

# Human Keypoint Detection on Android

Roberto Scolaro - [roberto.scolaro@studio.unibo.it](mailto:roberto.scolaro@studio.unibo.it)

27 giugno 2022

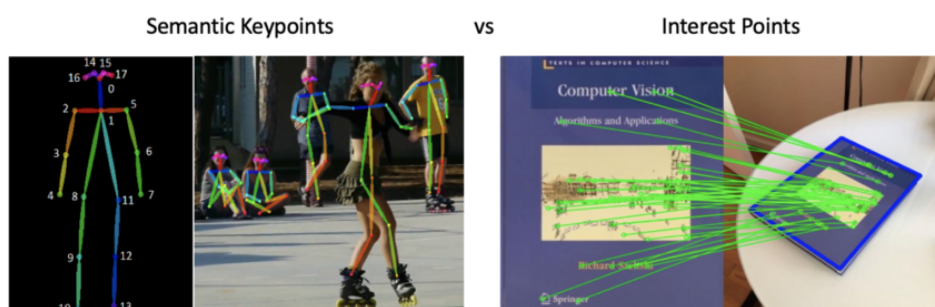
# Indice

<b>1</b>	<b>Analisi dell'obiettivo</b>	<b>3</b>
<b>2</b>	<b>Costruzione del modello</b>	<b>4</b>
2.1	Transfer Learning . . . . .	4
2.1.1	Dataset . . . . .	5
2.1.2	Costruzione del modello . . . . .	5
2.2	Problematiche riscontrate . . . . .	6
2.2.1	U-Net . . . . .	7
2.2.2	Heatmap . . . . .	8
2.2.3	Loss Function . . . . .	8
2.2.4	Dataset . . . . .	9
2.3	Training . . . . .	10
<b>3</b>	<b>App Android</b>	<b>12</b>
3.1	Quantizzazione . . . . .	12
<b>4</b>	<b>Risultati</b>	<b>15</b>

# Capitolo 1

## Analisi dell'obiettivo

L'obiettivo di questo progetto è quello di individuare gli “human keypoints”, come ad esempio le caviglie o i gomiti, su un dispositivo embedded che, in questo caso particolare, è uno smartphone Android.



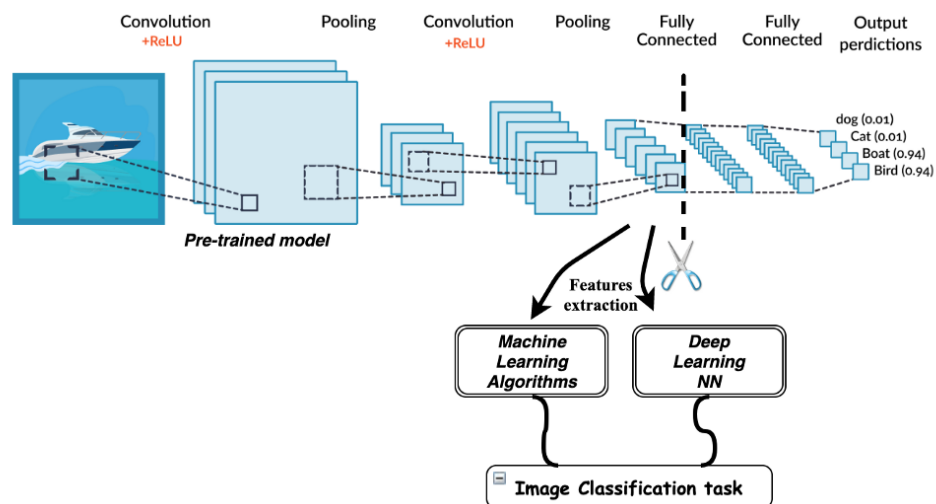
Questo problema non può essere risolto con i classici metodi di Computer Vision (quali corner detectors o keypoint detectors) dato che presuppone utilizzo di una determinata conoscenza semantica da parte dell'applicazione. Per questo motivo si rende necessario l'utilizzo di tecniche di Machine Learning per riuscire ad individuare i keypoints.

Inoltre, dato che l'inferenza andrà effettuata su un dispositivo embedded, andranno effettuate una serie di ottimizzazioni per sfruttare al meglio la limitata capacità computazionale.

# Capitolo 2

## Costruzione del modello

### 2.1 Transfer Learning



Il primo approccio utilizzato è stato quello del Transfer Learning che consiste nel prendere le features apprese da un modello per risolvere un determinato problema e utilizzarle per risolvere un problema che abbia dei punti in comune. In questo caso sono stati utilizzati come backbone (ossia come modello base) i modelli presenti in Tensorflow aventi a disposizione i pesi già allenati su imagenet tra cui:

- MobileNetV2
- MobileNetV3Small
- MobileNetV3Large

L'idea è stata quella di utilizzare le low level features (in particolare quelle relative al riconoscimento delle giunture degli uomini) apprese da un determinato modello e trasferire questa conoscenza a un nuovo modello per facilitare l'apprendimento.

Il metodo più comune per effettuare transfer learning consiste in 5 passi:

1. prendere i “bottom layer” di un modello già allenato
2. congelare i “bottom layer” per non distruggere quanto appreso precedentemente durante il successivo training
3. aggiungere dei nuovi layer che avranno il compito utilizzare la conoscenza dei “bottom layer” per trasformare le vecchie features in nuove necessarie per il nuovo dataset
4. allenare i nuovi layer con il nuovo dataset
5. infine, solitamente, si effettua il fine-tuning e cioè si scongelano i “bottom layer” per poi continuare il training con un learning rate molto basso (solitamente di almeno un ordine di grandezza inferiore) portando a un adattamento dei layer precedentemente congelati alle nuove features.

### 2.1.1 Dataset

Inizialmente è stato utilizzato il dataset COCO [3] congiuntamente al pacchetto “imgaug” che permette di effettuare data augmentation su immagini e keypoints senza dover applicare le omografie manualmente. COCO, infatti, comprende labels relative a object detection e segmentation e in particolare 250000 persone con relativi keypoints.

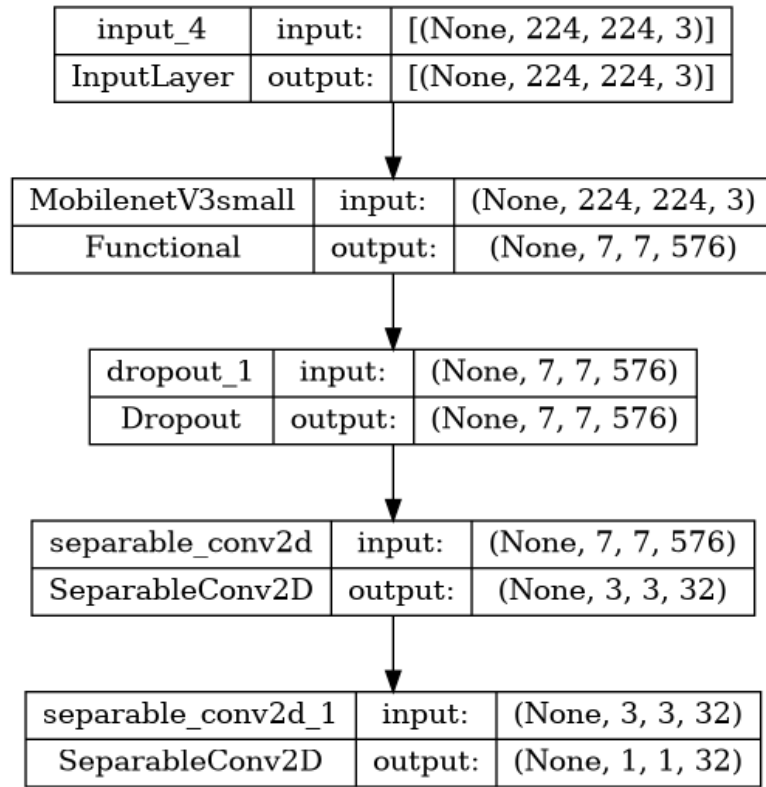
Solo successivamente è stato necessario cambiare dataset dato che COCO si è dimostrato inadatto nel caso di transfer learning ([5]).

Come nuovo dataset è stato scelto MPII [1]: questo nuovo dataset include 25000 immagini contenenti oltre 40000 persone annotate con le relative giunture; inoltre le immagini comprendono una tassonomia di oltre 410 attività quotidiane.

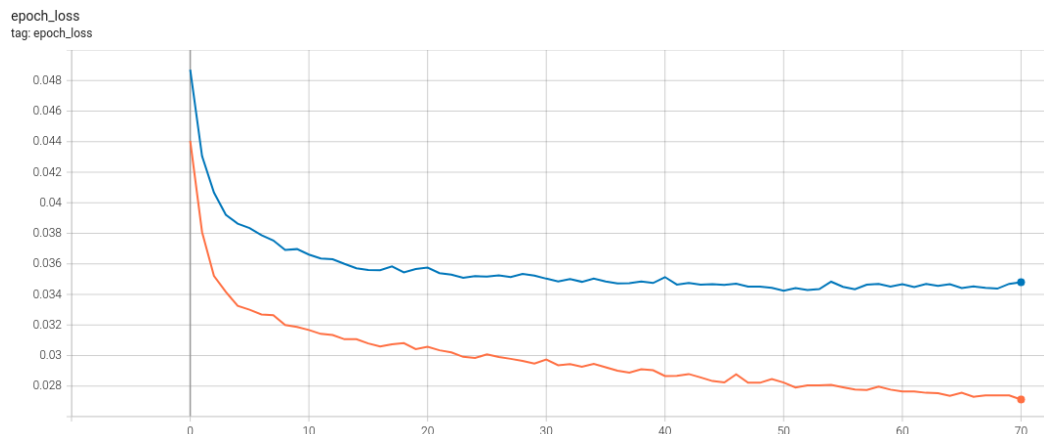
### 2.1.2 Costruzione del modello

Per effettuare la procedura di Transfer Learning sono stati utilizzati diversi backbone, come già anticipato, al fine di determinare la migliore base di partenza. Dopodiché, come da procedura, non sono stati inclusi i “top layer”

al fine di aggiungere una nuova serie di layer responsabile dell'apprendimento delle nuove feature.



## 2.2 Problematiche riscontrate

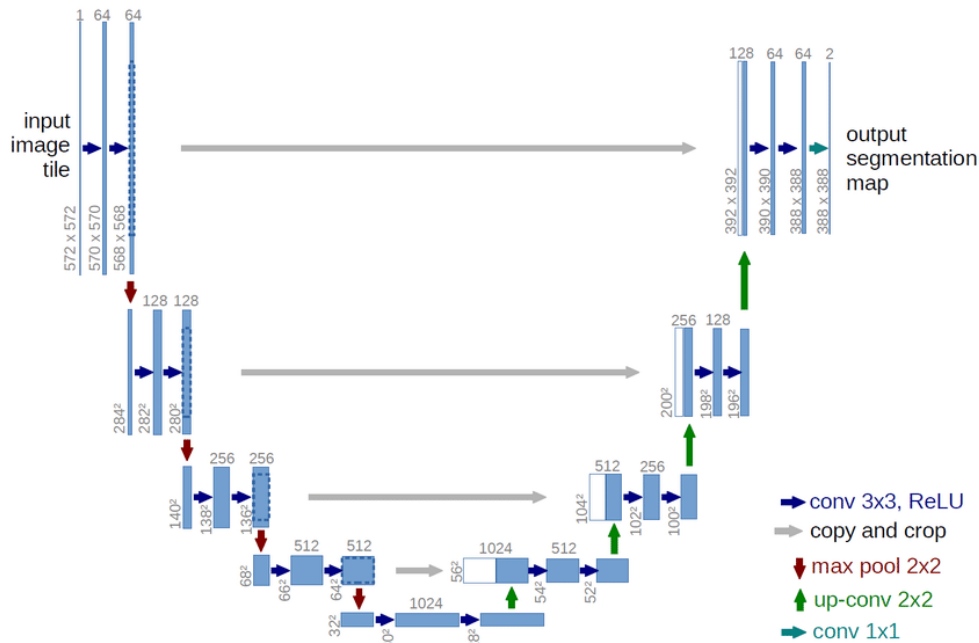


Dopo aver effettuato diversi tentativi con diversi backbone, iperparametri, top layers e dataset, dato che le performance del modello rimanevano pres-

soché invariate si è optato per nuovo approccio utilizzando un'architettura completamente diversa.

### 2.2.1 U-Net

U-Net [4] è un'architettura per “fully convolutional neural networks” (dato che contiene solo layer convoluzionali e nessun full-connected layer) specializzata in “image segmentation”. In particolare l’“image segmentation” è una procedura dove non solo si determina se un determinato soggetto si trova in un'immagine ma anche la sua posizione.



Il nome U-Net deriva dalla struttura del modello che è infatti costituito da due parti:

- **encoder**, la parte che si occupa di estrarre le features dall'immagine in input attraverso una sequenza di blocchi base; questi blocchi base sono costituiti da 2 convoluzioni  $3 \times 3$  ognuna delle quali viene seguita dalla funzione di attivazione ReLU. Ogni blocco base è poi seguito da un MaxPooling dove il numero delle feature viene dimezzato. Qui una classica rete utilizzata per classificazione aggiungerebbe dei dense layer per poi effettuare la vera e propria classificazione; nel caso di U-Net questo non è sufficiente ed è necessario estendere la rete per poter anche individuare i soggetti.

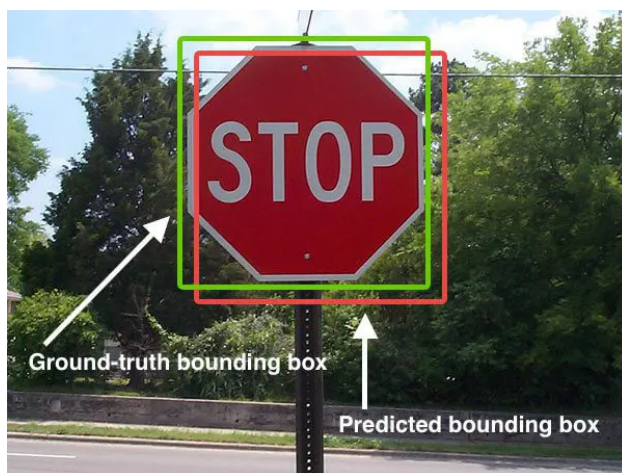
- **decoder**, la parte che si occupa di prendere le features generate dall'encoder e generare una maschera. Questa parte utilizza praticamente la stessa architettura dell'encoder (ad eccezione del MaxPooling) ma con la differenza che, per rendere la localizzazione più precisa, vengono usate delle "skip-connection" che consistono nel concatenare le feature map dell'encoder con l'output della "transposed convolution" (necessaria per ottenere una maschera della stessa dimensione dell'immagine di input) del livello corrispondente. L'utilizzo di "skip-connection" permette di sfruttare le feature apprese durante la fase di encoding per avere più informazioni utili durante la fase di decoding.

### 2.2.2 Heatmap

Dopo il tentativo, fallito, di usare direttamente le coordinate come output del modello è stata presa una strada alternativa che prevede l'utilizzo di una delle tecniche più utilizzate per stimare la posizione di keypoints: le **heatmap** e in particolare una heatmap per ogni keypoint.

La creazione delle heatmap è triviale: date le coordinate di un keypoint e le dimensioni dell'immagine si crea una matrice di zeri delle esatte dimensioni dell'immagine e si pone a 1 l'elemento avente le stesse coordinate del keypoint. A questo punto, per passare da un singolo punto a una zona di interesse avente pesi minori man mano che ci si allontana dal centro esatto, si applica un Gaussian filter. Infine si normalizza la maschera ottenuta per poter utilizzare la funzione sigmoid come ultimo layer.


### 2.2.3 Loss Function





Come loss function è stata scelta “Intersection Over Union”: solitamente questa funzione è utilizzata nel campo dell’image segmentation ma, in questo caso, è risultata comunque una buona scelta dato il problema analogo dato che in questo caso specifico al posto di due bounding box siamo in presenza di due heatmap.

L’equazione risulta piuttosto semplice ed intuitiva: al numeratore abbiamo l’area dell’intersezione delle due heatmap mentre al denominatore l’area della loro unione.

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$


Andando ad analizzare i singoli casi è immediatamente chiaro che l’unico il miglior caso è quello in cui le due heatmap coincidono che è esattamente l’obiettivo perseguito.



## 2.2.4 Dataset

Dopo aver allenato il modello con il dataset MPII sono stati ottenuti risultati discreti in fase di test ma, nel momento in cui è stata fatta inferenza su immagini reali l’accuratezza del modello non è risultata soddisfacente.

Dopo un’attenta analisi si è visto che il modello, pur essendo stato allenato su una singola persona alla volta riusciva a riconoscere più persone

nell'immagine: da qui è stato chiaro che probabilmente il dataset MPII non è il più adatto a questo task dato che contiene delle immagini con più persone ma con keypoint relativi soltanto a una.

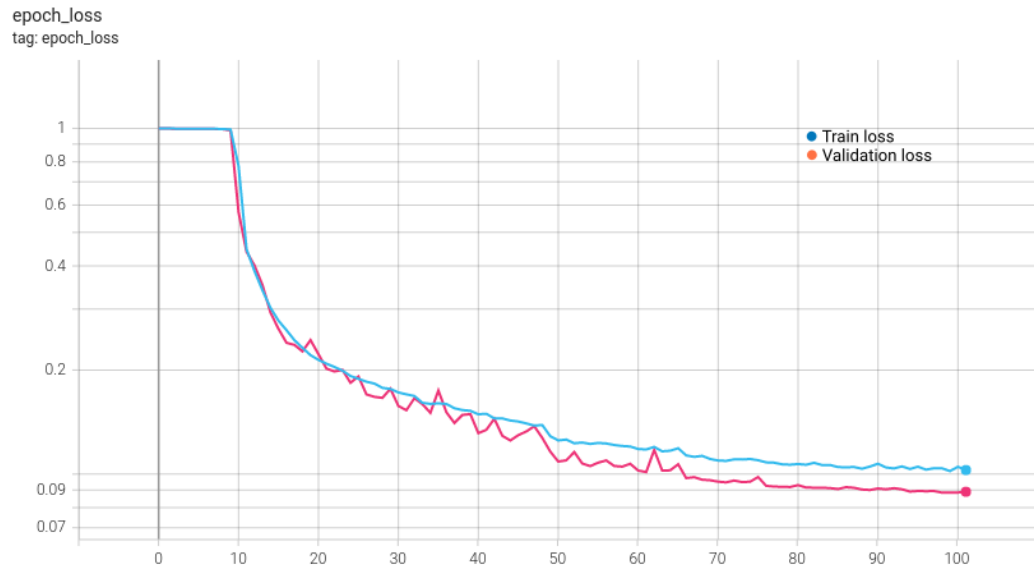
Da qui l'idea di utilizzare un terzo dataset che, invece, presenti solo una persona per ogni immagine: Human3.6M [2].

## 2.3 Training

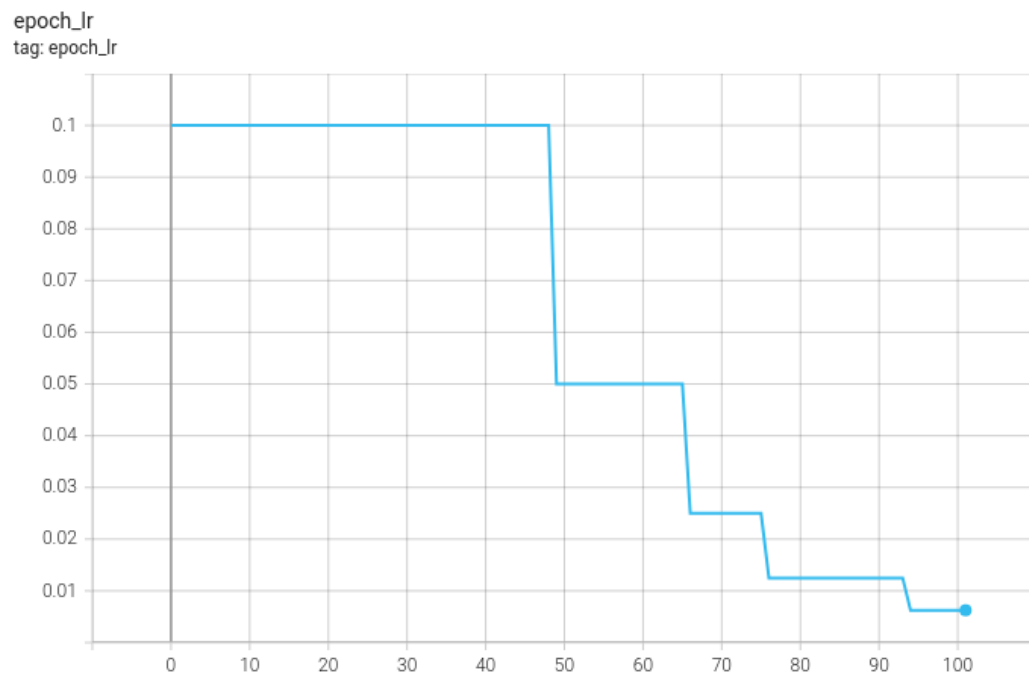
Per effettuare il training è stata utilizzata la piattaforma *slurm* messa a disposizione dall'università. In particolare si è fatto attenzione ad utilizzare la GPU (una NVidia RTX 2080 ti) per evitare che il training impiegasse tempi irragionevoli.

Quindi, per sfruttare al meglio le potenzialità della GPU, si è deciso di costruire una pipeline di input efficiente che potesse fornire i dati per il passaggio successivo prima che il passaggio corrente sia terminato. Per fare ciò è stata sfruttata l'API *tf.data.TFRecordDataset* che permette di convertire il dataset in un formato binario rendendo l'ingestione dei dati più efficiente.

Questa metodologia ha, però, portato a pessimi risultati e perciò si è deciso di utilizzare la più classi API *tf.keras.utils.Sequence* ed integrando già la data augmentation.



Succesivamente il modello creato è stato, poi, allenato per 100 epoche con l'accortezza di dimezzare il learning rate ogni 5 epoche nel caso in cui la loss del validation set non fosse diminuita.



# Capitolo 3

## App Android

Per utilizzare il modello creato si è deciso di creare un'app android a partire da uno skeleton fornito da Tensorflow.

In particolare sono state sfruttate le Android Neural Networks API (NNAPI) che permettono di accelerare i modelli Tensorflow Lite con acceleratori hardware quali:

- GPU
- Neural Processing Unit (NPU)
- Digital Signal Processor (DSP)

### 3.1 Quantizzazione

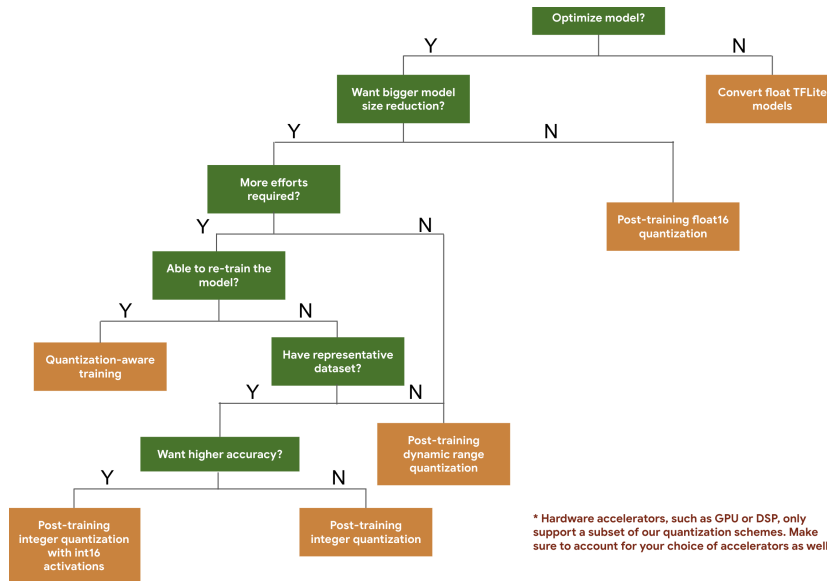
Dato che i dispositivi embedded hanno a disposizione una quantità limitata di risorse, quali batteria e memoria, spesso l'utilizzo diretto di modelli di deep learning risulta infattibile date le risorse richieste. Per questo bisogna il modello che si vuole utilizzare su tali dispositivi deve necessariamente avere i seguenti requisiti:

- dimensioni ridotte
- efficienti dal punto di vista dei consumi
- bassa latenza

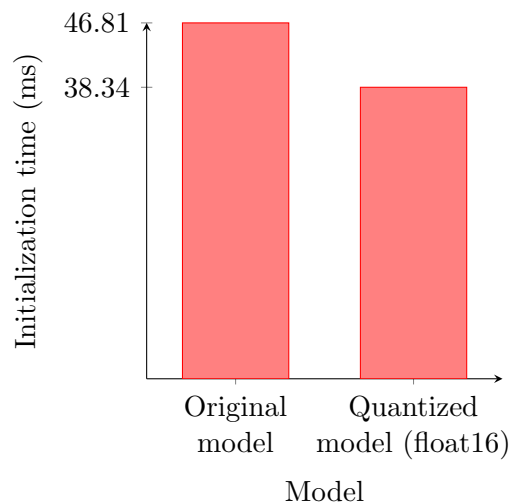
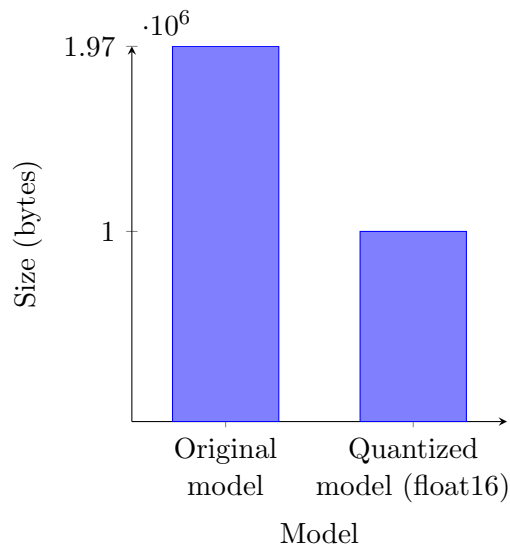
Per rispettare tali requisiti si applicano una serie di tecniche chiamate quantizzazione e in particolare Tensorflow fornisce un toolkit chiamato Tensorflow-Lite che, oltre a effettuare la quantizzazione fornisce le API necessarie per interfacciarsi con gli acceleratori hardware (NNAPI)

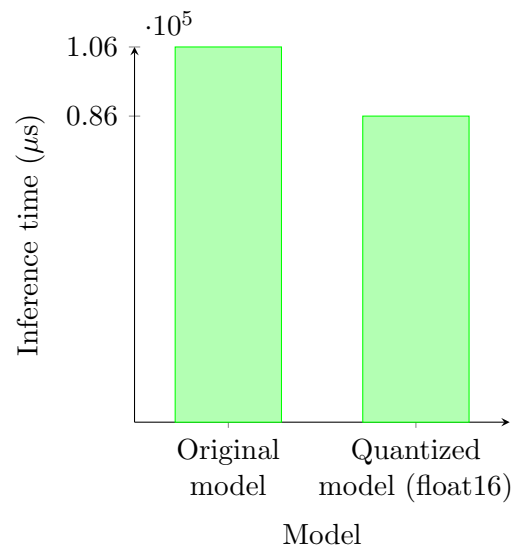
Tensorflow Lite è costituito da due componenti principali:

- l'interprete, che permette di utilizzare i modelli su diversi dispositivi embedded
- il convertitore, che si occupa di convertire i modelli Tensorflow in modelli TFLite che possono essere direttamente usati su tali dispositivi oltre e di applicare le ottimizzazioni.



Dato che il modello ottenuto è risultato già di piccole dimensioni ( $2MB$ ), seguendo le linee guida fornite da Tensorflow, si è proceduto effettuando una quantizzazione “float16” dopo il training ottenendo un modello di dimensioni inferiori a  $1MB$ .





# Capitolo 4

## Risultati

Come dall'esempio riportato, i risultati del training sembrano buoni.

Purtroppo effettuando predizioni su immagini al di fuori del dataset si ottengono pessimi risultati.

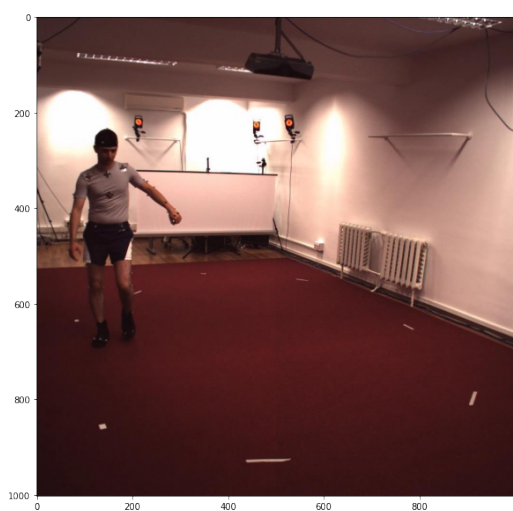


Figura 4.1: Immagine originale

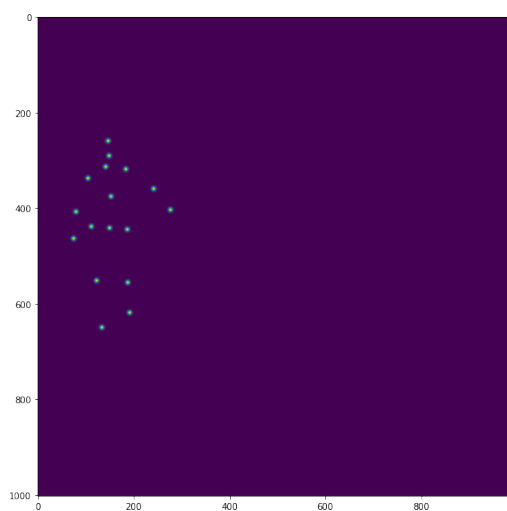


Figura 4.2: Keypoint reali

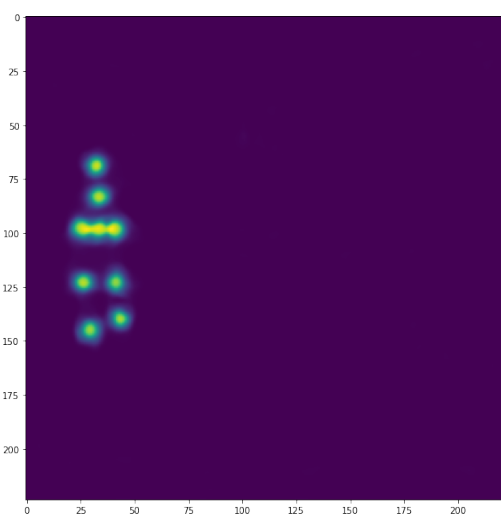


Figura 4.3: Keypoint predetti

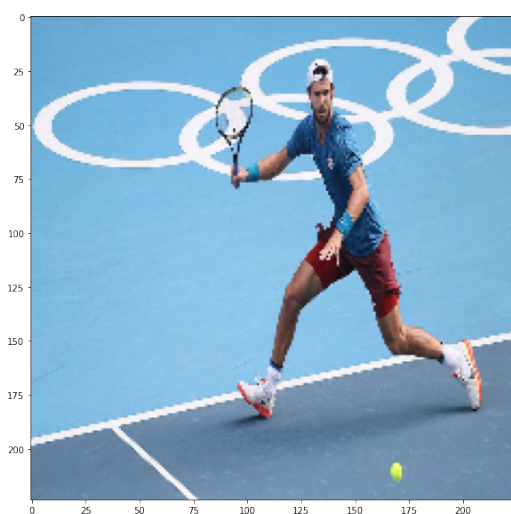


Figura 4.4: Immagine originale

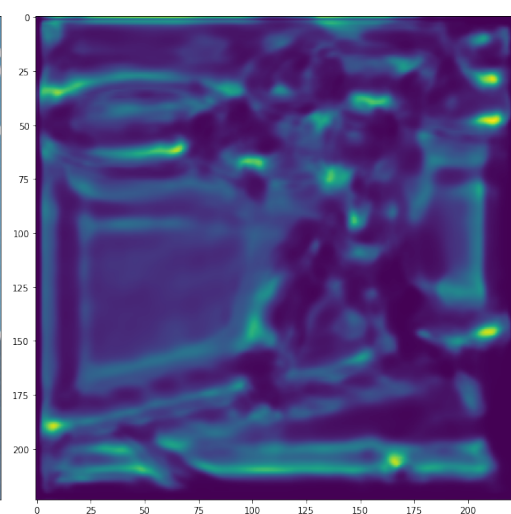


Figura 4.5: Keypoint reali

Considerato che erano state prese diverse precauzioni per evitare l'overfitting, tra cui early stopping, learning rate adattiva e data augmentation, la causa va, probabilmente, ricercata nel fatto di avere utilizzato solo parte del dataset per esigenze di tempo (utilizzando l'intero dataset una sola epoca avrebbe impiegato circa 5 ore).



# Bibliografia

- [1] Mykhaylo Andriluka et al. “2D Human Pose Estimation: New Benchmark and State of the Art Analysis”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Giu. 2014.
- [2] Catalin Ionescu et al. “Human3.6M: Large Scale Datasets and Predictive Methods for 3D Human Sensing in Natural Environments”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.7 (lug. 2014), pp. 1325–1339.
- [3] Tsung-Yi Lin et al. *Microsoft COCO: Common Objects in Context*. 2014. DOI: 10.48550/ARXIV.1405.0312. URL: <https://arxiv.org/abs/1405.0312>.
- [4] Olaf Ronneberger, Philipp Fischer e Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. DOI: 10.48550/ARXIV.1505.04597. URL: <https://arxiv.org/abs/1505.04597>.
- [5] Barret Zoph et al. “Rethinking Pre-training and Self-training”. In: *CoRR* abs/2006.06882 (2020). arXiv: 2006.06882. URL: <https://arxiv.org/abs/2006.06882>.