



# Relazione machine learning

Salvatore Mario Carota

March 2021

## Contents

<b>1 Task</b>	<b>4</b>
1.1 Room-based Localization . . . . .	4
<b>2 Dataset</b>	<b>5</b>
2.1 Organizzazione e Archiviazione . . . . .	5
2.2 Training set . . . . .	7
2.2.1 Training set usato nelle prime 4 prove . . . . .	7
2.2.2 Training set usato nella quinta prova . . . . .	8
2.3 Validation set . . . . .	9
2.4 Test set . . . . .	10
<b>3 Metodi</b>	<b>11</b>
3.1 Transfer Learning: Fine-tuning . . . . .	12
3.1.1 ResNet-18 . . . . .	12
3.2 Augmentation . . . . .	13
3.3 Training . . . . .	14
3.4 Bilanciamento classi . . . . .	15
<b>4 Valutazione</b>	<b>17</b>
4.1 Accuracy . . . . .	17
4.2 F1 score . . . . .	17
<b>5 Esperimenti</b>	<b>18</b>
5.1 Prima prova . . . . .	18
5.1.1 Prime 15 epoche . . . . .	18
5.1.2 Altre 15 epoche . . . . .	19
5.2 Seconda prova . . . . .	20
5.2.1 Prime 15 epoche . . . . .	20
5.2.2 Altre 15 epoche . . . . .	21
5.3 Terza prova . . . . .	22
5.3.1 Prime 15 epoche . . . . .	22
5.3.2 Altre 15 epoche . . . . .	23
5.3.3 Altre 10 epoche . . . . .	24
5.3.4 Altre 10 epoche . . . . .	25
5.3.5 Altre 15 epoche . . . . .	26

5.3.6	Valutazione terza prova . . . . .	27
5.4	Quarta prova . . . . .	28
5.4.1	Prime 15 epoche . . . . .	28
5.4.2	Altre 15 epoche . . . . .	29
5.4.3	Altre 15 epoche . . . . .	30
5.5	Quinta prova . . . . .	31
5.5.1	Prime 15 epoche . . . . .	31
5.5.2	Altre 15 epoche . . . . .	32
5.5.3	Altre 15 epoche . . . . .	33
5.5.4	Altre 10 epoche . . . . .	34
5.5.5	Valutazione quinta prova . . . . .	35
5.6	Tabella risultati . . . . .	35
<b>6</b>	<b>Demo</b>	<b>36</b>
<b>7</b>	<b>Codice</b>	<b>37</b>
7.1	Struttura codice . . . . .	37
<b>8</b>	<b>Conclusione</b>	<b>38</b>

# 1 Task

## 1.1 Room-based Localization

Il compito consiste nel determinare la stanza di un sito culturale in cui si trova il visitatore da immagini.



Sala 4



Cortile Parisio



Sala 11

Un sistema che riconosce stanze di luoghi specifici, attraverso l'acquisizione di immagini può avere svariate applicazioni e vantaggi, tra cui:

1. **Turismo e Guida:** Le applicazioni destinate ai turisti possono sfruttare questo sistema per offrire dettagli sul luogo fotografato. L'app potrebbe non solo identificare il luogo, ma anche fornire una descrizione dettagliata, elencare eventi in corso nelle vicinanze, suggerire luoghi di interesse simili e molto altro.
2. **Didattica e Cultura:** Può essere utilizzato in contesti educativi, per insegnare storia dell'arte o architettura. Gli studenti potrebbero fotografare e ricevere informazioni del luogo in tempo reale.
3. **Social Media e Condivisione:** Piattaforme social potrebbero utilizzare il riconoscimento di luoghi per suggerire tag automatici, creare album tematici o anche per suggerire eventi o post correlati al luogo.
4. **Archiviazione e Organizzazione:** Un sistema di riconoscimento visivo potrebbe facilitare la categorizzazione automatica delle fotografie in base ai luoghi rappresentati.
5. **Marketing e Pubblicità:** Aziende potrebbero sfruttare tale tecnologia per fornire annunci o promozioni mirate basate sui luoghi di interesse fotografati dagli utenti.
6. **Accessibilità:** Per le persone con disabilità visive, una combinazione di riconoscimento fotografico e sintesi vocale potrebbe fornire descrizioni dei luoghi circostanti.

## 2 Dataset

Il dataset è stato acquisito utilizzando un dispositivo Microsoft HoloLens montato sulla testa in un sito culturale situato in Sicilia (Siracusa), Italia “Palazzo Bellomo”

### EGO-CH: Palazzo Bellomo

- 22 ambienti
- Acquisizione video: 1280x720 a 29,97 fps
- 57 video di formazione
- Etichette temporali per indicare l’ambiente in cui si trova il visitatore

Per altre informazioni [link EGO-CH: Dataset](#)

### 2.1 Organizzazione e Archiviazione

Il dataset è stato diviso in training set, validation set, e test set. All’interno della cartella “Bellomo” è contenuta una cartella “Training” nella quale sono presenti i frames dei video divisi 57 cartelle una per ogni video acquisito.

Nomenclatura cartella:

```
1 "indice" + "_" + "nome sala" + "_" + "nome POI" + ".mp4" + ("_frames" ||  
" " || "_S")
```

- **indice**: è composto dalla coppia x.y dove  $x \in [1, 22]$  il quale indica la classe di appartenenza dei frame contenuti nella cartella e  $y \in [1, 3]$  che ne indica il POI. Le cartelle che terminano con “\_S” indicano che il video è stato acquisito di sera, non tutte le cartelle sono composte dalla coppia x.y.

$$x, y \in \mathbb{N}$$

Oltre alle 57 cartelle nella cartella “Training” sono presenti due file “training.txt” e “validation.txt”, i due file contengono liste di percorsi relativi al frame.

```
1 "/home/fragusa/Dataset/Bellomo/" + "nome sala" / "identificativo frame"  
+ ".jpg" + "identificativo classe"
```

- **identificativo frame**: è un valore numerico di 5 cifre.
- **Identificativo classe**: è un numero naturale  $n \in [1, 22]$

la lista “validation.txt” è stata divisa in due blocchi da  $m/2$  elementi dove il primo blocco è stato usato per il validation set e il secondo blocco per il test set.

- $m = \#\text{elementi contenuti in “validation.txt”}$

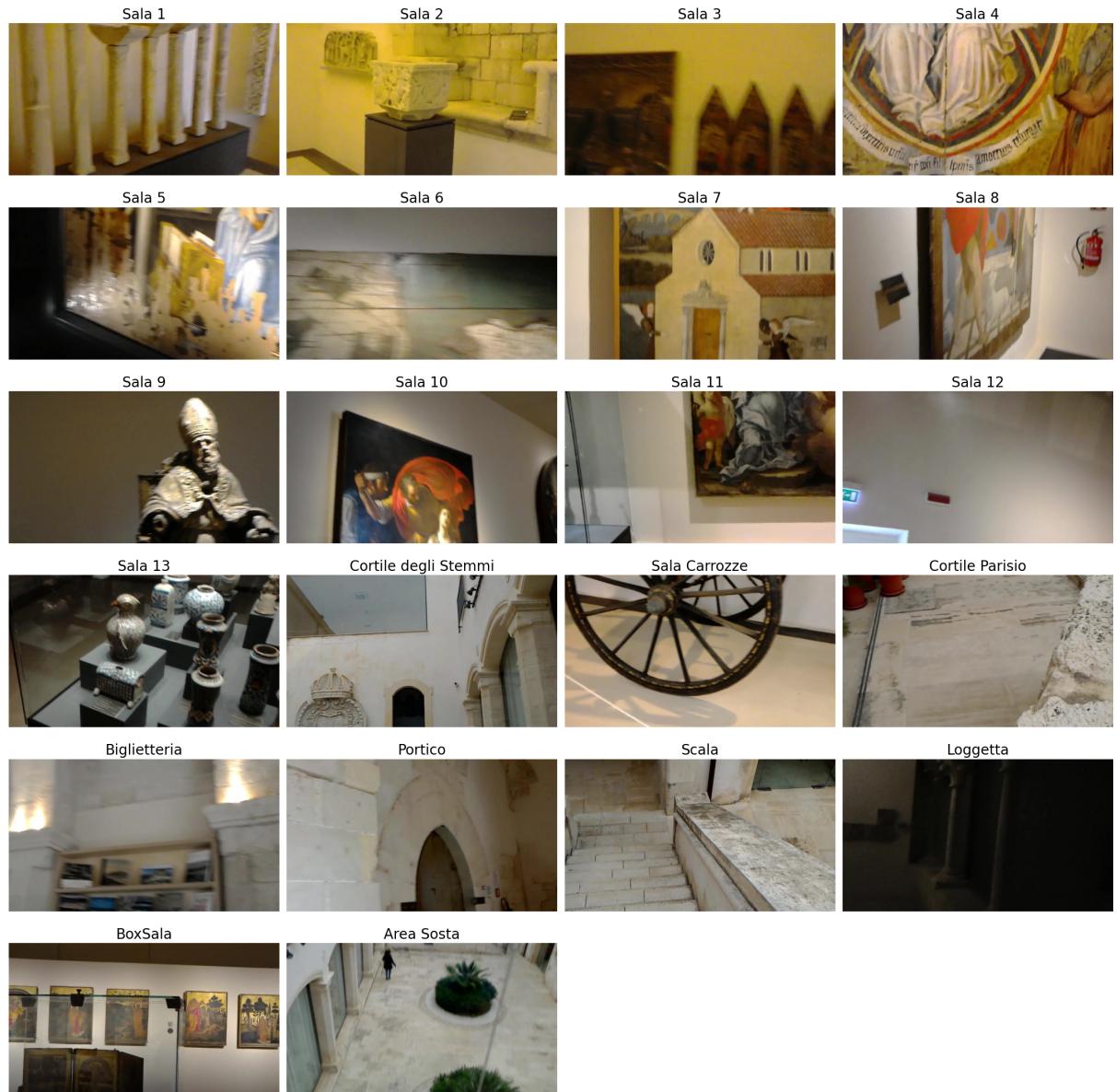
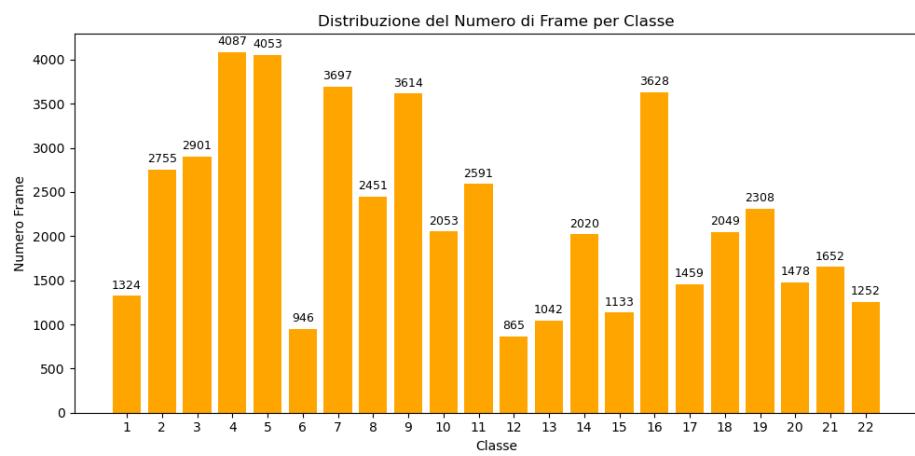
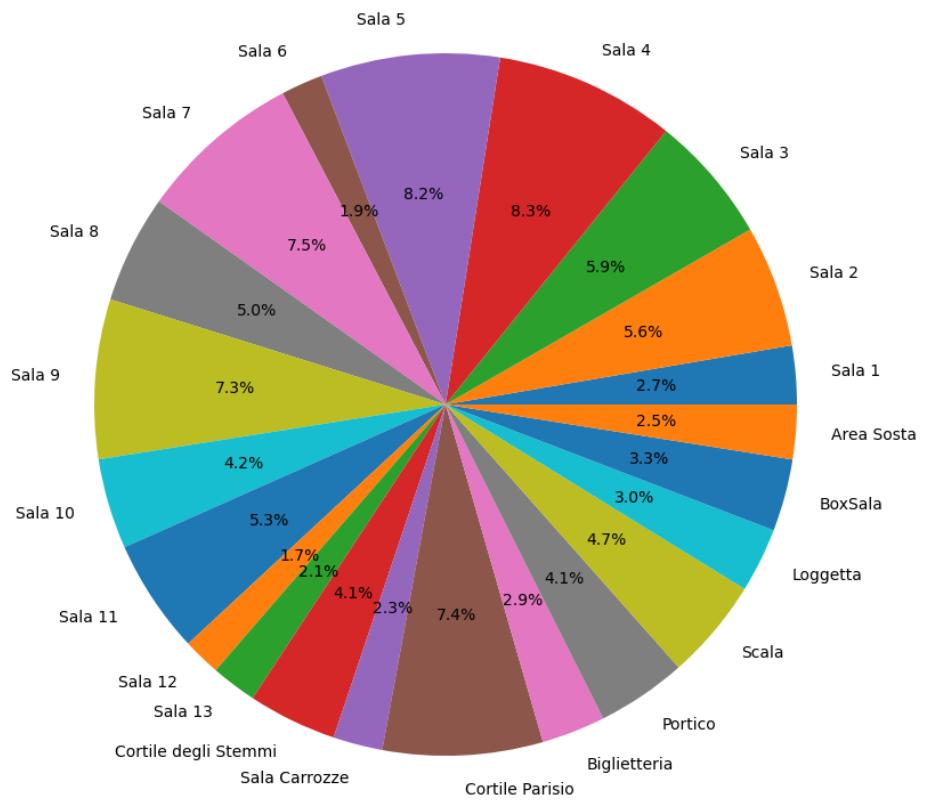


Figure 1: Campioni per ciascuna delle 22 classi

## 2.2 Training set

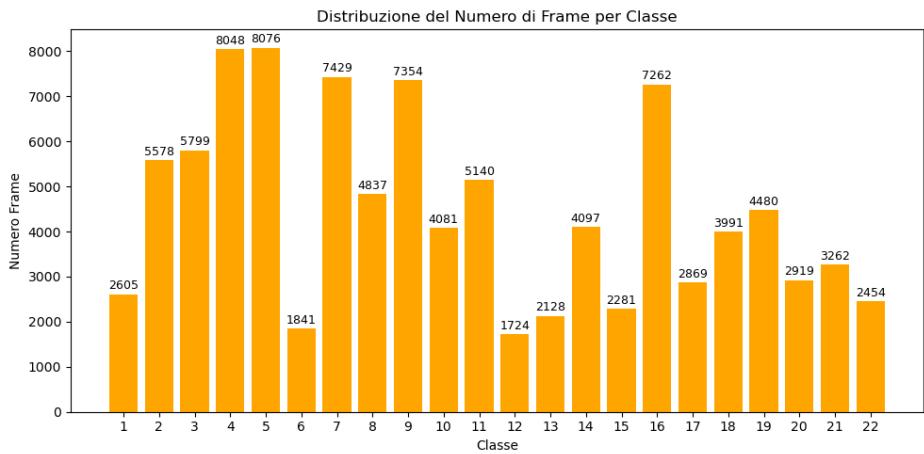
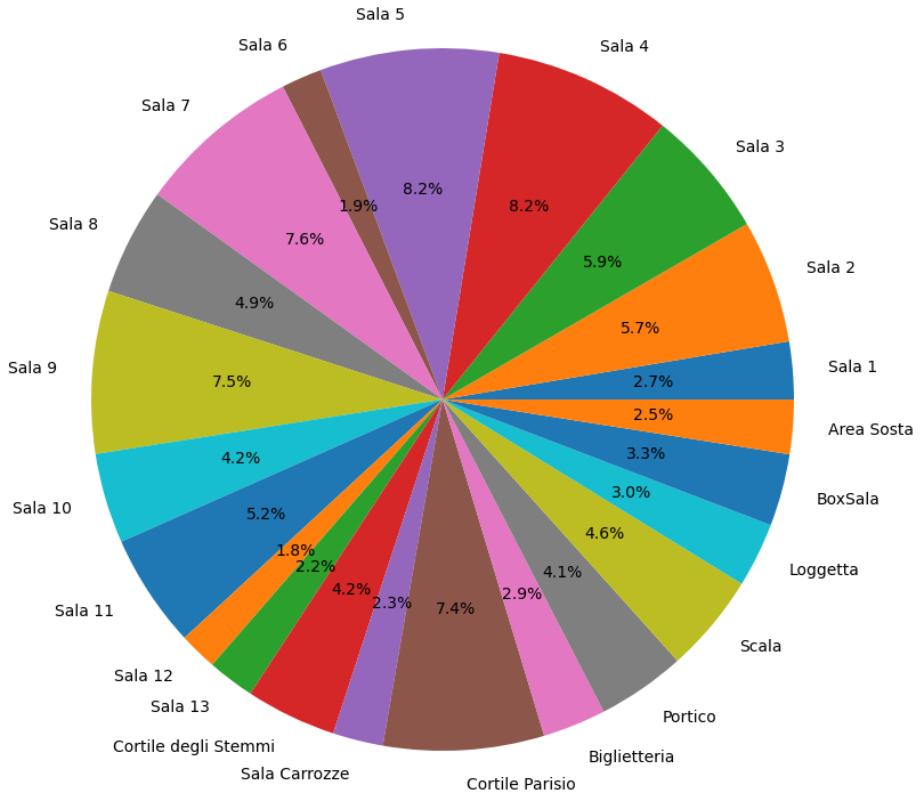
### 2.2.1 Training set usato nelle prime 4 prove

Distribuzione classi training set, 49127 frames



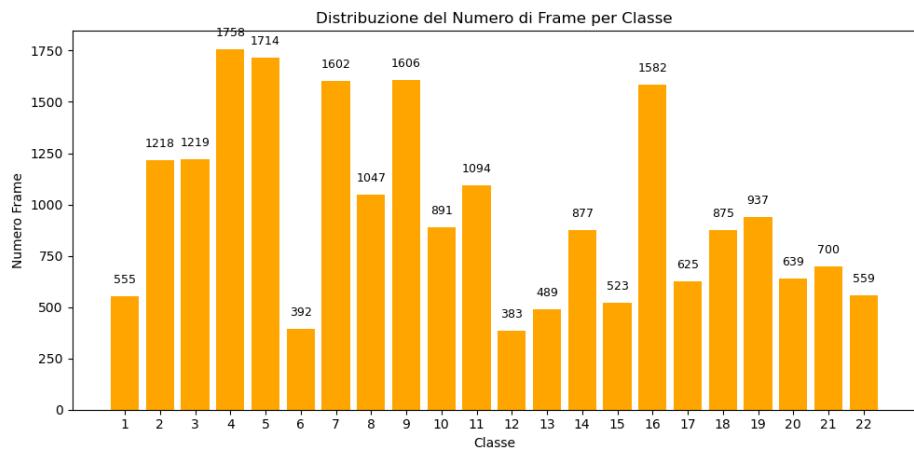
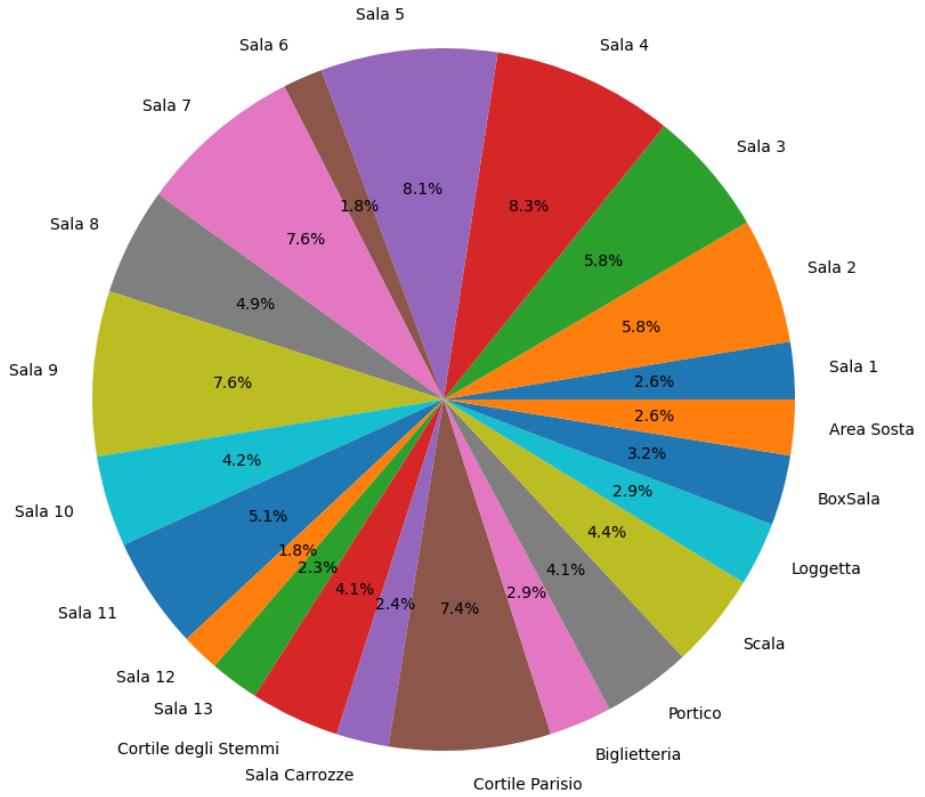
## 2.2.2 Training set usato nella quinta prova

Distribuzione classi training set, 98255 frames



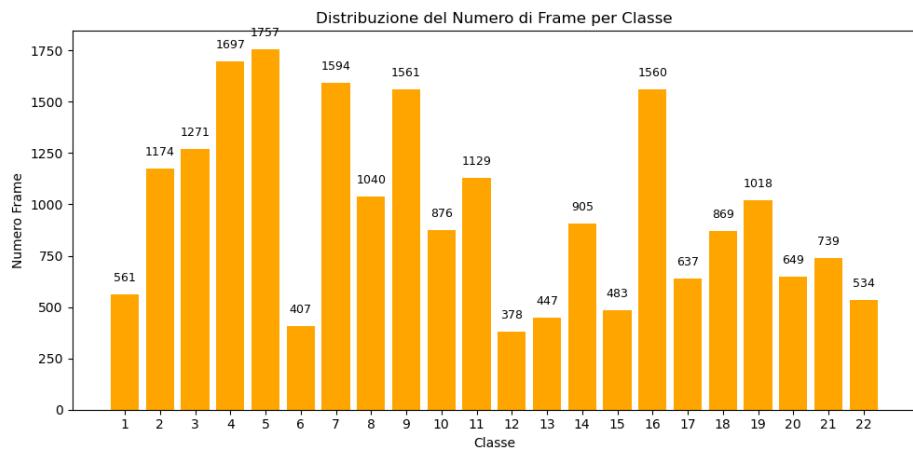
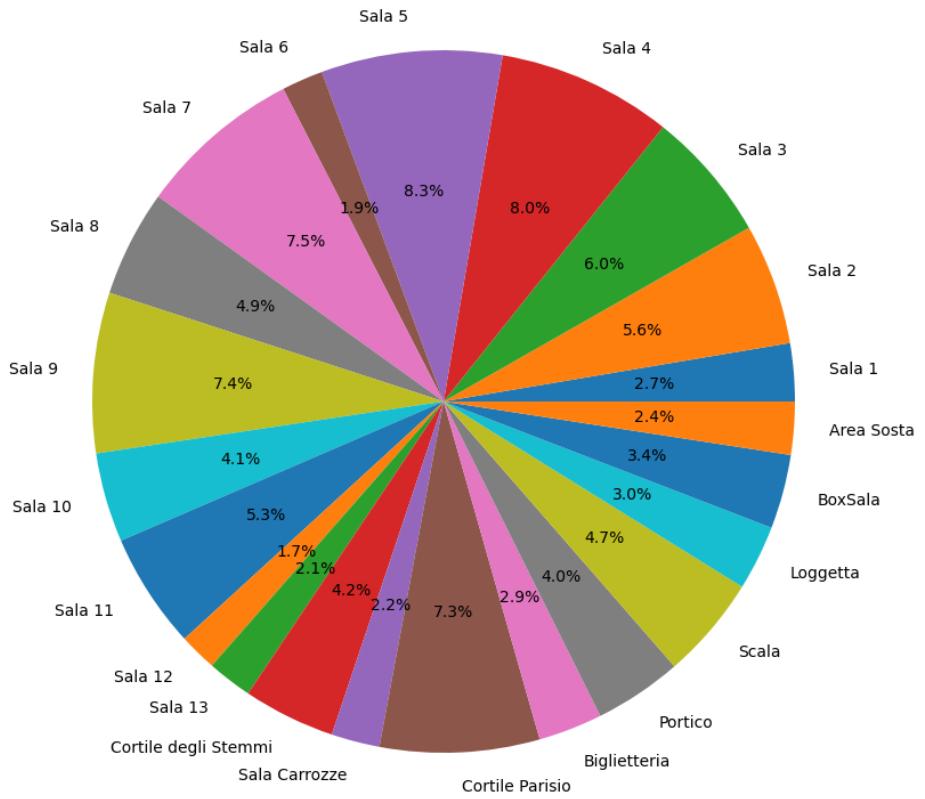
## 2.3 Validation set

Distribuzione classi validation set, 21054 frames



## 2.4 Test set

Distribuzione classi test set, 21055 frames



### 3 Metodi

Al fine di addestrare un modello destinato alla classificazione in un contesto con limitazioni hardware, sono state adottate le seguenti strategie:

1. **Utilizzo di un modello pre-allenato:** La scelta di utilizzare un modello pre-allenato sfrutta la potenza del "transfer learning". I modelli formati su vasti dataset, come ImageNet, hanno già assimilato caratteristiche visive generali che possono essere pertinenti a molti compiti di visione computerizzata. Pertanto, quando si adatta un modello pre-allenato a un nuovo compito con un numero limitato di classi, come la classificazione di 22 classi in questo caso, si può attingere alle caratteristiche apprese in precedenza. Questo solitamente significa che si necessita di meno dati per ottenere risultati accurati e che il tempo di addestramento è ridotto.
2. **Selezione di una versione semplificata del modello:** Modelli semplificati hanno il vantaggio di avere meno parametri, rendendo il processo di addestramento meno oneroso in termini di potenza computazionale e memoria.
3. **Sperimentazione con un training set ridotto:** L'uso di un subset dei dati per le fasi iniziali di sperimentazione permette di testare rapidamente diversi iperparametri. Una volta identificato un modello o una configurazione promettente, l'uso dell'intero dataset può aiutare a affinare ulteriormente le prestazioni e assicurare che tutti i possibili dettagli e variazioni nei dati siano considerati.
4. **Adozione di un elevato learning rate iniziale:** Partendo con un learning rate relativamente alto, il modello può rapidamente convergere verso una soluzione promettente, superando i minimi locali indesiderati. Riducendo il learning rate si possono effettuare aggiornamenti più fini man mano che ci avviciniamo alla convergenza.

### 3.1 Transfer Learning: Fine-tuning

La tecnica di fine-tuning consiste nel prendere un modello pre-addestrato e "continuare" l'addestramento di tutto o parte del modello sul nuovo dataset. In questo caso il livello **fc** della Resnet-18 è stato sostituito dal blocco di livelli (Listing 1) per adattare il modello al task della classificazione di 22 classi, e partendo dal modello pre-addestrato, sono stati addestrati tutti i livelli.

#### 3.1.1 ResNet-18

ResNet-18 è una delle versioni più leggere dell'architettura ResNet, dato che ha molti meno parametri rispetto alle versioni più profonde come ResNet-50, ResNet-101 o ResNet-152. Tuttavia, pur essendo più leggera, ResNet-18 ha il potenziale per fornire ottime risultati in numerose attività legate alla visione computazionale.

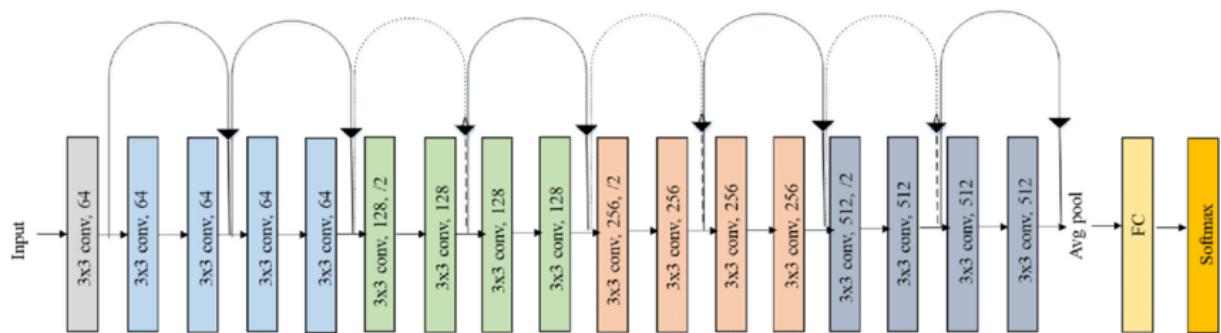


Figure 2: Architettura ResNet-18

l'ultimo strato relativo al classificatore è stato sostituito con i seguenti strati per la classificazione di 22 classi:

```
1 self.fc = nn.Sequential(  
2     nn.Linear(512, 256),  
3     nn.ReLU(),  
4     nn.Dropout(0.5),  
5     nn.Linear(256, 22)  
6 )
```

Listing 1: nuovo fc

## 3.2 Augmentation

L'augmentazione dei dati è una tecnica utilizzata per aumentare artificialmente la dimensione del set di dati tramite variazioni delle immagini esistenti. È particolarmente utile per prevenire l'overfitting, specialmente quando si dispone di un set di dati di dimensioni limitate o dati molto simili tra loro.

Operazioni usate :

- **Random Crop:** Effettua un ritaglio casuale dell'immagine di dimensione 224x224. Questo serve non solo come una forma di augmentazione (introducendo variazioni casuali nel set di addestramento) ma anche per portare l'immagine alla dimensione richiesta dal modello.
- **Random Horizontal Flip:** Con una probabilità del 50%, l'immagine viene ribaltata orizzontalmente. Questo introduce variazione e aiuta il modello a generalizzare meglio, specialmente se l'orientamento dell'oggetto nell'immagine non è fondamentale per la classificazione.

### 3.3 Training

Il modello viene addestrato usando un training loop e un validation loop, permettendo di monitorare le prestazioni sia sui dati di addestramento che di validazione.

- **Ottimizzazione con SGD:** L'ottimizzatore scelto è lo Stochastic Gradient Descent (SGD) con momentum. A differenza del Gradient Descent che calcola il gradiente utilizzando l'intero dataset, SGD stima il gradiente da ogni singolo esempio di addestramento e aggiorna i pesi del modello. Questo rende SGD molto più veloce e lo rende efficace per il training su dataset di grandi dimensioni. Il momentum aiuta a "smussare" gli aggiornamenti dei gradienti e ad accelerare la convergenza.
- **Prevenzione dell'Overfitting** L'uso della regolarizzazione L2 aiuta a prevenire l'overfitting.

$$J(\Theta) = Loss(X, \Theta) + \lambda \sum_i \Theta_i^2$$

- **Backpropagation:** La backpropagation sfrutta la regola della catena del calcolo differenziale per calcolare i gradienti delle funzioni di errore rispetto a tutti i parametri del modello. Ciò consente di aggiornare efficacemente tutti i parametri del modello in una sola passata attraverso la rete, a differenza di metodi naive che potrebbero richiedere un numero di passaggi proporzionale al numero di parametri. Accoppiata con tecniche di ottimizzazione, come la discesa del gradiente stocastico (SGD), la backpropagation può convergere rapidamente a soluzioni che minimizzano l'errore sul set di allenamento.
- **Dropout:** fornisce un modo per evitare che la rete impari singoli percorsi, forzandola a distribuire il peso delle informazioni su molteplici percorsi. Il dropout riduce la complessità effettiva del modello durante l'addestramento, aiudando a prevenire l'overfitting.
- **TensorBoard:** Il codice utilizza ‘SummaryWriter’ da ‘torch.utils.tensorboard’ per scrivere log, che possono poi essere visualizzati in TensorBoard. Questo è utile per monitorare l'addestramento in tempo reale e visualizzare grafici delle metriche come perdita e accuratezza.
- **Gestione Dispositivo (CPU/GPU):** Il codice è scritto per funzionare sia su CPU che su GPU (CUDA), garantendo flessibilità nell'addestramento. L'uso della GPU accelererà significativamente il processo di addestramento.
- **Salvataggio dei Modelli:** Dopo ogni epoca, il modello viene salvato su disco. Questo può essere utile per ripristinare il modello in un secondo momento o per fare inferenze utilizzando modelli da epoche specifiche.
- **Avvio Personalizzato dell'Epoca:** L'opzione 'start\_epochs' permette di iniziare l'addestramento da una determinata epoca, utile per riprendere l'addestramento.
- **Salvataggio Automatico:** Il salvataggio regolare dei pesi del modello garantisce che si possa sempre recuperare una versione addestrata, anche se l'addestramento viene interrotto.

### 3.4 Bilanciamento classi

- $N$  : *total\_samples* = Numero totale di campioni nel training set
- $C$  : *num\_classes* = Numero totale di classi
- $n_k$  : Numero di campioni nella classe  $k$

$$w_k = \frac{N}{C \cdot n_k}$$

```
1 class_weights = {k: total_samples / (num_classes * v) for k, v in
      indexlist.items()}
```

Listing 2: versione formula in codice python

- Se una classe ha esattamente la media dei campioni, allora il suo peso sarà  $w_k = 1$
- classi con meno campioni della media avranno un peso  $w_k > 1$
- classi con più campioni della media avranno un peso  $0 < w_k < 1$

```
1 sample_weights = [class_weights[train.getLabel(idx)] for idx in range(
      len(train))]
2 sampler = WeightedRandomSampler(weights=sample_weights, num_samples=len(
      train), replacement=True)
```

[link documentazione WeightedRandomSampler](#)

Utilizzando la funzione WeightedRandomSampler con i nuovi pesi, si otterrà una probabilità di selezione per ciascun campione  $i$  in classe  $k$  proporzionale a  $w_k$ , simulando così l'effetto combinato di oversampling e undersampling.

Questo approccio permette ai campioni delle classi meno rappresentate di avere una probabilità maggiore di essere selezionati, equilibrando così l'influenza di ciascuna classe sul modello in fase di training.

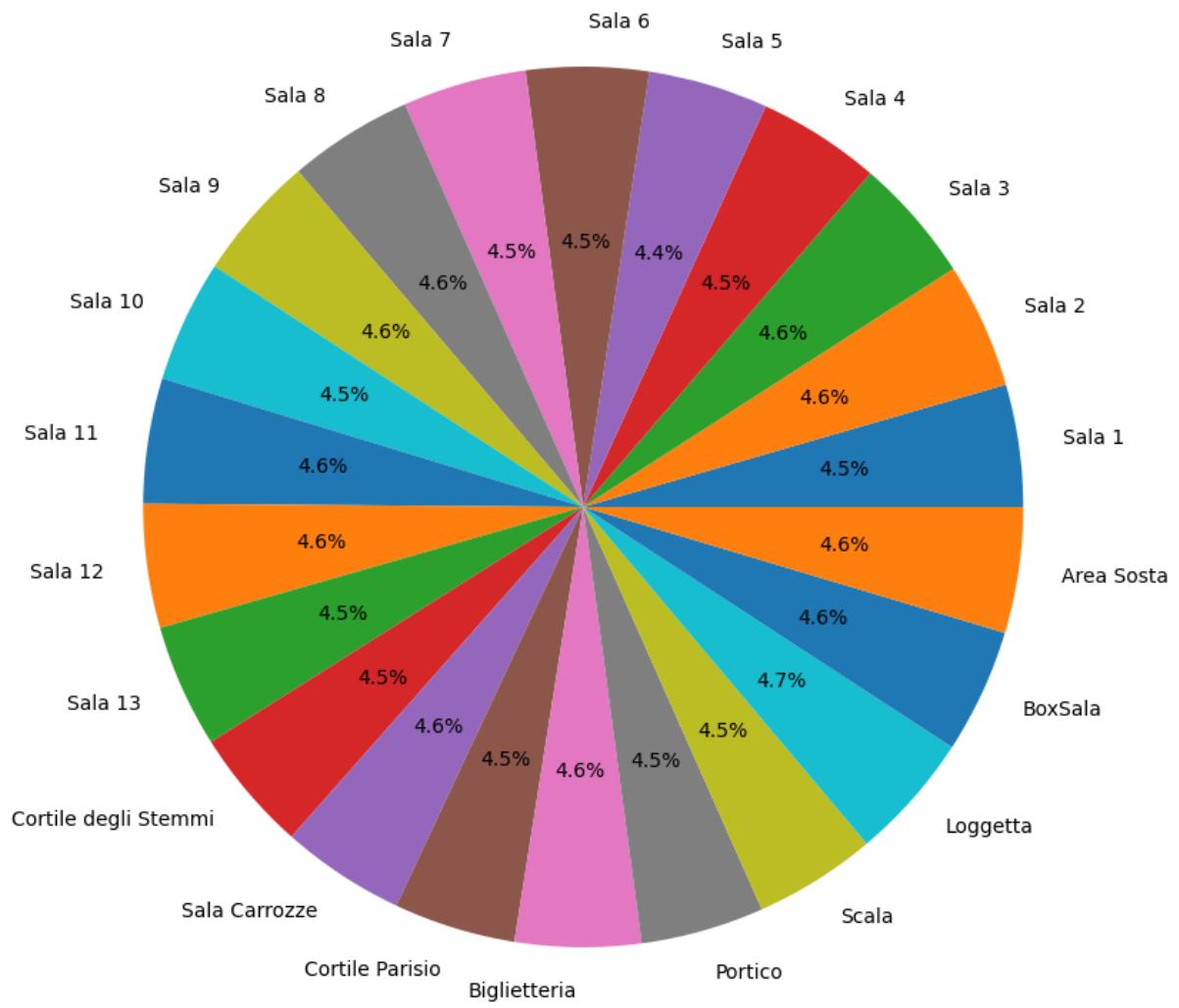


Figure 3: Distribuzione classi training set dopo il bilanciamento

## 4 Valutazione

### 4.1 Accuracy

$$Accuracy = \frac{\text{Numero totale di previsioni}}{\text{Numero di previsioni corrette}}$$

È una metrica intuitiva e di facile comprensione.

Può essere ingannevole in situazioni con classi sbilanciate. Ad esempio, se il 95% dei tuoi dati appartiene alla classe A e solo il 5% alla classe B, un modello che predice sempre la classe A avrà un'accuratezza del 95%, nonostante non sia un buon modello.

### 4.2 F1 score

$$F1\ Score = 2 \cdot \frac{Precision + Recall}{Precision \cdot Recall}$$

$$Precision = \frac{\text{Veri Positivi (TP)}}{\text{Veri Positivi (TP)} + \text{Falsi Positivi (FP)}}$$

$$Recall = \frac{\text{Veri Positivi (TP)}}{\text{Veri Positivi (TP)} + \text{Falsi Negativi (FN)}}$$

L'F1 score è particolarmente utile quando si ha un dataset sbilanciato. È una metrica equilibrata che considera sia falsi positivi che falsi negativi. Un valore di F1 score perfetto è 1, mentre il valore peggiore è 0. Se si desidera dare la stessa importanza a precisione e recall, l'F1 score è una buona metrica.

Durante la fase di training ho utilizzato l'accuracy in quanto più rapida da calcolare rispetto a F1 score, mentre F1 score è stato usato per valutazione in fase di testing.

## 5 Esperimenti

### 5.1 Prima prova

Modello	lr	Momentum	Regolarizzazione	Dropout
ResNet-18	0.01	0.99	0	50%

Table 1: Parametri del Modello

#### 5.1.1 Prime 15 epochhe

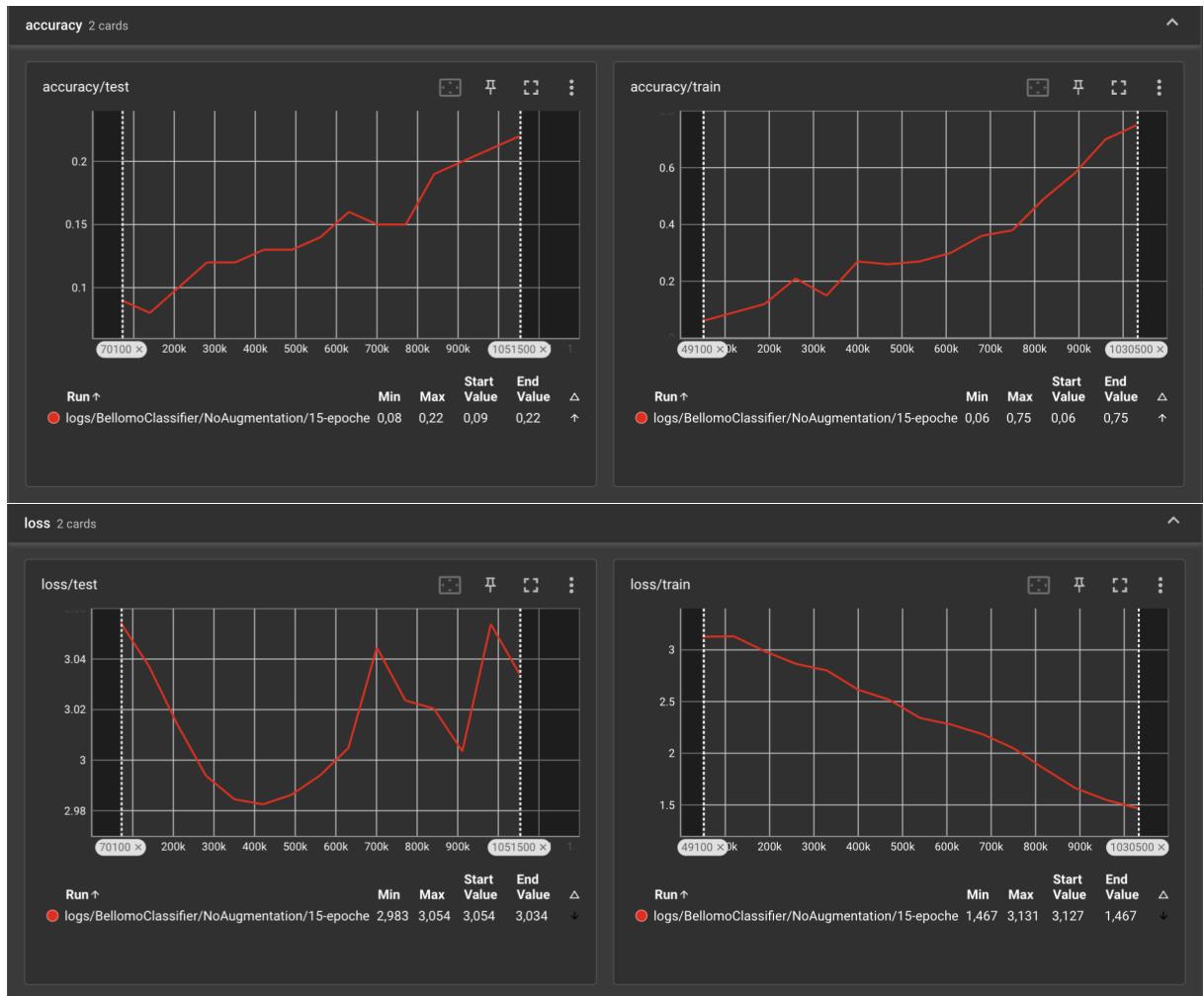


Figure 4: Monitoraggio fase di training

### 5.1.2 Altre 15 epochhe

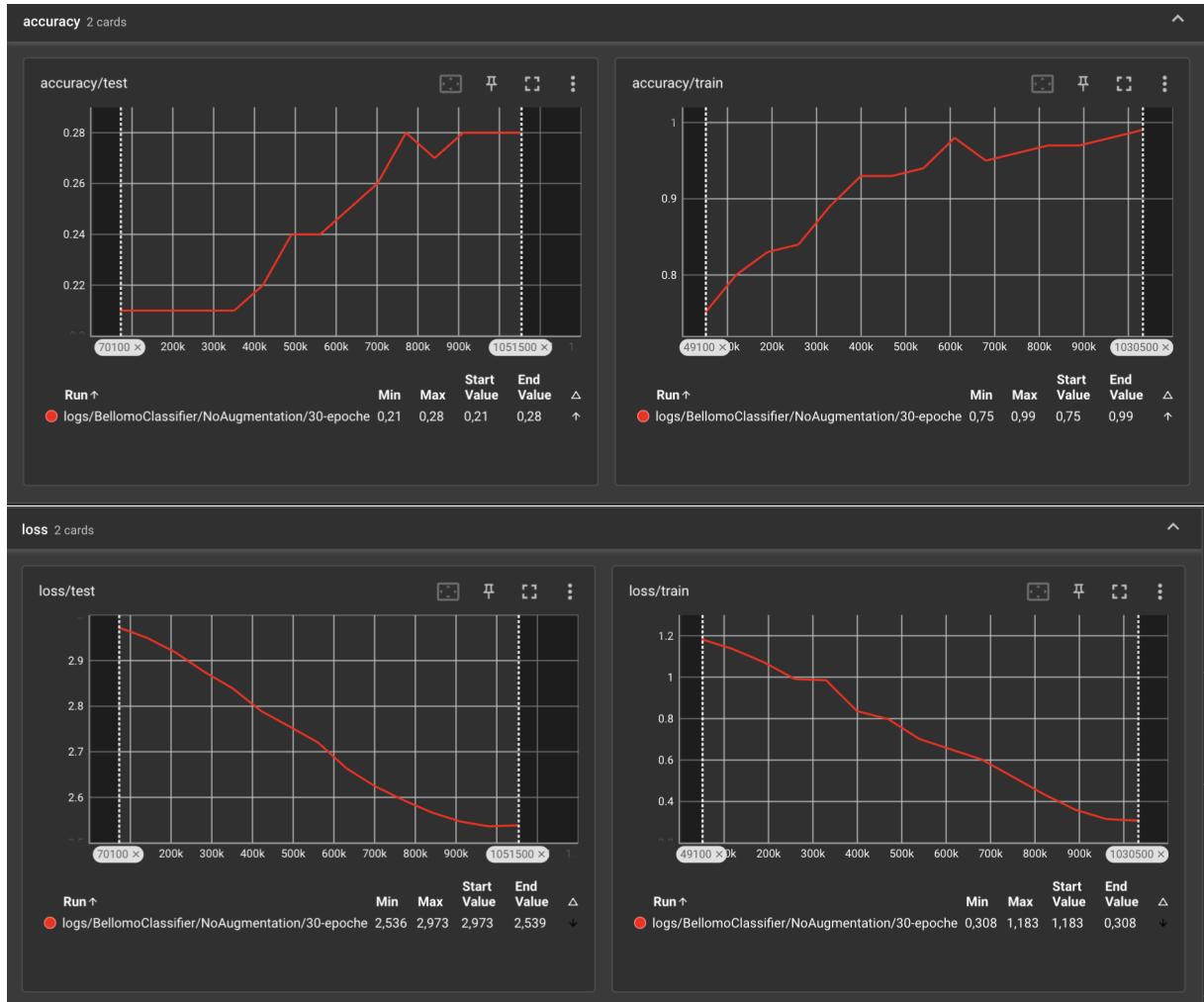


Figure 5: Monitoraggio fase di training

La discrepanza significativa tra l'accuratezza di test e quella di addestramento indica un evidente overfitting del modello. Un'accuratezza di addestramento tra 0,75 e 0,99 indica che il modello sta performingo molto bene sui dati con cui è stato addestrato. D'altro canto, un'accuratezza di test tra 0,21 e 0,28 suggerisce che il modello sta avendo grosse difficoltà a generalizzare su dati non visti durante la fase di addestramento. Per la prossima prova introduciamo regolazione norma 2 e augmentation dei dati del training set.

## 5.2 Seconda prova

Modello	lr	Momentum	Regolarizzazione	Dropout
ResNet-18	0.01	0.99	1e-3	50%

Table 2: Parametri del Modello

### 5.2.1 Prime 15 epocha



Figure 6: Monitoraggio fase di training

## 5.2.2 Altre 15 epochhe

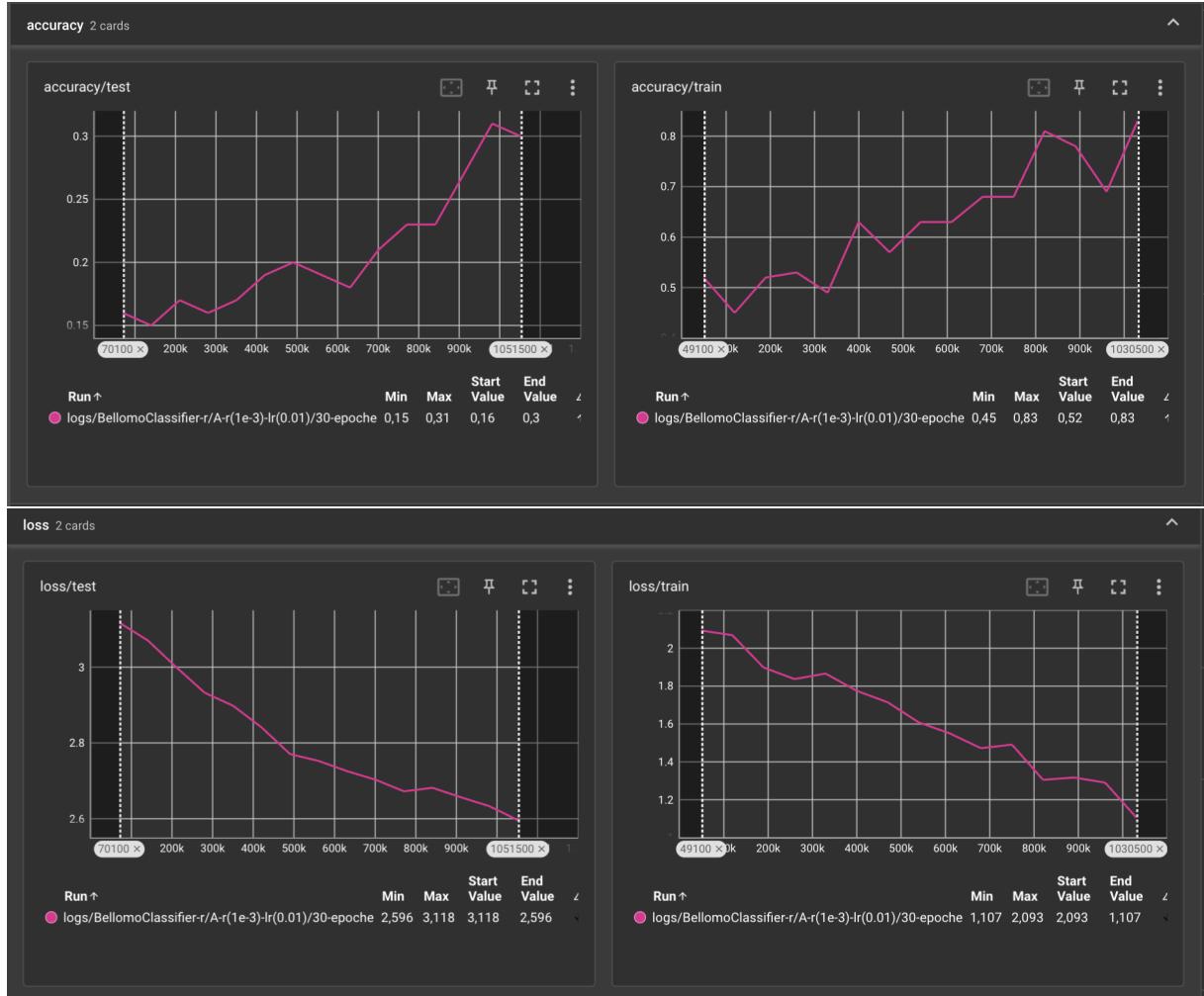


Figure 7: Monitoraggio fase di training

Osservando i dati possiamo notare che l'overfitting rispetto alla prima prova (5.1) è ridotto ma è comunque presente. Nella prossima prova aumenteremo il fattore di regolarizzazione.

### 5.3 Terza prova

Modello	lr	Momentum	Regolarizzazione	Dropout
ResNet-18	0.01	0.99	1e-2	50%

Table 3: Parametri del Modello

#### 5.3.1 Prime 15 epoche

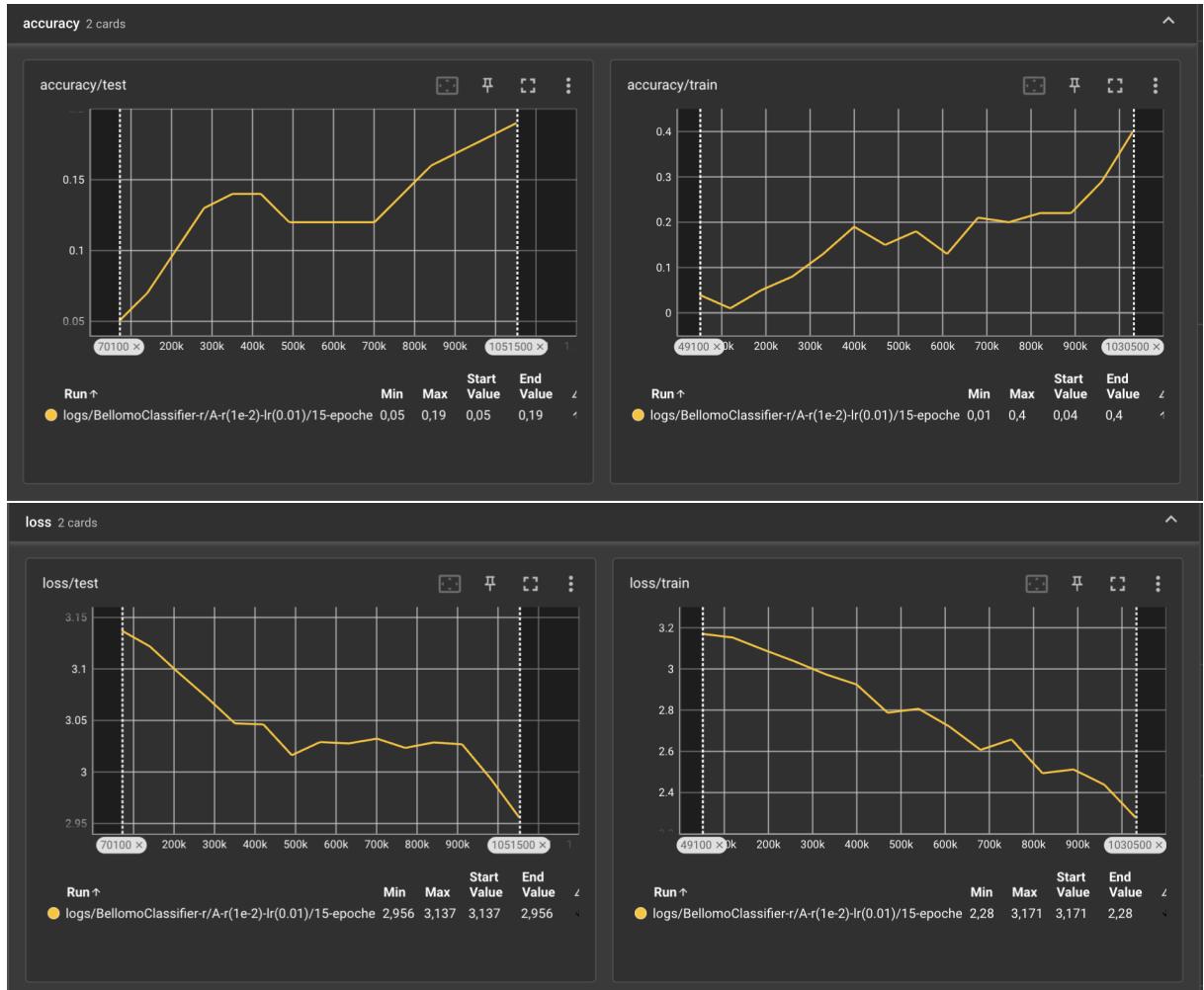


Figure 8: Monitoraggio fase di training

### 5.3.2 Altre 15 epochhe

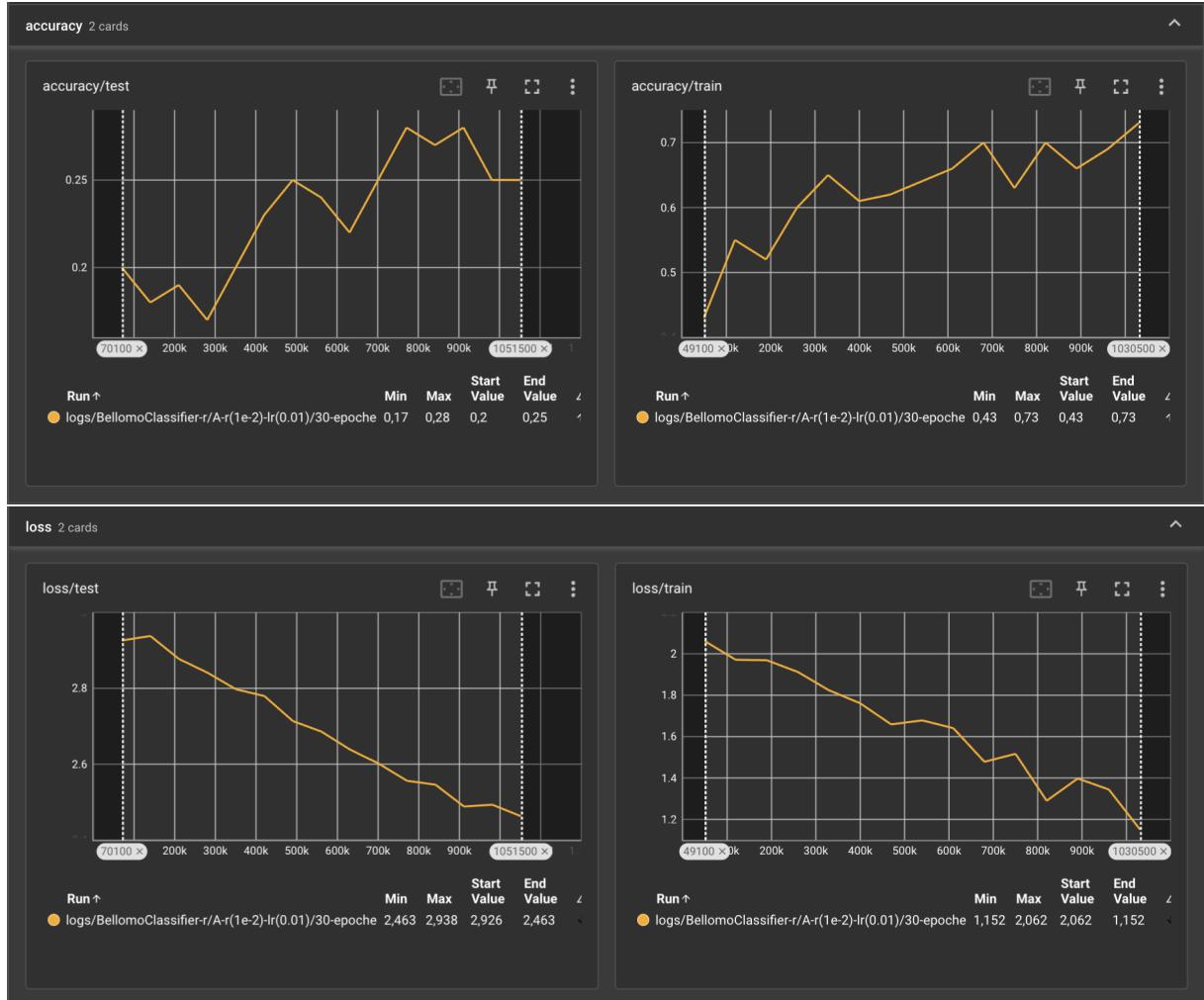


Figure 9: Monitoraggio fase di training

Pur notando segni di overfitting durante l'addestramento, ho deciso di continuare a allenare il modello. La mia decisione è stata guidata dall'osservazione che, nonostante l'overfitting, la funzione di loss continuava a diminuire, suggerendo che il modello stava migliorando nella sua capacità di generalizzazione. L'approccio che ho adottato ha previsto l'uso di un learning rate iniziale relativamente alto, per accelerare la fase di training. Successivamente, ho ridotto il learning rate, permettendo al modello di fare aggiustamenti più precisi e sottili man mano che si avvicinava alla convergenza.

### 5.3.3 Altre 10 epochhe

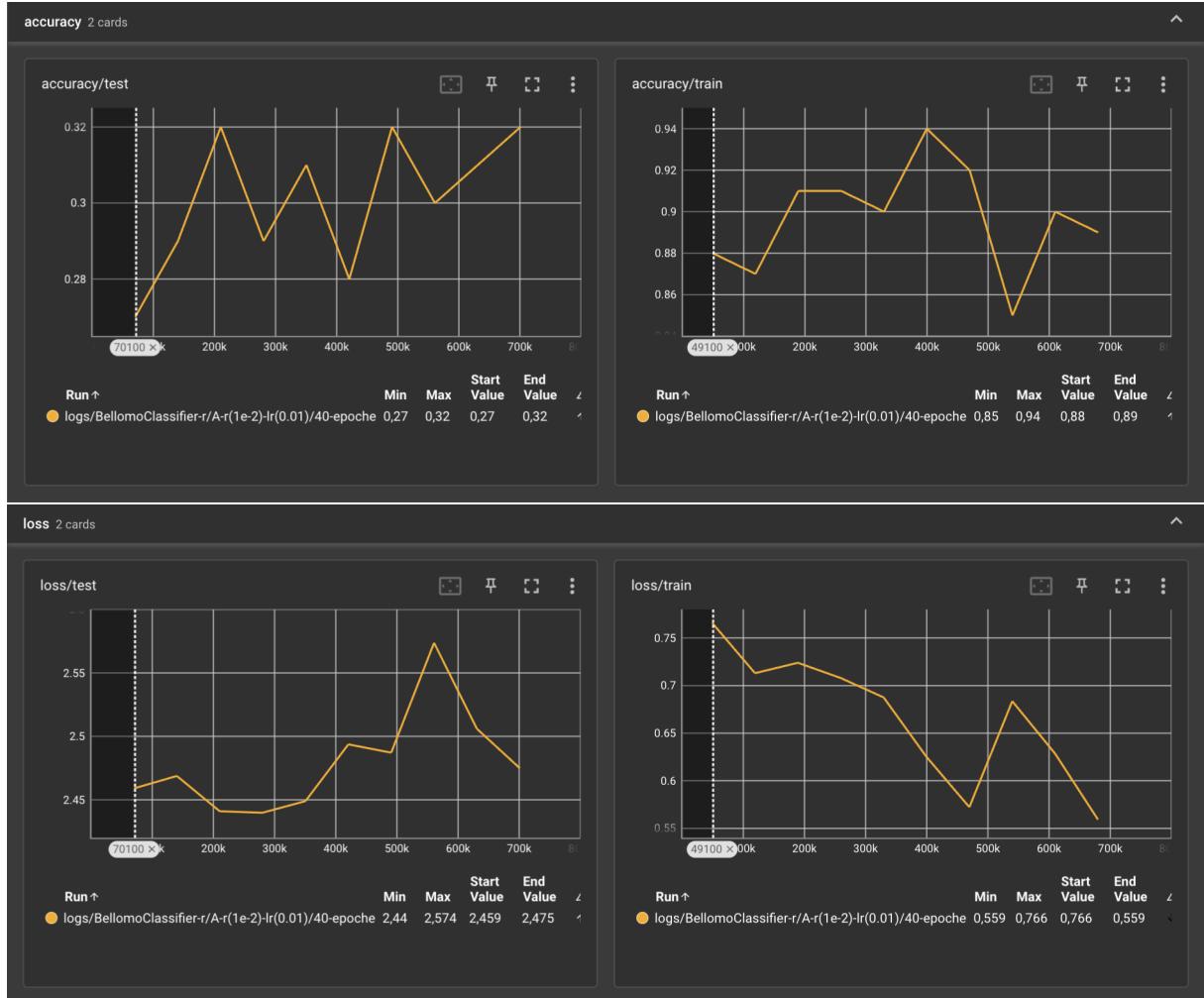


Figure 10: Monitoraggio fase di training

Nelle prossime epochhe passeremo da un lr = 0.01 a un lr = 0.001.

Modello	lr	Momentum	Regolarizzazione	Dropout
ResNet-18	0.001	0.99	1e-2	50%

Table 4: Parametri del Modello

### 5.3.4 Altre 10 epochhe

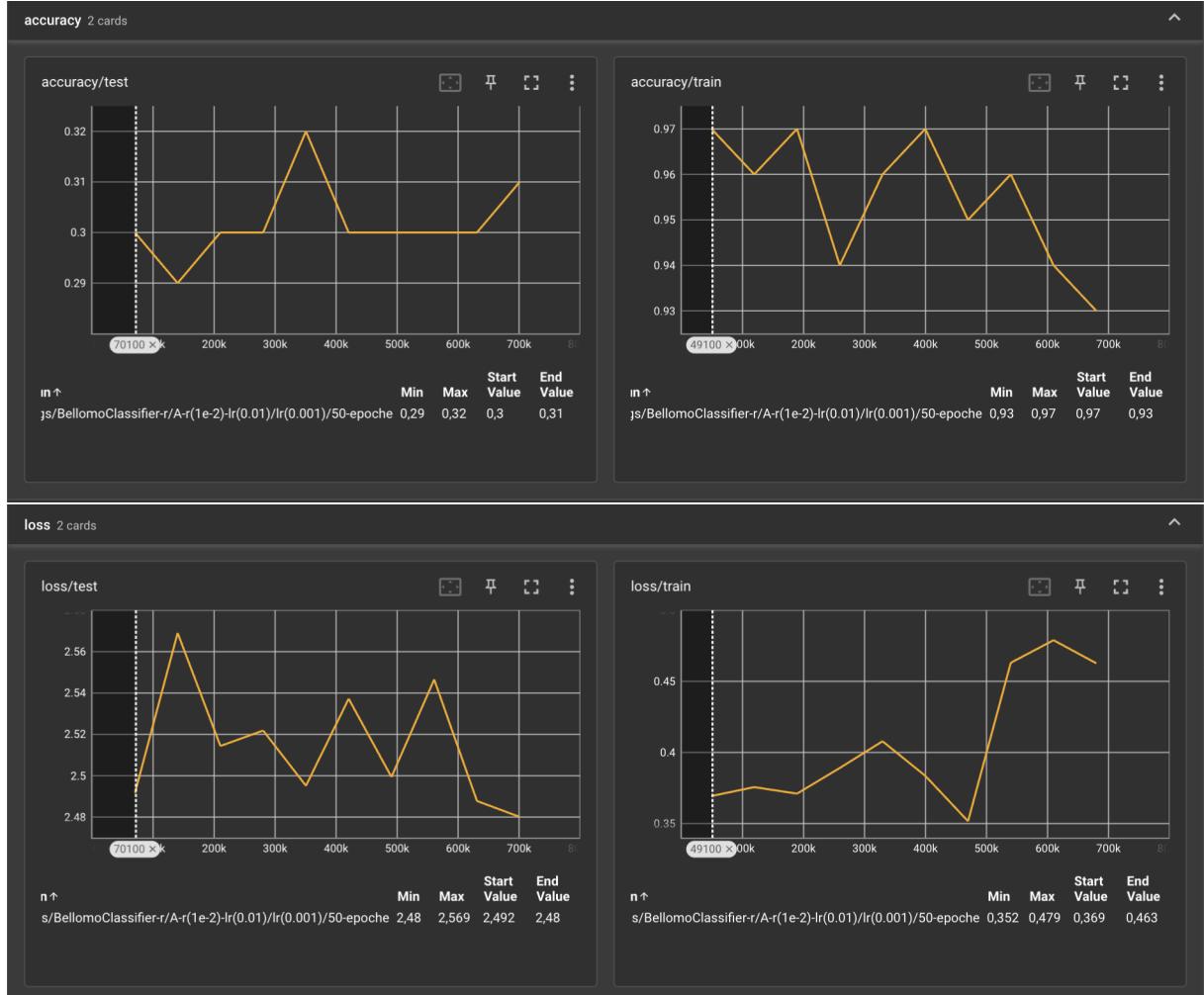


Figure 11: Monitoraggio fase di training

Ho notato che la funzione di perdita (loss) sul validation set sembra sul punto di stabilizzarsi, mentre quella relativa al training set continua a diminuire (grafico precedente 0.6, attuale 0.46). Considerando queste tendenze, ho scelto di estendere l'addestramento per altre 15 epochhe.

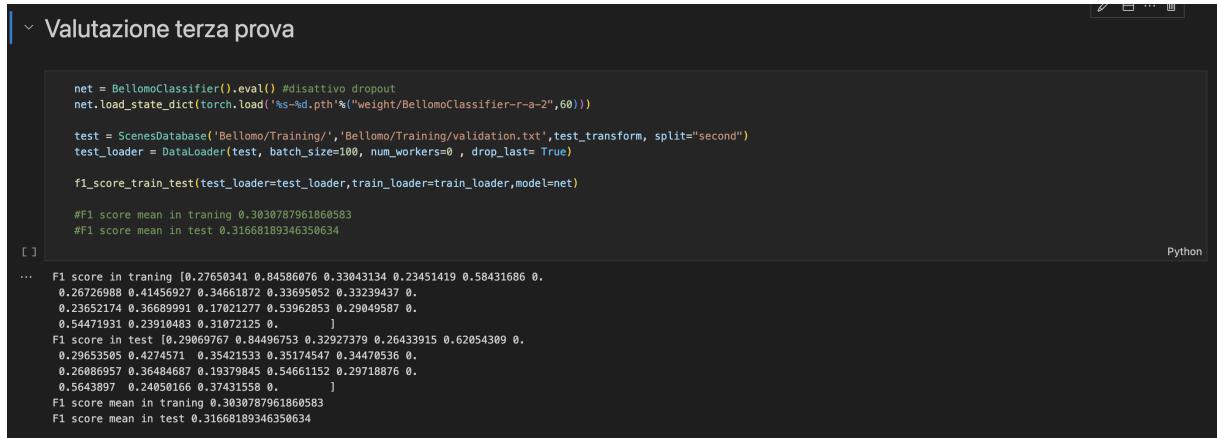
### 5.3.5 Altre 15 epochhe



Figure 12: Monitoraggio fase di training

Calcoliamo F1 score del training e test per ottenere una visione più completa delle prestazioni del modello.

### 5.3.6 Valutazione terza prova



```
net = BellomoClassifier().eval() #disattivo dropout
net.load_state_dict(torch.load("%s-%d.pth"%(weight/BellomoClassifier-r-a-2",60)))

test = ScenesDatabase('Bellomo/Training/','Bellomo/Training/validation.txt',test_transform, split="second")
test_loader = DataLoader(test, batch_size=100, num_workers=0 , drop_last= True)

f1_score_train_test(test_loader=test_loader,train_loader=train_loader,model=net)

#F1 score mean in training 0.3030787961860583
#F1 score mean in test 0.31668189346350634

[ ] F1 score in training [0.27650314 0.84586076 0.33043134 0.23451419 0.58431686 0.
0.26726988 0.41456927 0.34661872 0.33695852 0.33239437 0.
0.23652174 0.36689991 0.17021277 0.53962853 0.29049587 0.
0.54471931 0.23910483 0.31072125 0.]
F1 score in test [0.29869767 0.84496753 0.32927379 0.26433915 0.62054309 0.
0.29653505 0.4274571 0.35421533 0.35174547 0.34470536 0.
0.26086957 0.36484687 0.19379845 0.54661152 0.29718876 0.
0.5643897 0.24050166 0.37431558 0.]
F1 score mean in training 0.3030787961860583
F1 score mean in test 0.31668189346350634
```

- Accuracy in test 39.36%

Si osserva che i valori dell'F1 score ottenuti sia nel training che nel test sono molto simili, inoltre analizzando il vettore dell'F1 score, emergono discrepanze nell'F1 score tra le diverse classi. Supponendo che potrebbe esserci un problema legato al bilanciamento delle classi nel dataset, nella prossima prova bilanceremo il set di training, come descritto in dettaglio nella sezione 3.4.

## 5.4 Quarta prova

Modello	lr	Momentum	Regolarizzazione	Dropout
ResNet-18	0.01	0.99	1e-3	50%

Table 5: Parametri del Modello

### 5.4.1 Prime 15 epocha



Figure 13: Monitoraggio fase di training

Modello	lr	Momentum	Regolarizzazione	Dropout
ResNet-18	0.001	0.99	1e-3	50%

Table 6: Parametri del Modello

#### 5.4.2 Altre 15 epochhe

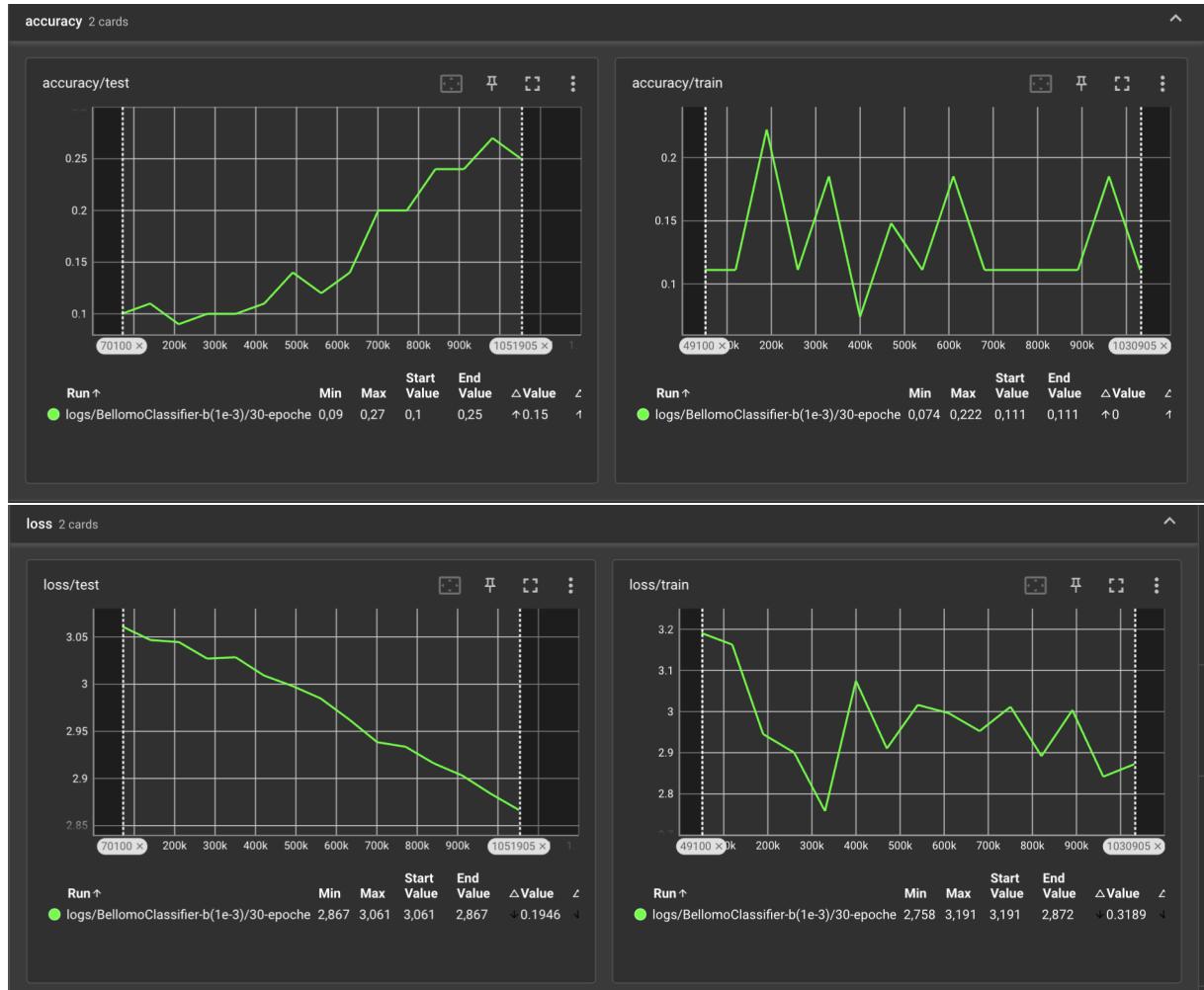


Figure 14: Monitoraggio fase di training

### 5.4.3 Altre 15 epochhe

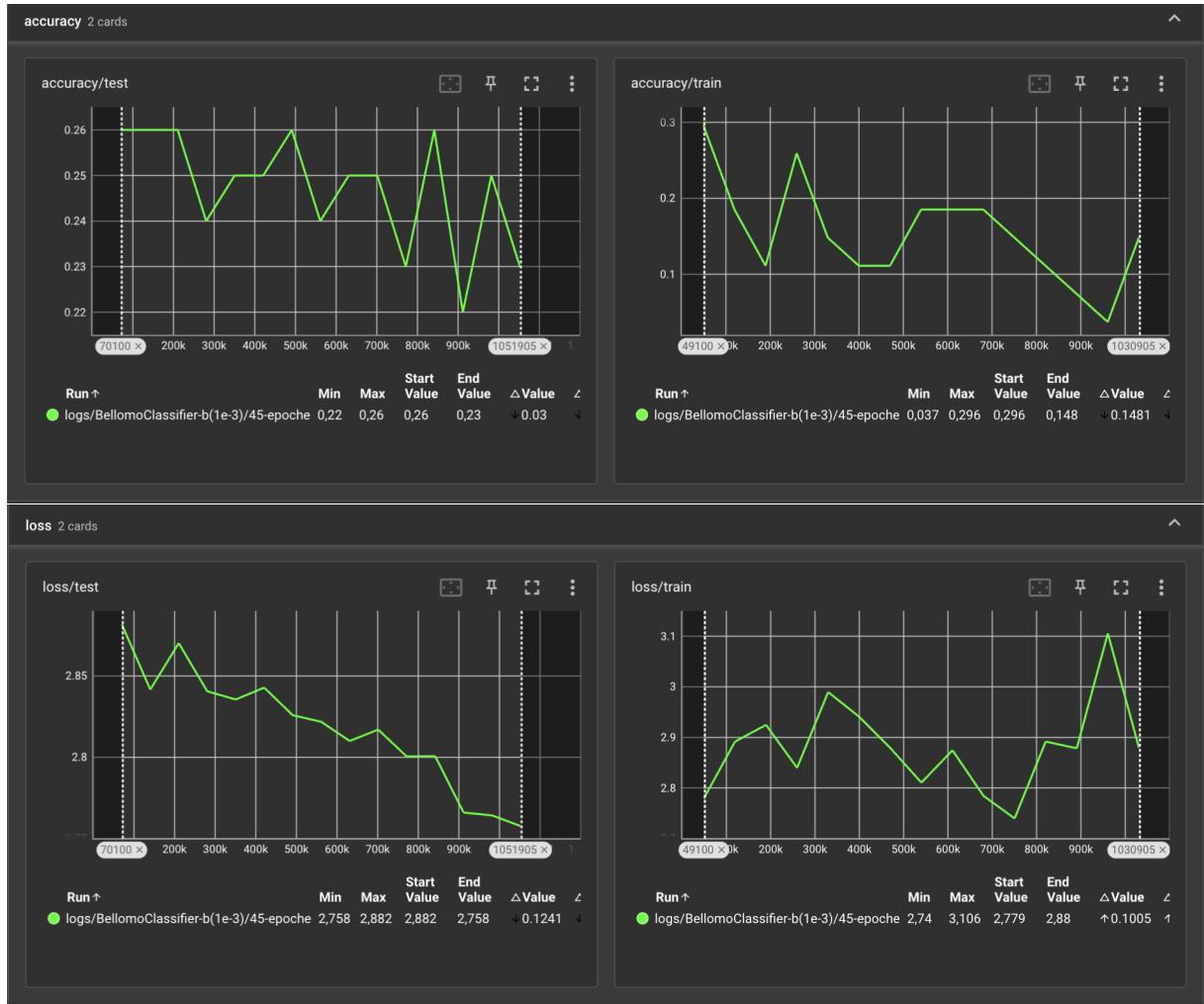


Figure 15: Monitoraggio fase di training

- F1 score mean in training 0.1827914073974613
- F1 score mean in test 0.18687599937993526
- Accuracy in test 23.66%

Notiamo che bilanciando il training set non abbiamo ottenuto i miglioramenti che ci aspettavamo, i valori di accuracy sono molto bassi sia in validation set sia in training set inoltre i valori f1 score suggeriscono che il modello soffre di underfitting.

Una soluzione potrebbe essere aumentare la complessità del modello per aumentare la capacità del modello a estrarre maggiori informazioni, utilizzando, ad esempio, una versione più profonda del ResNet-18 come ResNet-50.

Per motivi di limite di risorse (hardware) e limiti di tempo, non verrà addestrato un modello più complesso. Per cercare di migliorare le prestazioni ottenute partiremo dal modello allenato nella terza prova (5.3) il quale aveva ottenuto migliori valutazioni e aumenteremo il numero di dati del training set per permettere al modello di generalizzare meglio.

## 5.5 Quinta prova

In questa prova utilizzeremo il metodo proposto nella terza prova (5.3) con la differenza del training set di dimensione maggiore, sperando che aumentando i dati, il modello riesca ad ottenere più informazioni per generalizzare. Per velocizzare la fase di training partiremo dai parametri ottenuti nelle ultime epoche del modello allenato nella terza prova 5.3.5.

Modello	lr	Momentum	Regolarizzazione	Dropout
ResNet-18	0.01	0.99	1e-2	50%

Table 7: Parametri del Modello

### 5.5.1 Prime 15 epochhe



Figure 16: Monitoraggio fase di training

### 5.5.2 Altre 15 epochhe



Figure 17: Monitoraggio fase di training

Il modello ha mostrato miglioramenti in termini di accuratezza sul set di validazione, tuttavia, persiste il problema dell'overfitting. Nelle successive epochhe, prevediamo di aumentare il fattore di regolarizzazione e di ridurre il tasso di apprendimento (learning rate), dato che gli ultimi valori di loss sul set di training sono molto simili tra loro.

Modello	lr	Momentum	Regolarizzazione	Dropout
ResNet-18	0.001	0.99	5e-2	50%

Table 8: Parametri del Modello

### 5.5.3 Altre 15 epochhe



Figure 18: Monitoraggio fase di training

Modello	lr	Momentum	Regolarizzazione	Dropout
ResNet-18	1e-4	0.99	5e-2	50%

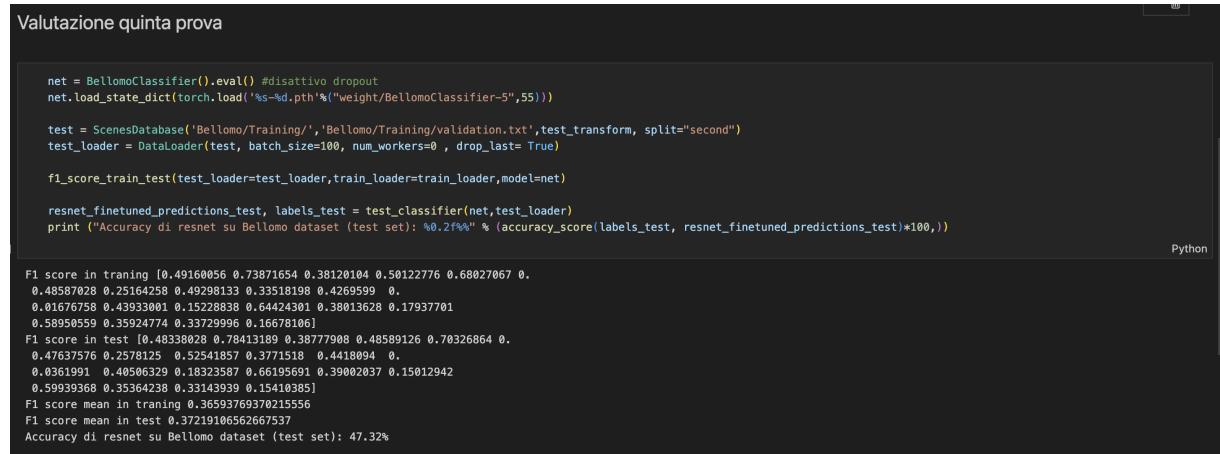
Table 9: Parametri del Modello

#### 5.5.4 Altre 10 epochhe



Figure 19: Monitoraggio fase di training

### 5.5.5 Valutazione quinta prova



```

net = BellomoClassifier().eval() #disattivo dropout
net.load_state_dict(torch.load('%s-%d.pth'%(weight/BellomoClassifier-5",55)))

test = ScenesDatabase('Bellomo/Training/','Bellomo/Training/validation.txt',test_transform, split="second")
test_loader = DataLoader(test, batch_size=100, num_workers=0 , drop_last= True)

f1_score_train_test(test_loader=test_loader,train_loader=train_loader,model=net)

resnet_finetuned_predictions_test, labels_test = test_classifier(net,test_loader)
print ("Accuracy di resnet su Bellomo dataset (test set): %.2f%%" % (accuracy_score(labels_test, resnet_finetuned_predictions_test)*100,))

F1 score in traning [0.49160056 0.73871654 0.38120104 0.50122776 0.68027067 0.
0.48587028 0.25164258 0.49298133 0.33518198 0.4269599 0.
0.01676758 0.43933001 0.15228838 0.64424301 0.38013628 0.17937701
0.58950559 0.35924774 0.33729996 0.16678106]
F1 score in test [0.48338028 0.78413189 0.38777908 0.48589126 0.70326864 0.
0.47637576 0.2578125 0.52541857 0.3771518 0.4418094 0.
0.0361991 0.40506329 0.18323587 0.66195691 0.39002037 0.15012942
0.59939368 0.35364238 0.33143939 0.15410385]
F1 score mean in training 0.36593769370215556
F1 score mean in test 0.37219106562667537
Accuracy di resnet su Bellomo dataset (test set): 47.32%

```

Figure 20: F1 score & accuracy

Dai dati ottenuti vediamo un miglioramento sia in F1 score che in accuracy.

### 5.6 Tabella risultati

I risultati fanno riferimento all'ultima epoca di addestramento.

Prova	Acc. TS	Acc. VS	Acc. TeS	F1 mean TS	F1 mean TeS
3	98%	31%	39.98%	0.303	0.316
4	29%	26%	23.66%	0.182	0.186
5	99%	43%	47.32%	0.365	0.372

Acronimo	Nome Completo
TS	Training Set
VS	Validation Set
TeS	Test Set
Acc.	Accuracy
F1	F1 Score

Table 10: Acronimi e nomi completi

## 6 Demo

L'applicazione demo offre un'interfaccia che permette agli utenti di osservare come il classificatore opera sulle immagini. Le immagini usate per la dimostrazione provengono da un test set. Una volta avviata l'applicazione, l'utente ha la possibilità di cliccare su un pulsante etichettato "Run Demo". Premendo questo pulsante, l'applicazione avvia l'inferenza del modello sulle immagini, mostrando quindi l'immagine prelevata, le etichette corrette e le previsioni del modello in tempo reale.

### Panoramica del codice:

- **Importazione delle librerie e dei moduli:** Vengono importate le librerie **torch**, **numpy**, **tkinter** e **PIL** e i moduli personalizzati **ClassDictionary**, **ScenesDatabase**, e **BellomoClassifier**.
- **Preprocessamento delle immagini:** Viene definita una sequenza di trasformazioni "test\_transform". Queste trasformazioni ridimensionano le immagini a 224x224, le convertono in tensori e le normalizzano con valori medi e standard specifici (m e s) al fine di renderle un input compatibile al modello (BellomoClassifier).
- **Caricamento del dataset:** Viene caricato il dataset personalizzato "ScenesDatabase" contenente le immagini del test set da classificare.
- **Creazione dell'applicazione GUI:** La classe BellomoApp estende tk.Tk e rappresenta la finestra principale dell'applicazione GUI. Al suo interno, vengono creati widget, come un canvas per visualizzare le immagini, due label per mostrare le previsioni e un pulsante per interagire con l'app. Il modello di classificazione BellomoClassifier viene caricato e preparato per l'inferenza.
- **Esecuzione demo:** Quando l'utente clicca sul pulsante "Run Demo", viene chiamata la funzione call\_demo che a sua volta chiama la funzione demo. La funzione demo prende un'immagine contenuta nel test set, la processa attraverso il modello di classificazione, e visualizza le immagini e le loro previsioni sull'interfaccia grafica.
- **Visualizzazione delle immagini e output:** Le funzioni display\_image\_on\_canvas e display\_prediction\_text sono utilizzate per visualizzare rispettivamente le immagini e le etichette previste (insieme alle etichette originali).
- **Esecuzione dell'applicazione:** Dopo aver eseguito il codice, si aprirà una finestra intitolata "Demo". In questa finestra, sarà presente un pulsante "Run Demo". Cliccando su questo pulsante, si avvierà il processo di inferenza sulle immagini.

## 7 Codice

Il codice è strutturato in sezioni e sottosezioni. Per farlo funzionare bisogna eseguire i primi due blocchi (sezioni) e successivamente le varie prove. Per la prima prova basta eseguire i primi due blocchi e successivamente il blocco relativo alla prima prova. Per le successive prove eseguire i primi due blocchi "Dataset", "Fine-tuning" successivamente i blocchi "Data augmentation", "Regolarizzazione", "Altri metodi per la valutazione" e infine la prova che si vuole eseguire e se presenti le relative valutazioni.

### 7.1 Struttura codice

- Dataset

- Fine-tuning

Resnet-18

- Prima prova

- Data augmentation

- Regolarizzazione

- Seconda prova

- Terza prova

- Altri metodi per la valutazione

- Valutazione terza prova

- Quarta prova

Bilanciamento distribuzione classi

Training quarta prova

Valutazione quarta prova

- Quinta prova

Training quints prova

Valutazione quinta prova

## 8 Conclusion

Durante la fase di sperimentazione e ottimizzazione del modello, sono stati affrontati vari problemi e tentate diverse soluzioni:

1. **Overfitting Iniziale:** I risultati iniziali hanno mostrato un evidente overfitting, con un'accuratezza del set di addestramento tra 0,75 e 0,99 mentre l'accuratezza del set di validation variava tra 0,21 e 0,28.  
(valori fanno riferimento alle prime 30 epoche)
2. **Regolarizzazione e Augmentazione dei Dati:** L'introduzione di regolarizzazione e augmentazione dei dati ha parzialmente mitigato l'overfitting, riducendo l'accuratezza del set di addestramento (acc. max 0,83) e migliorando leggermente quella del set di validation (acc. 0,31).  
(valori fanno riferimento alle prime 30 epoche)
3. **Aumento della Regolarizzazione:** Un ulteriore incremento del fattore di regolarizzazione ha reso i risultati più coerenti tra addestramento e validation, portando le accuratezze rispettivamente a 0,73 e 0,28.  
(valori fanno riferimento alle prime 30 epoche)
4. **Learning Decay:** L'implementazione del learning decay ha migliorato le prestazioni del set di test, portando l'accuratezza al 39%. Tuttavia, c'era ancora una notevole discrepanza rispetto all'accuratezza del set di addestramento.
5. **Bilanciamento del Dataset:** Il tentativo di bilanciare il dataset ha portato a un underfitting significativo, con un'accuratezza del test scesa al 23,66% e F1 scores medi molto bassi sia per l'addestramento (0,1828) che per il test (0,1869).
6. **Aumento della Dimensione del Dataset:** Tornando al training set sbilanciato ma raddoppiando la sua dimensione. L'accuratezza del set di addestramento ha raggiunto il 99%, mentre le accuratezze di validazione e test sono state rispettivamente del 43% e 47,32%. L'F1 score, però, ha mostrato una certa coerenza tra addestramento e test.

In sintesi, malgrado i tentativi di affinare le prestazioni, il modello attuale ha persistito nel mostrare limitati di precisione e accuracy sul test set.

Una potenziale strategia di miglioramento potrebbe essere: l'adozione di un modello più complesso: considerare un passaggio da ResNet-18, con i suoi 11.7 milioni di parametri, a strutture come ResNet-50, che vanta circa 25.6 milioni di parametri, o magari VGG-16, con un numero di circa 138 milioni di parametri. Dopo aver scelto una nuova architettura, si potrebbe perseverare l'approccio corrente, sfruttando tecniche come la data augmentation, la regolarizzazione, il decay del tasso di apprendimento, l'uso di SGD, ecc., che hanno precedentemente dimostrato di apportare benefici e l'uso di un training set bilanciato. Se l'introduzione di una maggiore complessità architettonica risulta fruttuosa, si potrebbe ulteriormente affinare il modello esplorando diverse configurazioni di iperparametri, al fine di raggiungere un'ottimizzazione ideale.

Durante le mie sperimentazioni nell'allenamento del modello, ho chiaramente percepito

l'importanza dell'hardware. Nonostante avessi implementato tecniche come SGD con weight decay per migliorare l'efficienza, ho riscontrato che addestrare modelli di maggiore complessità richiedeva risorse hardware di alto livello per ottenere risultati in tempi accettabili.

Inizialmente, ho cercato di addestrare modelli utilizzando solo la mia CPU standard. con la quale ho subito riscontrato limiti evidenti nell'addestramento.

Quando ho iniziato ad informarmi su framework come TensorFlow e PyTorch, ho capito che erano ottimizzati principalmente per GPU, anche se offrivano supporto per CPU, era chiaro che per sfruttarli al meglio, una GPU era quasi essenziale. Passando parte dell'elaborazione alla GPU (pur non essendo di fascia alta) riscontrando dei miglioramenti dal punto di vista della velocità di esecuzione.

Concludendo, il mio percorso nel machine learning mi ha chiaramente dimostrato l'importanza dell'hardware nel determinare non solo la velocità, ma anche la fattibilità dell'addestramento dei modelli. Sebbene abbia iniziato con risorse limitate, le lezioni apprese durante questo viaggio mi hanno guidato verso soluzioni più efficaci e efficienti.