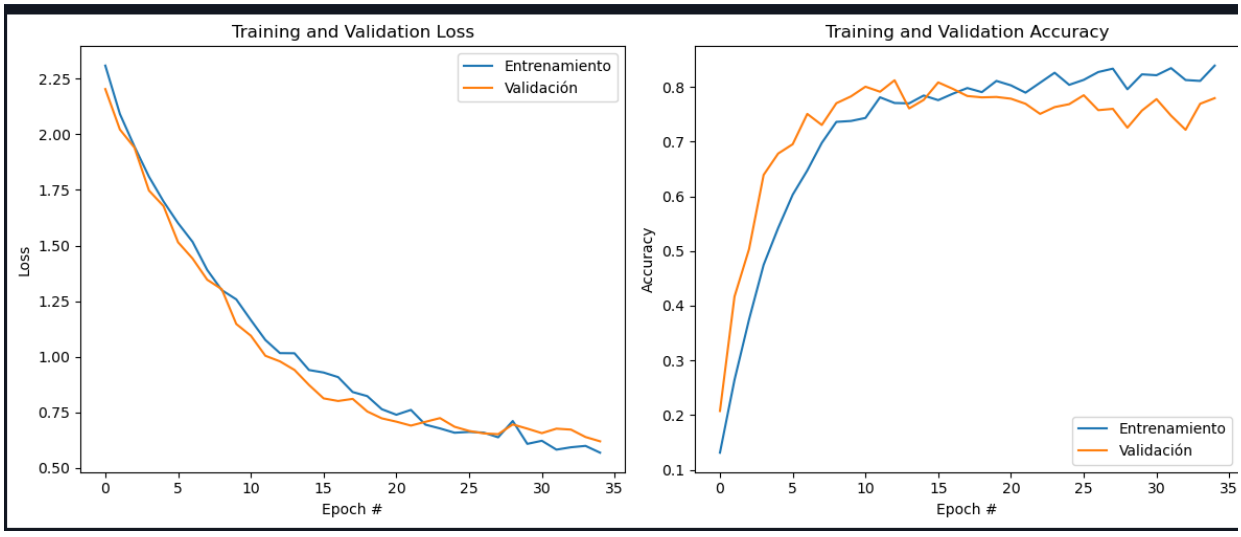
Por lo tanto, si se desea probar el modelo NetModel, es necesario ajustar los parámetros de transformación en el archivo app_dataset.py, específicamente en el método build de la clase DataBuilder, asegurándose de configurar correctamente el tamaño de las imágenes en la sección de transform.

El entrenamiento con el modelo B TransferLearning mostró resultados prometedores. Entre los valores obtenidos se destacan: **train_accuracy = 0.853**, **train_loss = 0.423**, **val_accuracy = 0.811**, y **val_loss = 0.516**, así como otra combinación significativa con **train_accuracy = 0.806**, **train_loss = 0.624**, **val_accuracy = 0.816**, y **val_loss = 0.576**. Estos resultados se pueden observar en la salida del estudio hp_TransferLearning_12, registrado en Optuna.

Un aspecto interesante observado durante el análisis es que la menor pérdida no siempre coincide con la mejor precisión, como se evidenció en los valores mencionados. Este comportamiento subraya la importancia de considerar múltiples métricas al evaluar el rendimiento de un modelo. A continuación, se presentan imágenes que ilustran estos resultados:

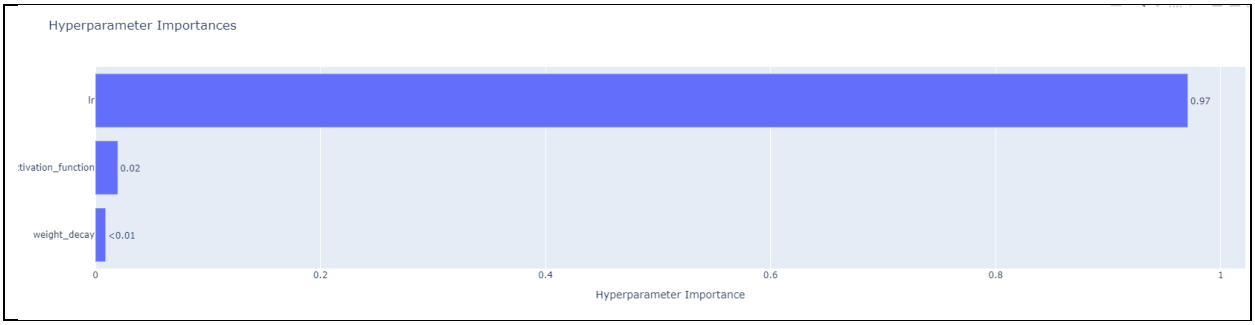



Estos valores (**train_accuracy = 0.853**, **train_loss = 0.423**, **val_accuracy = 0.811**, **val_loss = 0.516**) corresponden a la mejor configuración de parámetros obtenida durante el estudio de hiperparámetros. Sin embargo, al replicar el entrenamiento con esta configuración, los resultados exactos no se obtuvieron debido a factores como la aleatoriedad inherente al proceso de entrenamiento. A pesar de ello, se logró una aproximación razonable con los siguientes valores: **train_accuracy = 0.847**, **train_loss = 0.431**, **val_accuracy = 0.757**, y **val_loss = 0.590**.

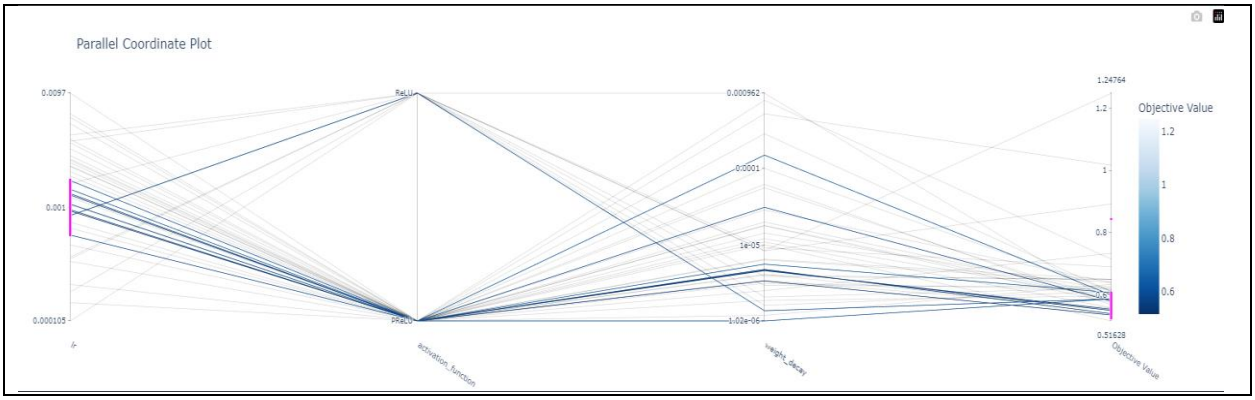



En esta imagen se corresponde con los parámetros del estudio que genera un train**_accuracy = 0.806**, **train_loss = 0.624**, **val_accuracy = 0.816**, y **val_loss = 0.576** en la replicación del training

se obtiene **train_accuracy=0.839, train_loss=0.568, val_accuracy=0.779, val_loss=0.619** que tiene mayor perdida, pero genera un Accuracy mejor.



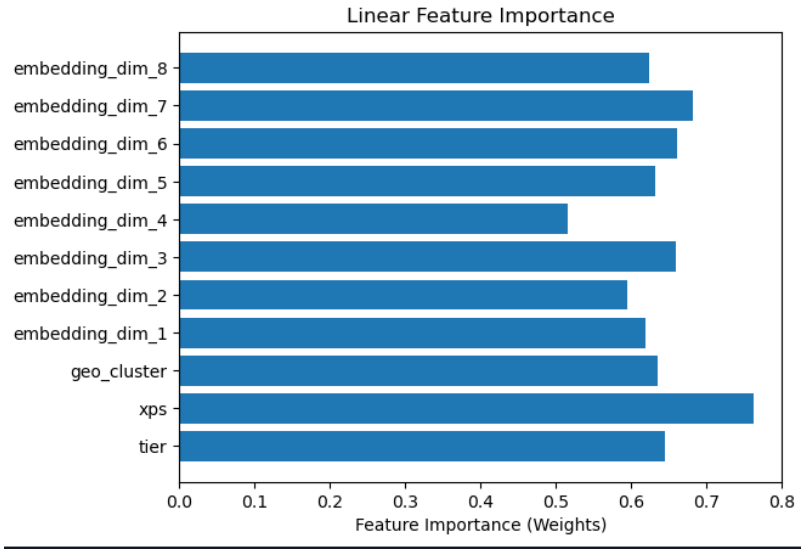

Dentro de la importancia de los parámetros el learning rate ocupa el 97% de importancia y en la figura Parallel Cordinate se puede observar que los mejores valores de perdida en validación se encontraban alrededor de lr=0.001.




**Evaluación y Análisis (app_test.py)**

Con el modelo guardado best_model_05183_B, identificado como el mejor modelo de acuerdo a las operaciones con Optuna, se evalúa el conjunto de validación, logrando una precisión (Accuracy) de 0.788, mientras que en el conjunto de prueba (Test) se obtiene una precisión de 0.75. Es importante destacar que, aunque el modelo best_model_0.5343_B, asociado a una menor pérdida, no fue el mejor en validación, muestra un mejor desempeño en el conjunto de prueba, superando al modelo seleccionado.

Dentro de las Importancias lineales se encuentran las siguientes:




El modelo presenta un amplio margen de mejora. Entre las acciones que podrían implementarse se incluyen las siguientes:

1. Incorporar a la parametrización de la clase HyperparameterOptimization indicadores adicionales como dropout, número de neuronas, y acotar el rango del learning rate **(lr)** a valores cercanos a 0.001 para una búsqueda más precisa.
2. Realizar un análisis basado en redes neuronales para la creación de clusters de geolocation, como alternativa al uso de KMeans, buscando una segmentación más adaptada al modelo.
3. Evaluar la cantidad óptima de capas a congelar en el modelo, identificando la configuración que brinde el mejor equilibrio entre tiempo de entrenamiento y rendimiento del modelo. EfficientNet
4. Experimentar con técnicas de data augmentation específicas para imágenes que puedan enriquecer la variabilidad del conjunto de entrenamiento, como rotaciones, recortes o ajustes de brillo.
5. Aplicar estrategias de fine-tuning adicionales para EfficientNet, permitiendo un ajuste más específico a las características de los datos tabulares y visuales.
6. Incorporar métricas adicionales durante la optimización, como F1-score o AUC, para evaluar el desempeño del modelo más allá de la precisión.