Introduzione

Setup dell’ambiente per eseguire la fase di training del modello di machine learning e collezione del Dataset relativo all’oggetto che vogliamo riconoscere.

Esecuzione della fase di train.

Esportazione del modello ottenuto e testing tramite jupyter notebook.

Il processo è stato già da me completamente eseguito per il riconoscimento della bottiglia d’acqua **San Bernardo**, tutti i file che si ottengono durate il procedimento e tutti quelli che sono stati necessari per la sua esecuzione sono reperibili nella cartella Bernardi\_Object\_Detection.

Di seguito viene mostrato come addestrare un modello per il riconosciemento di un oggetto specifico, ad ogni passo verrà mostrato quale file utilizzare della [cartella sopra indicata](#Bernardo_Object_Detection) o come configurare il proprio.

1. [Collezione Dataset ………………………………………………………………………………………………………………………2](#CollezioneDataset)
2. [Setup ambiente …..………………………………………………………………………………………………..……………………4](#SetupAmbiente)
3. [Esecuzione Train Tensorflow ……………………………………………………………………………………………………….5](#TrainTF)
4. [Esecuzione Train GoogleCloud …………………………………………………………………………………………………….6](#TrainGC)
5. [Esportazione …..………………………………………………………………………………………………………………………….8](#Export)
6. [Esecuzione Test …..………………………………………………………………………………………………………………………9](#Test)

Prima di iniziare le operazioni di raccolta del Data Set è necesario eseguire il download della seguente repository github, essa contiene l’API per l’Object Detection che permette di realizzare modelli di machine learning atti al riconosciemtno di oggetti all’interno di un’immagine.

<https://github.com/tensorflow/models>

Quando in seguito farò riferimento alla cartella models, indicherò la suddetta. L’API di object detection si trova in models/research/object\_detection

Collezione Dataset

Per eseguire il riconoscimento di un oggetto specifico è necessario collezionare un Dataset relativo all’oggetto che si vuole identificare. Su tale Dataset dovranno essere eseguite quindi le operazioni di train.

Un dataset completo si trova già all’interno della cartella images nella cartella da me fornita ed è costituito da 1000 immagini riguardanti una bottiglia di San Bernardo. Sono presenti anche i file .xml che in seguito verrà mostrato come ottenere e sono suddivise opportunamente in una cartella di train e di test, ma anche questo verrà mostrato a breve.

Creare innanzi tutto una cartella temporanea all’interno della quale si andrà a lavorare momentaneamente per poi spostarsi nella cartella models fornita da TensorFlow.

Chiamerò questa cartella temp, così quando nel seguito la nomierò saprete a cosa faccio riferimento.

1. Posizionarsi all’interno della cartella temp.
2. Eseguire il download da internet o scattare un insieme di foto dell’oggetto che si vuole riconoscere e collocarle in una cartella images all’interno di temp. Il numero di immagini maggiore risulta essere più preciso ed efficace sarà il riconoscimento.

A seconda dalle prospettive da cui è possibile vedere l’oggetto sarà opportuno scattare un numero maggiore o minore di foto, lo stesso vale per condizioni di luce differenti.

Se invece l’oggetto fosse trasparente (come la bottiglia da me riconosciuta) sarà opportuno scattare foto in situazioni cromatiche differenti per il fatto che l’oggetto acquisirebbe in foto il colore presente sullo sfondo.

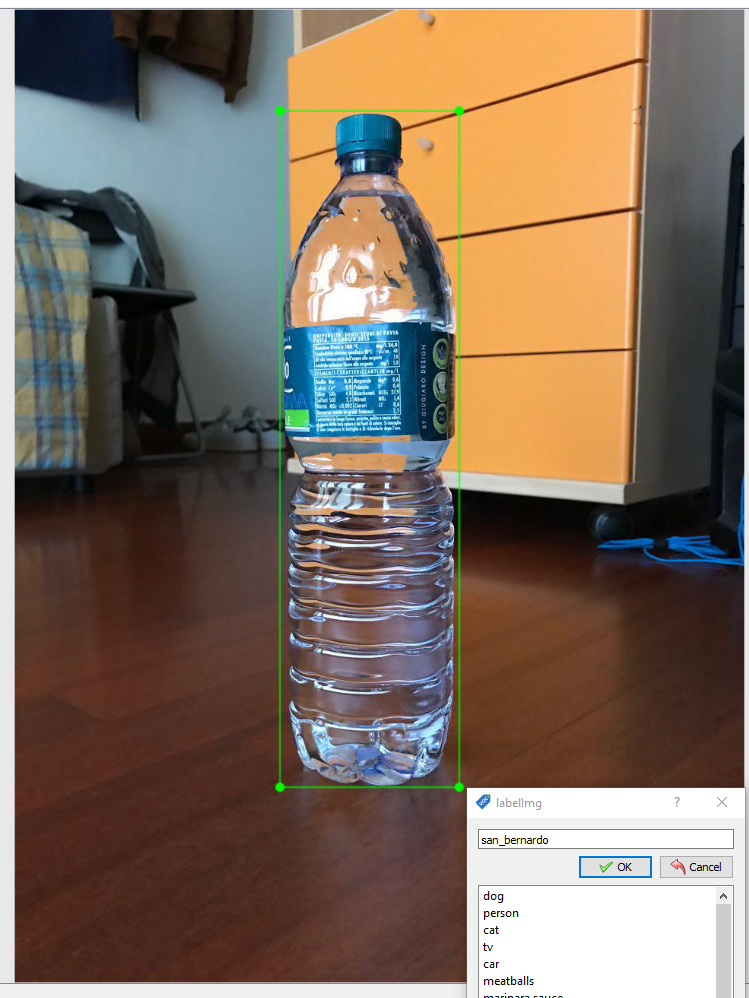
Ad esempio nel dataset da me fornito circa l’80% delle foto sono state scattate con la bottiglia in piedi o con vista frontale della stessa abbattuta e con numerose condizioni cromatiche e di luce.

1. Attraverso il tool LabelImg presente sempre all’interno della cartella Object\_Detection\_Bernardi dovrà essere generato per ogni immagine il file XML indicante la posizione dell’oggetto all’interno dell’immagine stessa.

Se il file fornito avesse problemi è possibile riscaricarlo o cambiare versione alla seguente pagina:

<https://tzutalin.github.io/labelImg/> (la versione 1.4.0 non dovrebbe dare problemi)

(Siccome ho riscontrato alcuni problemi di avvio del software su macchina virtuale potrebbe essere necessario eseguire la seguente operazione sul proprio sistema operativo ed esportare poi il risultato ottenuto su macchina virtuale. Per questo motivo è fornita anche la versione per windows del software).

Selezionare come cartella di import quella contenente le immagini e la medesima come cartella di salvataggio. Quindi andare a tracciare il riquadro attorno all’immagine e digitare il label relativo all’oggetto ovvero il nome dello stesso (nel caso l’oggetto sia uno dovremmo inserire sempre lo stesso label in alternativa lo stesso label per ogni classe).

Salvare dopo ogni foto e terminato si avrà per ogni immagine la presenza del relativo file xml indicante la posizione dell’oggetto all’interno dell’immagine. Tutti i file si troveranno all’interno della cartella images appena creata.

1. Creare due sottocartelle all’interno della cartella images chiamate train e test.

All’interno della cartella train copiare il 90% delle immagini e il corrispondente XML di ognuna.

All’interno della cartella test copiare il restante 10% delle immagini e il corrispondente XML di ognuna.

L’esempio di quale deve essere il risultato è mostrato nella cartella images da me fornita.

1. Copiare nella cartella temp i seguenti file che dovranno essere utilizzati per la conversione del dataset dalla cartella Bernardi\_Object\_Detection:

**xml\_to\_csv.py**

**generate\_tfrecord.py**

1. Il primo file è anche reperibile al seguente link <https://github.com/datitran/raccoon_dataset/blob/master/xml_to_csv.py>

Esso dovrà però essere modificato opportunamente sostituendo alle righe 29-30-31-32 le seguenti righe che permettono di iterare sulle due cartelle train e test appena create:

**for directory in ['train', 'test']:**

**image\_path = os.path.join(os.getcwd(), 'images/{}'.format(directory))**

**xml\_df = xml\_to\_csv(image\_path)**

**xml\_df.to\_csv('data/{}\_labels.csv'.format(directory), index=None)**

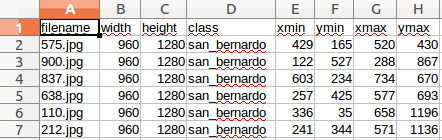
**print('Successfully converted xml to csv.')**

Per itale file se le cartelle sono state nominate correttamente train e test non è necessario modificare nulla.

1. Creare una cartella “data” all’interno della cartella temp.

Eseguire il primo file da terminale attraverso il seguente comando **python xml\_to\_csv.py**, nella cartella data dovrebbero comparire due file: **test\_label.csv** e **train\_directory.csv.**

Il formato di ciascuno di essi dovrebbe essere il seguente:



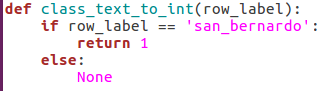
1. Il secondo file è invece reperibile al seguente link nella medesima repository del precedente:

<https://github.com/datitran/raccoon_dataset/blob/master/generate_tfrecord.py>

In esso basterà invece modificare solamente:

if row\_label == ‘*nome\_della\_classe\_da\_identificare*’

Inseriamo il nome della classe uguale a quello del label inserito in LabelImg.



1. Spostarsi nella cartella /models/research e da terminale digitare:

**sudo python setup.py install**

Così si andrà ad installare sul proprio sistema l’API per object detection

1. Quindi tornare nuovamente nella cartella tutorial e da terminale digitare i comandi che si trovano nell’intestazione del file generate\_tfrecord.py da me fornito (se il file viene scaricato e si esegue il comando nell’intestazione i file non vengono posizionati all’interno della cartella data e sarà necessario copiarli all’interno; è sempre possibile utilizzare il comando da me sotto riportato):

**python generate\_tfrecord.py --csv\_input=data/train\_labels.csv --output\_path=data/train.record**

**python generate\_tfrecord.py --csv\_input=data/test\_labels.csv --output\_path=data/test.record**

In questo modo i file .csv creati prima andranno ad essere convertiti in un file rispettivamente train.record e test.record che appariranno nella cartella data.

Una cartella data, con tutti i file in formato csv e record ottenuti dai passi precedenti, è presente all’interno della cartella Bernardi\_Object\_Detection.

Setup ambiente

1. All’interno della cartella Bernardi\_Object\_Detection sarà presente una cartella denominata

**ssd\_mobilenet\_v2\_coco\_2018\_03\_29** essa contiene un modello pre addestrato necessario per la fase di training.

Si tratta di un modello pre-addestrato per il riconoscimento di oggetti di uso comune sulla base del quale avverrà l’addestramento per il riconoscimento di uno specifico oggetto.

Eseguire infatti un addestramento partendo solamente dai file precedentemente generati richiederebbe un enorme quantitativo di tempo.

Nella cartella sarà presente anche un file **ssd\_mobilenet\_v2\_coco.config**

Dentro la cartella training sarà invece presente un file **.pbtxt.**

Nella cartella temp copiare la cartella sopracitata **ssd\_mobilenet\_v2\_coco\_2018\_03\_29**; creare una cartella training ed all’interno copiare i due file sopra nominati: **ssd\_mobilenet\_v2\_coco.config** e  **san\_bernardo.pbtxt**

Eseguire quindi le seguenti modifiche:

1. Nel file san\_bernardo.pbtxt modificare il campo name con il valore inserito in labelImg ed in gerate\_tfrecord.py. Quindi rinominare il file a proprio piacimento.
2. Nel file **ssd\_mobilenet\_v2\_coco.config** modificare invece le seguenti righe:

Alla riga 179: label\_map\_path: “training/nome\_file\_pbtxt”

Alla riga 193: label\_map\_path: “training/nome\_file\_pbtxt”

Alla riga 164 sarà possibile modificare il numero di passi di addestramento che si vogliono eseguire, commentando la riga l’addestramento proseguirà per un tempo indefinito e dovrà essere arrestato dall’utente.

**NOTA:** I file da me forniti e altri modelli ed altri file di configurazione sono presenti ai seguenti link:

**Modelli:** <https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md>

**Config:**

<https://github.com/tensorflow/models/tree/master/research/object_detection/samples/configs>

Il modello da me scelto presenta un buon compromesso tra precisione e velocità il che permette successivamente un buon utilizzo dell’applicazione Android. I parametri da tenere in considerazione nella scelta saranno la velocità (speed espressa in ms), la precisione (maP) e l’output (siamo interessati a boxes).

Eventuali scelte differenti dalle mie necessitano, oltre alle prcedenti, anche la modifica anche delle righe indicanti l’input path con il percorso indicato nel file da me fornito (nello specifico righe: 177 e 191).

**NOTA 2:** A seconda della tipologia di addestramento scelta potrebbe essere necessario modificare anche la riga 143 indicante la dimensione del batch ovvero il numero di immagini elaborate assieme ad ogni step.

Il valore di default è 24, un numero troppo grande può causare overfitting dei dati o uso eccessivo di memoria.

Esecuzione train tensorflow (CPU o GPU)

1. Spostare i i file ottenuti dalla cartella temp a models/research/object\_detection nel seguente modo:

data: copiare i dati interni collocarli dentro la cartella data esistente

images: copiare tutta la cartella nella cartella models/research/object\_detection

copiare la cartella contenente il modello pre-addestrato nella cartella models/research/object\_detection

copiare la cartella training nella cartella models/research/object\_detection

A questo punto si lavorerà esclusivamente all’interno della cartella models fornita da TensorFlow.

1. Eseguire dalla cartella models/research i seguenti comandi da terminale

**protoc object\_detection/protos/\*.proto --python\_out=.**

**export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim**

**NOTA** potrebbe essere generato un messaggio di errore all’esecuzione del primo dei due comandi (questo problema è dovuto alla versione di protobuf che dovrà quindi essere aggiornato). Per fare ciò:

eseguire il seguente comando da una cartella differente da models/research e scaricare la versione 3.2 di protobuf:

**curl -OL https://github.com/google/protobuf/releases/download/v3.2.0/protoc-3.2.0-linux-x86\_64.zip**

(potrebbe essere richiesto di installare curl, seguire istruzioni su terminale).

Quindi eseguire i seguenti step:

Da terminale estrarre il file dal pacchetto zip

**unzip protoc-3.2.0-linux-x86\_64.zip -d protoc3**

spostare il contenuto della cartella bin in usr/local/bin

**sudo mv protoc3/bin/\* /usr/local/bin/**

spostare il contenuto della cartella include in usr/local/include

**sudo mv protoc3/include/\* /usr/local/include/**

Verificare la versione di protoc installata digitando (la versione installata dovrebbe essere la 3.2.0):

**protoc --version**

**NOTA2 SE NEL CORSO DELLA GUIDA COMPARISSERO ERRORI DEL TIPO NO MODULE NAMED “ QUALCOSA” OPPURE IMPOSSIBLE TO IMPORT NAME “QUALCOSA” ESEGUIRE DALLA CARTELLA models/research IL SEGUENTE COMANDO export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim**

**E POI RIPROVARE CIO’ CHE HA GENERATO L’ERRORE.**

1. Dalla cartella models/research/object\_detection eseguiamo il seguente comando da terminale:

**python train.py --logtostderr --train\_dir=training/ --pipeline\_config\_path=training/ssd\_mobilenet\_v2\_coco.config**

**NOTA** Il processo potrebbe generare warning indicanti che viene ecceduto il 10% della memoria, questo non risulta essere un reale problema. Se si eccedesse la memoria disponibile ed il processo venisse killato sarebbe necessario ridurre la [batch size](#batch).

Se però il processo continuasse ad essere killato anche con batch size uguale a 1 allora non è possibile computazionalmente eseguire la fase di training in locale.

Un altro problema chesi potrebbe riscontrare è un eccessivo tempo impiegato per l’esecuzione di uno step di addestramento: considerando il fatto che, l’addestramento da me eseguito conta un numero di step pari a 200.000 con batch size a 24, è opportuno svolgere qualche conto sul tempo che potrebbe essere necessario a compierne a sufficienza (non è necessario eseguirne un numero così elevato ma qualche decina di migliaia la reputo quasi necessaria, in fase di testing si chiarirà quale è il numero necessario per avere il comportamento desiderato del modello).

L’addestramento può essere interrotto in qualunque momento digitanto CTRL + C

1. Una volta avviata è possibile verificare l’andamento della fase di training:

Aprire un nuovo terminale, spostarsi nella cartella models/research/object\_detection ed eseguire il seguente comando per poter utilizzare tensorboard, fondamentale per verificare l’andamento della fase di addestramento.

**tensorboard --logdir=/training --port=8888**

Digitare quindi da browser localhost:8888 per accedere alla pagina di tensorboard.

Durante il training verranno salvati periodicamente i valori assunti dai vari parametri che verranno rappresentati graficamente, mostrando il valore locale del parametro.

Sarà possibile analizzare differenti grafici, quello a cui prestiamo attenzione è il total lossì ovvero la funzione che viene minimizzata durante il training: il total loss dovrà mediamente scendere al di sotto di uno per considerare accettabile l’addestramento, più si avvicina a zero meglio è.

Il fatto che siano presenti dei picchi e che quindi il valore di loss locale salga al di sopra di 1 non significa che l’addestramento sia peggiorato, siamo interessati di fatti al valore medio del parametro.

Quando si ritiene che l’addestramento sia buono o quando sono stati compiuti tutti i passi impostati nel file di configurazione è possibile arrestarlo.

Sarà opportuno, nel caso della prima opzione controllare che nella cartella models/research/training siano presenti tra i vari file, tutti e tre i seguenti file riportanti lo stesso valore numerico riguardante i passi eseguiti.

model.ckpt-*numero\_passi\_eseguiti*.index

model.ckpt-*numero\_passi\_eseguiti*.data-00000-of-00001

model.ckpt-*numero\_passi\_eseguiti*.meta

Esecuzione train Google Cloud

Un’alternativa all’utilizzo delle proprie componenti hardware è l’utilizzo di Google Cloud.

Questo permette di eseguire la fase di train in cloud utilizzando un numero variabile di worker ed una potenza di calcolo nettamente superiore ad un singolo calcolatore.

Google cloud mette a disposizione 300 euro di credito ai nuovi iscritti che vengono consumati in base all’uso. La creazione di un account richiede l’inserimento della propria carta di credito ma fino a che è disponibile il credito fornito oppure per il tempo di prova di 365 giorni non viene prelevata alcuna somma di denaro.

1. Seguire il tutorial al seguente link per creare un nuovo progetto, attivare la fatturazione , abilitare le API di Google Cloud Machine Learning ed installare Google Cloud SDK.

<https://cloud.google.com/ml-engine/docs/tensorflow/getting-started-training-prediction>

1. Dal seguente link seguire il tutorial per eseguire il setup sdi Google SDK dal proprio terminale ed accedere al progetto precedentemente creato. (Il tutorial è realizzato per SO Ubuntu).

<https://cloud.google.com/sdk/docs/quickstart-debian-ubuntu>

1. Da Google Cloud creare un nuovo bucket.

Creare all’interno una cartella data in cui dovranno essere posizionati i seguenti file:

* File.config (ssd\_mobilenet\_v2\_coco.config o quello scelto da voi)
* File train.record
* File test.record
* File .pbtxt (Il nome sarà quello assegnato ed utilizzato in fase di setup dell’ambiente)
* Modello pre addestrato: i tre file nella cartella **ssd\_mobilenet\_v2\_coco\_2018\_03\_29** definita in fase di setup: model.ckpt.info, model.ckpt.meta, model.ckpt.data-00000-of-00001 dovranno essere collocati tutti nella cartella data del bucket.

1. Sarà necessario modificare il file di configurazione definendo come percorsi dei file i seguenti:

**gs://nome\_bucket/data/nome\_file.formato\_file**

Questo è da applicare per le righe mostrate sopra nella fase di configurazione quindi input\_path e label\_map\_path.

Un esempio è comunque presente nella cartella Bernardi\_Object\_Detection/training ed è il file ssd\_mobilenet\_v2\_coco.config

1. Configurare il path da terminale

**export PROJECT=$(gcloud config list project --format "value(core.project)")**

**export YOUR\_GCS\_BUCKET="gs://nome\_bucket\_creato"**

1. Eseguire da terminale i comandi seguenti dalla cartella tutorial/models/research:

**protoc object\_detection/protos/\*.proto --python\_out=.**

**export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim**

1. Avviare l’esecuzione da terminale copiando ed incollando il comando presente nella cartella Bernardi\_Object\_Detection/GoogleCloudUtil chiamato **google\_cloud.txt**

Il comando specifica l’uso di tensorflow versione 1.8, (è comunque opportuno verificare l’ultima versione supportata da google cloud) e l’uso del file di configurazione mostrato in precedenza.

Eventuali differenze necessitano ovviamente di opportuna modifica.

1. Avviata la fase di training sarà possibile monitorare differenti fattori da google cloud:

Dal menù laterale selezionando ML-Engine e Job sarà possibile vedere lo stato del/dei job attualmente in esecuzione.

Cliccando su visualizza log di ognuno di essi sarà possibile visualizzare i log stampati durante l’addestramento; questo permetterà di visualizzare, una volta avviato il processo, il worker, il numero di passi, il total loss (ovvero la funzione che vuole essere minimizzata) ed il tempo impiegato per ciasuno step.

Su fatturazione sarà visibile il proprio credito residuo.

In fine su storage e poi browser è possibile vedere il contenuto del porprio bucket: durante la fase di addestramento verrà creata la cartella train, contenente il modello addestrato, e verranno periodicamente caricati i tre file rappresentanti il modello. In questo modo si a avranno sempre a disposizione gli ultimi 4/5 file prodotti. Questi vengono sovrascritti mano a mano che ne vengono salvati di nuovi per non causare un’eccessiva occupazione di memoria (anche la quantità di spazio occupato infatti avrà un costo mensile seppure veramente molto ridotto).

1. Dal terminale di google cloud accessibile dal pulsante in alto a destra “attiva google cloud shell” sarà possibile attivare tensorboard e monitorare l’andamento della fase di train digitando il seguente comando:

**tensorboard - -logdir gs://nome\_bucket/train - -port 8080**

quindi sulla barra superiore del terminale di google cloud premere il pulsante vicino alla matita “visualizza anteprima web”.

Analogamente a quanto detto per [l’addestramento eseguito sul proprio calcolatore](#tensorboardo) il parametro da seguire sarà il total loss.

L’addestramento può essere interrotto in qualunque momento dalla pagina dei job.

Esportazione

Questa fase ci permettà di ottenere l’informazione contenuta nei file di checkpoint in un unico file in formato .pb.

1. Una volta terminata la fase di training a seconda della metodologia di addestramento seguita saranno

presenti rispettivamente nella cartella training all’interno di models/research/object\_detection o nella cartella train del bucket i tre file costituenti il modello addestrato con indicato in numero di step a cui sono stati salvati:

Nel caso di google cloud:

train\_model.ckpt-*numero\_step*.data-00000-of-00001

train\_model.ckpt-*numero\_step*.index

train\_model.ckpt-*numero\_step*.meta

Nel caso di TF in locale:

model.ckpt-*numero\_step*.data-00000-of-00001

model.ckpt-*numero\_step*.index

model.ckpt-*numero\_step*.meta

I file ottenuti dal mio addestramento sono presenti nela cartella Bernardi\_Object\_Detection/training, sono stati forniti sia i file del modello salvato allo step 32.909 sia quelli ottenuti in conclusione della fase di training allo step 192.586 (l’addestramento è proseguito fino allo step 200.000 ma ho scelto il modello

che tra gli utlimi salvati aveva il valore di loss locale più piccolo, 0,3779).

1. Se l’addestramento è avvenuto con Google Cloud scaricare i tre file più recenti, o quelli vicini alla terminazione del processo ma con un buon valore di loss, e collocarli nella cartella models/research/object\_detection/training
2. Dalla cartella models/research/object\_detection aprire il file **export\_inference\_graph.py** e da

terminale eseguire il comando presente dalla riga 53 alla 57 modificandolo opportunamente nel seguente modo:

Alla prima riga aggiungere l’estensione del file: **export\_inference\_graph.py**

Alla seconda riga del comando inserire il percorso al file di conifgurazione utilizzato, nel caso mostrato in esempio: **training/ssd\_mobilenet\_v2\_coco.config**

La terza riga del comando inserire il percorso dei file ckpt mostrati sopra ed inserire il numero di step, modificando nel seguente modo: **training/model.ckpy-*numero\_step*** (questo se l’addestramento è avvenuto in locale) oppure **training/train\_model.ckpy-*numero\_step*** (nel caso di Google Cloud).

L’ultima riga il percorso in cui creare una cartella con i file prodotti dall’esportazione

Un esempio del file è quello da me utilizzato e fornito nella cartella Object\_Detection\_Bernardi

E’ possibile, nel caso in cui non si fosse riusciti ad ottenere un buon modello o non si potesse eseguire l’addestramento per insufficiente capacità di calcolo, eseguire tale operazione utilizzando i file presenti nella cartella Object\_Detection\_Bernardi/training.

Per fare questo ed evitare ogni tipo di errore copiare la cartella training e il file export\_inferece\_graph.py sostituendo quelli esistenti nella cartella models/research/object\_detection. Quindi eseguire il comando sopracitato dalla riga 53 alla 57 nel file export\_inferece\_graph.py da me fornito modificando eventualemte il numero di passi a seconda di quale si voglia esportare.

1. Nel caso in cui la cartella di output non fosse presente essa sarà creata automaticamente e conterrà i tre file distinti rappresentanti il modello addestrato che possono essere utlizzati per far partire un nuovo addestramento dal punto in cui si è interrotto.

Ad essi si aggiunge il file **frozen\_inference\_graph.pb** che contiene in esso l’informazione dei tre file appena citati, costituendo quindi il modello. Questo file sarà necessario nelle prossime fasi.

La cartella di output ottenuta dal mio addestramento è presente nella cartella Object\_Detection\_Bernardi ed è denominata **exported\_model\_directory.** Il contenuto sarà lo stesso spiegato sopra ma il modello sarà quello da me addestrato e sarà opportunamente rinominato per la fase successiva di esportazione nell’applicazione android: **ssd\_mobilenet\_v1\_android\_export.pb**.

Test addestramento (Jupyter notebook)

1. Una volta terminata la fase di addestramento è possibile verificare come è avvenuto lo stesso utilizzando jupyter notebook.

Per fare ciò è necessario da terminale spostarsi nella cartella /models/research/object\_detection ed eseguire **jupyter notebook.**

1. Si aprirà una pagina web, selezionare da essa il file **object\_detection\_tutorial.ipynb** ed eseguire alcune modifiche :

MODEL\_NAME = ‘*nome\_cartella\_contenente\_frozen\_inference\_graph”*

Cancellare:

MODEL\_FILE = MODEL\_NAME + '.tar.gz'

DOWNLOAD\_BASE = '<http://download.tensorflow.org/models/object_detection/>'

Modificare:

PATH\_TO\_LABELS = os.path.join(‘training', *nome\_file*.pbtxt’)

NUM\_CLASSES = 1

Modificare eventualemente se si rinominasse il modello addestrato:

PATH\_TO\_CKPT = MODEL\_NAME + '/nome\_modello\_adddestrato’

Cancellare tutto download model

Modificare:

TEST\_IMAGE\_PATHS = [ os.path.join(PATH\_TO\_TEST\_IMAGES\_DIR, 'image{}.jpg'.format(i)) for i in range(1, N) ]

Dove N sarà il numero di immagini +1 presenti nella cartella models/research/object\_detection/test\_images

1. Aggiungiamo alla cartella test\_images delle immagini che chiamiamo ‘image*i’* (i è un numero crescente per ogni immagine) e cambiamo il range a seconda del numero di immagini che andiamo ad utilizzare.
2. Eseguiamo cell -> run all per vedere come si comporta.

È normale la presenza di un warning nella sezione di import.

Se ci fosse un errore in fase di importazione che impedisce la corretta esecuzione, eseguire dalla cartella models/research il seguente comando **export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim.**

Quindi tentare nuovamente cell -> run all.