



UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO



Face to BMI

Documentazione
Caso di Studio
Sistemi ad Agenti
AA 2023-24

Vincenzo Monopoli
MAT. 760451
v.monopoli8@studenti.uniba.it

Sommario

Introduzione	3
Descrizione del dominio e dei dati utilizzati	4
Preparazione dei dati.....	4
Preparazione dei dati (immagini)	6
Apprendimento supervisionato e scelta del modello.....	10
Contesto e Obiettivi.....	10
Idea 1: Rete Neurale	13
Idea 2: Rete Neurale Convoluzionale (CNN).....	15
Idea 3: Uso di Dropout	17
Idea 4: CNN con meno layers.....	18
Idea 5: Modifiche al learning rate.....	19
Idea 6: Regolarizzazione L2	20
Idea 7: Augmentation	21
Idea 8: Modifiche al learning rate.....	22
Idea 9: Modifiche al modello	23
Idea 10: Fine Tuning.....	24
Conclusione e considerazioni finali	26

Introduzione

Il volto è il primo elemento che notiamo in una persona e può rivelare informazioni significative riguardo a età, sesso, condizioni psicologiche, povertà e condizioni di salute.

In questo contesto, il Body Mass Index (BMI) emerge come un indicatore ampiamente utilizzato per valutare lo stato nutrizionale e, di conseguenza, la condizione di salute di un individuo.

Studi recenti hanno dimostrato che un BMI elevato può portare a molte malattie come malattie cardiache, diabete, ictus, cancro, apnee notturne, ipertensione, malattie del fegato, malattie renali, depressione e nelle donne anche problemi di gravidanza.

Normalmente, il BMI viene calcolato a partire da due misurazioni, quali: altezza e peso.

$$BMI = \frac{\text{peso (kg)}}{\text{altezza (m)}^2}$$

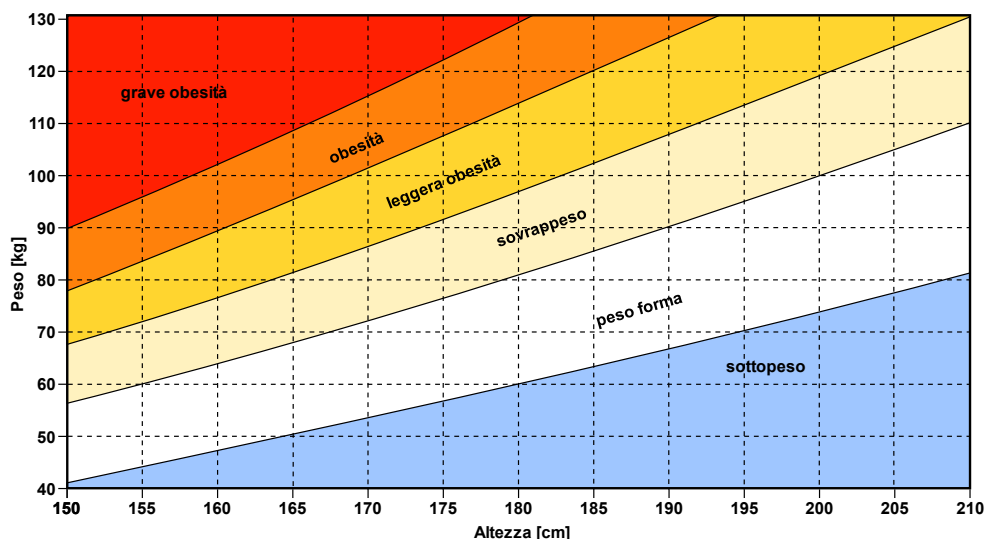
Tuttavia, questi metodi possono essere invasivi e/o richiedono la collaborazione attiva dell'individuo.

Negli ultimi anni, l'avvento dell'intelligenza artificiale e della visione artificiale ha aperto nuove possibilità per la valutazione dello stato di salute.

In questo lavoro, verrà presentato un modello in grado di predire il BMI di una persona direttamente da un'immagine del suo volto.

È fondamentale riconoscere che calcolare il BMI a partire dal volto non è sempre una soluzione accurata, infatti ha delle sue limitazioni, poiché non tiene conto della composizione corporea, della distribuzione del grasso e di altri fattori come ad esempio il livello di attività fisica.

Pertanto, è importante considerare che il BMI calcolato è strumento complementare e non come l'unica indicazione della salute di un individuo.



Descrizione del dominio e dei dati utilizzati

Il dominio è rappresentato dal dataset in formato csv con allegate immagini in formato *jpg* scaricato da [Kaggle](#).

Si tratta di un dataset formato da foto segnaletiche di detenuti e dei loro dati associati, quali: altezza, peso, ecc.

Il copyright del dataset impiegato è di dominio pubblico poiché prodotto dal governo (*Illinois Dept. of Corrections*).

Dal dataset originale è stato impiegato solo l'uso del data-frame *person.csv*, contenenti le informazioni per ogni soggetto, e la cartella *front* contenente le foto frontali di ogni soggetto.

Le feature presenti nel data-frame erano molte, ma sono state identificate solo tre come importanti per lo sviluppo del progetto:

Nome della feature	Descrizione
id	Identificatore univoco per ogni soggetto. Ogni identificatore è correlato al nome del file dell'immagine che rappresenta il volto del soggetto.
weight	Peso del soggetto espresso in <i>libbre</i> .
height	Altezza del soggetto espressa in <i>pollici</i> .

Preparazione dei dati

Il primo passo è stato come descritto poc'anzi quello di escludere tutte le feature irrilevanti ed isolare: id, peso e altezza per ogni soggetto, che permetteranno dunque, attraverso la formula del BMI, di calcolarci questo dato essenziale.

$$BMI = \frac{\text{peso (kg)}}{\text{altezza (m)}^2}$$

Poiché il peso è espresso in *libbre* è stata applicata una conversione per trasformarlo in *kilogrammi*, usando la seguente formula:

$$\text{peso (kg)} = \frac{\text{peso (lbs)}}{2.205}$$

Anche l'altezza è espressa in *pollici*, per questo motivo è stata applicata la conversione per trasformarla in *centimetri*, usando la seguente formula:

$$\text{altezza (cm)} = \text{altezza (m)} * 2.54$$

Una volta effettuate le necessarie conversioni, il BMI per ciascun soggetto è stato calcolato utilizzando la formula precedentemente indicata, dopo aver convertito l'altezza in *metri*.

Sono state quindi escluse tutte le colonne nulle o contenenti valori pari a zero nella colonna del BMI, se presenti.

Durante una prima analisi del dataset risultante, è emersa la presenza di valori incongruenti, con alcune misurazioni estremamente elevate o basse, che risultavano irrealistiche per la sopravvivenza umana:

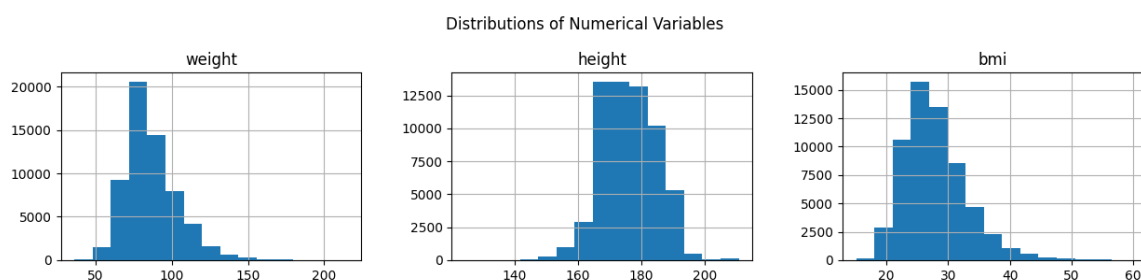
	weight	height	bmi
count	60715.000000	60715.000000	60715.000000
mean	86.519460	176.126315	27.877602
std	17.422152	8.425256	5.204424
min	7.709751	124.460000	2.828435
25%	74.829932	170.180000	24.323504
50%	83.900227	175.260000	27.119956
75%	95.238095	182.880000	30.645550
max	226.757370	241.300000	78.296865

Questo suggeriva la presenza di errori nei dati di origine.

Dunque è stato pensato di escludere dal dataset tutti i soggetti con un bmi inferiore o uguale a 15 (estrema magrezza) e soggetti con un bmi superiore o uguale a 60 (estrema obesità) ed inoltre arrotondare tutti i valori del bmi alla prima cifra decimale:

	weight	height	bmi
count	60673.000000	60673.000000	60673.000000
mean	86.485469	176.124235	27.866781
std	17.270021	8.418153	5.148376
min	36.281179	124.460000	15.100000
25%	74.829932	170.180000	24.300000
50%	83.900227	175.260000	27.100000
75%	95.238095	182.880000	30.600000
max	215.419501	210.820000	59.400000

Con queste operazioni di pulizia, il dataset risulta ora privo di anomalie e pronto per ulteriori analisi.



Grafici che rappresentano la distribuzione dei valori rispettivamente per: "peso", "altezza", "bmi"

Preparazione dei dati (immagini)

Il passo successivo ha riguardato la preparazione delle immagini per l'analisi da parte del modello.

Si è deciso di iniziare identificando il volto della persona presente nell'immagine, utilizzando tecniche di rilevamento facciale.

Una volta individuato il volto, si è proceduto effettuando un'operazione di parsing dell'immagine, che consiste nell'isolare l'area del volto, evidenziandola per migliorare la qualità dei dati in input.

Questo approccio permette di concentrare l'analisi su caratteristiche rilevanti, riducendo al contempo il rumore visivo e aumentando l'accuratezza delle previsioni del modello.

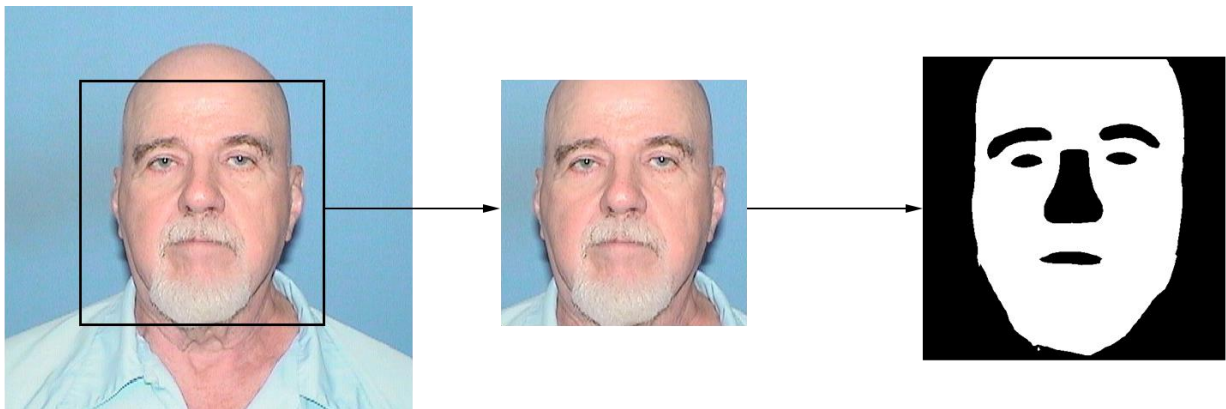


Immagine che mostra i passaggi compiuti durante l'elaborazione della immagine (da sinistra, verso destra)

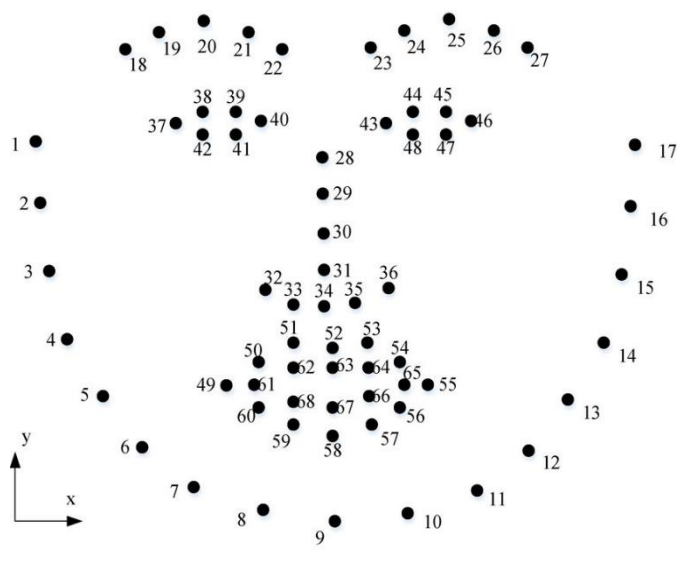
Per l'individuazione e estrazione del volto da un'immagine, si è fatto uso di librerie come *OpenCV* e *dlib*.

Inizialmente, viene caricata un'immagine e convertita in scala di grigi per facilitare il rilevamento.

Utilizzando un [classificatore pre-addestrato](#), il programma cerca di identificare un volto nell'immagine.

Se il sistema rileva un volto e ne trova esattamente uno, il passo successivo prevede l'utilizzo di un [modello di dlib](#) che analizza i punti di riferimento facciali, focalizzandosi in particolare sul punto che rappresenta la parte inferiore del volto.

Questa informazione è fondamentale perché permette di determinare l'altezza corretta del

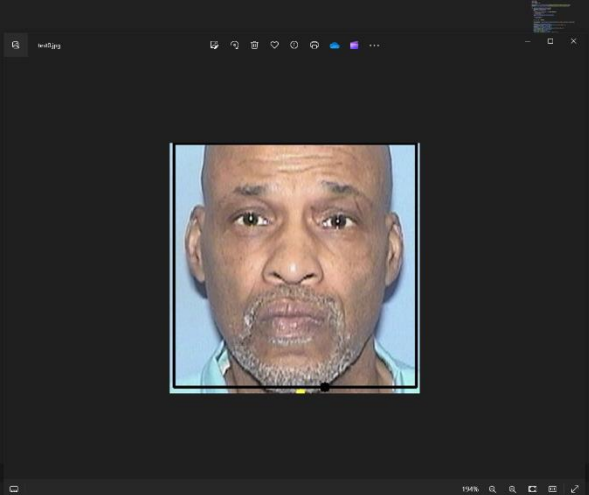


ritaglio, assicurando che il volto venga catturato in modo completo.

Una volta definite le dimensioni del ritaglio, il codice apporta eventuali regolazioni per mantenere il volto centrato e proporzionato.

A tal proposito è stato scoperto che il modello usato sembrava percepire meglio come parte finale del volto non il *landmark* 9, ma il *landmark* 8; infatti come si può ben vedere dalla immagine sottostante

```
28 cv2.circle(img, (landmark_9.x, landmark_9.y), 5, (0, 0, 0), -1)
29 print("posizione pt9: ", landmark_9.y)
30 h_max = abs(y - landmark_8.y)
31 print("h_max con punto 8: ", h_max)
32 print("h_max con punto 9: ", abs(y - landmark_9.y))
33 print("h_max del programma: ", h)
34 print("larghezza w: ", w)
35 max_d = max(w, h_max)
36 h_9 = max(abs(y - landmark_9.y), w)
37 if max_d == h_max:
38     h = h_max
39     aux = int(abs(w - h)/2)
40     aux_9 = int(abs(w - h_9)/2)
41     x_9 = x - aux_9
42     w_9 = w + 2*aux_9
43     x -= aux
44     w += 2*aux
45     if x < 0 or x+w > img.shape[1]:
46         (x, y, w, h) = faces[0]
47         h = w
48     cv2.rectangle(img, (x_9, y), (x_9 + w_9, y + h_9), (0, 0, 0), 2)
49 else:
50     h = w
51     face_region = img[y:y + h, x:x + w]
52     print("dim finali usando pt9: " + h + " | " + w)
53
posizione pt8: 367
posizione pt9: 360
h_max con punto 8: 257
h_max con punto 9: 250
h_max del programma: 229
larghezza w: 229
dim finali usando pt8: 257 | 257
[Finished in 1.6s]
```



Esempio fatto con una immagine presa casualmente dal dataset (punto nero: landmark 9 - punto giallo: landmark 8)

la cornice nera rappresenta come l'immagine sarebbe stata tagliata se si avesse fatto uso del *landmark* 9 come altezza massima; quindi, di una dimensione ridotta rispetto a come invece è stata tagliata e mostrata in video usando il *landmark* 8.

Poiché lo scopo prefissato era di cercare di prendere più porzione del volto possibile (quindi più informazioni possibili riguardanti il voto della persona) si è optato per usare il *landmark* 8 come altezza massima del volto.

Infine, l'area del volto estratta viene salvata in un percorso specificato.

Tuttavia, nonostante l'efficacia del modello, ci sono stati casi in cui l'individuazione del volto risultava imprecisa.

Per garantire l'accuratezza dei dati, è stato necessario eseguire un controllo su ogni immagine esportata.

A tal fine, è stato nuovamente impiegato il [classificatore pre-addestrato](#) per verificare se l'immagine estratta contenesse effettivamente un volto.

Questo passaggio di verifica si è rivelato cruciale per assicurare che solo le immagini valide venissero incluse nel dataset finale, contribuendo così a migliorare la qualità dei dati.

Ovviamente sono state anche applicate modifiche al dataframe contenenti gli id, peso, altezza e bmi; eliminando dunque tutte le tuple dei soggetti che non avevano una immagine identificata come valida.

Questa operazione ha garantito che il dataset rimanesse consistente.

L'ultimo passo nell'elaborazione delle immagini ha coinvolto la produzione di un'immagine parsata, in cui l'area del volto è stata ritagliata e isolata, evidenziata in bianco.

Per realizzare questo passaggio, è stato utilizzato un [modello di face parsing](#).

Quest'ultimo è in grado di segmentare il volto in diverse aree, distinguendo tra le caratteristiche come occhi, naso, bocca, sopracciglia, collo, vestiti, capelli e identificando anche lo sfondo.

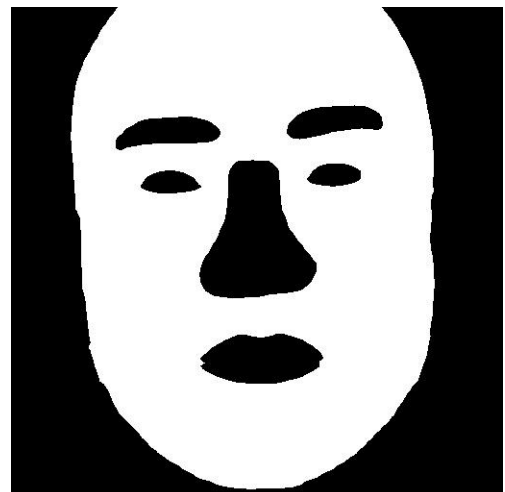


Il funzionamento del modello si basa su una rete neurale profonda che, addestrata su un ampio dataset di volti, è in grado di apprendere le peculiarità di ciascuna regione del volto.

Durante il processo di face parsing, il modello genera una maschera di segmentazione, in cui ogni area del volto può essere colorata in base alla sua classe specifica.

Per l'immagine finale, è stata applicata una maschera nera per escludere tutti i dettagli non rilevanti, lasciando solo l'area del volto ben definita.

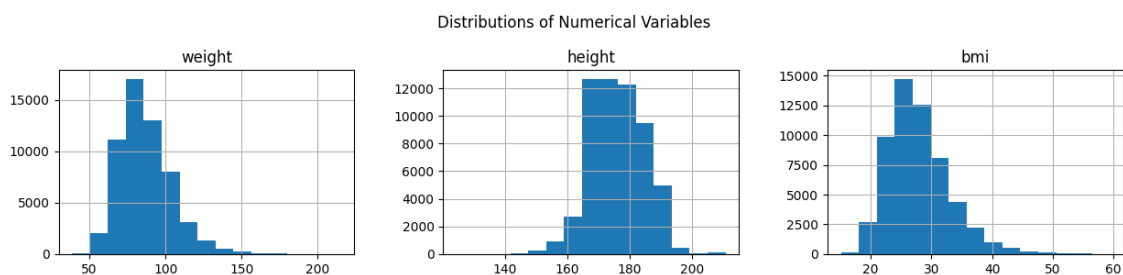
E' utile menzionare che l'immagine finale viene sempre salvata dal modello in un grandezza fissa: 512×512 px (.jpg), questo approccio migliora ulteriormente la qualità dei dati, poiché tutti della stessa dimensione.



I dati finali ottenuti sono stati quindi rianalizzati, ottenendo come si può notare una loro diminuzione rispetto alla fase precedente (circa 4000 *entries* in meno):

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 56694 entries, 0 to 56693
Data columns (total 4 columns):
#   Column  Non-Null Count  Dtype  
---  -
0   id       56694 non-null   object  
1   weight   56694 non-null   float64  
2   height   56694 non-null   float64  
3   bmi       56694 non-null   float64  
dtypes: float64(3), object(1)
memory usage: 1.7+ MB
None
```

	weight	height	bmi
count	56694.000000	56694.000000	56694.000000
mean	86.532890	176.118360	27.884632
std	17.233163	8.412165	5.138844
min	38.548753	124.460000	15.100000
25%	74.829932	170.180000	24.300000
50%	83.900227	175.260000	27.100000
75%	95.238095	182.880000	30.700000
max	215.419501	210.820000	59.400000



Ora i dati sono pronti per la fase successiva, ovvero, essere usati per addestrare il modello.

Apprendimento supervisionato e scelta del modello

Contesto e Obiettivi

L'obiettivo principale di questo progetto, come già accennato, è lo sviluppo e la creazione di un modello capace di predire il BMI di una persona partendo da una foto del suo volto.

- **Fase Iniziale:** si è partiti dalla costruzione di un modello di base, con l'obiettivo di stabilire un punto di partenza solido per successive ottimizzazioni.
- **Iterazioni e Miglioramenti:** successivamente, il modello è stato raffinato e perfezionato attraverso diverse iterazioni, cercando di incrementare costantemente le sue performance.
- **Confronto con altre Soluzioni:** parallelamente, sono state sviluppate e valutate altre architetture di rete neurale, al fine di identificare la soluzione più efficace ed efficiente (rispetto al test-set usato).

Per misurare l'accuratezza dei modelli sviluppati, sono state utilizzate le seguenti metriche:

- **Mean Squared Error (MSE):** Questa metrica calcola la media degli scarti quadratici tra i valori predetti dal modello e i valori reali del BMI. Fornisce una misura della dispersione degli errori, penalizzando maggiormente gli errori più grandi.

$$MSE = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$

- **Mean Absolute Error (MAE):** Il MAE calcola la media degli errori assoluti tra i valori predetti e i valori reali. Questa metrica è più intuitiva, in quanto rappresenta la deviazione media assoluta tra le previsioni e i dati reali.

$$MAE = \frac{1}{n} \sum_{i=1}^n |Y_i - \hat{Y}_i|$$

Perché queste metriche? Entrambe le metriche, MSE e MAE, sono comunemente utilizzate nella valutazione di modelli di regressione, come quello sviluppato in questo progetto. La scelta di utilizzare entrambe le metriche

permette di ottenere una visione più completa delle performance del modello, considerando sia la dispersione degli errori (MSE) sia la loro entità media (MAE).

Numero di epoche e dimensione del batch:

- Epoche (epochs): è stato impostato il modello ad allenarsi per 10 epoche. Un'epoca rappresenta una completa iterazione sull'intero dataset di addestramento. Aumentando il numero di epoche, il modello ha più opportunità di apprendere le caratteristiche più sottili dei dati.
- Dimensione del batch (batch_size): è stato scelto di utilizzare un batch size di 32. Il batch size indica il numero di esempi di addestramento che vengono processati in una singola iterazione dall'algoritmo di ottimizzazione.
Un batch size più piccolo può portare a un addestramento più stabile, ma richiede più iterazioni per processare l'intero dataset.

Divisione del dataset:

- È stato diviso il dataset in un insieme di addestramento (X_train, y_train) e un insieme di test (X_test, y_test) utilizzando una proporzione del 80% per l'addestramento e del 20% per il test.

Questa configurazione è stata scelta in seguito a una fase di sperimentazione preliminare, al fine di bilanciare la necessità di un addestramento accurato con la limitazione delle risorse computazionali.

La divisione del dataset è stata effettuata in modo casuale, mantenendo il 20% dei dati per la valutazione delle prestazioni del modello.

(Il parametro random_state=42 è stato fissato per garantire la riproducibilità degli esperimenti).

Ottimizzatore:

- La scelta dell'ottimizzatore è ricaduta su *Adam*, un ottimizzatore ampiamente usato in questi contesti di apprendimento supervisionato.

Per quanto riguarda l'ambiente di sviluppo è stato scelto [Python](#) (3.12.5) e la scelta per libreria (principale) usata per sviluppare il modello si è ricaduta su [TensorFlow](#) (2.17.0 per linux con supporto GPU) , libreria open-source, sviluppata da Google, è ampiamente utilizzata nella comunità scientifica per la ricerca e lo sviluppo di modelli di machine learning. La sua flessibilità, semplicità e la vasta gamma di strumenti offerti l'hanno resa la scelta ideale per questo progetto.

La potenza computazionale è stata “offerta” dalla scheda video Nvidia GeForce

RTX 3060; si è rilevata estremamente efficiente per eseguire calcoli numerici intensivi, come quelli richiesti dall'addestramento delle reti neurali.

La RTX 3060, grazie alla sua architettura e alla sua memoria dedicata, ha accelerato significativamente i tempi di training del modello (rispetto all'uso del processore).

Di seguito viene riportato in formato schematico lo sviluppo e l'evoluzione del progetto (ovvero i diversi Training checkpoints):

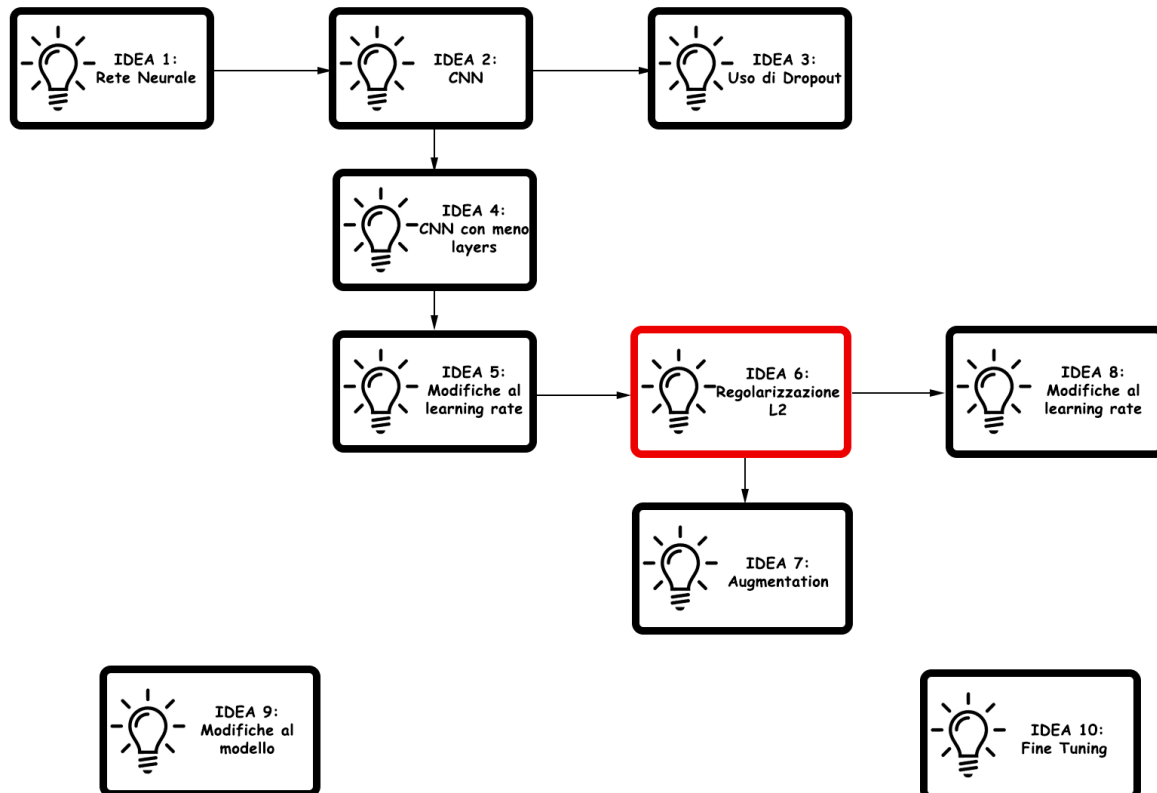


Immagine che rappresenta lo sviluppo del progetto, evidenziato in rosso la soluzione "ottimale" tra tutte quelle sviluppate.

Idea 1: Rete Neurale

La prima idea progettuale è stata quella di una creazione di un modello basato su una rete neurale “semplice”.

```
model = Sequential([
    Input((IMG_DIM, IMG_DIM, 1)),
    Dense(640, activation='relu'),
    Dropout(0.5),
    Dense(320, activation='relu'),
    Dense(160, activation='relu'),
    Dropout(0.5),
    Dense(64, activation='relu'),
    Dense(1)
])
```

1. Layer di Input:
 - Il modello inizia con un layer di input che accetta dunque immagini di dimensioni $\text{IMG_DIM} \times \text{IMG_DIM}$ formate solo da un canale, ovvero quello in scala di grigi.
2. Strati Dense:
 - Tutti i layer di tipo Dense sono neuroni con una funzione di attivazione ReLu, che permettono al modello di apprendere rappresentazioni complesse
3. Layer Dropout:
 - Tra il primo e il penultimo layer sono stati aggiunti layer di Dropout con un tasso del 50%, usato principalmente per prevenire overfitting durante la fase di training.
4. Layer di Output
 - Ultimo layer di tipo Dense, ha un solo neurone, progettato per compito di regressione e quindi di stimare il BMI finale a partire dalla foto presa in input

Per quanto riguarda le prestazioni, questo modello non ha performato tanto bene (bensì si tratti di un primo approccio), infatti, da come sui può ben vedere dalle immagini sottostanti, il modello oltre ad aver performato male è andato decisamente in overfitting, rispetto al training set e quindi durante la fase di validazione ha performato sempre peggio, in ogni epoca.

```

1418/1418 — 237s 162ms/step - loss: 93.3961 - mae: 6.7260 - val_loss: 46.0814 - val_mae: 5.0365
Epoch 2/10
1418/1418 — 217s 153ms/step - loss: 33.0318 - mae: 4.4238 - val_loss: 86.1939 - val_mae: 7.7404
Epoch 3/10
1418/1418 — 217s 153ms/step - loss: 29.2790 - mae: 4.1499 - val_loss: 179.3145 - val_mae: 12.3389
Epoch 4/10
1418/1418 — 217s 153ms/step - loss: 27.9303 - mae: 4.0427 - val_loss: 192.5587 - val_mae: 12.8643
Epoch 5/10
1418/1418 — 217s 153ms/step - loss: 26.9839 - mae: 4.0000 - val_loss: 182.2330 - val_mae: 12.4566
Epoch 6/10
1418/1418 — 217s 153ms/step - loss: 27.0035 - mae: 3.9889 - val_loss: 186.8740 - val_mae: 12.6415
Epoch 7/10
1418/1418 — 217s 153ms/step - loss: 26.9039 - mae: 3.9779 - val_loss: 184.5211 - val_mae: 12.5481
Epoch 8/10
1418/1418 — 215s 152ms/step - loss: 26.8673 - mae: 3.9901 - val_loss: 177.1385 - val_mae: 12.2505
Epoch 9/10
1418/1418 — 215s 151ms/step - loss: 26.5787 - mae: 3.9644 - val_loss: 162.0965 - val_mae: 11.6206
Epoch 10/10
1418/1418 — 217s 153ms/step - loss: 26.6063 - mae: 3.9639 - val_loss: 162.2442 - val_mae: 11.6269
355/355 — 14s 41ms/step - loss: 160.0669 - mae: 11.5577
MSE | MAE: [162.24420166015625, 11.626935005187988]

```

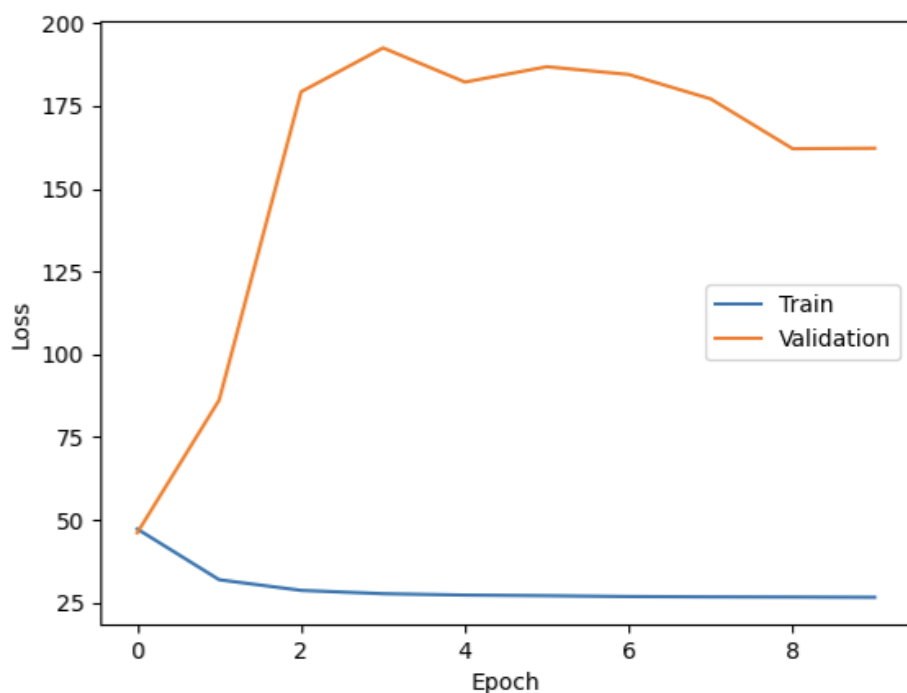


Grafico che rappresenta la metrica MSE a variare delle epoche

Idea 2: Rete Neurale Convolutionale (CNN)

Proseguendo con una seconda idea si è deciso di sviluppare una rete neurale convuluzionale al fine di cercare di migliorare le prestazioni.

```
model = Sequential([
    Input((IMG_DIM, IMG_DIM, 1)),
    Conv2D(256, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(1)
])
```

1. Layer di Input:

- Il modello inizia con un layer di input che accetta dunque immagini di dimensioni $IMG_DIM \times IMG_DIM$ formate solo da un canale, ovvero quello in scala di grigi.

2. Layer Conv2D:

- I layer Conv2D si occupano della convoluzione bidimensionale all'input, dove: il primo parametro rappresenta il numero di filtri convoluzionali; il secondo parametro (formato da una coppia di numeri), rappresenta la dimensione dei filtri convoluzionali; e il terzo e ultimo parametro rappresenta la funzione di attivazione ReLU.

3. Layer MaxPooling2D:

- I layer MaxPooling2D, si occupano di ridurre la dimensione dell'immagine tramite pooling massimo, dove la dimensione della finestra di pooling è per l'appunto (2, 2).

4. Funzione Flatten:

- Converte l'output bidimensionale in un vettore unidimensionale, per i layer successivi.

5. Strati Dense:

- Tutti i layer di tipo Dense sono neuroni con una funzione di attivazione ReLu, che permettono al modello di apprendere rappresentazioni complesse

6. Layer di Output

- Ultimo layer di tipo Dense, ha un solo neurone, progettato per compito di regressione e quindi di stimare il BMI finale a partire dalla foto presa in input

Per quanto riguarda le prestazioni, questo modello ha performato molto bene rispetto al primo approccio, infatti, da come si può ben vedere dalle immagini sottostanti, il modello risulta avere un MSE e un MAE inferiore.

Però in linea generale, non è ancora una soluzione da ritenere ottimale; infatti, dal grafico si può notare che il modello è andato decisamente in overfitting, rispetto al training set, e quindi durante la fase di validazione ha performato sempre peggio, in ogni epoca.

```
1418/1418 — 82s 50ms/step - loss: 43.7657 - mae: 4.5893 - val_loss: 27.8525 - val_mae: 4.3262
Epoch 2/10
1418/1418 — 64s 45ms/step - loss: 22.7385 - mae: 3.6481 - val_loss: 21.3759 - val_mae: 3.5214
Epoch 3/10
1418/1418 — 65s 46ms/step - loss: 21.3329 - mae: 3.5201 - val_loss: 23.2583 - val_mae: 3.5605
Epoch 4/10
1418/1418 — 64s 45ms/step - loss: 20.7487 - mae: 3.4927 - val_loss: 21.9466 - val_mae: 3.6099
Epoch 5/10
1418/1418 — 64s 45ms/step - loss: 19.6151 - mae: 3.3931 - val_loss: 21.1359 - val_mae: 3.4811
Epoch 6/10
1418/1418 — 64s 45ms/step - loss: 18.7119 - mae: 3.3114 - val_loss: 21.8861 - val_mae: 3.5304
Epoch 7/10
1418/1418 — 64s 45ms/step - loss: 17.2517 - mae: 3.1942 - val_loss: 23.9035 - val_mae: 3.6494
Epoch 8/10
1418/1418 — 64s 45ms/step - loss: 14.5602 - mae: 2.9541 - val_loss: 24.1517 - val_mae: 3.6833
Epoch 9/10
1418/1418 — 64s 45ms/step - loss: 12.0179 - mae: 2.6853 - val_loss: 24.2345 - val_mae: 3.7198
Epoch 10/10
1418/1418 — 64s 45ms/step - loss: 10.0371 - mae: 2.4637 - val_loss: 25.7714 - val_mae: 3.8897
355/355 — 5s 13ms/step - loss: 25.2639 - mae: 3.8587
MSE | MAE: [25.7713565826416, 3.889650344848633]
```

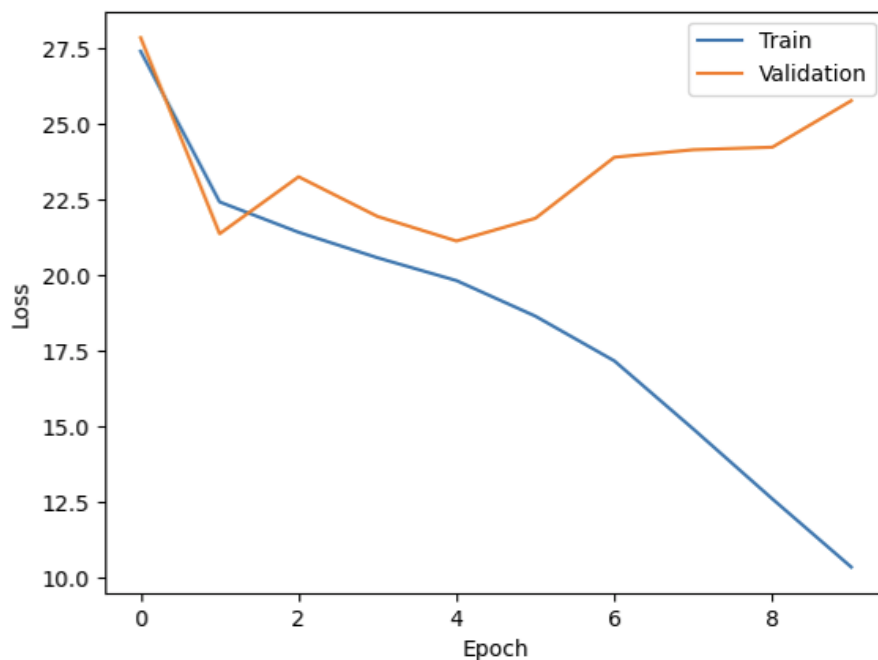


Grafico che rappresenta la metrica MSE a variare delle epoche

Idea 3: Uso di Dropout

Si è cercato di migliorare ulteriormente la soluzione Idea 2, facendo uso dei layer dropout, utili in caso di overfitting del modello.

Questa soluzione sarà fallimentare, da come si potrà pure evincere dai grafici.

```
model = Sequential([
    Input((IMG_DIM, IMG_DIM, 1)),
    Conv2D(256, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Dropout(0.3),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Dropout(0.3),
    Flatten(),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(32, activation='relu'),
    Dense(1)
])
```

1. Layer Dropout:

- Sono stati aggiunti layer di Dropout con un tasso del 30% e 50%, usati principalmente per prevenire overfitting durante la fase di training.

Per quanto riguarda le prestazioni da come si può notare dal grafico, il modello (può sembrare strano) ancora in overfitting, anzi è peggiorato rispetto alla soluzione precedente, per questo motivo questa soluzione è stata abbandonata. **[FALLIMENTARE]**

```
1418/1418 81s 52ms/step - loss: 51.0827 - mae: 5.3408 - val_loss: 301.6556 - val_mae: 16.6451
Epoch 2/10
1418/1418 66s 47ms/step - loss: 27.9455 - mae: 4.0278 - val_loss: 279.8839 - val_mae: 15.9946
Epoch 3/10
1418/1418 66s 47ms/step - loss: 25.1210 - mae: 3.8197 - val_loss: 265.7859 - val_mae: 15.5581
Epoch 4/10
1418/1418 66s 47ms/step - loss: 24.0575 - mae: 3.7376 - val_loss: 242.8065 - val_mae: 14.8161
Epoch 5/10
1418/1418 66s 47ms/step - loss: 23.3937 - mae: 3.6865 - val_loss: 239.6324 - val_mae: 14.7060
Epoch 6/10
1418/1418 66s 47ms/step - loss: 22.5572 - mae: 3.6128 - val_loss: 230.3995 - val_mae: 14.3899
Epoch 7/10
1418/1418 66s 47ms/step - loss: 21.7148 - mae: 3.5566 - val_loss: 218.7142 - val_mae: 13.9955
Epoch 8/10
1418/1418 66s 47ms/step - loss: 21.6472 - mae: 3.5574 - val_loss: 209.6991 - val_mae: 13.6798
Epoch 9/10
1418/1418 66s 47ms/step - loss: 20.7765 - mae: 3.4886 - val_loss: 217.4335 - val_mae: 13.9523
Epoch 10/10
1418/1418 67s 48ms/step - loss: 20.7119 - mae: 3.4633 - val_loss: 201.1243 - val_mae: 13.3607
355/355 5s 13ms/step - loss: 198.9837 - mae: 13.2965
MSE | MAE: 201.12432861328125, 13.360734939575195]
```

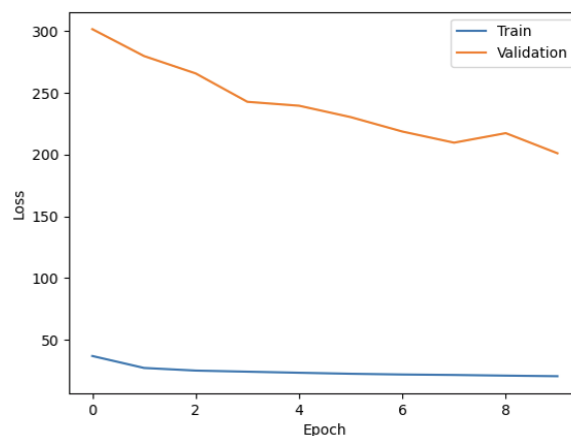


Grafico che rappresenta la metrica MSE a variare delle epoche

Idea 4: CNN con meno layers

Partendo dunque nuovamente dalla *idea 2*, si è pensato di “snellire” la rete rendendola meno complicata, così cercando di migliorare le sue prestazioni.

```
model = Sequential([
    Input((IMG_DIM, IMG_DIM, 1)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(32, activation='relu'),
    Dense(16, activation='relu'),
    Dense(1)
])
```

Come si può notare ora ci sono molti meno neuroni all'interno della rete, il che la rende meno complessa e anche più veloce nell'eseguire operazioni.

Questa modifica ha apportato miglioramenti rispetto alla *idea 2*, infatti come si può vedere dai grafici il modello è leggermente migliorato e il suo MSE (e anche il suo MAE) è diminuito.

```
1418/1418 — 39s 24ms/step - loss: 36.1279 - mae: 4.2636 - val_loss: 22.8408 - val_mae: 3.5371
Epoch 2/10
1418/1418 — 31s 22ms/step - loss: 22.6002 - mae: 3.6270 - val_loss: 22.5614 - val_mae: 3.7170
Epoch 3/10
1418/1418 — 31s 22ms/step - loss: 21.0922 - mae: 3.5091 - val_loss: 21.9458 - val_mae: 3.4776
Epoch 4/10
1418/1418 — 31s 22ms/step - loss: 20.7412 - mae: 3.4737 - val_loss: 21.6011 - val_mae: 3.5911
Epoch 5/10
1418/1418 — 31s 22ms/step - loss: 19.5751 - mae: 3.3880 - val_loss: 23.3731 - val_mae: 3.7984
Epoch 6/10
1418/1418 — 31s 22ms/step - loss: 18.7388 - mae: 3.3260 - val_loss: 21.6147 - val_mae: 3.5098
Epoch 7/10
1418/1418 — 31s 22ms/step - loss: 17.8876 - mae: 3.2597 - val_loss: 22.6916 - val_mae: 3.6339
Epoch 8/10
1418/1418 — 31s 22ms/step - loss: 16.1477 - mae: 3.1001 - val_loss: 23.3363 - val_mae: 3.6212
Epoch 9/10
1418/1418 — 31s 22ms/step - loss: 14.1756 - mae: 2.9063 - val_loss: 23.9160 - val_mae: 3.7335
Epoch 10/10
1418/1418 — 31s 22ms/step - loss: 12.5424 - mae: 2.7483 - val_loss: 24.7416 - val_mae: 3.8398
355/355 — 2s 6ms/step - loss: 24.6567 - mae: 3.8453
MSE | MAE: [24.741558074951172, 3.8397958278656006]
```

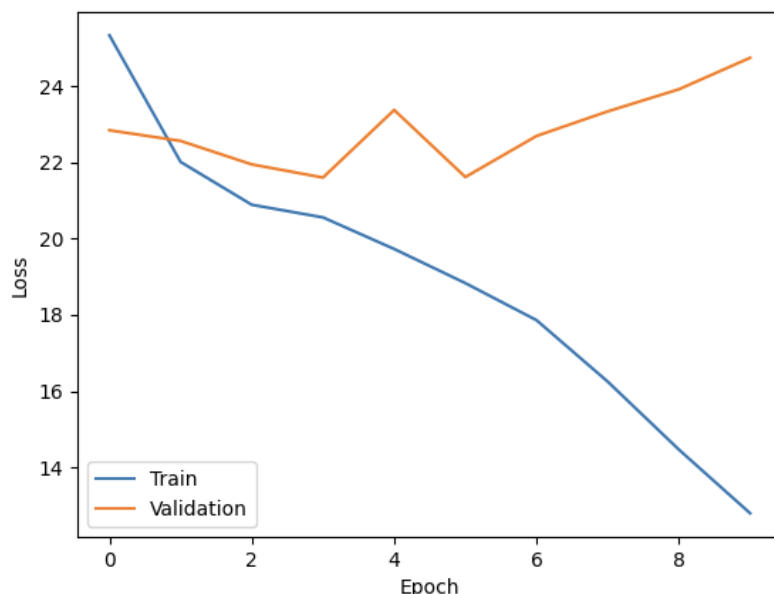


Grafico che rappresenta la metrica MSE a variare delle epoche

Idea 5: Modifiche al learning rate

Il passo successivo è stato quello di cercare di vedere se modificando il learning rate del modello, le sue performance migliorassero.

Per questo motivo il learning rate dal valore di *default* è stato impostato su 0.0001.

```
from tensorflow.keras.optimizers import Adam

optimizer = Adam(learning_rate=0.0001)
model.compile(optimizer=optimizer, loss='mean_squared_error', metrics=['mae'])
```

Per quanto riguarda le prestazioni, questa modifica ha apportato delle migliorie rispetto alla soluzione precedente; infatti, da come si può notare dai grafici, sia durante la fase di training che durante la fase di test, il modello performa più o meno nella stessa maniera ed inoltre le sue prestazioni sono nettamente migliorate.

```
1418/1418 — 39s 25ms/step - loss: 91.8737 - mae: 6.1296 - val_loss: 23.2590 - val_mae: 3.6714
Epoch 2/10
1418/1418 — 31s 22ms/step - loss: 22.8591 - mae: 3.6657 - val_loss: 22.9292 - val_mae: 3.7172
Epoch 3/10
1418/1418 — 31s 22ms/step - loss: 22.1463 - mae: 3.5981 - val_loss: 22.1251 - val_mae: 3.5335
Epoch 4/10
1418/1418 — 31s 22ms/step - loss: 21.3972 - mae: 3.5324 - val_loss: 21.7352 - val_mae: 3.5522
Epoch 5/10
1418/1418 — 31s 22ms/step - loss: 21.1852 - mae: 3.5194 - val_loss: 21.8411 - val_mae: 3.6147
Epoch 6/10
1418/1418 — 31s 22ms/step - loss: 20.8832 - mae: 3.4976 - val_loss: 22.3694 - val_mae: 3.5038
Epoch 7/10
1418/1418 — 31s 22ms/step - loss: 21.1592 - mae: 3.5043 - val_loss: 21.3946 - val_mae: 3.5093
Epoch 8/10
1418/1418 — 31s 22ms/step - loss: 20.5551 - mae: 3.4610 - val_loss: 22.5345 - val_mae: 3.5213
Epoch 9/10
1418/1418 — 31s 22ms/step - loss: 20.9387 - mae: 3.4981 - val_loss: 21.4354 - val_mae: 3.5632
Epoch 10/10
1418/1418 — 31s 22ms/step - loss: 21.0628 - mae: 3.4887 - val_loss: 21.3482 - val_mae: 3.5424
355/355 — 2s 6ms/step - loss: 21.0509 - mae: 3.5171
mse | MAE: [21.348169326782227, 3.5423858165740967]
```

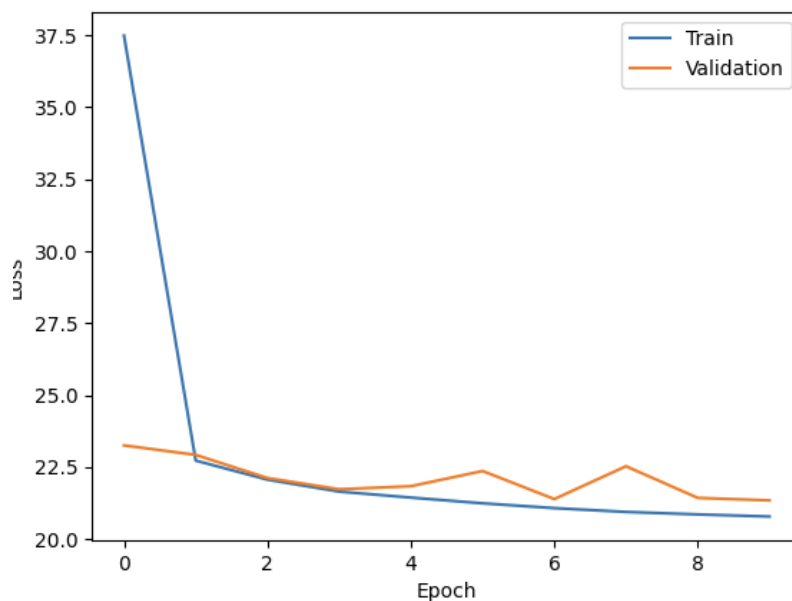


Grafico che rappresenta la metrica MSE a variare delle epoche

Idea 6: Regularizzazione L2

Per cercare di migliorare ulteriormente la soluzione descritta nell'*idea 5* è stato pensato di usare regolarizzatori L2 negli ultimi due layer della rete.

```
model = Sequential([
    Input((IMG_DIM, IMG_DIM, 1)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(32, activation='relu', kernel_regularizer=l2(0.01)),
    Dense(16, activation='relu', kernel_regularizer=l2(0.01)),
    Dense(1)
])
```

La regolarizzazione L2 aggiunge una penalità alla funzione di costo della rete, basata sulla somma dei quadrati dei pesi dei neuroni. In questo caso, la penalità è proporzionale a 0.01. Un valore più alto della penalità L2 incoraggia la rete a “imparare” pesi più piccoli, riducendo il rischio di overfitting.

Per quanto riguarda le prestazioni, come si può vedere dai grafici, questa modifica apporta ulteriori miglioramenti alla rete (e come vedremo pure inseguito si riterrà la soluzione ottimale del lavoro).

```
1418/1418 — 45s 25ms/step - loss: 111.8380 - mae: 6.7098 - val_loss: 24.7050 - val_mae: 3.6548
Epoch 2/10
1418/1418 — 31s 22ms/step - loss: 23.5484 - mae: 3.6809 - val_loss: 22.8413 - val_mae: 3.6181
Epoch 3/10
1418/1418 — 31s 22ms/step - loss: 22.4401 - mae: 3.5896 - val_loss: 22.4334 - val_mae: 3.6133
Epoch 4/10
1418/1418 — 41s 22ms/step - loss: 22.4020 - mae: 3.5848 - val_loss: 22.9656 - val_mae: 3.7257
Epoch 5/10
1418/1418 — 31s 22ms/step - loss: 21.6132 - mae: 3.5233 - val_loss: 23.6614 - val_mae: 3.8270
Epoch 6/10
1418/1418 — 31s 22ms/step - loss: 21.9268 - mae: 3.5491 - val_loss: 22.3993 - val_mae: 3.4924
Epoch 7/10
1418/1418 — 31s 22ms/step - loss: 21.4213 - mae: 3.4974 - val_loss: 22.4604 - val_mae: 3.6758
Epoch 8/10
1418/1418 — 31s 22ms/step - loss: 21.3067 - mae: 3.4919 - val_loss: 21.9545 - val_mae: 3.5951
Epoch 9/10
1418/1418 — 31s 22ms/step - loss: 21.2865 - mae: 3.4910 - val_loss: 21.8383 - val_mae: 3.5830
Epoch 10/10
1418/1418 — 31s 22ms/step - loss: 21.0758 - mae: 3.4746 - val_loss: 21.4970 - val_mae: 3.5008
355/355 — 2s 6ms/step - loss: 21.1868 - mae: 3.4762
MSE | MAE: [21.496950149536133, 3.500758647918701]
```

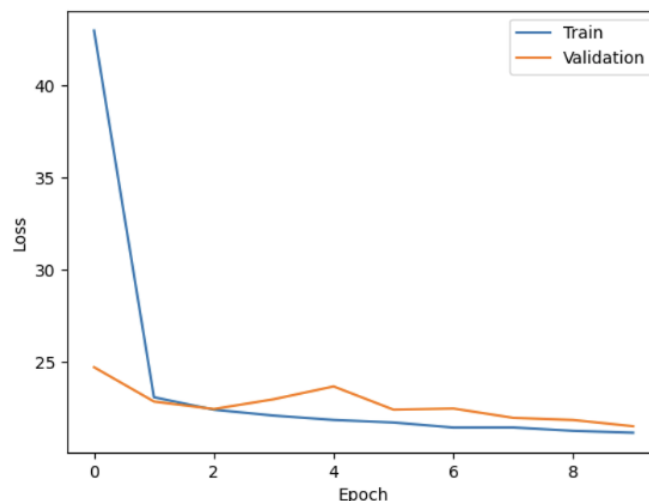


Grafico che rappresenta la metrica MSE a variare delle epoche

Idea 7: Augmentation

Per cercare di incrementare ancora di più le prestazioni della rete neurale si è pensato di usare la tecnica della augmentation; ovvero, di aggiungere oltre alle immagini di training altre immagini “nuove”, dove le immagini “nuove” erano immagini prese dal training set e modificate secondo alcuni parametri randomici.

```
train_datagen = ImageDataGenerator(  
    rotation_range=20,  
    width_shift_range=0.2,  
    height_shift_range=0.2,  
    zoom_range=0.2,  
    horizontal_flip=True  
)  
  
history = model.fit(  
    train_datagen.flow(X_train, y_train, batch_size=32),  
    epochs=10,  
    validation_data=(X_test, y_test)  
)
```

Questo però non ha prodotto ai risultati desiderati, anzi ha peggiorato le prestazioni della rete, da come si può ben notare dai grafici riportati.

[FALLIMENTARE]

Epoch	Time	loss	mae	val_loss	val_mae	
1418/1418	44s	28ms/step	loss: 61.6963	mae: 5.2889	val_loss: 29.8577	val_mae: 4.4169
Epoch 2/10						
1418/1418	37s	26ms/step	loss: 26.0458	mae: 3.8864	val_loss: 26.3497	val_mae: 4.0155
Epoch 3/10						
1418/1418	41s	26ms/step	loss: 25.6894	mae: 3.8652	val_loss: 26.0187	val_mae: 3.9617
Epoch 4/10						
1418/1418	37s	26ms/step	loss: 26.2520	mae: 3.9176	val_loss: 25.6401	val_mae: 3.9556
Epoch 5/10						
1418/1418	36s	26ms/step	loss: 25.9051	mae: 3.8747	val_loss: 25.1225	val_mae: 3.8714
Epoch 6/10						
1418/1418	37s	26ms/step	loss: 25.7892	mae: 3.8606	val_loss: 24.7516	val_mae: 3.8180
Epoch 7/10						
1418/1418	37s	26ms/step	loss: 25.7181	mae: 3.8683	val_loss: 24.3857	val_mae: 3.7795
Epoch 8/10						
1418/1418	37s	26ms/step	loss: 25.0005	mae: 3.8208	val_loss: 24.4650	val_mae: 3.8025
Epoch 9/10						
1418/1418	37s	26ms/step	loss: 25.1473	mae: 3.8149	val_loss: 24.5425	val_mae: 3.6900
Epoch 10/10						
1418/1418	37s	26ms/step	loss: 24.9625	mae: 3.8162	val_loss: 28.2963	val_mae: 4.3178
355/355	2s	6ms/step	loss: 28.1535	mae: 4.2978		
MSE MAE: [28.296337127685547, 4.31775990692139]						

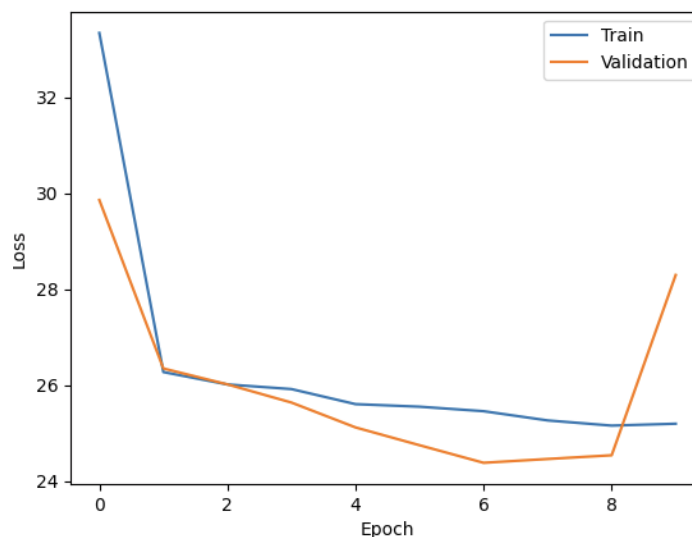


Grafico che rappresenta la metrica MSE a variare delle epoche

Idea 8: Modifiche al learning rate

Dati i risultati ottenuti con l'*idea 5*, si è pensato di modificare nuovamente il learning rate, però tenendo in considerazione l'*idea 6* come riferimento di modello.

Il learning rate, quindi, è stato impostato su *0.00001*, ancora più basso del precedente, cercando così di migliorare il modello.

```
model = Sequential([
    Input((IMG_DIM, IMG_DIM, 1)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(32, activation='relu', kernel_regularizer=l2(0.01)),
    Dense(16, activation='relu', kernel_regularizer=l2(0.01)),
    Dense(1)
])

from tensorflow.keras.optimizers import Adam

optimizer = Adam(learning_rate=0.00001)
model.compile(optimizer=optimizer, loss='mean_squared_error', metrics=['mae'])
```

Questa modifica ha portato alla creazione di un modello stabile, ma senza ulteriori miglioramenti, rispetto alla *idea 6*, come si può ben vedere dai grafici. Per questo motivo si è deciso di abbandonare l'*idea* e quindi cercare di non migliorarla ulteriormente.

```
1418/1418 ————— 40s 25ms/step - loss: 198.0517 - mae: 9.8353 - val_loss: 25.1377 - val_mae: 3.7473
Epoch 2/10
1418/1418 ————— 32s 22ms/step - loss: 24.3680 - mae: 3.7201 - val_loss: 24.2853 - val_mae: 3.6892
Epoch 3/10
1418/1418 ————— 31s 22ms/step - loss: 23.2829 - mae: 3.6449 - val_loss: 23.7957 - val_mae: 3.7205
Epoch 4/10
1418/1418 ————— 31s 22ms/step - loss: 23.2451 - mae: 3.6380 - val_loss: 23.5083 - val_mae: 3.6038
Epoch 5/10
1418/1418 ————— 31s 22ms/step - loss: 22.8117 - mae: 3.6000 - val_loss: 23.0882 - val_mae: 3.5835
Epoch 6/10
1418/1418 ————— 31s 22ms/step - loss: 22.7116 - mae: 3.5956 - val_loss: 22.7466 - val_mae: 3.6189
Epoch 7/10
1418/1418 ————— 31s 22ms/step - loss: 21.7631 - mae: 3.5385 - val_loss: 22.5120 - val_mae: 3.5884
Epoch 8/10
1418/1418 ————— 32s 22ms/step - loss: 21.3228 - mae: 3.4995 - val_loss: 22.3621 - val_mae: 3.5813
Epoch 9/10
1418/1418 ————— 32s 22ms/step - loss: 21.6782 - mae: 3.5205 - val_loss: 22.2340 - val_mae: 3.5587
Epoch 10/10
1418/1418 ————— 32s 22ms/step - loss: 21.3376 - mae: 3.4876 - val_loss: 22.2885 - val_mae: 3.5153
355/355 ————— 2s 6ms/step - loss: 21.9251 - mae: 3.4878
MSE | MAE: [22.288515090942383, 3.515286445617676]
```

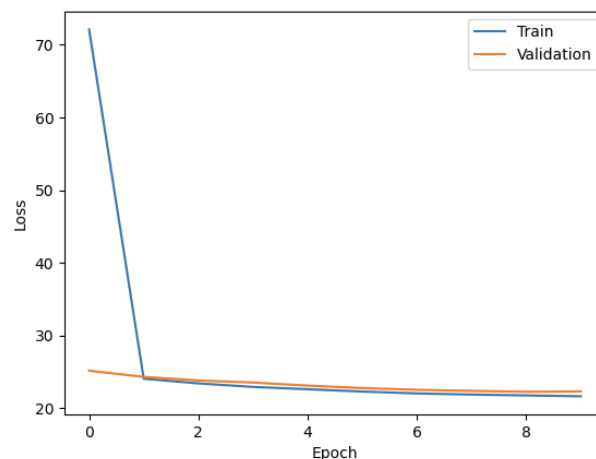


Grafico che rappresenta la metrica MSE a variare delle epoche

Idea 9: Modifiche al modello

A questo punto, per vedere se si potesse creare un altro modello migliore, si è optato per cambiare totalmente soluzione, cercando di crearne una nuova prendendo in considerazione anche tutti gli aspetti positivi degli esperimenti fatti precedentemente.

```
model = Sequential([
    Input((IMG_DIM, IMG_DIM, 1)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(32, activation='relu'),
    Dense(16, activation='relu'),
    Dense(1)
])

from tensorflow.keras.optimizers import Adam

optimizer = Adam(learning_rate=0.0001)
model.compile(optimizer=optimizer, loss='mean_squared_error', metrics=['mae'])
```

Quindi è stato creato un modello meno “denso” rispetto ai precedenti, mantenendo con un *learning rate* pari a 0.0001.

Questo modello, non ha portato a risultati desiderati, infatti come si vede dai grafici il modello non performa meglio rispetto ai precedenti, rimane sempre un “buon” modello (rispetto a quelli creati) ma non migliore.

```
1418/1418 — 25s 15ms/step - loss: 104.9275 - mae: 6.6064 - val_loss: 23.7244 - val_mae: 3.6856
Epoch 2/10
1418/1418 — 17s 12ms/step - loss: 22.9458 - mae: 3.6818 - val_loss: 24.3833 - val_mae: 3.6345
Epoch 3/10
1418/1418 — 17s 12ms/step - loss: 22.5997 - mae: 3.6176 - val_loss: 22.2073 - val_mae: 3.5913
Epoch 4/10
1418/1418 — 17s 12ms/step - loss: 21.9201 - mae: 3.5837 - val_loss: 22.5573 - val_mae: 3.5250
Epoch 5/10
1418/1418 — 17s 12ms/step - loss: 21.4717 - mae: 3.5514 - val_loss: 23.4000 - val_mae: 3.8408
Epoch 6/10
1418/1418 — 17s 12ms/step - loss: 21.0392 - mae: 3.4961 - val_loss: 21.4898 - val_mae: 3.5152
Epoch 7/10
1418/1418 — 17s 12ms/step - loss: 21.2153 - mae: 3.5134 - val_loss: 21.5711 - val_mae: 3.5842
Epoch 8/10
1418/1418 — 17s 12ms/step - loss: 21.1823 - mae: 3.4985 - val_loss: 21.2032 - val_mae: 3.4937
Epoch 9/10
1418/1418 — 17s 12ms/step - loss: 21.1121 - mae: 3.5034 - val_loss: 21.1372 - val_mae: 3.5041
Epoch 10/10
1418/1418 — 17s 12ms/step - loss: 20.6606 - mae: 3.4562 - val_loss: 21.8320 - val_mae: 3.6451
355/355 — 1s 3ms/step - loss: 21.5673 - mae: 3.6212
MSE | MAE: [21.831960678100586, 3.645148992538452]
```

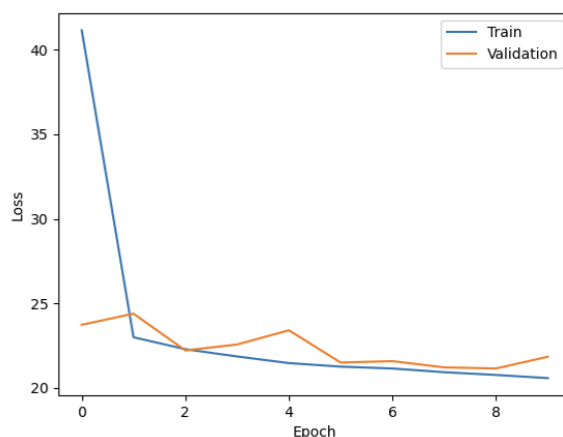


Grafico che rappresenta la metrica MSE a variare delle epoche

Idea 10: Fine Tuning

Ultima idea è stata quello di creare un altro modello, facendo fine tuning su un altro modello pre-addestrato nel riconoscimento di volti.

```
def get_model():
    base_model = tf.keras.applications.ResNet50(weights='imagenet', include_top=False, input_shape=(IMG_DIM, IMG_DIM, 3))

    for layer in base_model.layers:
        layer.trainable = False

    x = base_model.output
    x = tf.keras.layers.Flatten()(x)
    x = BMIHead()(x)

    model = tf.keras.models.Model(inputs=base_model.input, outputs=x)
    return model
```

Il modello preso come base è stato *ResNet50*, con i pesi di *imagenet*, escludendo il livello di classificazione finale presente nella rete neurale.

Questo modello rispetto alle soluzioni precedenti prende in input direttamente immagini in formato vettoriale in formato RGB, quindi con 3 canali.

Sono stati impostati tutti i suoi layer non addestrabili durante l'addestramento, , poiché si riteneva che i livelli iniziali della rete hanno già appreso caratteristiche generali per analisi di immagini .

E' quindi stato aggiunto e accodato il modello ideato, ovvero: *BMIHead*.

```
import tensorflow as tf

@tf.keras.utils.register_keras_serializable()
class BMIHead(tf.keras.layers.Layer):
    def __init__(self, trainable=True, **kwargs):
        super(BMIHead, self).__init__(**kwargs)
        self.trainable = trainable
        self.layer1 = tf.keras.layers.Dense(128, activation='relu')
        self.layer2 = tf.keras.layers.Dense(64, activation='relu')
        self.layer3 = tf.keras.layers.Dense(32, activation='relu')
        self.layer4 = tf.keras.layers.Dense(16, activation='relu')
        self.layer_out = tf.keras.layers.Dense(1)

    def call(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        out = self.layer_out(x)
        return out
```

Il modello creato è una rete neurale “semplice”, formata quindi da 5 layer, di cui uno di output che predice il BMI.

Per quanto riguarda le prestazioni del modello, quest'ultimo non si è rilevato migliore rispetto a soluzioni precedentemente discusse; infatti, da come si può ben vedere dai grafici il suo MAE e MSE sono più alti di soluzioni come, ad esempio, la *idea 6* [21.49, 3.50].

```

473/473 ————— 40s 61ms/step - loss: 103.4040 - mae: 6.7545 - val_loss: 25.5427 - val_mae: 3.8229
Epoch 2/10
473/473 ————— 13s 28ms/step - loss: 25.0803 - mae: 3.8548 - val_loss: 25.0957 - val_mae: 3.7568
Epoch 3/10
473/473 ————— 14s 29ms/step - loss: 24.9674 - mae: 3.8363 - val_loss: 25.9541 - val_mae: 3.7688
Epoch 4/10
473/473 ————— 14s 29ms/step - loss: 23.9118 - mae: 3.7540 - val_loss: 24.0728 - val_mae: 3.7311
Epoch 5/10
473/473 ————— 14s 29ms/step - loss: 24.4275 - mae: 3.7856 - val_loss: 24.8298 - val_mae: 3.9103
Epoch 6/10
473/473 ————— 14s 30ms/step - loss: 23.6977 - mae: 3.7542 - val_loss: 24.2180 - val_mae: 3.6796
Epoch 7/10
473/473 ————— 14s 30ms/step - loss: 24.6193 - mae: 3.7771 - val_loss: 25.1895 - val_mae: 3.9705
Epoch 8/10
473/473 ————— 14s 30ms/step - loss: 24.1122 - mae: 3.7565 - val_loss: 27.1842 - val_mae: 3.8281
Epoch 9/10
473/473 ————— 14s 29ms/step - loss: 23.6164 - mae: 3.7105 - val_loss: 23.5114 - val_mae: 3.7039
Epoch 10/10
473/473 ————— 14s 29ms/step - loss: 22.9652 - mae: 3.6681 - val_loss: 25.8942 - val_mae: 3.7521
119/119 ————— 3s 23ms/step - loss: 26.3712 - mae: 3.8146
MSE | MAE: [25.894237518310547, 3.752124786376953]

```

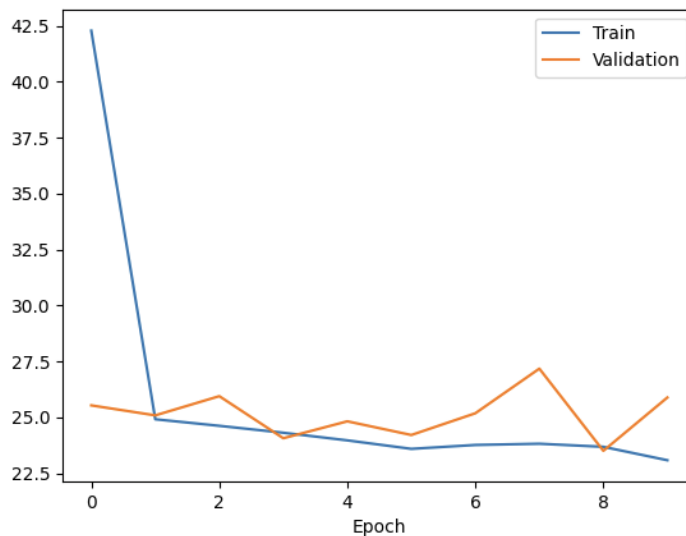


Grafico che rappresenta la metrica MSE a variare delle epoche

Per quanto riguarda l'addestramento però non è stato possibile addestrare la rete su tutti i dati poiché la memoria a disposizione del dispositivo usato per effettuare calcoli non aveva memoria a sufficienza, per questo motivo i dati totali (train + test) sono stati limitati.

```

num_images = int(len(os.listdir(image_dir)) / 3)
X = np.empty((num_images, IMG_DIM, IMG_DIM, 3))
y = np.empty(num_images)

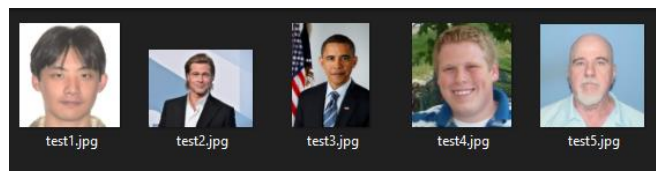
```

Poiché il modello è molto più pesante a livello computazionale, di memoria occupata sul disco (circa 130MB) e le sue prestazioni non sono migliorate rispetto alle precedenti è stato deciso di abbandonare l'idea di migliorarlo ulteriormente.

Conclusione e considerazioni finali

Dopo aver analizzato tutte le soluzioni prodotte e discusse, si è deciso di prendere come soluzione migliore la *idea 6*.

Per provare la sua funzionalità, si è provveduto a trovare e analizzare altre immagini sia trovate su internet, che prese direttamente dal dataset.



I risultati ottenuti dal modello *dell'idea 6* si sono rilevati abbastanza buoni; infatti, come si può vedere i BMI predetti dal modello si avvicinano al *Ground Truth* relativi alle immagini.

NOME FILE	Ground Truth	BMI predetto
test.jpg	27.1	27.069
test2.jpg	25.2	27.180805
test3.jpg	24.5	25.315533
test4.jpg	32.8	30.149044
test5.jpg	29.0	27.85595

È fondamentale sottolineare che l'obiettivo di questo progetto è puramente accademico e di ricerca.

L'utilizzo di un modello di questo tipo per scopi diagnostici o clinici richiederebbe ulteriori sviluppi, validazioni e regolamentazioni. Inoltre, è importante considerare le implicazioni etiche legate alla privacy e alla discriminazione nell'utilizzo di algoritmi di riconoscimento facciale.

[GitHub](#)