Semi-supervised stamps anomaly detection for Zwicky Transient Facility data

(PI: Konstantin Malanche)
phy240031p
Pittsburgh Supercomputing Center (PSC)
Documentación

LINK a este documento:
https://docs.google.com/document/d/1I8Ouj12LCZo40jEchqyez0ByAVBiomrNamo3gFm5CJo/edit?usp=sharing

A continuación se describen consideraciones para el trabajo con los archivos del proyecto de detección de anomalías desarrollado por el equipo de Konstantin Malanche.

ssh frannou@bridges2.psc.xsede.org
cd /ocean/projects/phy240031p/shared

Datos

Se requieren imágenes astronómicas para el proyecto. Estas provienen del repositorio de ZTF, desde donde se pueden descargar de forma directa en formato .avro. En el directorio del proyecto en Ocean se pueden encontrar ya empaquetados en formato .tfrecord todos los datos empleados en la carpeta:

```
TFRecords
```

Total: 102.494 .tfrecord Aprox 10.000GB = 10TB
Alertas totales = 209.907.712
Cada .tfrecord contiene 2048 alertas .avro descargadas del repositorio de ZTF.
Para replicar entrenamiento de manera local, se usa 1% de datos (1.024 TFRecords - 2.048.000 Alertas, Aprox 100GB)
Data descargada: data_batch_220_* aprox hasta data_batch_224_*
data_batch_380_2* aprox hasta data_batch_381_*
Para descargar alertas actualizadas desde ZTF y comprimirlas como .tfrecords, **modificar** script 0.SIMPLE_ALL_DOWNLOAD.py

Cantidad de .tfrecords usados en los scripts originales:
scripts SiSAD_DEEPER & SiSAD_MULTIE_GPU usan 100.000 TFRecords
script SiSAD_ATTENTION_UNET usa 10.000 TFRecords
script anomalies usa 1000 TFRecords
script de Similarity_search toman la carpeta entera de TFRecords

Ver el contenido de los TFRecords

Para ello, se puede usar el jupyter notebook **OpenTFRecords_TEST.ipynb**
Se puede observar el contenido de un .tfrecord en específico o cargar los primeros *n* deseados y mostrarlos junto a su Bogus_Score

Modelo

Los modelos entrenados se encuentran en la carpeta:

```
wandb
```

Específicamente, el best-model obtenido para el resto del proceso se encuentra en:

`wandb/run-20240707_174614-pmxt3b1m/files/model-best.h5`

De forma local, se encuentra en:

`'/home/nicosepulveda/astro/bridge/model/bridge_best_model/model-best.h5'`

El cual se obtuvo corriendo el script:

```
2.SiSAD_NETWORK_1_MULTIPLE_GPUs.py
```

Con los parámetros:

```
config: {'batch_size': 2048, 'model_name': 'U-Net Simple', 'epochs': 100,
'init_learning_rate': 0.0001, 'lr_decay_rate': 0.1, 'optimizer': 'adam',
'loss_fn': 'mean_squared_error', 'earlystopping_patience': 10,
```

CAPAS de la red UNET:

Input Layer

1. **Input:** imágenes de tamaño 48,48,2

Encoder

2. **Conv2D + Activation (c1)**: convolución 32 filtros 3x3 ReLU, He, Dropout 0.1
3. **MaxPooling2D (p1)**
4. **Conv2D + Activation (c2)**: convolución 64 filtros 3x3 ReLU, He, Dropout 0.1
5. **MaxPooling2D (p2)**
6. **Conv2D + Activation (c3)**: convolución 128 filtros 3x3 ReLU, He, Dropout 0.2
7. **MaxPooling2D (p3)**
8. **Conv2D + Activation (c4)**: convolución 256 filtros 3x3 ReLU, He, Dropout 0.2
9. **MaxPooling2D (p4)**

Bottleneck

10. **Conv2D + Activation (bn)**: dos convoluciones 512 filtros 3x3

Decoder

11. **Conv2DTranspose (u6)**: transposición de convolución 256 filtros 2x2, skip connection c4
12. **Conv2D + Activation (c6)**: dos convoluciones 256 filtros 3x3
13. **Conv2DTranspose (u7)**: transposición de convolución 128 filtros 2x2, skip connection c3
14. **Conv2D + Activation (c7)**: dos convoluciones 128 filtros 3x3
15. **Conv2DTranspose (u8)**: transposición de convolución 64 filtros 2x2, skip connection c2
16. **Conv2D + Activation (c8)**: dos convoluciones 64 filtros 3x3
17. **Conv2DTranspose (u9)**: transposición de convolución 32 filtros 2x2, skip connection c1
18. **Conv2D + Activation (c9)**: dos convoluciones 32 filtros 3x3

Output Layer

19. **Conv2D (outputs)**: Convolución 1 filtro 1x1 lineal

Conv2D: filtros convolucionales sobre imagen para extracción de características locales según tamaño de filtro.

Conv2DTranspose: aumentar dimensiones espaciales de imagen (upsampling) según tamaño del filtro

MaxPooling2D: reducir dimensiones espaciales extrayendo características importantes (valores máximos) de región de filtro (2x2). Comprime información

Skip connections: conecta capas del encoder con el decoder de forma directa.

Activación ReLU: función no lineal. Establece en 0 entradas negativas, para aprendizaje de relaciones no lineales en los datos. Usado en conjunto con kernel_initializer='he_normal' para que activaciones tengan varianza similar producto de activación ReLU que puede establecer en 0 muchos valores.
Dropout: prevención de sobreajuste, desactivando aleatoriamente un conjunto de neuronas durante el entrenamiento (por ejemplo 0.1->10%, 0.2->20%)

Métricas monitoreadas:
KLDivergence (KLD): medida de la diferencia entre dos distribuciones de probabilidad. Puede usarse para medir la diferencia entre la distribución de la salida de la red y la distribución de las etiquetas verdaderas.
Mean Absolute Error (MAE): diferencia promedio entre predicciones del modelo y valores reales. Más robusta frente a errores individuales muy grandes que métricas como MSE penalizan más. Puede usarse para medir qué tan cerca están las predicciones de los valores reales.
Mean Absolute Percentage Error (MAPE): Porcentaje de error respecto a los valores reales. Similar al anterior, pero de más fácil interpretación.
Mean Squared Error (MSE): medida de los cuadrados de las diferencias entre predicciones y valores reales, penalizando errores más grandes en comparación a MAE, siendo sensible a valores atípicos. Útil si se desea que el modelo se enfoque en minimizar errores más grandes.

Normalización y extracción de imágenes:
Mismo procedimiento seguido por scripts: extract_bottleneck_features, SiSAD, anomalies y Similarity(sin considerar metadata)

- Ingresan tripletas de 63x63, cualquier NaN se reemplaza por 0s
- Science y Reference normalizadas de 0 a 1 usando min y max de combinación img1,img2
- Difference se normaliza a parte entre -1 y 1, usando min y max de la diferencia entre img1 e img2
- Crop a 48x48 centrado de las tres imágenes
- Se entregan como resultado dos imágenes: stack [img1, img2] , difference
- Trainset (70%), valset (15%), testset (15%), usando como data los archivos .tfrecord (ya sean todas las imágenes o un subconjunto de ellas, como 100.000, 10.000, etc).

Scripts disponibles en Ocean:

extract_bottleneck_features

Extrae características del dataset usando un modelo previamente entrenado. Utiliza la capa **pre_bottleneck** del modelo

Genera archivo val_bottleneck_features.h5 donde se guardan características latentes (features) extraídas por el modelo a partir del dataset de tfrecords de entrada. Se obtiene un conjunto de vectores 1x512 para cada tripleta, identificada por el Object ID de la imagen, por lo que es posible posteriormente rastrear las características y las imágenes originales y realizar búsquedas de similaridad.

0.SIMPLE_ALL_DOWNLOAD.py

Funciones para descargar data desde ZTF

Extraer imágenes FITS desde archivos avro descargados

Guardar tripletas con su metadata (record) como batches TFrecord

1.SingleBatchTEST

Llama a script anterior para descargar la data desde ZTF y guardar en directorio deseado

Tamaño del batch o cantidad de tripletas/records por cada TFrecord: 2048

2.SiSAD_NETWORK

Entrenamiento del modelo U-NET. Monitoreo y guardado del modelo usando wandb. Configurar batch size a un tamaño manejable. Entrenamiento original tiene tamaño 2048. Para entrenamiento local puede ser 128 o 64.

```
# Adjust the batch size by the number of GPUs
num_gpus = strategy.num_replicas_in_sync
scaled_batch_size = int(2048) # * num_gpus)
# Configuration dictionary
config = dict(
    batch_size=scaled_batch_size,
    model_name='U-Net Simple',
    epochs=100,
    init_learning_rate=0.0001,
    lr_decay_rate=0.1,
    optimizer='adam',
    loss_fn='mean_squared_error',
    earlystopping_patience=10,
    metrics=[tf.keras.metrics.KLDivergence(),
tf.keras.metrics.MeanAbsoluteError(),
             tf.keras.metrics.MeanAbsolutePercentageError()]
)
```

Ajustar **batch_size** según sea necesario

Considerar ajustes al **lr_decay_rate** para ver comportamiento del entrenamiento

```
    def lr_scheduler(epoch, lr):
        # log the current learning rate onto W&B
```

```
        if wandb.run is None:
            raise wandb.Error("You must call wandb.init() before
WandbCallback()")
        wandb.log({'learning_rate': lr}, commit=False)
        if epoch < 7:
            return lr
        else:
            return lr * tf.math.exp(-config['lr_decay_rate'])
```

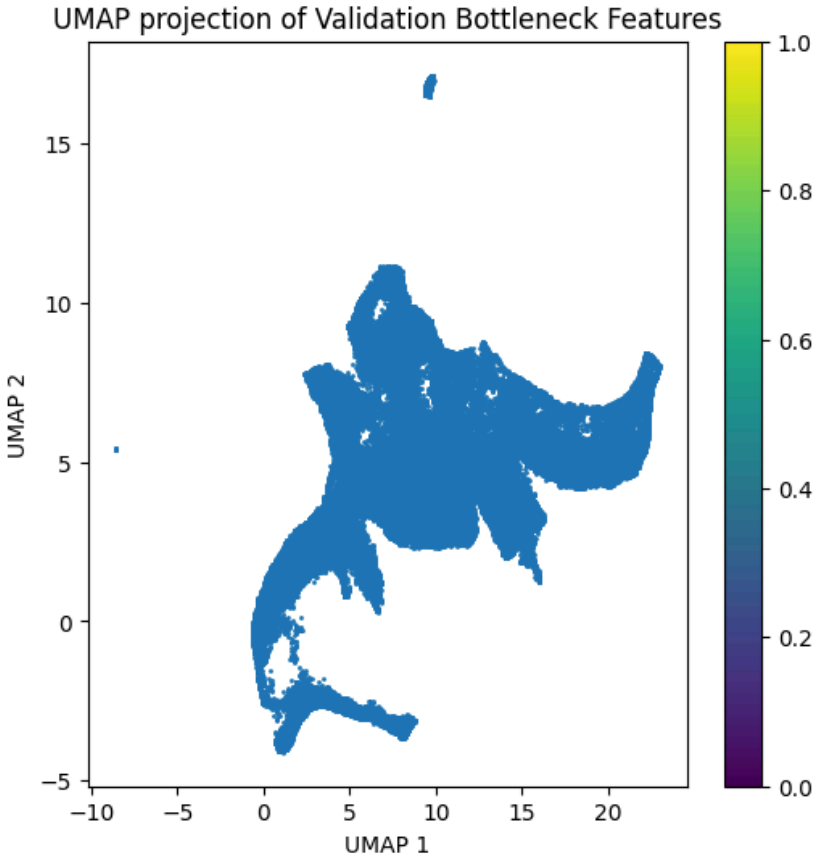
Considerar otras funciones a parte de exp para ir cambiando el learning rate

wandb-metadata.json: características del equipo corriendo la simulación
wandb-summary.json: estadísticas de parámetros del entrenamiento

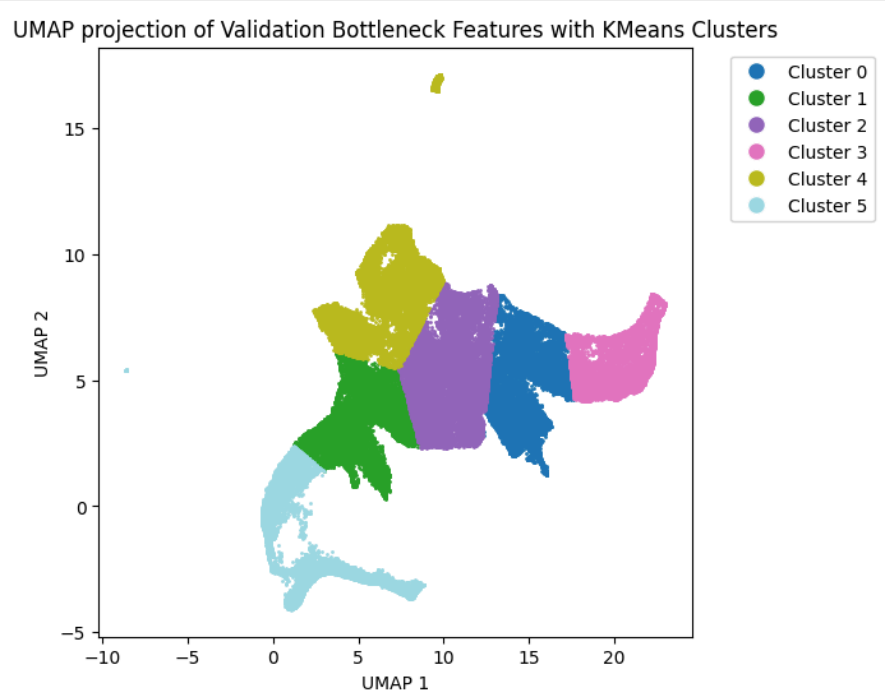
Tiempo estimado de entrenamiento: En Loki, con batch size 128, el tiempo estimado por epoch es de 13 min empleando los 1000 tfrecords disponibles.
40 epoch: aprox 9 horas

3. anomalies.ipynb

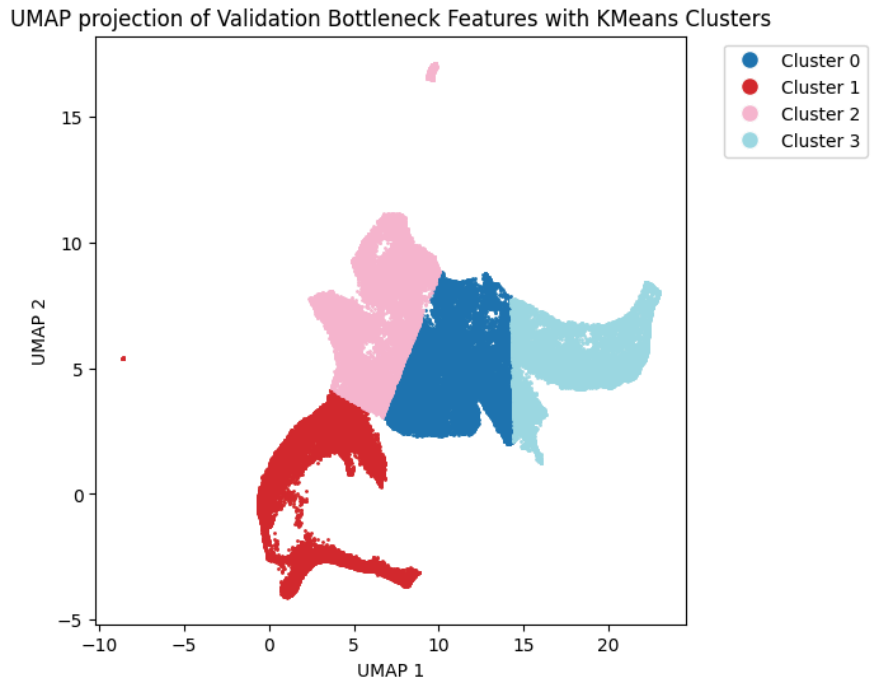
Se extrae la capa bottleneck de un modelo entrenado para usarlo como modelo extractor de características, utilizado para extraer características de las imágenes normalizadas de un conjunto de tfrecords. La última celda permite plotear gráficos y ejemplos de una imagen por cluster, dependiendo del tipo de clusterización que se ejecutó por última vez.



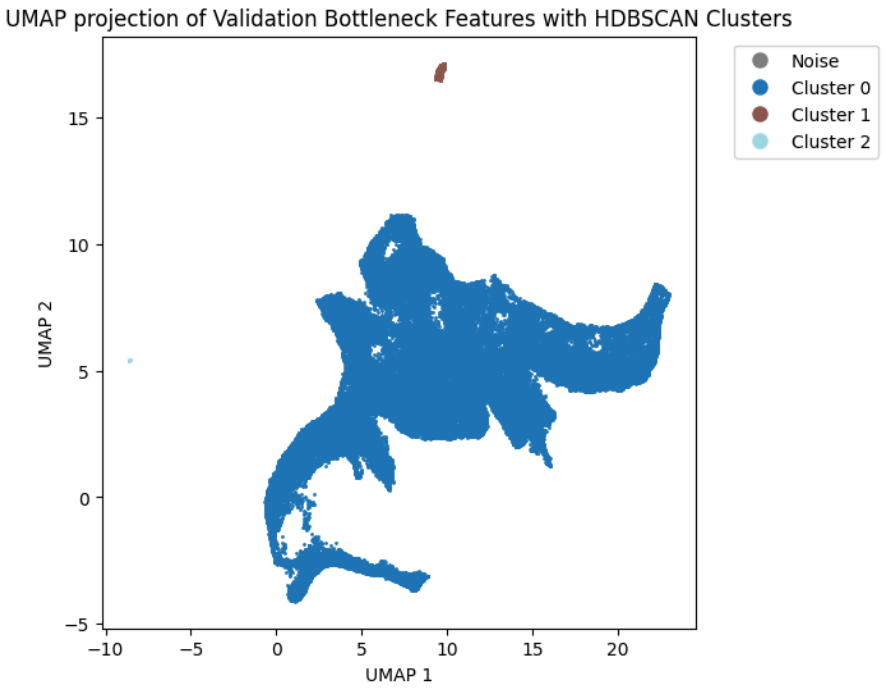
Resultado del clustering con KMeans:
Ejemplo de 6 clusters:



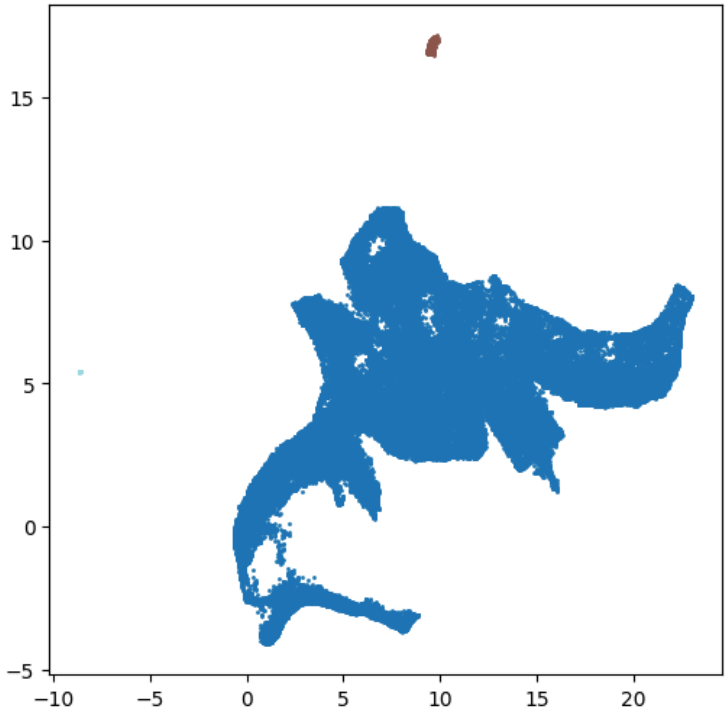
Ejemplo de 4 clusters:



Resultado del clustering con HDBSCAN



Este último gráfico debiera mostrar proyección UMAP de Validation Bottleneck Features con Extreme Noise Points, pero no se hallaron extreme noise points dentro del data set empleado para el entrenamiento local.



4.Similarity_search_ANNOY
(usa Annoy "Approximate Nearest Neighbors Oh Yeah" de Spotify)
4.Similarity_search
(usa FAISS "Facebook AI Similarity Search")
Ambos scripts tienen una función similar, pero en uno se usa ANNOY (búsqueda de similaridad más aproximada) y en otro FAISS (búsqueda más exacta)

1. Permite probar la realización de predicciones de alguna imagen fuera del data set. Se descarga una imagen desde Alerce (`input_img`) y se predice su diferencia. Se muestran luego las imágenes science, reference y difference junto a la predicción.
2. Usa espacio latente (val_bottleneck_features.h5) para crear un **índice Annoy** o **índices FAISS** para posterior búsqueda rápida.
3. Se usa `input_img` descargada para generar una predicción con el modelo. Recupera imágenes (k=10) similares en el espacio latente (vecinos más cercanos) y obtiene los metadatos de los vecinos.
4. Visualiza las tripletas con los ID obtenidos junto a la predicción para cada una de ellas.

****** RUTAS LOCALES**
Cambiar la ruta de **`tfrecord_folder_path`** por la ruta local en todos los scripts y jupyter notebook
Cambiar la ruta de **`model_path`** por la ruta local en los jupyter notebook anomalies.ipynb, Similarity_search.ipynb y Similarity_search_ANNOY.ipynb
Cambiar la ruta de **`hdf5_file`** por la ruta local en los jupyter notebook Similarity_search.ipynb y Similarity_search_ANNOY.ipynb

ANEXO: Entorno virtual y Singularity

Para ejecutar los script de entrenamiento se puede crear un entorno virtual con conda y las dependencias necesarias, o en su defecto instalar Singularity, descargar el contenedor disponible por el proyecto y utilizarlo para correr los script.

1. **Entorno virtual con conda (máquina con CUDA 11.6, cuDNN 8.4)**

```
conda create -n tf-gpu python=3.9 -y
conda activate tf-gpu
pip install tensorflow==2.11
pip install "numpy<2"
pip install matplotlib astropy scipy pandas scikit-learn wandb
pip install umap-learn hdbscan alerce faiss annoy
python3 -c "import tensorflow as tf; print(tf.config.list_physical_devices('GPU'))"
```

COMPATIBILIDAD:
(se puede verificar en https://www.tensorflow.org/install/source?hl=es-419#gpu)
equipo Loki tiene CUDA 11.6
Combinación que funcionó: tensorflow 2.11 + CUDA 11.6 + cuDNN 8.4 + python 3.9

********Instalación de cuDNN**
descargar cuDNN 8.4.1 de: https://developer.nvidia.com/rdp/cudnn-archive

```
sudo dpkg -i cudnn-local-repo-ubuntu1804-8.4.1.50_1.0-1_amd64.deb
sudo cp /var/cudnn-local-repo-ubuntu1804-8.4.1.50/cudnn-local-E2A6B588-keyring.gpg
/usr/share/keyrings/
sudo apt-get update
sudo apt-get install libcudnn8 libcudnn8-dev libcudnn8-samples
cat /usr/include/cudnn_version.h | grep CUDNN_MAJOR -A 2
```

(esto último para verificar, debiera mostrar CUDNN_MAJOR 8, CUDNN_MINOR 4, CUDNN_PATCHLEVEL 1)

Verificaciones:
1. Configuración de tensorflow:

```
python3 -c "import tensorflow as tf; print(tf.sysconfig.get_build_info())"
```

2. Versión de tensorflow:

```
pip3 list | grep tensorflow
```

3. Versión de CUDA (no necesariamente coincide con lo que muestra nvidia-msi)

```
nvcc -version
```

4. Versiones de cuDNN

```
ls /usr/local/cuda-11.6/lib64 | grep libcudnn
```

Detección de GPU:

```
python3 -c "import tensorflow as tf
print('Num GPUs Available:', len(tf.config.list_physical_devices('GPU')))
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())"
```

Realizar un cálculo para comprobar el uso de la GPU

```
python3 -c " import tensorflow as tf tf.debugging.set_log_device_placement(True) #
Create some tensors a = tf.constant([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]]) b =
tf.constant([[1.0, 2.0], [3.0, 4.0], [5.0, 6.0]]) c = tf.matmul(a, b) print(c) "
```

2. **Singularity**

Descarga del contenedor: Descargar de contenedor de tensorflow usado para la ejecución de los scripts (aprox 7,7 GB)

scp frannou@bridges2.psc.xsede.org:/ocean/containers/ngc/tensorflow/tensorflow_latest.sif .

Instalar singularity

Descargar singularity para ejecutar el script usando el contenedor tensorflow

1. Instalar dependencias necesarias para preparar la instalación

```
sudo apt-get update && sudo apt-get install -y \
    build-essential \
    libssl-dev \
    uuid-dev \
    libgpgme11-dev \
    squashfs-tools \
    libseccomp-dev \
    pkg-config
```

2. Instalar go

```
export VERSION=1.23 OS=linux ARCH=amd64 && \
    wget https://dl.google.com/go/go$VERSION.$OS-$ARCH.tar.gz && \
    sudo tar -C /usr/local -xzvf go$VERSION.$OS-$ARCH.tar.gz && \
    rm go$VERSION.$OS-$ARCH.tar.gz
```

3. Instalar una versión actualizada de singularity:

```
export VERSION=3.0.3 && # adjust this as necessary \
    mkdir -p $GOPATH/src/github.com/sylabs && \
    cd $GOPATH/src/github.com/sylabs && \
    wget
https://github.com/sylabs/singularity/releases/download/v4.2.1/singularity-ce-4.2.1.tar.gz
&& \
    tar -xzf singularity-ce-4.2.1.tar.gz && \
    cd singularity-ce-4.2.1 && \cd ..
    ./mconfig

./mconfig && \ make -C ./builddir && \ sudo make -C ./builddir install
```

4. Observación: Instalar todas las dependencias que vayan faltando a medida que se realiza el intento de instalación y ocurran fallos…

Contenedor de singularity:

Ver en:
singularity exec tensorflow_latest.sif bash / pip list
o en:
requeriments.txt dentro de carpeta model

Pueden usarse los script de bash que hay en la carpeta del proyecto para correr los modelos usando Singularity. Cambiar las en el script rutas según necesidad (por ejemplo:)

```
SCRIPT_PATH="/home/nicolas/astro/bridge/extract_bottleneck_features.py"
SCRIPT_PATH="/home/nicolas/astro/bridge/2.SiSAD_NETWORK_1_DEEPER.py"
SCRIPT_PATH="/home/nicolas/astro/bridge/2.SiSAD_NETWORK_1_MULTIPLE_GPUs.py"
SCRIPT_PATH="/home/nicolas/astro/bridge/2.SiSAD_NETWORK_2_ATTENTION_UNET.py"
CONTAINER_PATH="/home/nicolas/astro/bridge/container/tensorflow_latest.sif"
```

Correr script con screen:

Crear sesión, activar entorno, correr script
screen -S entrenamiento
conda activate tf-gpu
python 2.SiSAD_NETWORK_1_MULTIPLE_GPUs.py

(Ctrl + A, D para detach)
Reconectarse:
screen -r entrenamiento
Salir:
exit

GPU detectada por wandb:
En loki


```
"gpu_nvidia": [
  {
    "name": "NVIDIA GeForce RTX 2070 SUPER",
    "memoryTotal": "8366784512",
    "architecture": "Turing"
  }
],
"cudaVersion": "11.4"
```


En local IdeaPad


```
"gpu_nvidia": [
  {
    "name": "NVIDIA GeForce GTX 1650 Ti",
    "memoryTotal": "4294967296",
    "cudaCores": 1024,
    "architecture": "Turing"
  }
],
"cudaVersion": "12.5"
```


Conectarse al PSC bridge
Conectarse por
ssh frannou@bridges2.psc.xsede.org
cd /ocean/projects/phy240031p/shared
scp frannou@bridges2.psc.xsede.org:/ocean/projects/phy240031p/shared/<filename> .