**EMBEDDING MODELS**

Pregunta que me faltó responder en el análisis de los datos:

8. Verificar si el modelo es robusto o sensible a outliers

Preguntas que me hizo el mtro Rafael:

a) ¿vas a entrenar un modelo desde cero o vas a tomar uno entrenado?

b)      ¿cuál es la arquitectura de la red para DL?

c)       ¿En qué capas vas a tomar los valores pre-entrenados?

d)      ¿vas a evaluar lo nuevos especímenes hasta tal capa?

e)      ¿Cómo se “resuelve” o “evalúa” la “similaridad” o “parecido” (métrica)?

f)        ¿Con qué herramientas lo tienes pensado llevar a cabo?

# ----------------------------------

INVESTIGACIÓN:

Modelos que hasta ahora he investigado que son open source:

1. OpenL3 [[GitHub](https://github.com/marl/openl3?tab=readme-ov-file)][[Doc](https://openl3.readthedocs.io/en/latest/tutorial.html#introduction)]:
   1. Los tipos/formatos de imágenes que soporta este modelo son aquellos que soporta a la vez la libreria *scikit-image* (PNG, JPEG, etc.)
   2. El modelo genera un array de dimensión D (toma valores 8192 o 512) a partir de una sola imagen
   3. Si se desea pasar una colección de imágenes se debe pasar como un *numpy array.* El modelo va a regresar un array de dimensión NxD donde N es el número de embeddings creados (1 por cada imagen) y D es la dimensión del embedding (8192 o 512)
   4. Por default, cada que se llama al modelo, el archivo correspondiente al modelo es cargado desde el disco. Para evitar esto cuando se está trabajando con múltiples imágenes podemos cargar el modelo una sola vez y después pasarlo como parámetro
   5. Se pueden procesar varias imágenes a la vez mediante arrays y después éstos se guardan dentro de la misma carpeta donde están las imágenes
   6. Este modelo utiliza dos tipos de audio frontends:
      1. Kapre (GPU): Implementa el procesamiento de audio/media como capas de redes neuronales y procesa el audio frontend como parte de la inferencia de su modelo. Cuando un GPU está disponible entonces el procesamiento sucede en ese GPU
      2. Librosa (CPU): Te permite paralelizar operaciones computacionales a través de múltiples CPUs.
   7. Te permite realizar las operaciones con la terminal.
2. Contrastors [[GitHub](https://github.com/nomic-ai/contrastors?tab=readme-ov-file)]:
   1. Este proceso utiliza el método *Flash Attention* para un mejor procesamiento de los datos y, por consiguiente, un entrenamiento de modelo más eficiente
   2. Permite el entrenamiento del modelo a través de múltiples GPUs
   3. Utiliza el método de Gradient Cache para poder hacer procesamiento de los datos con el mínimo uso de memoria posible. Este método incluso puede llegar a utilizar un solo GPU
   4. Es un modelo pre entrenado de tipo Masked Language Modeling (MLM)
   5. Utiliza el método **Matryoshka Representation Learning (MRL)** el cual codea la información en diferentes granularidades y permite un solo embedding que se adapta a las diferentes limitantes de las tareas de embedding. Esto permite no utilizar muchas dimensiones en los vectores resultantes para el entrenamiento del modelo [[GitHub](https://github.com/RAIVNLab/MRL)]
   6. Utiliza CLIP (Contrastive Language-Image-Pre-Training), el cual es una red neuronal entrenada en una variedad de pares de imágenes. Se utiliza para predecir textos dada una imagen [[GitHub](https://github.com/OpenAI/CLIP?tab=readme-ov-file)]
   7. Utiliza LiT (Locked-image Tuning) el cual enseña a un modelo de texto a leer buenas representaciones a partir de un modelo de imágenes pre entrenada para nuevas tareas
   8. Soporta modelos ViT (como los modelos [timm](https://huggingface.co/models?library=timm&sort=trending)
3. Sports [[GitHub](https://github.com/roboflow/sports?tab=readme-ov-file)]:
   1. Contiene una librería grande modelos y notebooks que nos pueda ayudar a utilizar diferentes métodos para nuestros propósitos
   2. Uno de nuestro interés es un método de Image Embeddings [[Colab](https://colab.research.google.com/github/roboflow-ai/notebooks/blob/main/notebooks/image_embeddings_analysis_part_1.ipynb)]
   3. Modelo de clasificación de imágenes [[Colab](https://colab.research.google.com/github/roboflow-ai/notebooks/blob/main/notebooks/dinov2-classification.ipynb)]
   4. Modelo de inferencia (Roboflow) diseñado para simplificar el desarrollo y aplicación de modelos de vision computacional para realizar detección de objetos, clasificación y segmentación mediante modelos como CLIP, Segment Anything y YOLO World
4. ResNet [[GitHub](https://github.com/pytorch/vision/blob/main/torchvision/models/resnet.py)][[Doc](https://pytorch.org/vision/stable/models/resnet.html)]:
5. VGG [[GitHub](https://github.com/pytorch/vision/blob/main/torchvision/models/vgg.py)][[Doc](https://pytorch.org/vision/stable/models/vgg.html)]:
6. Inception V3 [[Github](https://github.com/pytorch/vision/blob/main/torchvision/models/inception.py)][[Doc](https://pytorch.org/vision/stable/models/inception.html)]:
7. EfficientNet [[GitHub](https://github.com/pytorch/vision/blob/main/torchvision/models/efficientnet.py)][[Doc](https://pytorch.org/vision/stable/models/efficientnet.html)]:
8. ViT [[GitHub](https://github.com/lukemelas/PyTorch-Pretrained-ViT)]:

Página de GitHub con lista de modelos:

<https://github.com/topics/image-embeddings>