**3. INTRODUZIONE GENERICA AL TRAINING**

La computer vision è l’insieme di tutti quei task che consentono di automatizzare i comportamenti di un computer, permettendogli di analizzare direttamente un’immagine.

Considerando l’andamento delle performance degli algoritmi che operano nell’ambito della computer vision, è impossibile non considerare come negli ultimi anni, questo strumento abbia fatto passi da gigante nel campo dell’object detection.

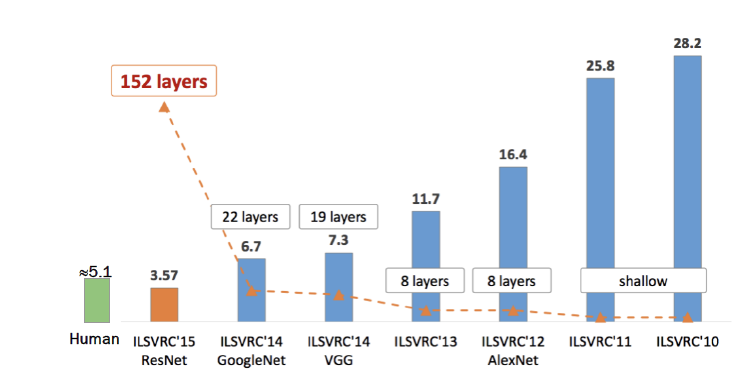


Figura 3.1 – immagine relativa alla progressione della percentuale di errore all’interno dei sistemi di computer vision.

Già dal 2015, possiamo notare come la percentuale relativa ad errori di classificazione, sia diminuita drasticamente, arrivando addirittura a superare l’errore umano.

Uno dei principali fattori, responsabili di questo successo e soprattutto senza il quale il concetto di computer vision non avrebbe ragione di esistere, è il training.

La fase di training dipende fortemente dalla capacità computazionale della macchina su cui viene eseguita, non è

quindi un caso se la straordinaria evoluzione della computer vision coincida con l’introduzione, all’interno del training, delle GPU che permetteno di velocizzare i tempi di allenamento della rete.

Il training è la fase del processo di object detection, nella quale gruppi di dati vengono utilizzati per addestrare un sistema a riconoscere determinati oggetti, nel caso specifico si tratta di una mano.

Ancora prima, però, di poter parlare di training è necessario citare due attività ad esso correlate: la creazione del dataset[[1]](#footnote-1) di immagini contenenti l’oggetto da rilevare e il concetto di classificazione delle immagini.

Quest’ultima viene svolta grazie ad un classificatore, il cui scopo primario è quello di analizzare un’immagine e restituire la probabilità che contenga una mano.

Per fare ciò, il classificatore deve essere stato addestrato su un insieme di dati contenente immagini etichettate come mani ed è proprio a questo punto che entra in gioco il training.

Tramite il training, si va ad allenare il classificatore, su un set di dati sintetici, al fine di rilevare la posizione ma soprattutto la presenza di una mano nell’immagine.

Il training è basato su una tecnica di apprendimento supervisionato, ovvero un metodo che, tramite una serie di step di allenamento, permette al sistema di elaborare automaticamente previsioni dell’output sulla base dell’input che viene fornito.

Il sistema composto da algoritmo addestrato, dati e parametri operativi costituisce il modello.

**Un aspetto strettamente connesso con il training è la loss function, ovvero** una quantità che mette in relazione il **valore predetto**dal modello e il valore **corretto**, del dataset di training.

Ovviamente più la loss è piccola più il modello è accurato e tendenzialmente il suo valore tende a diminuire progressivamente con l’aumentare degli step effettuati.

Tornando al concetto di training, tutti i dati utilizzati per l’addestramento, prendono il nome di training set e consistono in un vettore di immagini in input a cui viene successivamente associata una risposta o una determinata classificazione.

Una volta eseguito, l'algoritmo di training apprende, in base alla risposta o alla classificazione, quali sono le caratteristiche fondamentali che permettono di individuare gli elementi appartenenti alla categoria d’interesse (la mano) all’interno dell’immagine.

Successivamente alla fase di apprendimento, è necessario verificare la correttezza e le prestazioni del modello, eseguendolo su un insieme di dati, chiamato test set, per rilevare eventuale overfitting o underfitting rispetto al training set.

In base alle performance del modello si può parlare di:

* Underfitting: il modello ha prestazioni scarse sul training set perché non è in grado di ridurre l’errore tra i dati in input e i valori di uscita.
* Overfitting: il modello funziona bene con il training set, ma non con il test set, perché memorizza i dati che ha osservato, ma non è in grado di generalizzare il modello con dati mai osservati.
* Well–fitted: il modello restituisce buoni risultati sia con il training set che con il test set.

Per il progetto, sono stati utilizzati per il training set, dati sintetici creati con Unity, mentre come test set, le immagini delle nostre mani riprese tramite webcam del computer.

**3.1 BASI PER IL TRAINING**

Il punto di partenza per effettuare il training è la cartella di progetto *tesi*, nella quale si va ad effettuare la clonazione della directory *models*, fornita da GitHub per TensorFlow 1.8.0, tramite il comando: *git clone https://github.com/tensorflow/models.git*.

*Models* non è altro che una repository contenente una serie di diverse implementazioni di modelli e soluzioni di modellizzazione per gli utilizzatori di TensorFlow.

In particolare, la raccolta di modelli di nostro interesse è *research* che contiene implementazioni di codice e modelli pre–addestrati.

In questa cartella si trovano le due subdirectories che verranno utilizzate per effettuare la fase di training: *UnityStuff* e *object\_detection*.

**3.2 UnityStuff**

*UnityStuff* è la cartella, creata in ambiente Unity, che contiene la maggior parte delle informazioni necessarie per procedere con il training.

Questa directory non è presente in *research,* già dal momento in cui si effettua la clonazione di *models,* ma viene aggiunta in un secondo momento, a seguito della generazione del training set.

All’interno di *UnityStuff* troviamo alcune cartelle indispensabili per il training: *trainOutput*, *train*, *finalOutput* e *TFUtils*

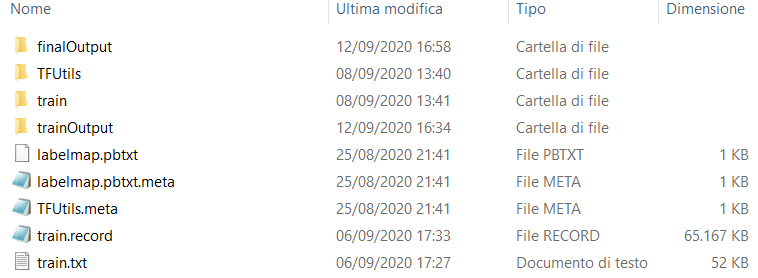


Figura 3.2 – contenuto della cartella UnityStuff.

**3.2.1 CARTELLA** **trainOutput**

Sostanzialmente il training è costituito da una serie di step, ognuno dei quali ha associato un valore, relativo alla loss, e un numero che lo identifica univocamente.

Nel momento in cui si esegue il comando relativo al training viene creata la cartella *trainOutput*.

All’interno in *trainOutput* si trovano una serie di file con una denominazione del tipo model.ckpt–XXXX dove “XXXX” corrisponde al un valore numerico che, come appena descritto, identifica lo step.

Il training non va a salvare in *trainOutput* un file model.ckpt–XXXX per ogni step effettuato ma, dopo ogni esecuzione, salva il valore dell’ultimo step effettuato, detto checkpoint.

Per ogni esecuzione, il training realizza un certo numero di step, valore che può essere modificato nel file *.config* del modello.

Quando il training viene riavviato non riparte quindi da zero ma dall’ultimo checkpoint memorizzato.

Infine, sono presenti una serie di file .LAPTOP-PAPUPM4Q che fanno riferimento ai grafici riguardanti l’andamento della loss al procedere degli step.

**3.2.2 CARTELLA train**

*Train* è la cartella nella quale vengono salvate tutti i file *.jpg,* relativi alle immagini che andranno a comporre il dataset sintetico, e i corrispettivi file .meta generati automaticamente da Unity.

Questo aspetto è stato già ampiamente approfondito nei paragrafi precedenti.

**3.2.3 CARTELLA finalOutput**

La directory *finalOutput* è la cartella che viene creata quando, una volta ultimato il training, si esegue l’inference del modello addestrato.

Il concetto di inference verrà approfondito in seguito, durante la definizione di tutti i comandi utilizzati per il training.

File:

* *frozen\_inference\_graph.pb*: è un file che fa riferimento ad un grafico congelato, ovvero un grafico sul quale non è possibile eseguire nuovamente un training poiché le sue variabili sono diventate costanti.

Inoltre è serializzato, cioè la sua struttura è stata trasformata in un set di dati binari al fine di renderla facilmente trasmissibile e memorizzabile.

Questo file costituisce il modello usato per l’object detection.

* *saved\_model.pb*: è un modello che deve essere importato nella sessione, contiene il grafico completo con tutti i pesi relativi all’allenamento, proprio come il grafico congelato, ma a differenza di quest’ultimo può essere allenato più volte perché le sue variabili non vengono memorizzate all’interno del file, inoltre non è serializzato.
* *model.ckpt*: questi file sono i checkpoint, generati durante l'allenamento, che vengono utilizzati per riprendere il training o per avere una copia di backup nel momento in cui qualcosa vada storto dopo l'allenamento.

Cartelle:

* *saved\_model*: è la cartella all’interno della quale troviamo il file *saved\_model.pb* descritto precedentemente.

**3.2.4 CARTELLA TFUtils**

Questa cartella è, sotto certi aspetti, il cuore del progetto, ovvero quella che ci permette di realizzare il training.

È proprio all’interno di questa directory che, tramite CMD, vengono eseguiti i comandi necessari ad avviare, non solo il training, ma anche l’inference e il test effettivo per verificare la bontà del progetto.

File:

* *ssdlite\_mobilenet\_v2\_coco*: questo file contiene principalmente un modello di rilevamento rapido degli oggetti, addestrato sul set di dati coco.

SSD è un rilevatore di oggetti abbastanza veloce da poter essere utilizzato, come nel nostro caso, su video in tempo reale.

Esistono molte varianti di SSD, in particolare quella utilizzata si avvale di MobileNet\_V2 come spina dorsale e prende il nome di **SSDLite** perché è dotata di convoluzioni separabili in profondità per i livelli SSD.

MobileNet\_V2[[2]](#footnote-2) è un’architettura di rete neurale che funziona in modo molto efficiente su dispositivi mobili.

Coco[[3]](#footnote-3) è un set di dati di rilevamento, segmentazione e didascalia di oggetti su larga scala.

* *CreateTFRecord.py*: durante un processo di object detection può essere utile serializzare i dati e archiviarli in una serie di file che possono essere letti in modo lineare.

TFRecord è un formato semplice per memorizzare una sequenza di record binari che utilizza buffer di protocollo come libreria per una serializzazione efficiente dei dati strutturati.

I messaggi di protocollo, detti Protobuf, sono definiti da file *.proto* e sono un tipo di messaggio flessibile che rappresenta una mappatura del tipo {“string”: value}.

I Protobuf sono progettati per essere utilizzati con TensorFlow e in tutte le API di livello superiore.

Il file TFRecord può essere letto solo in sequenza e contiene una sequenza di record, dove ogni record è costituito da una stringa di byte, per il payload dei dati, più la lunghezza dei dati e hash CRC32C per il controllo dell'integrità.

Infine i record vengono concatenati assieme per produrre il file.

* *testDetection.py*: è il file che contiene gli script di python che verranno eseguiti durante la fase di inference.

Questi script contengono, innanzitutto, il numero di classi da utilizzare, ovvero le tipologie di oggetti che si vogliono rilevare.

Ci sono inoltre il percorso relativo al file *frozen\_inference\_graph.pb*, che costituisce il modello usato per l’object detection, e la lista delle stringhe da utilizzare per aggiungere la label corretta ad ogni box contenente la mano rilevata.

Nel progetto troviamo un numero di classi pari a 1 perché si deve identificare solo un oggetto, ossia la mano, mentre la lista di stringhe consiste, in realtà, nel solo valore “hand”.

Infine una parte del codice di questo file è riservata all’attivazione della videocamera USB, necessaria per effettuare il test finale di rilevamento della mano.

Cartelle:

* *ssdlite\_mobilenet\_v2\_coco*: in questa cartella troviamo gli stessi elementi descritti nel paragrafo precedente, riguardante la directory *finalOutput*, perché è proprio da *ssdlite\_mobilenet\_v2\_coco* che il comando di inference preleva i dati necessari alla creazione di *finalOutput* per verificare il corretto funzionamento del modello addestrato.

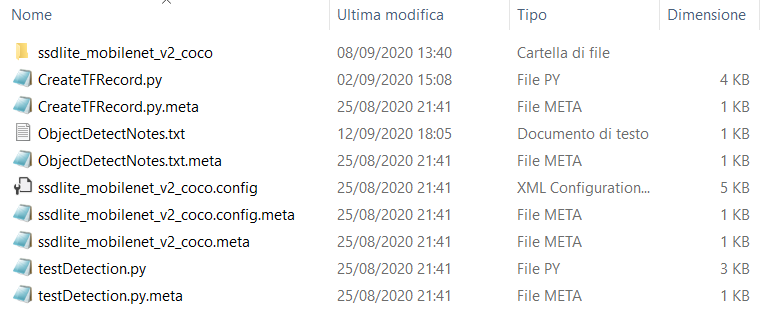


Figura 3.3 – contenuto della cartella TFUtils.

**3.3 Object\_detection**

*Object\_detection* è la cartella, già presente in *research* al momento della clonazione di *models*, che racchiude al suo interno l’API TensorFlow Object Detection[[4]](#footnote-4), ovvero un

framework open source, basato su TensorFlow, che semplifica la creazione, l'addestramento e la distribuzione di modelli di rilevamento degli oggetti.

Fondamentalmente, questa cartella, viene utilizzata nel

momento in cui si vanno ad eseguire i comandi relativi a

training e inference e prende in considerazione il file

*export\_inference\_graph.py* e le cartelle *protos* e *legacy*[[5]](#footnote-5).

**3.4 TRAINING**

Terminata la carrellata di file e cartelle che fanno da requisiti fondamentali, si procede col descrivere la parte pratica della fase di training.

Il training, di per sé, inizia nel momento in cui si ha a disposizione il training set di Unity.

La prima cosa da fare, partendo dalla cartella *tesi*, è spostarsi, tramite linea di comando, all’interno di *models\research* ed eseguire il comando

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

All’interno della cartella *protos* si trova l'API Tensorflow Object Detection che utilizza Protobuf[[6]](#footnote-6) per configurare i parametri del modello e di addestramento.

Per poter utilizzare il framework, è necessario compilare le librerie Protobuf ed è possibile farlo tramite il comando di cui sopra.

A questo punto, spostandosi all’interno di *UnityStuff\TFUtils*, si procede con il comando *python* *createTFrecord.py* che va ad eseguire, tramite python, il file *CreateTFrecord.py*[[7]](#footnote-7).

Una volta creato il TFrecord inizia il training vero e proprio.

Sempre all’interno di *TFUtils* si digita *python ../../object\_detection/legacy/train.py --pipeline\_config\_path=ssdlite\_mobilenet\_v2\_coco.config --train\_dir=../trainOutput/ --logtostderr.*

Tramite questo comando viene eseguito il file *train.py*, presente in *object\_detection\legacy*, che contiene gli script di python che permettono di allenare il modello.

Il file *train.py* non fa altro che andare a settare come path, per la configurazione della pipeline, il file *ssdlite\_mobilenet\_v2\_coco.config* che, come visto in precedenza, è un modello in grado di rilevare oggetti basandosi su una rete neurale e un dataset coco.

Nel momento in cui il training ha inizio viene creata la cartella *TrainOutput*, nella quale vengono memorizzati i checkpoint step, con relativo valore di loss, e inviati i log al file standard STDERR tramite la porzione di codice logtostderr.

Nel momento in cui si dovesse omettere questo parametro, i log verrebbero inviati a STDOUT che non registrerebbe più nulla e i log stessi apparirebbero sullo schermo, poiché STDOUT non viene reindirizzato a nessun file.

Una volta completato il training, è necessario testare il modello per verificare che funzioni nella maniera desiderata.

Per fare ciò si deve esportare il grafico di inferenza tramite lo script *export\_inference\_graph.py*, contenuto

nella cartella *object\_detection*, eseguendo il comando:

*python ../../object\_detection/export\_inference\_graph.py --input\_type image\_tensor --pipeline\_config\_path ssdlite\_mobilenet\_v2\_coco.config --trained\_checkpoint\_prefix ../trainOutput/model.ckpt-????? --output\_directory ../finalOutput.*

I parametri da passare sono sostanzialmente 3:

* il valore del checkpoint più alto tra i file che si trovano nella cartella *trainOutput*, considerando che per il modello sono stati eseguiti circa 75.000 step di allenamento.
* il path per la configurazione della pipeline che, come già descritto per il comando di training, è sempre *ssdlite\_mobilenet\_v2\_coco.config.*
* La posizione in cui si desidera che venga posizionato il grafico di inferenza, ovvero *finalOutput*.

A questo punto non rimane altro da fare che testare realmente il funzionamento del modello digitando il comando *python* *testDetection.py* che va ad eseguire il file *testDetection*.py[[8]](#footnote-8).

**4. RISULTATI**

Un fattore importante, per il corretto funzionamento del progetto, in termini di accuratezza nel rilevamento della mano, è la loss function.

All’inizio del training il valore della loss, relativo ai primi step di addestramento, è risultato essere abbastanza elevato, variando continuamente tra 8 e 10.

Il motivo di ciò è dovuto al fatto che, avendo iniziato da veramente poco l’allenamento, il rilevatore non si era ancora adattato al dataset sintetico fornito.

Già dopo un centinaio di step, la loss è calata notevolmente assestandosi su un valore di circa 1 e, continuando con il training, è arrivata a mantenersi costantemente attorno a 0.4/0.5 come valore ultimo.

Di seguito sono riportati i grafici dell’andamento della loss function, generati da TensorBoard[[9]](#footnote-9), tramite comando *python C:\Users\Matte\anaconda3\envs\tesi\Lib\site-packages\tensorboard\main.py --logdir=../trainOutput*.



Figura 3.4 – immagine del grafico relativo a Loss/classification\_loss.

Tag: Losses/Loss/ classification\_loss

Figura 3.5 – immagine del grafico relativo a Loss/localization\_loss. Tag: Losses/Loss/ localization\_loss

Figura 3.6 – immagine del grafico relativo a TotalLoss.

Tag: Losses/ TotalLoss.

Figura 3.7 – immagine del grafico relativo a clone\_loss.

Tag: Losses/ clone\_loss.



Figura 3.7 – immagine del grafico relativo a regularization\_loss.

Tag: Losses/ regularization\_loss.

Il testing si conclude con l’attivazione della webcam USB del pc, nella quale vengono inquadrate le mani utilizzate come test set.

I risultati sono i seguenti:

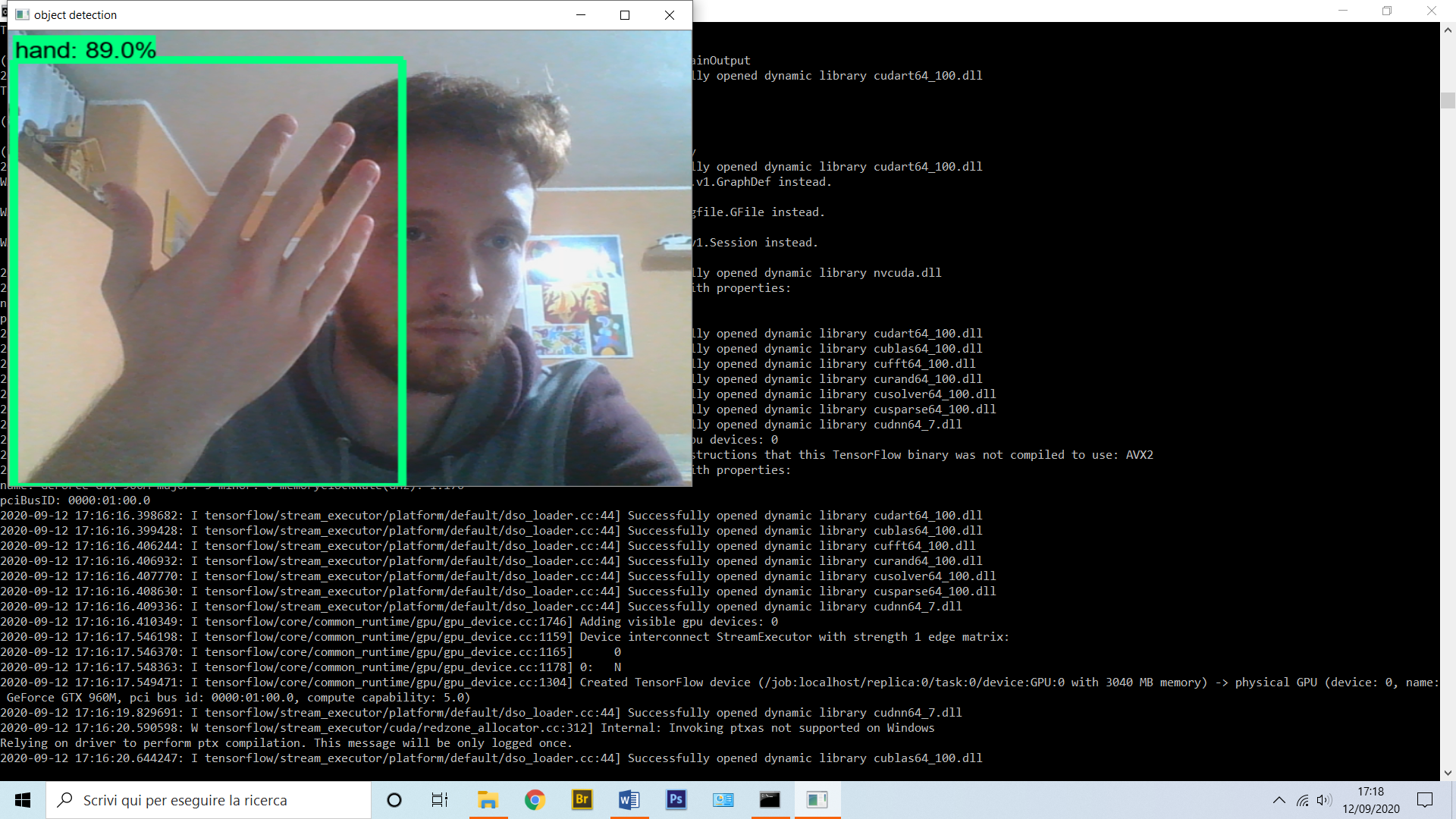


Figura 3.8 – rilevamento del dorso della mano.

.

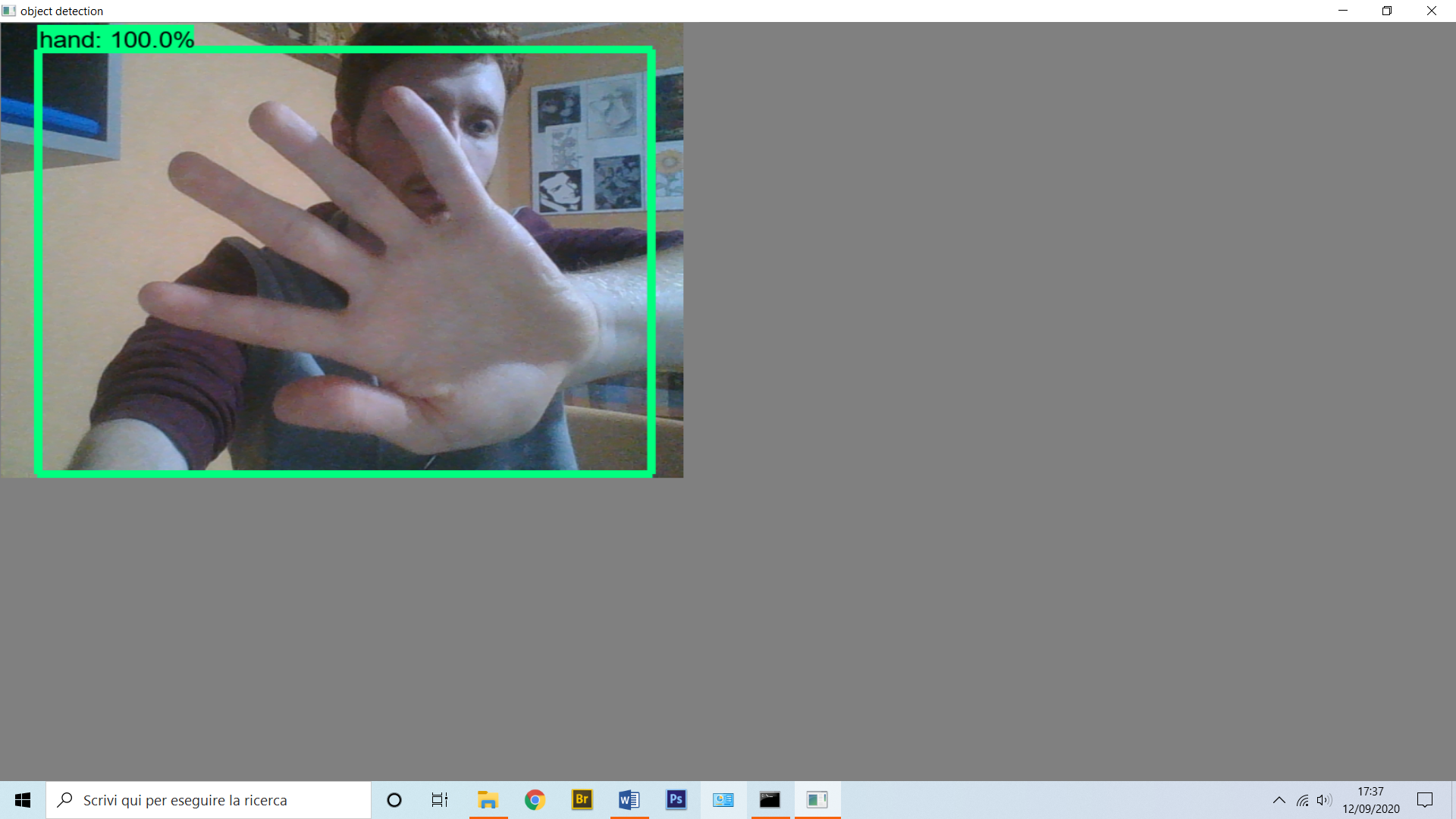


Figura 3.9 – rilevamento del palmo della mano.

.

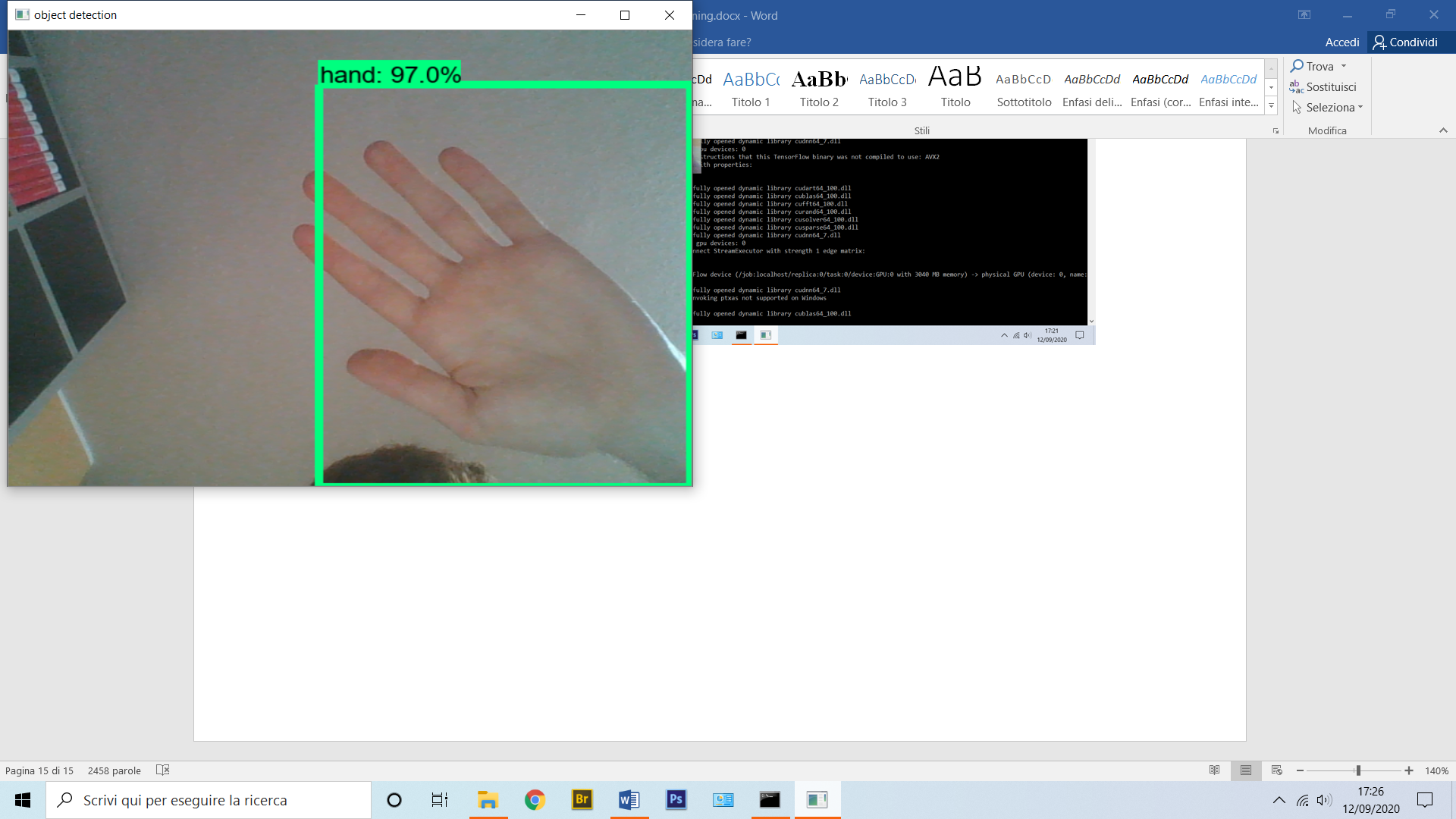


Figura 3.10 – rilevamento del palmo della mano.

.

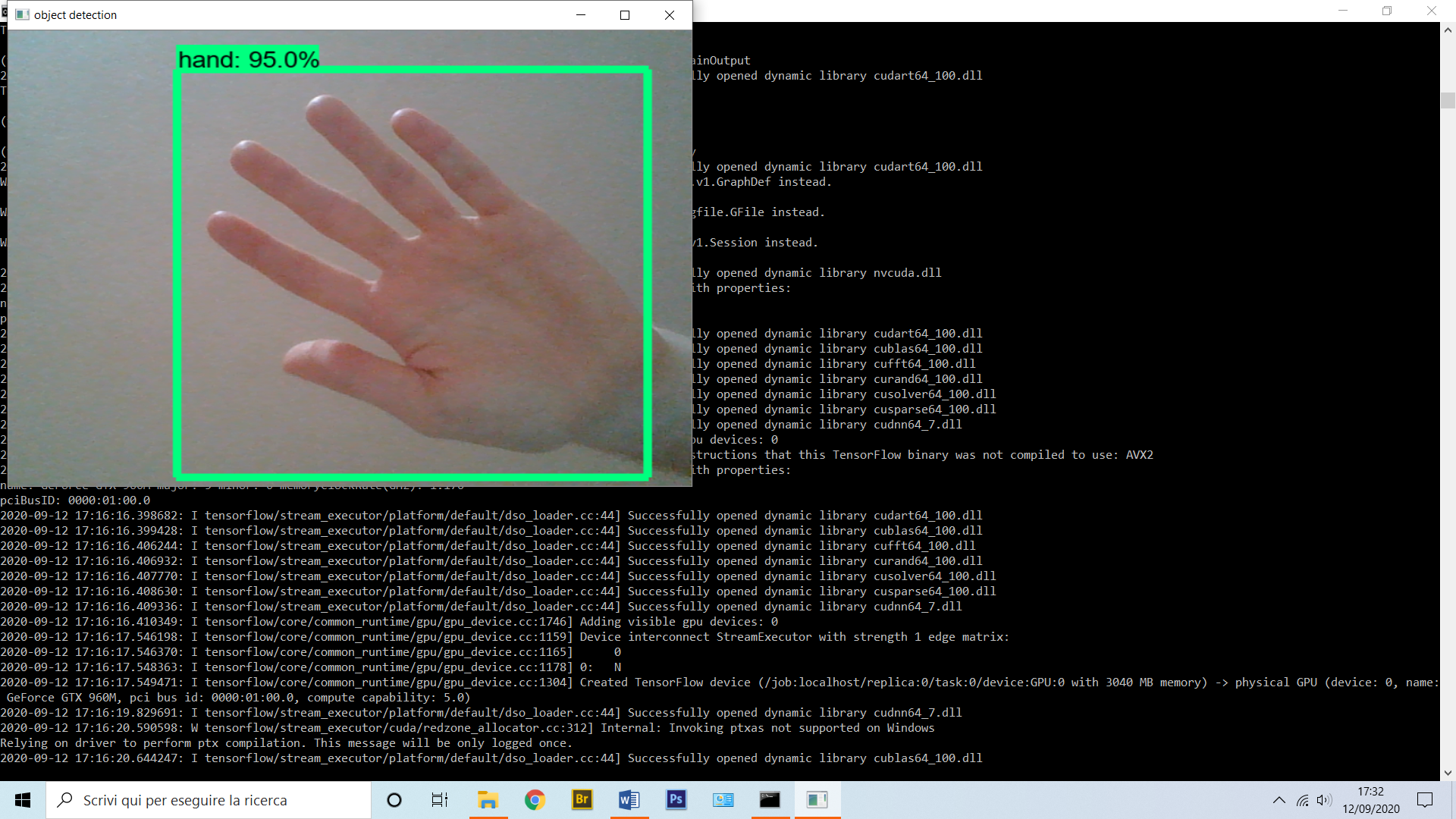


Figura 3.11 – rilevamento del palmo della mano.

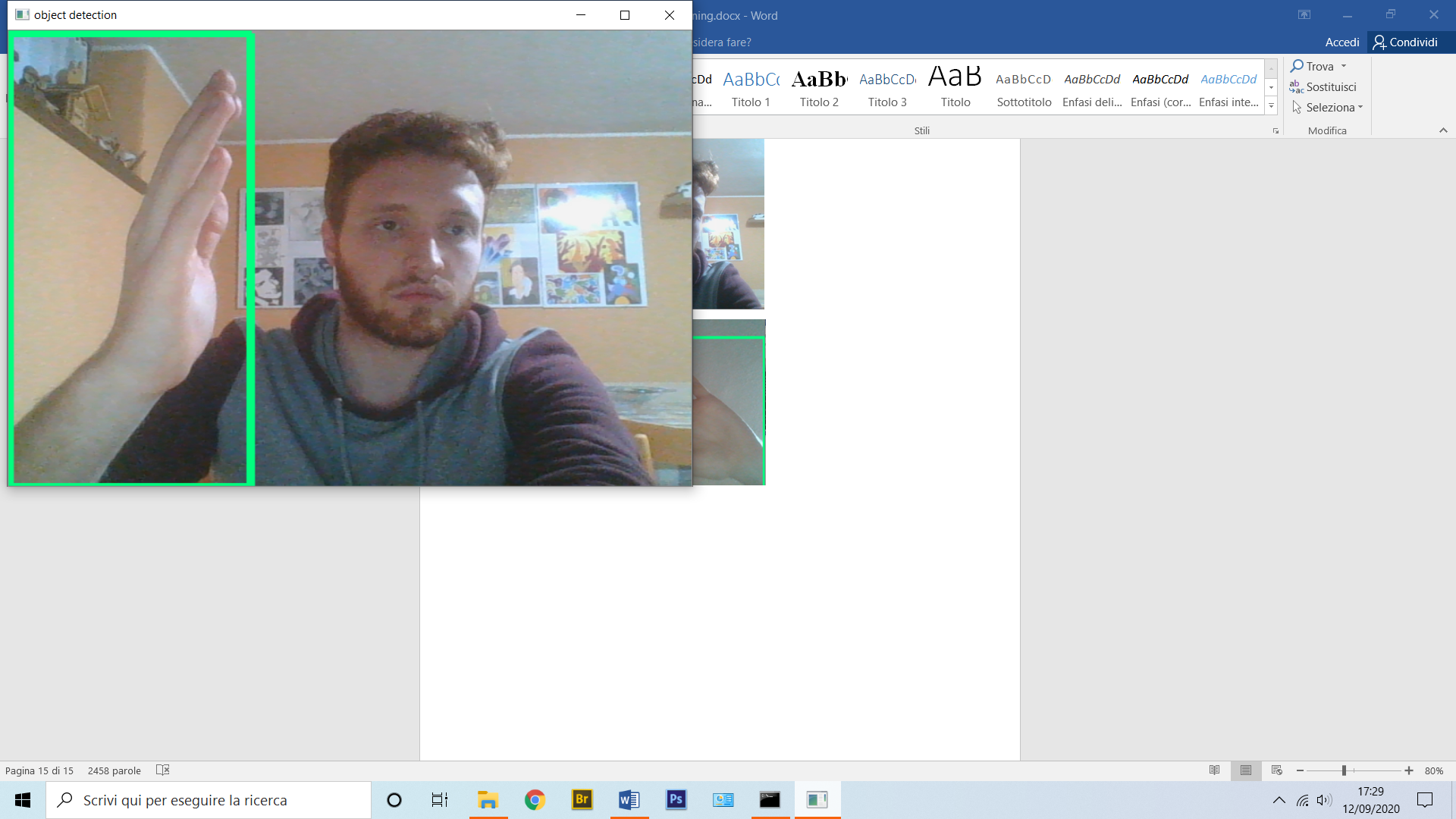


Figura 3.12 – rilevamento del profilo della mano.

1. La creazione del dataset è stata ampiamente analizzata nel capitolo 2 [↑](#footnote-ref-1)
2. Fonte: https://github.com/tensorflow/models/tree/master/research/slim/nets/ mobilenet [↑](#footnote-ref-2)
3. Fonte: https://cocodataset.org [↑](#footnote-ref-3)
4. Fonte: https://github.com/tensorflow/models/tree/master/research/object\_detection [↑](#footnote-ref-4)
5. Questi tre elementi verranno descritti nel prossimo paragrafo dedicato alla realizzazione del training e ai comandi utilizzati. [↑](#footnote-ref-5)
6. Fonte:https://github.com/tensorflow/models/blob/d530ac540b0103caa194b4824af353f1b07355 3b/research/object\_detection/g3doc/installation.md [↑](#footnote-ref-6)
7. La descrizione del file e i relativi effetti sono stati definiti nel paragrafo 3.3.4 [↑](#footnote-ref-7)
8. La descrizione del file e i relativi effetti sono stati definiti nel paragrafo 3.3.4 [↑](#footnote-ref-8)
9. TensorBoard è un toolkit di visualizzazione di TensorFlow per il tracciamento e la visualizzazione di metriche come perdita e accuratezza, fonte: https://www.tensorflow.org/tensorboard [↑](#footnote-ref-9)