



UNIVERSIDAD SANTO TOMÁS

PRIMER CLAUSTRO UNIVERSITARIO DE COLOMBIA

SECCIONAL TUNJA

VIGILADA MINEDUCACIÓN - SNIES 1732



Acreditación Institucional
Internacional

OTORGADA POR EL IAC CINDE ACUERDO 55 DEL 6 DE MAYO-VIGENCIA 5 AÑOS



ACREDITACIÓN
INSTITUCIONAL
DE ALTA CALIDAD
MULTICAMPUS

Vigencia por seis años



QS STARS
RATED FOR EXCELLENCE



ISO 9001
Icontec
SC4289-1



CERTIFIED
IONet
MANAGEMENT SYSTEM



UNIVERSIDAD SANTO TOMÁS
PRIMER CLAUSTRO UNIVERSITARIO DE COLOMBIA
SECCIONAL TUNJA

VIGILADA MINEDUCACIÓN - SNIES 1732

Faculty: systems engineer

Course: Deep Learning

Topic: CNN-visión por computadora (development environment)

Professor: Luis Fernando Castellanos Guarín

Email: Luis.castellanosg@usantoto.edu.co

Phone: 3214582098



CONTENIDO

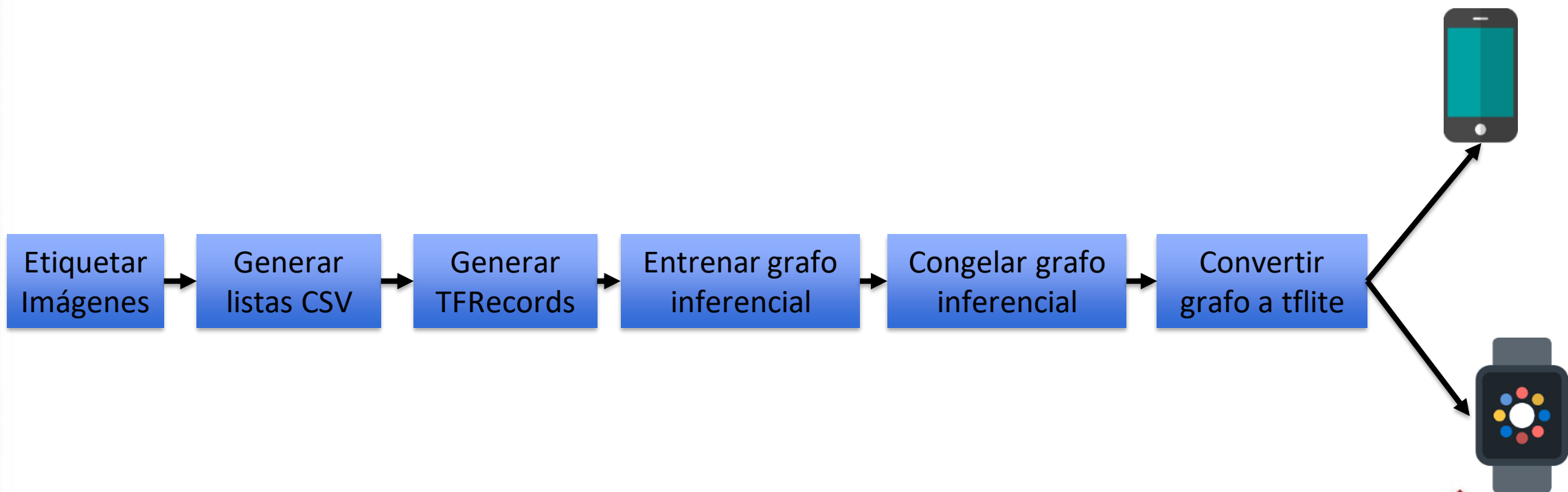
1. *Teoría e historia.*
2. *Crear Dataset de entrenamiento y pruebas*
3. **Preparando el ambiente para entrenamiento**
4. Realizar entrenamiento (usando tensorflow 1.15 con redes CNN)
5. Congelar/exportar el modelo de inferencia(model.ckpt)
6. Descargar modelo a pc
7. Convertir el modelo para uso en dispositivos de bajo rendimiento (celulares/Raspberry)
8. Usando OPENCV y el modelo entrenado:
 1. Crear una app para Raspberry para visión por computadora
 2. Crear una app para android

¡Siempre
hacia lo alto!





Proceso Global



¡Siempre
hacia lo alto!



Preparando el ambiente para entrenamiento

Podemos entrenar en Windows/ Linux o MAC...pero es necesario tener el siguiente hardware:

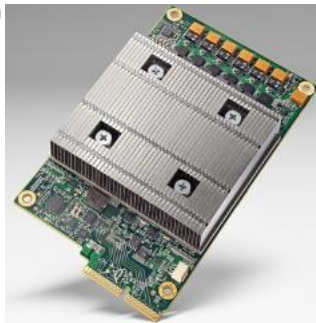
- Una buena CPU: mínimo i7



- GPU



- TPU (Tensor Processing Unit)



**Estudiantes de la
USTA**

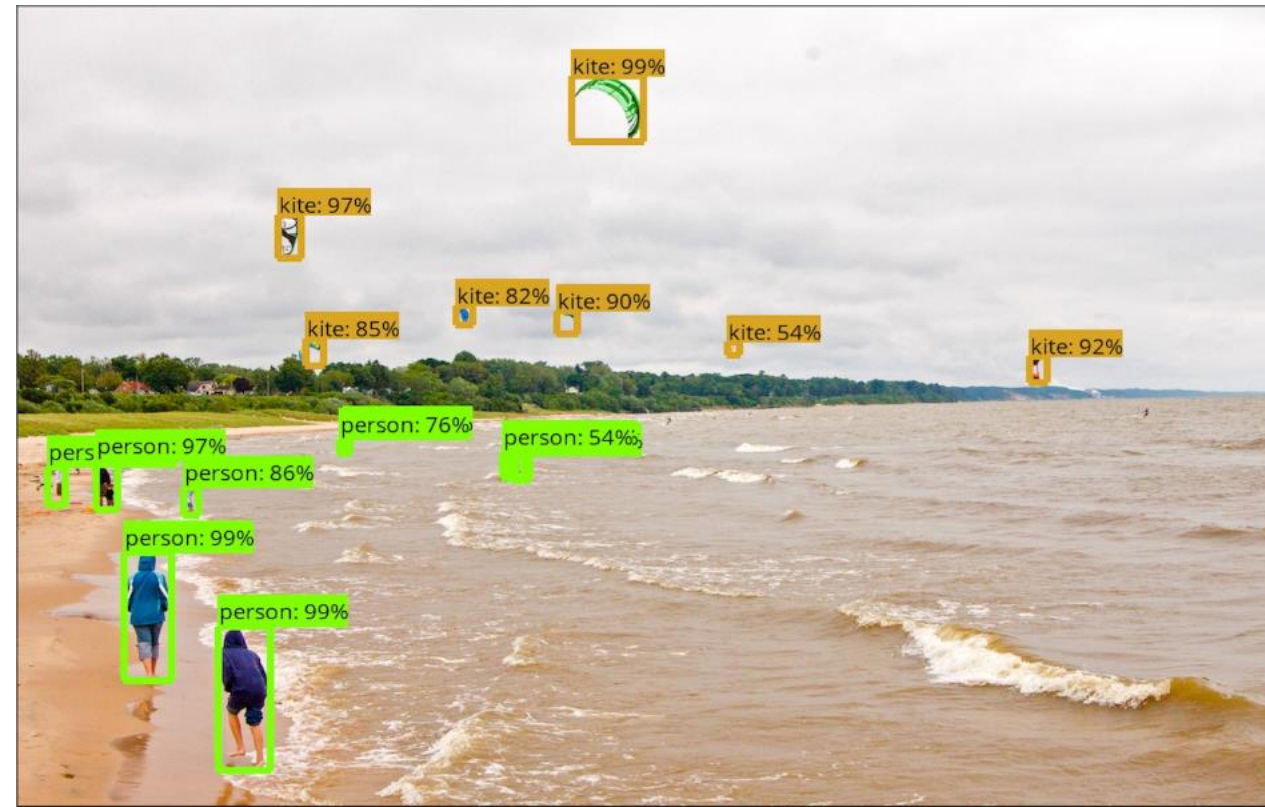


Preparando el ambiente para entrenamiento

Utilizaremos como base el proyecto:

https://github.com/tensorflow/models/tree/master/research/object_detection

Cabe resaltar que funciona únicamente en la versión 1.15 de tensorflow, debemos verificar que se tenga instalado esa versión en el ambiente de trabajo.





3.1. instalar librerías necesarias

Trabajaremos en Google Colaboratory, y necesitaremos hacer algunos cambios en el entorno de ejecución para entrenar un CNN.

Instalaremos la versión 1.15 de tensorflow (*aunque ya está la versión 2.2, aun no se ha optimizado el código para trabajar con las versiones 2.x*)

```
pip install tensorflow==1.15
```

Una vez instalado la versión 1.15, es prudente reiniciar el “entorno de ejecución”

Instalaremos unas librerías adicionales que se requieren para trabajar con tensorflow

```
!apt-get install -qq protobuf-compiler python-pil python-lxml python-tk #serialización de grafos inferenciales  
!pip install -q Cython contextlib2 pillow lxml matplotlib #librerías para visualizar imágenes  
!pip install -q pycocotools #para trabajar con unas herramientas de http://cocodataset.org/  
!pip install -q watermark #imprimir marcas de fecha y hora, números de versión e información de hardware.
```




3.1. ver librerías instaladas (opcional)

Si queremos ver que hardware y software tenemos instalado:

```
!python3 --version
%load_ext watermark
print("--Computer vision(hardware)--")
%watermark
%watermark -a "--Computer vision(libraries)--" -u -d -v -p numpy,tensorflow,pycocotools
```

Si queremos ver que la GPU esta activa para tensorflow o no

```
import tensorflow as tf
device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
    print('GPU NO esta activa para el entorno')
tf.__version__
```




3.2. Clonar repositorio de object_detection

habilitamos el acceso de Google colabory a Google drive

```
from google.colab import drive
drive.mount('/content/gdrive')
```

Clonaremos el repositorio de GITHUB **object_detection** (detección de objetos) :
(https://github.com/tensorflow/models/tree/master/research/object_detection)

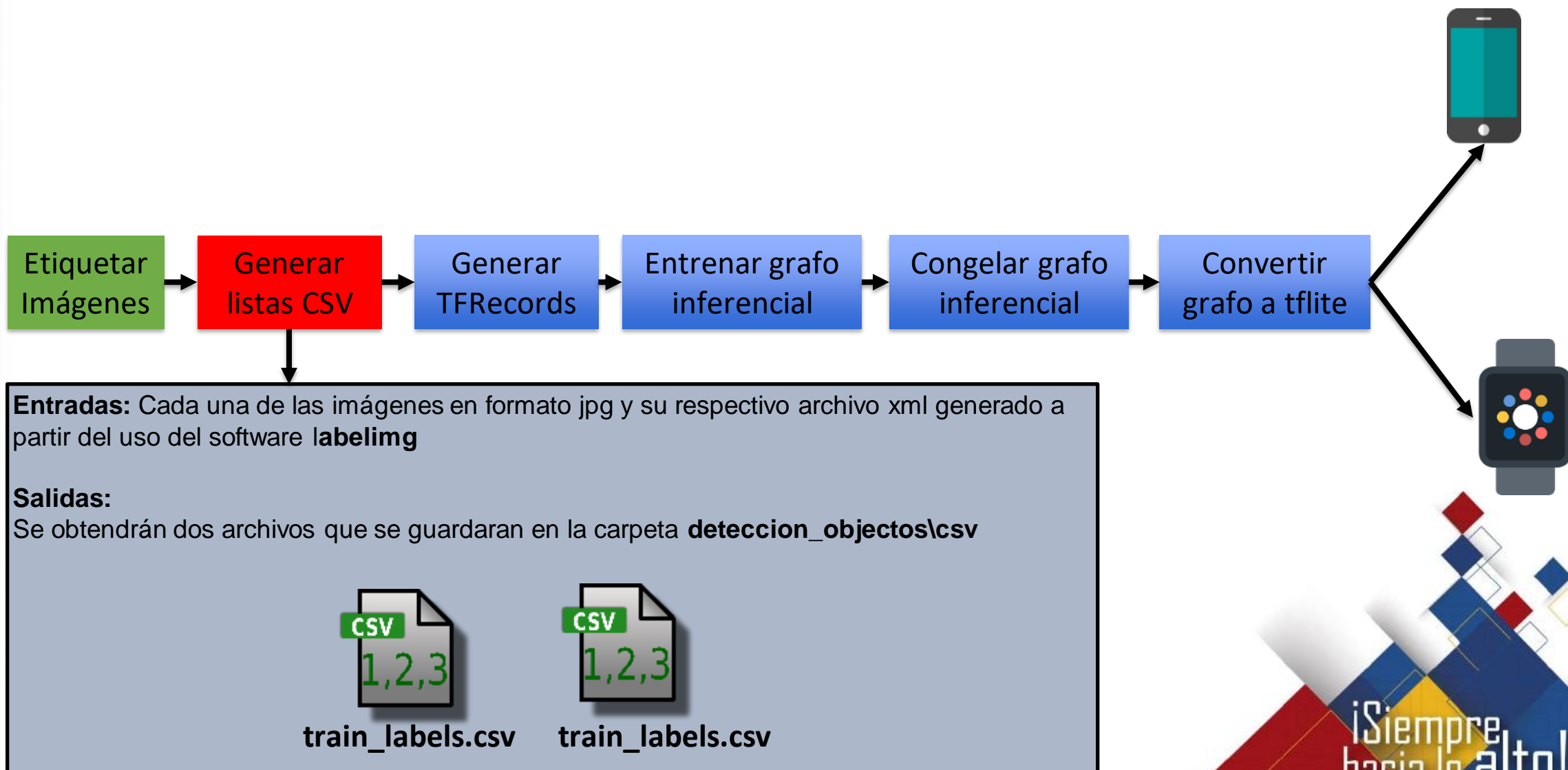
```
#descargamos el repositorio de object_detection que esta en GITHUB
%cd /content/gdrive/My\ Drive/deteccion_objetos

!git clone --quiet https://github.com/tensorflow/models.git

%cd /content/gdrive/My\ Drive/deteccion_objetos/models/research
!protoc object_detection/protos/*.proto --python_out=.
#cambiamos la variable de entorno para que sea ahora la del repositorio clonado
import os
os.environ['PYTHONPATH'] += ':/content/gdrive/My Drive/deteccion_objetos/models/research://content/gdrive/My
Drive/deteccion_objetos/models/research/slim/'
!echo $PYTHONPATH
#probamos si quedo bien descargado y funcionando el repositorio de object_detection
!python object_detection/builders/model_builder_test.py
```



Proceso Global



¡Siempre
hacia lo alto!



3.4. Crear listas CSV

Generaremos una lista (CSV) con las imágenes, sus respectivas etiquetas (labels) y la posición dentro de la imagen

Necesitaremos crear en la carpeta “**detección_objectos**” las siguientes subcarpetas:

- **csv**, donde se van a generar la lista de los dataset (entrenamiento y prueba).
- **Configuracion**, donde guardaremos algunos archivos de configuración del entrenamiento
- **TfRecords**, donde guardaremos los dataset que necesitaremos para entrenar

En la carpeta de compartida de drive “**USTA-202001_7°_DEEP_LEARNING\Computer_vision\deteccion_objectos**”, copie los dos archivos a su carpeta del proyecto:

- **xml_a_csv_v2.py** (con el creamos las listas donde estarán los nombres de las imágenes + la información de xml) y el **label_maps.pbtxt**
- **csv_a_tf_v2.py** (a partir de los CSV que se generan crearemos unos Tfrecords (array con Imágenes + xml)

Nombre

- configuracion
- csv
- imagenes
- img_entrenamiento
- img_test
- TfRecords
- csv_a_tf.py
- csv_a_tf_v2.py
- xml_a_csv.py
- xml_a_csv_v2.py



3.4.1 lista CSV de entrenamiento

Crear archivos **CSV** con listado de imágenes y sus respectivos archivos XML.

Imágenes de entrenamiento

```
# Convierte los archivos xml que estan en la carpeta de entrenamiento a una lista CSV
# y genera el archivo label_map.pbtxt en el directorio configuracion
!python /content/...../deteccion_objectos/xml_a_csv_v2.py --
inputDir /content/...../deteccion_objectos/img_entrenamiento --
outputFile /content/...../deteccion_objectos/csv/train_labels.csv --
labelMapDir /content/...../deteccion_objectos/configuracion
```

El código anterior le creara dos archivos:

- Uno, en la carpeta csv denominado “train_labels.csv”
- Otro, en la carpeta configuración denominado **label_map.pbtxt**

Nota:

recuerde que los espacios en las rutas debe agregarles el backslash \ , por ejemplo:

- Ruta= /content/drive/My Drive/
- Cambio en ruta= /content/drive/My\ Drive/

¡Siempre
hacia lo alto!



3.4.2 lista CSV de entrenamiento

Crear archivos **CSV** con listado de imágenes y sus respectivos archivos XML.

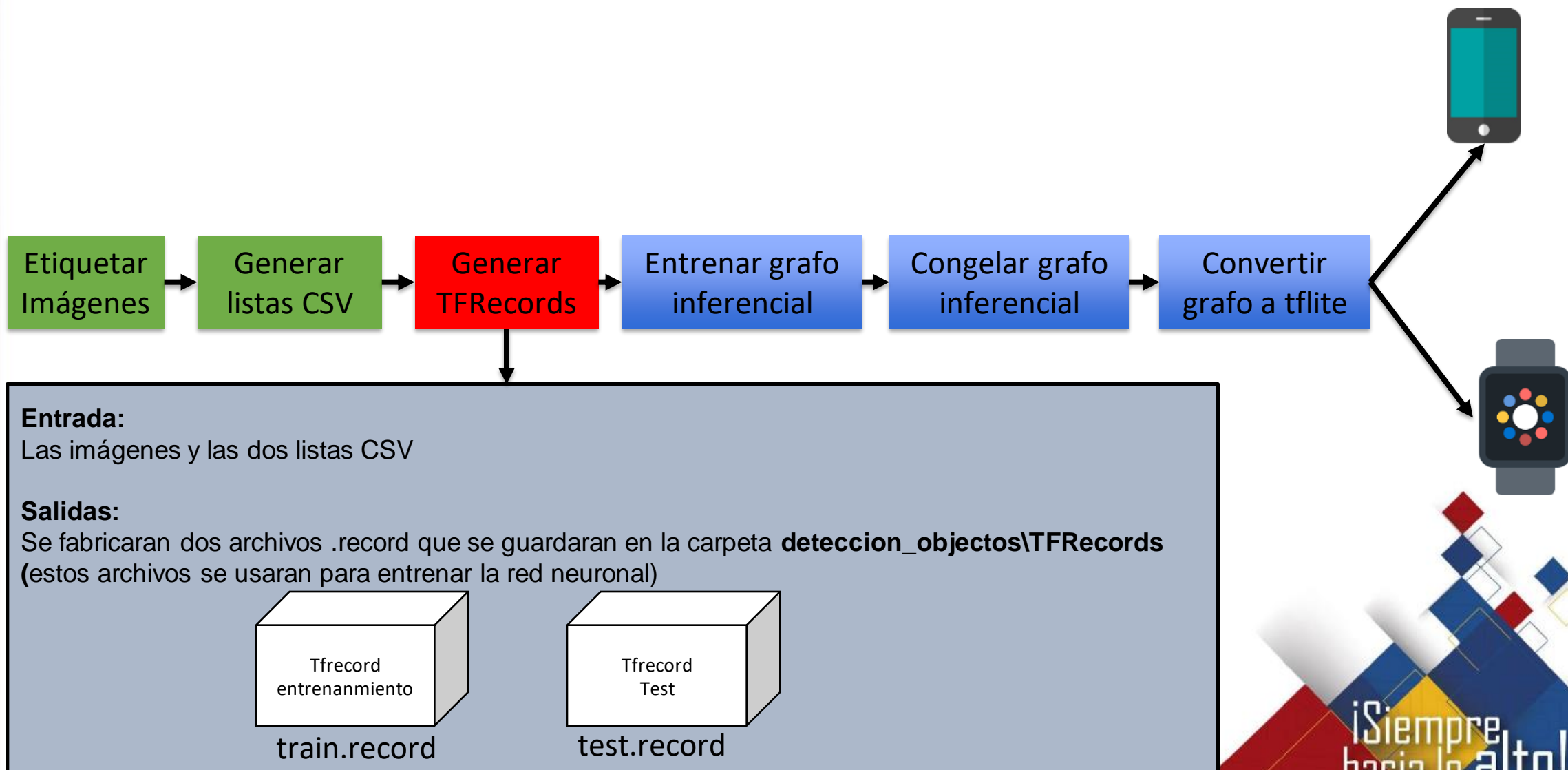
Imágenes de Test

```
# Convierte los archivos xml que estan en la carpeta de test a una lista CSV
!python /content/gdrive/My\ Drive/deteccion_objectos/xml_a_csv_v2.py --
inputDir /content/gdrive/My\ Drive/deteccion_objectos/img_test --
outputFile /content/gdrive/My\ Drive/deteccion_objectos/csv/test_labels.csv
```

El código anterior solo creara un archivo y estará en la carpeta csv denominado “test_labels.csv”



Proceso Global



¡Siempre
hacia lo alto!



3.5. Crear dataframe para tensorflow

Creando un **dataframe** (.record) con los archivos de imágenes y sus labels

Ahora le diremos a PYTHON que lea los dos archivos csv (**train_labels.csv** y **test_labels.csv**) de forma independiente y usando **Tensorflow** tomara cada archivo de imagen y su **xml** para convertir en una gran matriz donde :

- Cada fila es la información del archivo de imagen
- Cada columna representan características de la imagen(height, width, encoded, format, filename) y los labels asociados (xmin, xmax, ymin, ymax, text, label).



3.5. Crear dataframe para tensorflow

Etiquetas (label_map.pbtxt)

En este archivo (configuracion/label_map.pbtxt) le dirá a nuestro algoritmo cuales son las etiquetas sobre el cual lo entrenaremos. El nombre que pongamos en las etiquetas debe ser el mismo que usamos en la herramienta labelImg (incluyendo mayúsculas y espacios). Básicamente este archivo tiene una serie de elementos 'item' con su respectivo identificador 'id' y nombre de clase 'name'.

He aquí un ejemplo, esto cambia según el número de elementos que quieras aprender a detectar.

```
item {  
  id: 1  
  name: 'Auto'  
}  
item {  
  id: 2  
  name: 'Semaforo'  
}  
item {  
  id: 3  
  name: 'Paso Peatonal'  
}
```

```
label_map.pbtxt  
1 item {  
2   id: 1  
3   name: 'lion'  
4 }  
5  
6 item {  
7   id: 2  
8   name: 'panther'  
9 }  
10  
11 item {  
12   id: 3  
13   name: 'tiger'  
14 }
```

label_map.pbtxt

¡Siempre
hacia lo alto!



3.5.1 Crear dataframe de entrenamiento

Creando un **dataframe** de entrenamiento (train.record) con los archivos de imágenes y sus labels.

```
# Generando el archivo train.record
!python /content/drive/My\ Drive/deteccion_objetos/csv_a_tf_v2.py --
csv_input=/content/drive/My\ Drive/deteccion_objetos/csv/train_labels.csv --
output_path=/content/drive/My\ Drive/deteccion_objetos/TFRecords/train.record --
img_path=/content/drive/My\ Drive/deteccion_objetos/img_entrenamiento --
label_map /content/drive/My\ Drive/deteccion_objetos/configuracion/label_map.pbtxt
```

Donde:

- **Csv_input**, es la ubicación del archivo CSV con la lista de imágenes
- **Output_path**, es la ubicación donde se va a generar el archivo .record.
- **Img_path**, es la ubicación donde están las imágenes de entrenamiento
- **Label_map**, es la ubicación del archivo label_map.pbtxt



3.5.2 Crear dataframe de test

Creando un **dataframe** de test (test.record) con los archivos de imágenes y sus labels.

```
# Generando el archivo test.record
!python /content/drive/My\ Drive/deteccion_objetos/csv_a_tf_v2.py --
csv_input=/content/drive/My\ Drive/deteccion_objetos/csv/test_labels.csv --
output_path=/content/drive/My\ Drive/deteccion_objetos/TFRecords/test.record --
img_path=/content/drive/My\ Drive/deteccion_objetos/img_test --
label_map /content/drive/My\ Drive/deteccion_objetos/configuracion/label_map.pbtxt
```

Donde:



- **Csv_input**, es la ubicación del archivo CSV con la lista de imágenes
- **Output_path**, es la ubicación donde se va a generar el archivo .record.
- **Img_path**, es la ubicación donde están las imágenes de test
- **Label_map**, es la ubicación del archivo label_map.pbtxt



3.5.2 Crear dataframe de test

Los dos procesos anteriores nos deben crear dos archivo:

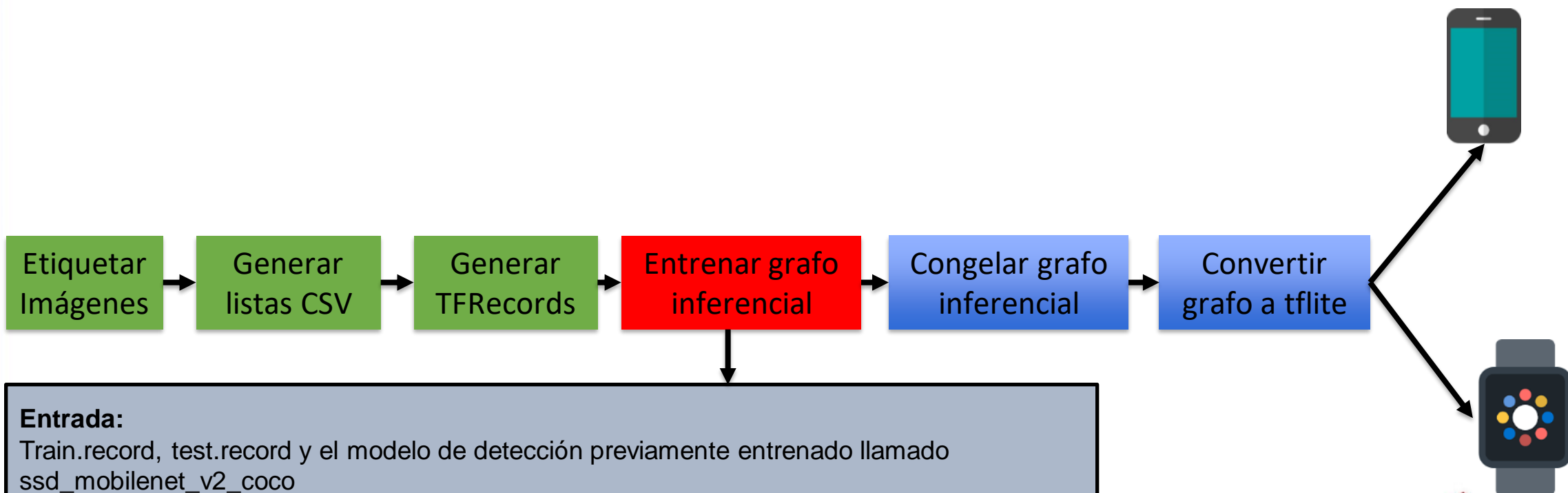
- Test.record
- Train.record

Nombre	Tamaño	Fecha de modificación	Tipo
 test.record	521 KB	24/05/2020 2:17 p. m.	Archivo RECORD
 train.record	2.184 KB	24/05/2020 2:15 p. m.	Archivo RECORD

Estos dos archivos contienen la información de todas las imágenes y de las coordenadas de los objetos que marcamos



Proceso Global



Entrada:

Train.record, test.record y el modelo de detección previamente entrenado llamado `ssd_mobilenet_v2_coco`

Salidas:

Con las imágenes y las dos listas CSV, se fabricaran dos archivos .record que se guardaran en la carpeta **deteccion_objectos\TFRecords** (estos archivos se usaran para entrenar la red neuronal)

¡Siempre
hacia lo alto!



4. Entrenar modelo de inferencia.

Para realizar el proceso de entrenamiento, necesitaremos:

1. Elegir el modelo que entrenaremos (usaremos uno de Google).
1. Definir hiperparámetros.
1. Configurar el proceso de entrenamiento (pipeline)
1. Habilitar tensorboard (para ver el avance del entrenamiento)
1. Entrenar el modelo



4.1 Seleccionar un modelo

Debemos decidir que modelo es el que queremos entrenar, algunos nos ofrecen detecciones más veloces, sacrificando certeza o viceversa.

Para ver todos los modelos podemos ingresar a:

https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md

Usaremos el modelo [ssd_mobilenet_v2_coco](#) el cual nos brinda predicciones más veloces (requeridas en celulares) aunque no las mas efectivas (eso depende de que tanto lo entrenemos).



4.1 Seleccionar un modelo

Descargamos el modelo a nuestro entorno.

```
%cd /content/models/research

import os
import shutil
import glob
import urllib.request
import tarfile
MODEL_FILE = MODEL + '.tar.gz'
DOWNLOAD_BASE = 'http://download.tensorflow.org/models/object_detection/'
DEST_DIR = '/content/models/research/pretrained_model'

if not (os.path.exists(MODEL_FILE)):
    urllib.request.urlretrieve(DOWNLOAD_BASE + MODEL_FILE, MODEL_FILE)

tar = tarfile.open(MODEL_FILE)
tar.extractall()
tar.close()

os.remove(MODEL_FILE)
if (os.path.exists(DEST_DIR)):
    shutil.rmtree(DEST_DIR)
os.rename(MODEL, DEST_DIR)
```




4.1 Seleccionar un modelo

Verificamos que los archivos estén descargados

```
!echo {DEST_DIR}  
!ls -alh {DEST_DIR}
```

```
📁 /content/models/research/pretrained_model  
total 135M  
drwxr-xr-x  3 345018 89939 4.0K Mar 30 2018 .  
drwxr-xr-x 63 root    root  4.0K May 24 20:24 ..  
-rw-r--r--  1 345018 89939   77 Mar 30 2018 checkpoint  
-rw-r--r--  1 345018 89939 67M Mar 30 2018 frozen_inference_graph.pb  
-rw-r--r--  1 345018 89939 65M Mar 30 2018 model.ckpt.data-00000-of-00001  
-rw-r--r--  1 345018 89939 15K Mar 30 2018 model.ckpt.index  
-rw-r--r--  1 345018 89939 3.4M Mar 30 2018 model.ckpt.meta  
-rw-r--r--  1 345018 89939 4.2K Mar 30 2018 pipeline.config  
drwxr-xr-x  3 345018 89939 4.0K Mar 30 2018 saved_model
```

El listado de archivos corresponden al modelo **ssd_mobilenet_v2_coco** de detección previamente entrenado.

¡Siempre
hacia lo alto!



4.2 definir hiperparametros

Necesitaremos definir varios parámetros para el entrenamiento de la red neuronal.

```
# ubicación del archivo donde esta el modelo con el que iniciaremos el entrenamiento
fine_tune_checkpoint = os.path.join(DEST_DIR, "model.ckpt")
fine_tune_checkpoint

# número de pasos (epochs) que usaremos para entrenar
num_steps = 2000 # recomiendo mínimo 200000 pero probaremos con 2000

# número de evaluaciones por pasos (cada 50 pasos evaluamos el modelo).
num_eval_steps = 50

# Seleccionamos el archivo de configuración del modelo
selected_model = 'ssd_mobilenet_v2'

# Nombre del modelo de detección de objetos que usaremos
MODEL = MODELS_CONFIG[selected_model]['model_name']

# Nombre del archivo de canalización en la API de detección de objetos de tensorflow.
pipeline_file = MODELS_CONFIG[selected_model]['pipeline_file']

# definimos el tamaño del lote de entrenamiento para uso en la memoria de
# la GPU Tesla K80 de Colaborate para el modelo seleccionado.
batch_size = MODELS_CONFIG[selected_model]['batch_size']
```



4.2 definir hiperparametros

Por último la configuración del modelo que usaremos

```
MODELS_CONFIG = {
    'ssd_mobilenet_v2': {
        'model_name': 'ssd_mobilenet_v2_coco_2018_03_29',
        'pipeline_file': 'ssd_mobilenet_v2_coco.config',
        'batch_size': 12
    },
    'faster_rcnn_inception_v2': {
        'model_name': 'faster_rcnn_inception_v2_coco_2018_01_28',
        'pipeline_file': 'faster_rcnn_inception_v2_pets.config',
        'batch_size': 12
    },
    'rfcn_resnet101': {
        'model_name': 'rfcn_resnet101_coco_2018_01_28',
        'pipeline_file': 'rfcn_resnet101_pets.config',
        'batch_size': 8
    }
}

#variables con las rutas de nuestras dataset de entrenamiento
test_record_fname = '/content/drive/My Drive/deteccion_objectos/TFRecords/test.record'
train_record_fname = '/content/drive/My Drive/deteccion_objectos/TFRecords/train.record'
label_map_pbtxt_fname = '/content/drive/My Drive/deteccion_objectos/configuracion/label_map.pbtxt'
```




4.3 definir el canalización (pipeline) de la API

Cargamos el archivo para agregar los hiperparametros.

```
import os
pipeline_fname = os.path.join('/content/models/research/object_detection/samples/configs/', pipeline_
file)

assert os.path.isfile(pipeline_fname), '`{}` not exist'.format(pipeline_fname)
```

Creamos una función que nos permitirá leer las etiquetas y categorías del archivo de configuración de la pipeline.

```
def get_num_classes(pbtxt_fname):
    from object_detection.utils import label_map_util
    label_map = label_map_util.load_labelmap(pbtxt_fname)
    categories = label_map_util.convert_label_map_to_categories(
        label_map, max_num_classes=90, use_display_name=True)
    category_index = label_map_util.create_category_index(categories)
    return len(category_index.keys())
```



4.3 definir el canalización (pipeline) de la API

Agregamos nuestras clases al archivo de configuración de la pipeline

```
import re

num_classes = get_num_classes(label_map_pbtxt_fname)
with open(pipeline_fname) as f:
    s = f.read()
with open(pipeline_fname, 'w') as f:

    # punto de control de inicio
    s = re.sub('fine_tune_checkpoint: ".*?"', 'fine_tune_checkpoint: "{}"'.format(fine_tune_checkpoint), s)

    # ubicación de los tfrecords de train y test
    s = re.sub('(input_path: ".*?" (train.record) (.*)"')', 'input_path: "{}"'.format(train_record_fname), s)
    s = re.sub('(input_path: ".*?" (val.record) (.*)"')', 'input_path: "{}"'.format(test_record_fname), s)

    # ubicación del labelmap.pbtxt
    s = re.sub('label_map_path: ".*?"', 'label_map_path: "{}"'.format(label_map_pbtxt_fname), s)

    # Establecemos el tamaño del lote de entrenamiento
    s = re.sub('batch_size: [0-9]+', 'batch_size: {}'.format(batch_size), s)

    # Establecemos la cantidad de pasos de entrenamiento
    s = re.sub('num_steps: [0-9]+', 'num_steps: {}'.format(num_steps), s)

    # establecemos el número de clases (etiquetas)
    s = re.sub('num_classes: [0-9]+', 'num_classes: {}'.format(num_classes), s)
f.write(s)
```



4.3 definir el canalización (pipeline) de la API

Revisamos que todo haya quedado como se requiere

```
!cat {pipeline_fname}
```

Número de clases

```
model {
  faster_rcnn {
    num_classes: 3 (Aquí van el número de objetos a detectar Lion, tiger y panther)
    image_resizer {
      keep_aspect_ratio_resizer {
        min_dimension: 600
        max_dimension: 1024
      }
    }
  }
}
```

Ubicación del modelo

```
train_config: {
  batch_size: 1
  gradient_clipping_by_norm: 10.0
  fine_tune_checkpoint: "modelo/model.ckpt"
  from_detection_checkpoint: true
}
```

¡Siempre
hacia lo alto!



4.3 definir el canalización (pipeline) de la API

- Ubicación de los archivos de tensorflow (entrenamiento y test) y el labelmap.pbtxt

```
train_input_reader: {  
  tf_record_input_reader {  
    input_path: "TFRecords/entrenamiento.record"  
  }  
  label_map_path: "configuracion/label_map.pbtxt"  
}  
eval_input_reader: {  
  tf_record_input_reader {  
    input_path: "TFRecords/test.record"  
  }  
  label_map_path: "configuracion/label_map.pbtxt"  
  shuffle: false  
  num_readers: 1  
  num_epochs: 1  
}
```

- Número de épocas o “ciclos” de entrenamiento

```
# never decay). Remove the below line to train indefinitely.  
num_steps: 200000  
data_augmentation_options {  
  random_horizontal_flip {  
  }  
}
```



4.5 Limpiar salida (opcional)

Creamos el directorio donde se va a generar el modelo de inferencia.

```
model_dir = 'training/'  
# borrar archivos de la carpeta donde se guardara el modelo  
!rm -rf {model_dir}  
os.makedirs(model_dir, exist_ok=True)
```

Opcionalmente (en caso que desee entrenar desde cero el modelo), se debe eliminar el contenido del directorio del modelo de salida para comenzar de cero.

No se recomienda si lo que se va a realizar es un reentrenamiento.



4.4 Tensorboard (opcional)

Tensorboard es una interfaz de tensorflow que nos permite hacer seguimiento al proceso de entrenamiento de un grafo inferencia.

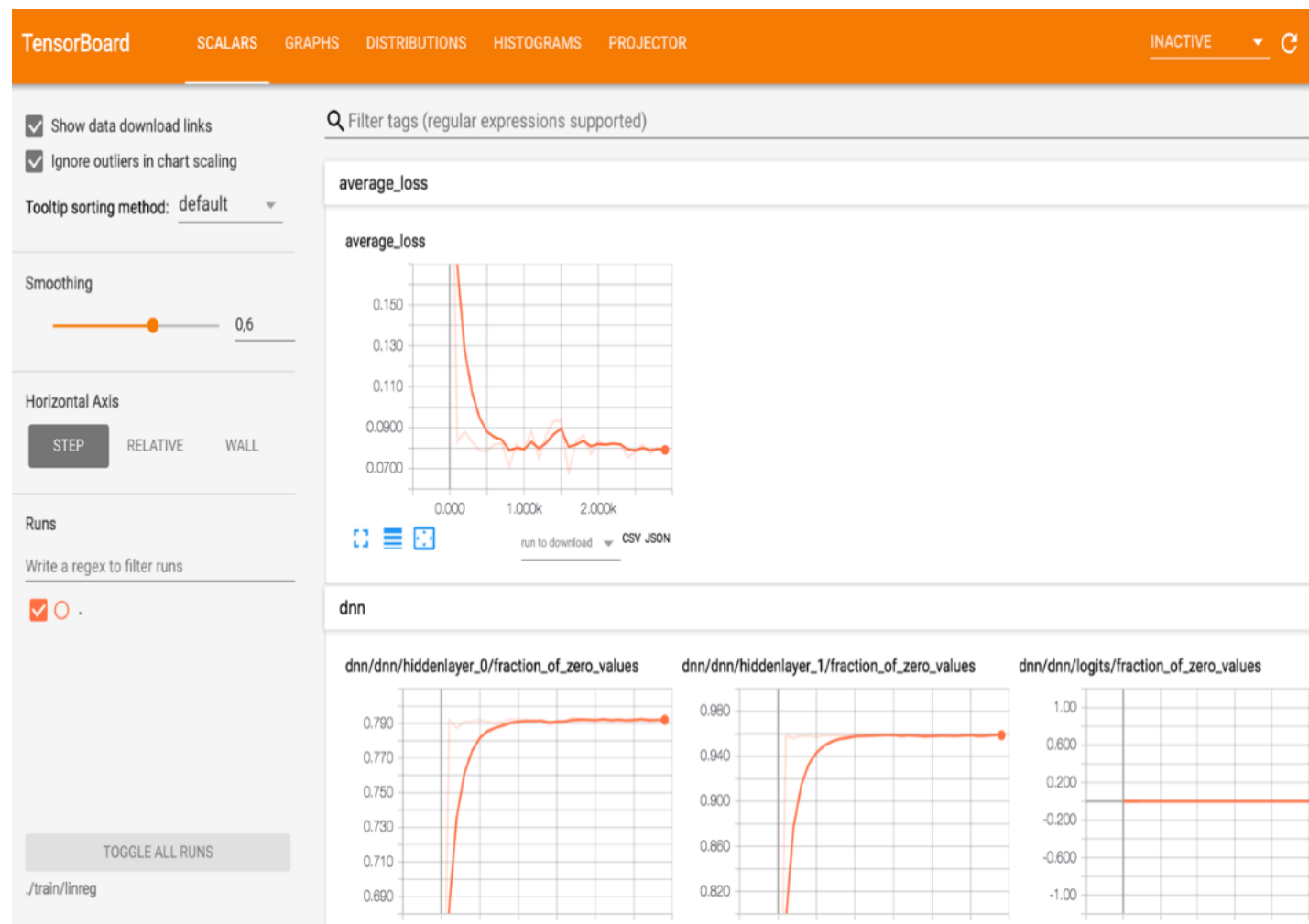
<https://guru99.es/tensorboard-tutorial/>

El panel tiene las siguientes fichas que están vinculadas a cada nivel.

- **Escalares:** Muestra diferentes informaciones útiles durante el entrenamiento del modelo
- **Gráficos:** Mostrar el modelo

Las siguientes no están disponibles en Google collaborate:

- *Histograma: muestra los pesos con un histograma*
- *Distribución: Muestra la distribución del peso*
- *Proyector: Mostrar análisis de componentes principales y algoritmo T-SNE. La técnica utiliza para la reducción de dimensionalidad*





4.4.1 Instalar tensorboard

Descargar y descomprimir tensorboard.

```
!wget https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-amd64.zip
!unzip -o ngrok-stable-linux-amd64.zip
```

Almacenar el log del proceso de entrenamiento y sus checkpoints del entrenamiento.

```
LOG_DIR = model_dir
get_ipython().system_raw(
    'tensorboard --logdir {} --host 0.0.0.0 --port 6006 &'
    .format(LOG_DIR)
)
get_ipython().system_raw('./ngrok http 6006 &')
```

Generar el link de tensorboard

```
! curl -s http://localhost:4040/api/tunnels | python3 -c \
    "import sys, json; print(json.load(sys.stdin)['tunnels'][0]['public_url'])"
```




4.5 Entrenar el modelo de inferencia

Por fin!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!in.

Ahora si viene lo que tanto necesitábamos (entrenar el modelo).

```
!python /content/models/research/object_detection/model_main.py \  
  --pipeline_config_path={pipeline_fname} \  
  --model_dir={model_dir} \  
  --alsologtostderr \  
  --num_train_steps={num_steps} \  
  --num_eval_steps={num_eval_steps}
```

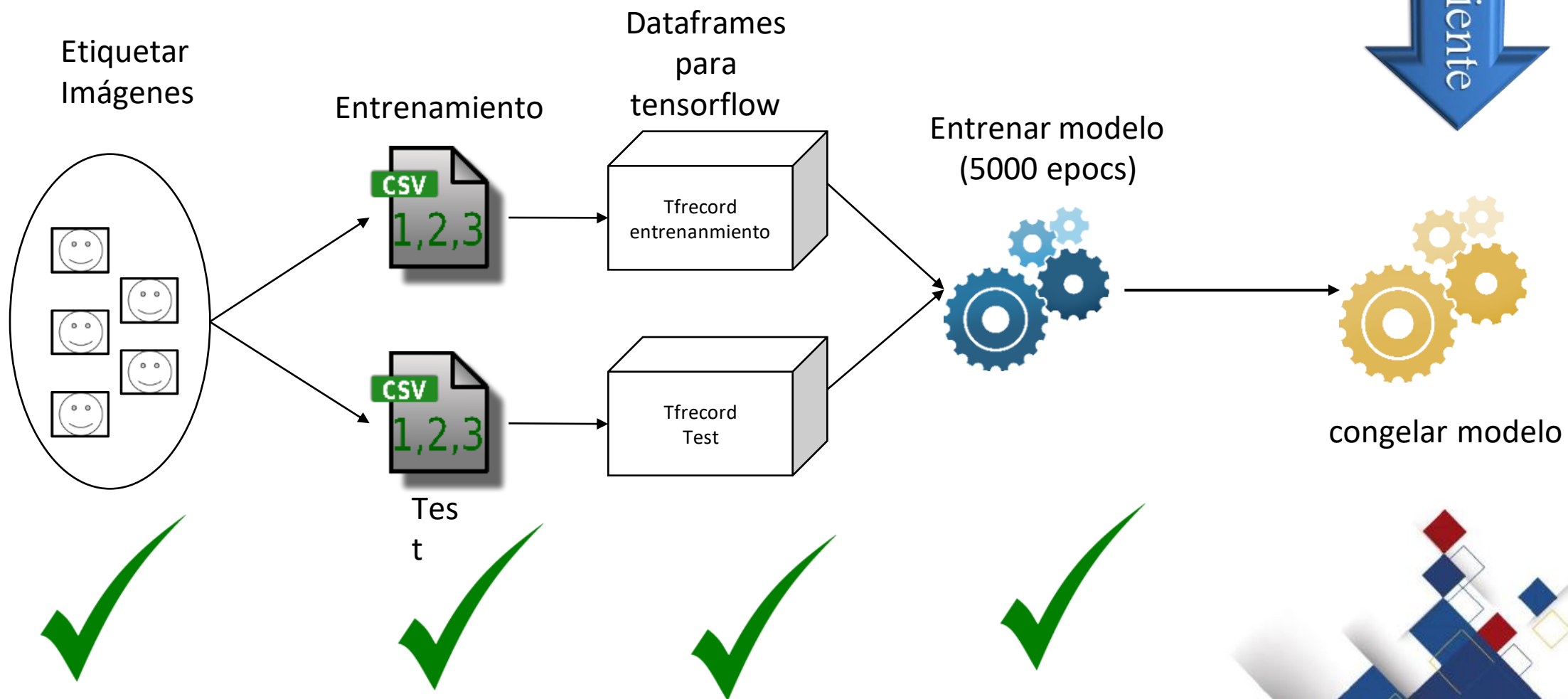
Este proceso es baaaaaaaaaaaaaaaaaastante lento, tengan paciencia. En el tensorboard pueden ver el avance del entrenamiento.

Para ver listado de archivos creados después del entrenamiento:

```
!ls {model_dir}
```



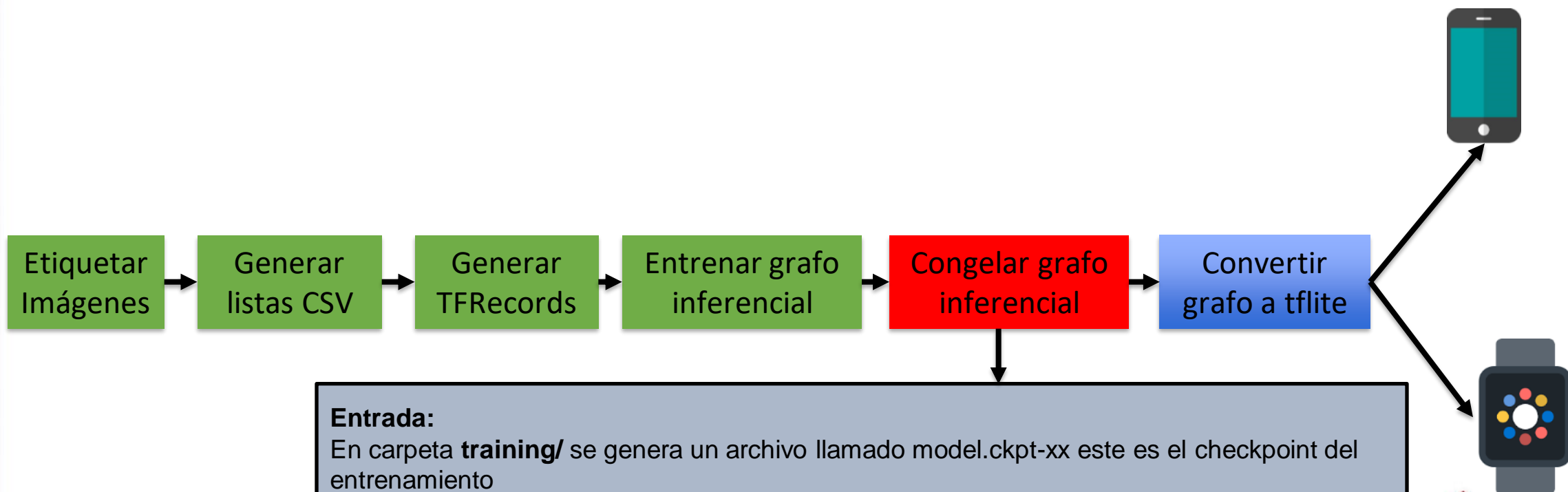
5 Congelar/Exportar modelo



¡Siempre
hacia lo alto!



Proceso Global



Entrada:

En carpeta **training/** se genera un archivo llamado `model.ckpt-xx` este es el checkpoint del entrenamiento

Salidas:

Con el checkpoint se realizara un congelamiento para obtener un archivo llamado

`fine_tuned_model/frozen_inference_graph.pb`

Con este archivo se pueden hacer inferencias pero se requiere un gran hardware.

mpre
nada lo alto!



5 Congelar/Exportar modelo

Una vez que se completa su entrenamiento, debemos extraer el gráfico de inferencia recién entrenado, que luego se utilizará para realizar la detección de objetos. Esto puede hacerse de la siguiente manera:

```
import re
import numpy as np

output_directory = './fine_tuned_model'

lst = os.listdir(model_dir)
lst = [l for l in lst if 'model.ckpt-' in l and '.meta' in l]
steps=np.array([int(re.findall('\d+', l)[0]) for l in lst])
last_model = lst[steps.argmax()].replace('.meta', '')

last_model_path = os.path.join(model_dir, last_model)
print(last_model_path)
!python /content/models/research/object_detection/export_inference_graph.py \
    --input_type=image_tensor \
    --pipeline_config_path={pipeline_fname} \
    --output_directory={output_directory} \
    --trained_checkpoint_prefix={last_model_path}
```

Para ver listado de archivos exportados.

```
!ls {model_dir}
```




6 descargar modelo .pb

Recordemos en Google collaborate es un ambiente temporal una vez se cierre sesión se pierde todo, por lo tanto vamos a exportar el modelo

```
import os

pb_fname = os.path.join(os.path.abspath(output_directory), "frozen_inference_graph.pb")
assert os.path.isfile(pb_fname), '{} not exist'.format(pb_fname)
!ls -alh {pb_fname}
from google.colab import files
#descargamos el modelo
files.download(pb_fname)
#descargamos el mapa de etiquetas.
files.download(label_map_pbt.txt)
```

Ya tenemos nuestro modelo, por fiiiin....ahora probemos haber si sirve.

¡Siempre
hacia lo alto!



6 probar modelo .pb

Creamos una subcarpeta dentro de **deteccion_objectos**, a la que llamaremos **img_prueba**, en ella colocamos unas imágenes que tenga alguna pantera, león o tigre pero que no estén entre las imágenes que usamos para entrenamiento ni prueba.

```
import os
import glob

# Ruta del gráfico de detección congelado. Este es el modelo real que se utiliza para la detección de o
bjetos.
PATH_TO_CKPT = pb_fname

# Lista de las cadenas que se utilizan para agregar la etiqueta correcta para cada cuadro.
PATH_TO_LABELS = label_map_pbtxt_fname

# ruta donde estan las imágenes para probar
PATH_TO_TEST_IMAGES_DIR = "/content/drive/My Drive/deteccion_objectos/img_prueba"

assert os.path.isfile(pb_fname)
assert os.path.isfile(PATH_TO_LABELS)
TEST_IMAGE_PATHS = glob.glob(os.path.join(PATH_TO_TEST_IMAGES_DIR, "*.jpg"))
assert len(TEST_IMAGE_PATHS) > 0, 'No image found in {}'.format(PATH_TO_TEST_IMAGES_DIR)
print(TEST_IMAGE_PATHS)
```



6 probar modelo .pb

Pasamos cada una de las imágenes por el grafo y luego las visualizamos

```
%cd /content/models/research/object_detection
```

```
import numpy as np
import os
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile
from collections import defaultdict
from io import StringIO
from matplotlib import pyplot as plt
from PIL import Image
```

```
sys.path.append("..")
from object_detection.utils import ops as utils_ops
```

```
#Esto es necesario para mostrar las imágenes.
%matplotlib inline
```

```
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as vis_util
```

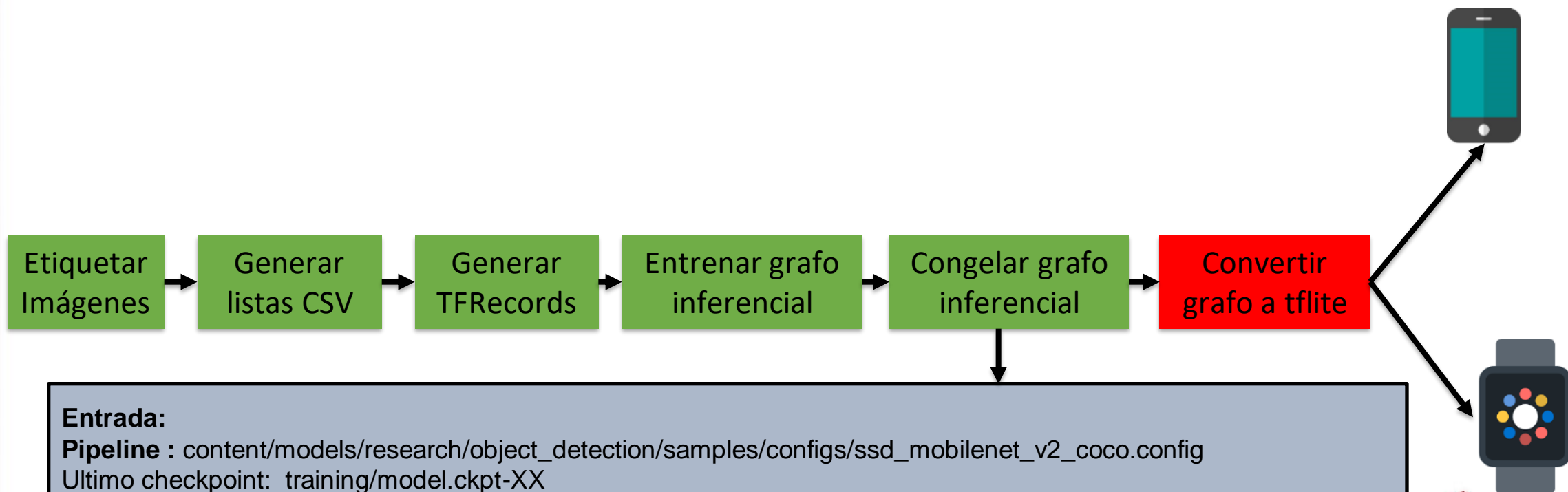
```
detection_graph = tf.Graph()
with detection_graph.as_default():
    od_graph_def = tf.GraphDef()
    with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
        serialized_graph = fid.read()
        od_graph_def.ParseFromString(serialized_graph)
        tf.import_graph_def(od_graph_def, name='')
```

```
label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(label_map, max_num_classes=num_classes, use_display_name=True)
category_index = label_map_util.create_category_index(categories)
```

```
def load_image_into_numpy_array(image):
    (im_width, im_height) = image.size
    return np.array(image.getdata()).reshape(
```



Proceso Global



Entrada:

Pipeline : content/models/research/object_detection/samples/configs/ssd_mobilenet_v2_coco.config

Ultimo checkpoint: training/model.ckpt-XX

Salidas:

Se creara una carpeta llamada **tflite** donde se generara un grafo liviano denominado **tflite_graph.pb** y su respectivo **label_map.pbtxt**

Estos dos últimos pueden pesar máximo 5 mb haciendo un grafo que se puede usar en celulares o cualquier otro dispositivo de recursos limitados.

mpre
nada lo alto!



Proceso Global

Listemos los archivos que vamos a necesitar:

```
print("fine_tune_checkpoint: "+fine_tune_checkpoint)
print("pb_fname: "+pb_fname)
print("pipeline_fname: "+pipeline_fname)
print("model_dir: "+model_dir)
!ls -alh {model_dir}
```

Convertir a grafo liviano (tflite):

```
!python /content/gdrive/My\ Drive/deteccion_objetos/models/research/object_detection/export_tflite_ssd_graph.py \
--
pipeline_config_path=/content/gdrive/My\ Drive/deteccion_objetos/models/research/object_detection/samples/configs/s
sd_mobilenet_v2_coco.config \
--trained_checkpoint_prefix=/content/gdrive/My\ Drive/deteccion_objetos/models/research/training/model.ckpt-3690 \
--output_directory=/content/gdrive/My\ Drive/deteccion_objetos/models/research/tflite \
--add_postprocessing_op=true
```



UNIVERSIDAD SANTO TOMÁS
PRIMER CLAUSTRO UNIVERSITARIO DE COLOMBIA
SECCIONAL TUNJA

VIGILADA MINEDUCACIÓN - SNIES 1732

¡Siempre
hacia lo alto!

USTATUNJA.EDU.CO



@santotomastunja