19.03.2025

Numero di classi da riconoscere: 5 ==> samples per il training (800 * 5) = 4_000 samples totali e 1_000 samples di CrossValidation

Prova del modello TinyVGG con le seguenti caratteristiche:

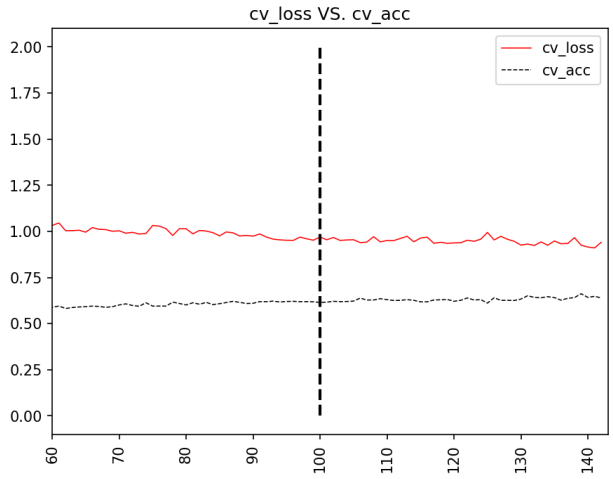
```
Model0(
  (first_conv): Sequential(
    (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
    (1): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (second_conv): Sequential(
    (0): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
    (1): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (third_conv): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
    (1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=1024, out_features=1024, bias=True)
    (2): Linear(in_features=1024, out_features=1024, bias=True)
    (3): Linear(in_features=1024, out_features=5, bias=True)
  )
)
```

Impostando un LR pari a 1e-4
BatchSize = 2 ** 10

Si ottiene un outcome come di seguito:

```
{'train_losses': 0.8115352243185043,
 'train_acc': 0.6955482219827587,
 'test_losses': 0.9513975977897644,
 'test_acc': 0.622}
```

Aumentare le epochs da 100 a 150 (circa) non ha avuto effetti di sorta. La loss e l'accuracy stallano attorno ai valori già assunti alla epoch nr.100:



21.03.2025

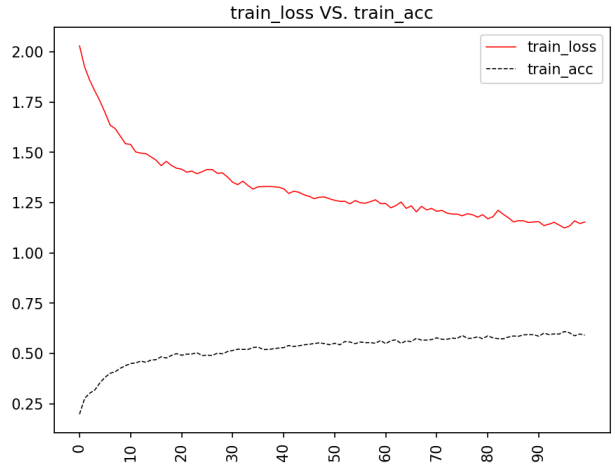
Training fatto con 8 classi per vedere se aumentando il numero di samples potesse migliorare il risultato finale.

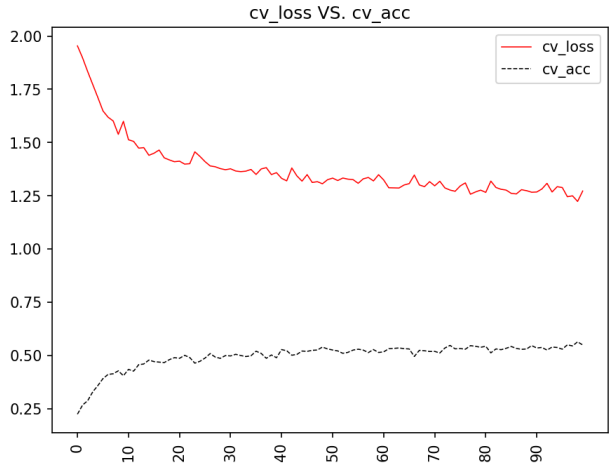
Modello0 come sopra.
LR invariato (1e-4)
BatchSize invariato (2 ** 10)

Outcome del modello dopo training:

```
{'train_losses': 1.178324580192566,
 'train_acc': 0.5864955357142857,
 'test_losses': 1.2726932168006897,
 'test_acc': 0.5496419270833334}
```

Questo il grafico della loss / accuracy





Modello salvato in:
`/Users/thomaspierantozzi/PycharmProjects/PyTorch_Train/05_PyTorch_Food101/saved_models/model0_8classes_0.0001_1024.pkl`

Provando ad aumentare il numero di epochs a 150 come nel caso precedente purtroppo non ci sono miglioramenti della loss; le prime 14 epochs (da 100 a 114) non forniscono il minimo miglioramento della percentuale di accuracy né della loss.

22.03.2025

Testato un modello pari a quello di cui sopra in cui i layers fully connected sono stati impostati per errore con un numero di nodi inferiore a quello che prevede l'archiettura (VGG) cui ci si sta ispirando (sotto in rosso):

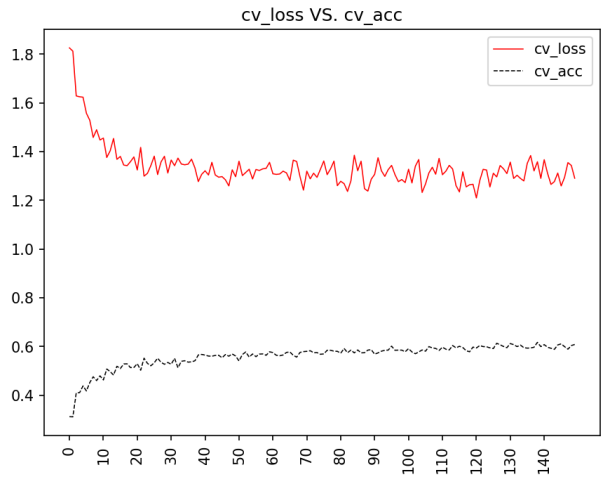
```
Model0(
  (first_conv): Sequential(
    (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
    (1): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (second_conv): Sequential(
    (0): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
    (1): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
    (2): ReLU()
```

```
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (third_conv): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
    (1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=9216, out_features=1024, bias=True)
    (2): Linear(in_features=1024, out_features=1024, bias=True)
    (3): Linear(in_features=1024, out_features=8, bias=True)
  )
)
```

La differenza sostanziale col modello di sopra sta nel batch_size. Qui è stato usato un batch size di:
Batch_size=32
LEarning_rate invariato (1e-4)

Il training è stato piuttosto veloce e non si sono rilevati grossi problemi di rumore nella loss di training; al contrario la loss di cross_validation è stata piuttosto affetta da questo tipo di problema, tanto che si è reso necessario usare una Moving Average (poco sotto il dettaglio) per estrapolare un trend dai dati di CrossValidation.

Nonostante l'errore il modello ha svolto un training su 150 iterazioni portando ai seguenti risultati:

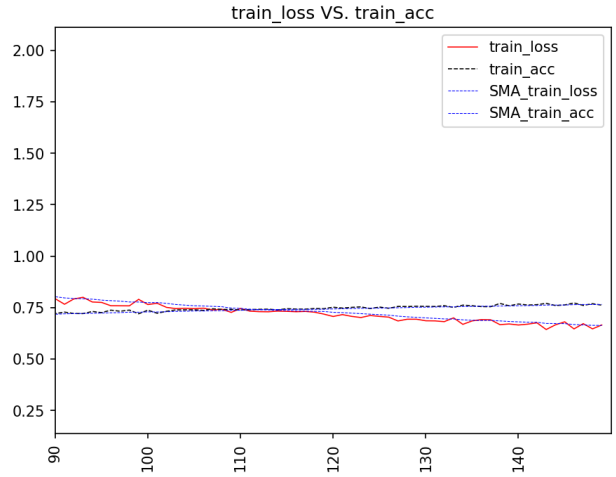


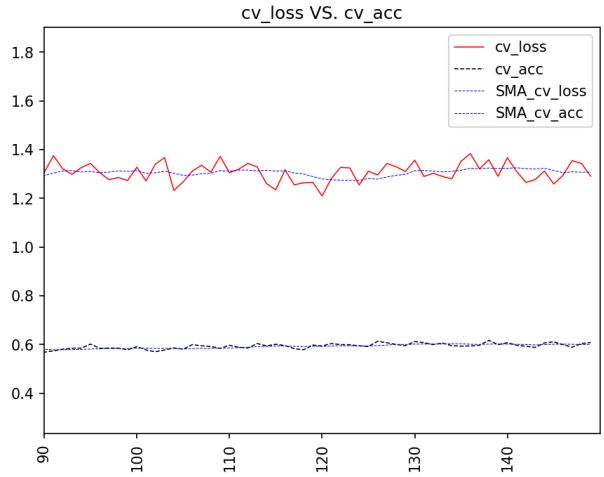

```
{'train_losses': 0.6093571089953184,
 'train_acc': 0.78546875,
 'test_losses': 1.29103190690279,
 'test_acc': 0.60875}
```

Per quanto possa sembrare rassicurante la discesa della train_loss a destra, anche durante le ultime iterazioni, se si va a plottare la SimpleMovingAverage fatta su una finestra di 10 rilevazioni, i trend risultano più evidenti. Se per quanto riguarda la train_loss si conferma quanto visibile a occhio nudo, è anche vero che si dà anche conferma al fatto che la cv_loss sembra avere un trend praticamente a zero decrescita (così come anche la cv_acc):

Qui di sotto riporto un'analisi fatta, per mezzo della funzione polyfit di Numpy, che non fa altro che fare una regressione sulla SMA dando in uscita il valore del coefficiente della X, della regressione stessa:

```
Considering the last 10 epochs, the cv_loss trend is: -2.369e-03
Considering the last 15 epochs, the cv_loss trend is: -1.143e-03
```






```
Considering the last 20 epochs, the cv_loss trend is: -1.680e-04
Considering the last 25 epochs, the cv_loss trend is: 1.028e-03
Considering the last 30 epochs, the cv_loss trend is: 1.561e-03
Considering the last 35 epochs, the cv_loss trend is: 9.900e-04
Considering the last 40 epochs, the cv_loss trend is: 5.064e-04
Considering the last 45 epochs, the cv_loss trend is: 4.166e-04
Considering the last 50 epochs, the cv_loss trend is: 2.851e-04
```

Si nota come la decrescita sia vicino allo zero e addirittura di segno positivo (CRESCITA quindi) se si considerano dalle ultime 30 alle ultime 50 epochs.

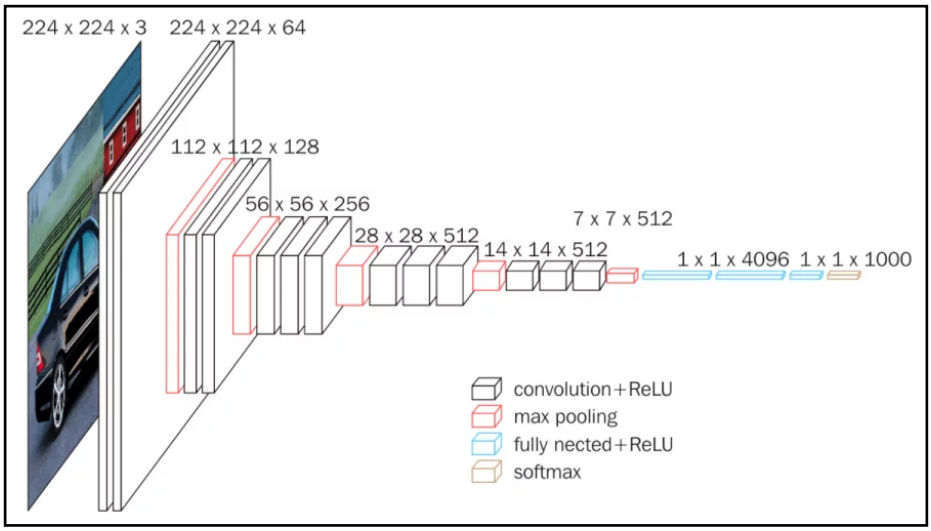
Di sicuro il training è finito se si tengono in considerazione questi segnali.

I risultati ad ogni modo NON SONO confortanti e si passa ad analizzare un nuovo modello, molto più complesso e molto più simile al modello VGG16.

24.03.2025

Il modello VGG16 è stato lanciato su server Amazon AWS per un primo run da 50 epochs.

La struttura del modello è simile a quella VGG16 standard:



La differenza principale sta nel numero di canali per layer convolutivo. Per ragioni di memoria non ho potuto fare un training con un numero di parametri così alto. Il modello è stato leggermente modificato come riportato sotto dal summary stampato da TorchInfo.

```
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
Model0                                   [1, 8]                    --
├─Sequential: 1-1                        [1, 16, 126, 126]         --
│    └─Conv2d: 2-1                       [1, 16, 254, 254]         448
│    └─Conv2d: 2-2                       [1, 16, 252, 252]         2,320
│    └─ReLU: 2-3                         [1, 16, 252, 252]         --
│    └─MaxPool2d: 2-4                    [1, 16, 126, 126]         --
├─Sequential: 1-2                        [1, 32, 60, 60]           --
│    └─Conv2d: 2-5                       [1, 32, 124, 124]         4,640
│    └─Conv2d: 2-6                       [1, 32, 122, 122]         9,248
│    └─Conv2d: 2-7                       [1, 32, 120, 120]         9,248
│    └─ReLU: 2-8                         [1, 32, 120, 120]         --
│    └─MaxPool2d: 2-9                    [1, 32, 60, 60]           --
├─Sequential: 1-3                        [1, 64, 27, 27]           --
│    └─Conv2d: 2-10                      [1, 64, 58, 58]           18,496
│    └─Conv2d: 2-11                      [1, 64, 56, 56]           36,928
│    └─Conv2d: 2-12                      [1, 64, 54, 54]           36,928
│    └─ReLU: 2-13                        [1, 64, 54, 54]           --
│    └─MaxPool2d: 2-14                   [1, 64, 27, 27]           --
├─Sequential: 1-4                        [1, 128, 10, 10]          --
│    └─Conv2d: 2-15                      [1, 128, 25, 25]          73,856
│    └─Conv2d: 2-16                      [1, 128, 23, 23]          147,584
│    └─Conv2d: 2-17                      [1, 128, 21, 21]          147,584
│    └─ReLU: 2-18                        [1, 128, 21, 21]          --
│    └─MaxPool2d: 2-19                   [1, 128, 10, 10]          --
├─Sequential: 1-5                        [1, 8]                    --
│    └─Flatten: 2-20                     [1, 12800]                --
│    └─Linear: 2-21                      [1, 12800]                163,852,800
│    └─Linear: 2-22                      [1, 12800]                163,852,800
│    └─Linear: 2-23                      [1, 8]                    102,408
==========================================================================================
Total params: 328,295,288
Trainable params: 328,295,288
Non-trainable params: 0
Total mult-adds (Units.GIGABYTES): 1.32
==========================================================================================
Input size (MB): 0.79
Forward/backward pass size (MB): 34.48
Params size (MB): 1313.18
Estimated Total Size (MB): 1348.45
==========================================================================================
```

I risultati sono più promettenti di quanto atteso.

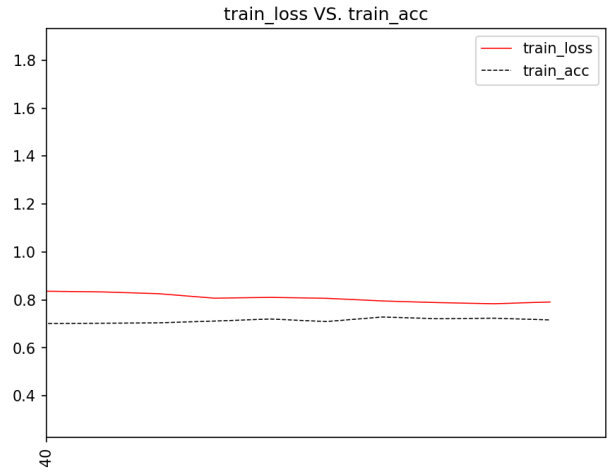
LEARNING RATE = 1e-5

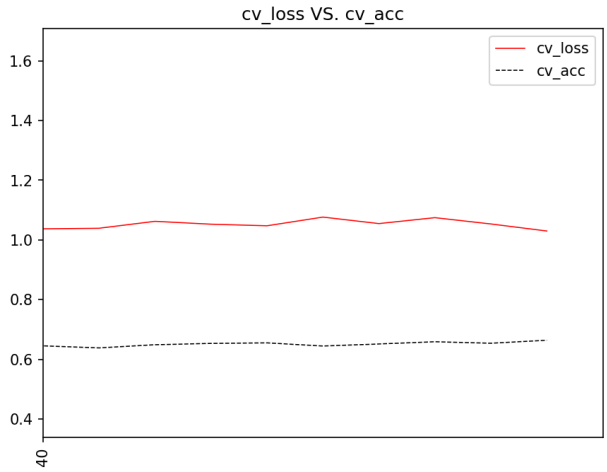
BATCH_SIZE = 32

Di seguito l'output calcolato sul batch di test:

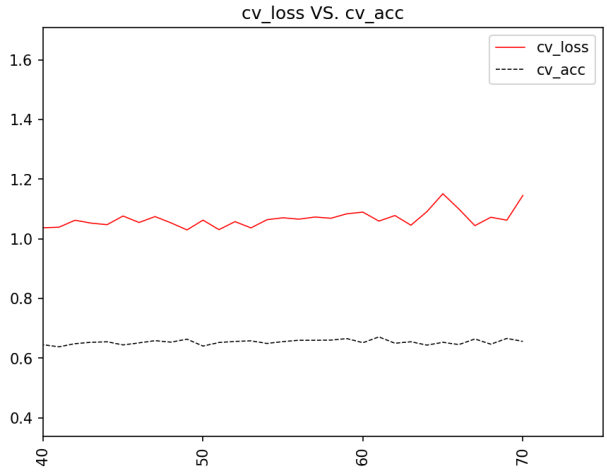
```
{'train_losses': 0.7334156341850757,
 'train_acc': 0.73953125,
 'test_losses': 1.0298687088489533,
 'test_acc': 0.66375}
```

Infine uno zoom sull'andamento della loss e accuracy su batches di training e test, durante le ultime 10 epochs. L'andamento sembra andare verso uno stallo. È stata lanciato un training per ulteriori 50 epochs, per capire se l'andamento possa o meno migliorare ancora o se si è giunti all'asintoto.





La discesa continua per le metrics di Train, ma sono praticamente all'asintoto per quanto riguarda i valori di cross_validation. Addirittura si inizia ad evincere un andamento positivo nella variazione della accuracy del set di cross_validation ==> il modello sta iniziando a mostrare segni di OVERFITTING.



Dopo ulteriori 21 epochs siamo al seguente risultato:

```
{'train_losses': 0.6659356257319451,
 'train_acc': 0.7590625,
 'test_losses': 1.0947986608743667,
 'test_acc': 0.648125}
```

Il modello relativo a queste prove è salvato in:

```
/Users/thomaspierantozzi/PycharmProjects/PyTorch_Train/05_PyTorch_Food101/saved_models/
model0_VGG16_1e-05_32
```

24.03.2025 - pt.2

Nel nuovo test proviamo a modificare il transform di importazione. Qui sotto il setting di partenza

```
transform = nn.Sequential(
    v2.ToImage(),
    v2.ToDtype(torch.float32),
    v2.Resize(RESIZE),
    v2.randomhorizontalflip(p=0.5),
    v2.randomverticalflip(p=0.5),
    v2.randomrotation(degrees=30),
    v2.randomperspective(p=0.5, distortion_scale=0.2),
)

transform_test = nn.Sequential(
    v2.ToImage(),
```

```
    v2.ToDtype(torch.float32, scale=True),
    v2.Resize(RESIZE),
)
```

Di seguito il nuovo setting:

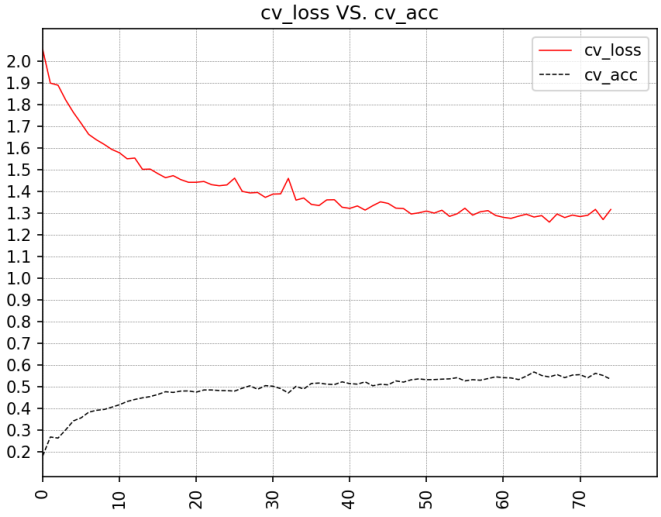
```
transform = nn.Sequential(
    v2.ToImage(),
    v2.ToDtype(torch.float32, scale=True),
    v2.Resize(RESIZE),
    v2.TrivialAugmentWide(31)
)

transform_test = nn.Sequential(
    v2.ToImage(),
    v2.ToDtype(torch.float32, scale=True),
    v2.Resize(RESIZE),
)
```

Proviamo sia ad inserire una tecnica di data augmentation diversa, sia ad avere dei valori TUTTI fra 0 e 1, in quanto il parametro `v2.ToDtype(torch.float32, scale=True)` serve proprio a convertire i dati importati all'interno del range di valori atteso per il dtype stesso; di seguito una indicazione:

- torch.uint8 ==> valori fra 0 e 255
- Torch.float32 ==> valori fra 0 e 1

Il risultato del training fino a 75 epochs è il seguente:



Il training è piuttosto stabile in discesa sia in termini di training che di CV, anche se l'accuracy di CV inizia ad essere asintotica attorno alla 75esima epoch (ca.55% di acc. CV).

Qui sotto si riporta l'outcome del modello fino alla 75 epoch:

```
{'train_losses': 1.0087706530094147,
 'train_acc': 0.6609375,
 'test_losses': 1.3170705097913742,
 'test_acc': 0.53375}
```

Il training continua fino alla 95esima epoch, quando si registrano i seguenti dati e, non vedendo grossi miglioramenti della accuracy di CV, si decide di interrompere il training. L'overfitting e la bassa accuracy di training potrebbero suggerire che il dataset di addestramento sia troppo scarno.

```
Epoch number: 98
       Train loss: 0.898 | Train Accuracy: 70.578%
       CV loss:    1.330 | CV Accuracy:    56.063%
       Time taken: 85.35 seconds
```

Proviamo ad aumentare il numero di classi a 50.

25.03.2025

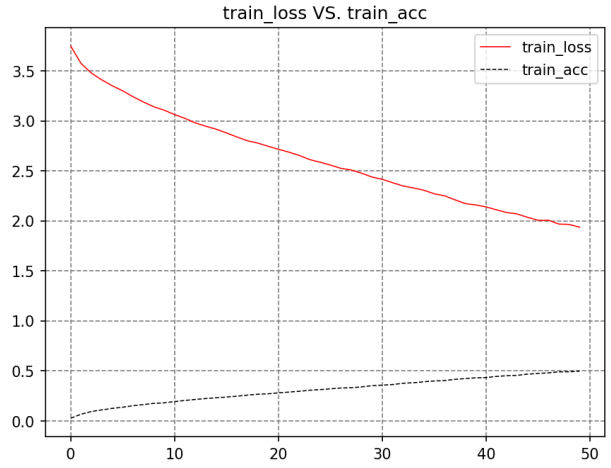
La prova su 50 classi non ha avuto esiti positivi:

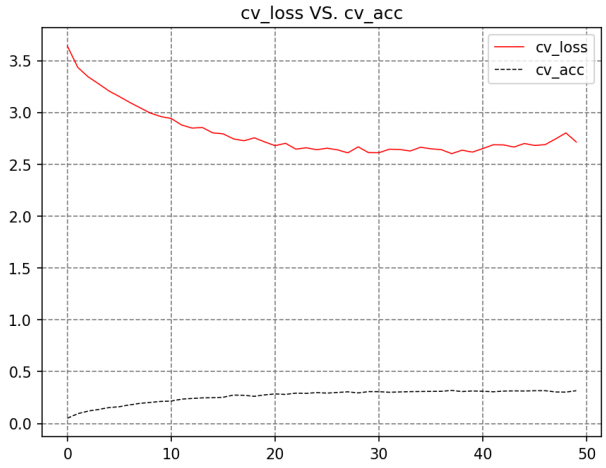
Il modello è il medesimo del punto di cui sopra.

LEARNING RATE = 1e-5

BATCH_SIZE = 32

L'history delle metrics è quella riportata di sotto, con una pessima performance non solo sul CV ma anche sul dataset di training. L'overfitting è evidente, nonché la scarsa accuracy raggiunta dopo 50 iterazione. Seppur il training sembri avere margini di miglioramento ampissimi, l'accuracy e la loss di CV stanno rispettivamente stallando e risalendo.





I valori dell'outcome non vengono riportati per questo training chiaramente fallimentare.

Debug necessario.

Modello salvato in:

```
/Users/thomaspierantozzi/PycharmProjects/PyTorch_Train/05_PyTorch_Food101/saved_models/
03_model0_VGG16_1e-05_32_50Classes
```

26.03.2025

Si nota un errore nell'architettura della rete VGG16 finora usata. In realtà mancava un intero blocco di convoluzione (4 layers in totale).

Ho rispettato anche i valori di padding e striding (1 per entrambi). La CNN adesso è come di seguito. Usando un batch_size di 64 (per velocizzare nei limiti dell'hardware a disposizione) il modella ha un peso gestibile.

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
Model0                                   [64, 8]                   --
├─Sequential: 1-1                        [64, 16, 128, 128]        --
```

```
│    └─Conv2d: 2-1                       [64, 16, 256, 256]        448
│    └─Conv2d: 2-2                       [64, 16, 256, 256]        2,320
│    └─ReLU: 2-3                         [64, 16, 256, 256]        --
│    └─MaxPool2d: 2-4                    [64, 16, 128, 128]        --
├─Sequential: 1-2                        [64, 32, 64, 64]          --
│    └─Conv2d: 2-5                       [64, 32, 128, 128]        4,640
│    └─Conv2d: 2-6                       [64, 32, 128, 128]        9,248
│    └─ReLU: 2-7                         [64, 32, 128, 128]        --
│    └─MaxPool2d: 2-8                    [64, 32, 64, 64]          --
├─Sequential: 1-3                        [64, 64, 32, 32]          --
│    └─Conv2d: 2-9                       [64, 64, 64, 64]          18,496
│    └─Conv2d: 2-10                      [64, 64, 64, 64]          36,928
│    └─Conv2d: 2-11                      [64, 64, 64, 64]          36,928
│    └─ReLU: 2-12                        [64, 64, 64, 64]          --
│    └─MaxPool2d: 2-13                   [64, 64, 32, 32]          --
├─Sequential: 1-4                        [64, 128, 16, 16]         --
│    └─Conv2d: 2-14                      [64, 128, 32, 32]         73,856
│    └─Conv2d: 2-15                      [64, 128, 32, 32]         147,584
│    └─Conv2d: 2-16                      [64, 128, 32, 32]         147,584
│    └─ReLU: 2-17                        [64, 128, 32, 32]         --
│    └─MaxPool2d: 2-18                   [64, 128, 16, 16]         --
├─Sequential: 1-5                        [64, 256, 8, 8]           --
│    └─Conv2d: 2-19                      [64, 256, 16, 16]         295,168
│    └─Conv2d: 2-20                      [64, 256, 16, 16]         590,080
│    └─Conv2d: 2-21                      [64, 256, 16, 16]         590,080
│    └─ReLU: 2-22                        [64, 256, 16, 16]         --
│    └─MaxPool2d: 2-23                   [64, 256, 8, 8]           --
├─Sequential: 1-6                        [64, 8]                   --
│    └─Flatten: 2-24                     [64, 16384]               --
│    └─Linear: 2-25                      [64, 16384]               268,451,840
│    └─Linear: 2-26                      [64, 16384]               268,451,840
│    └─Linear: 2-27                      [64, 8]                   131,080
==========================================================================================
Total params: 538,988,120
Trainable params: 538,988,120
Non-trainable params: 0
Total mult-adds (Units.GIGABYTES): 133.11
==========================================================================================
Input size (MB): 50.33
Forward/backward pass size (MB): 2332.04
Params size (MB): 2155.95
Estimated Total Size (MB): 4538.32
==========================================================================================
```

LEARNING RATE = differenti e inclusi fra 1e-5 e 3e0

BATCH_SIZE = 64

Dopo diversi tentativi di tuning del learning rate e con differenti numeri di classi da apprendere (per avere un maggior numero di esempi in ingresso), comunque la rete non riesce a migliorare la loss iniziale. Sembra impossibile apprendere un segnale da un cos' eseguo numero di esempi, seppur aumentati dalla DataAugmentation.

L'argomento andrebbe approfondito cercando in qualche paper.

27.03.2025

Sotto si riporta l'architettura di un modello semplificativo rispetto al VGG con quattro blocchi convoluzionali.

Testando il modello su 3, 8, 25 e 50 classi, si ottiene sempre lo stesso risultato: loss costante e accuracy attorno alla baseline del 'random guessing'. Il modello è ancora troppo complesso.

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
Model0                                   [32, 3]                   --
├─Sequential: 1-1                        [32, 64, 64, 64]          --
│    └─Conv2d: 2-1                       [32, 64, 128, 128]        1,792
│    └─Conv2d: 2-2                       [32, 64, 128, 128]        36,928
│    └─ReLU: 2-3                         [32, 64, 128, 128]        --
│    └─MaxPool2d: 2-4                    [32, 64, 64, 64]          --
├─Sequential: 1-2                        [32, 128, 32, 32]         --
│    └─Conv2d: 2-5                       [32, 128, 64, 64]         73,856
│    └─Conv2d: 2-6                       [32, 128, 64, 64]         147,584
│    └─ReLU: 2-7                         [32, 128, 64, 64]         --
│    └─MaxPool2d: 2-8                    [32, 128, 32, 32]         --
├─Sequential: 1-3                        [32, 256, 16, 16]         --
│    └─Conv2d: 2-9                       [32, 256, 32, 32]         295,168
│    └─Conv2d: 2-10                      [32, 256, 32, 32]         590,080
│    └─Conv2d: 2-11                      [32, 256, 32, 32]         590,080
│    └─ReLU: 2-12                        [32, 256, 32, 32]         --
│    └─MaxPool2d: 2-13                   [32, 256, 16, 16]         --
├─Sequential: 1-4                        [32, 512, 8, 8]           --
│    └─Conv2d: 2-14                      [32, 512, 16, 16]         1,180,160
│    └─Conv2d: 2-15                      [32, 512, 16, 16]         2,359,808
│    └─Conv2d: 2-16                      [32, 512, 16, 16]         2,359,808
│    └─ReLU: 2-17                        [32, 512, 16, 16]         --
│    └─MaxPool2d: 2-18                   [32, 512, 8, 8]           --
├─Sequential: 1-5                        [32, 3]                   --
│    └─Flatten: 2-19                     [32, 32768]               --
│    └─Linear: 2-20                      [32, 256]                 8,388,864
│    └─Linear: 2-21                      [32, 128]                 32,896
│    └─Linear: 2-22                      [32, 3]                   387
==========================================================================================
Total params: 16,057,411
Trainable params: 16,057,411
Non-trainable params: 0
Total mult-adds (Units.GIGABYTES): 146.27
==========================================================================================
Input size (MB): 6.29
Forward/backward pass size (MB): 1107.40
Params size (MB): 64.23
Estimated Total Size (MB): 1177.92
==========================================================================================
```

02.04.2025

Dopo giorni di test sulla rete VGG16, si è deciso di rendere il modello molto più semplice. L'obiettivo è quello di rendere in qualche modo comparabili la grandezza del modello e quella del dataset: a quanto pare un modello estremamente complesso e con un numero elevato di parametri, è difficile da lanciare in apprendimento da zero, su un dataset così esiguo come quello rappresentato dal Food101.

Si riparte quindi dal TinyVGG suggerito dal corso di Daniel Bourke (Udemy) su PyTorch, e che si basa sul modello del sito:

`https://poloclub.github.io/cnn-explainer/`

Il modello di partenza è quello riportato sotto:

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
```

```
Model0                                   [16, 5]                   --
├─Sequential: 1-1                        [16, 10, 112, 112]        --
│    └─Conv2d: 2-1                       [16, 10, 224, 224]        280
│    └─Conv2d: 2-2                       [16, 10, 224, 224]        910
│    └─ReLU: 2-3                         [16, 10, 224, 224]        --
│    └─MaxPool2d: 2-4                    [16, 10, 112, 112]        --
├─Sequential: 1-2                        [16, 10, 56, 56]          --
│    └─Conv2d: 2-5                       [16, 10, 112, 112]        910
│    └─Conv2d: 2-6                       [16, 10, 112, 112]        910
│    └─ReLU: 2-7                         [16, 10, 112, 112]        --
│    └─MaxPool2d: 2-8                    [16, 10, 56, 56]          --
├─Sequential: 1-3                        [16, 5]                   --
│    └─Flatten: 2-9                      [16, 31360]               --
│    └─Linear: 2-10                      [16, 5]                   156,805
==========================================================================================
Total params: 159,815
Trainable params: 159,815
Non-trainable params: 0
Total mult-adds (Units.GIGABYTES): 1.32
==========================================================================================
Input size (MB): 9.63
Forward/backward pass size (MB): 160.56
Params size (MB): 0.64
Estimated Total Size (MB): 170.84
==========================================================================================
```

Per i primi due modelli riportati sotto il modello è identico.

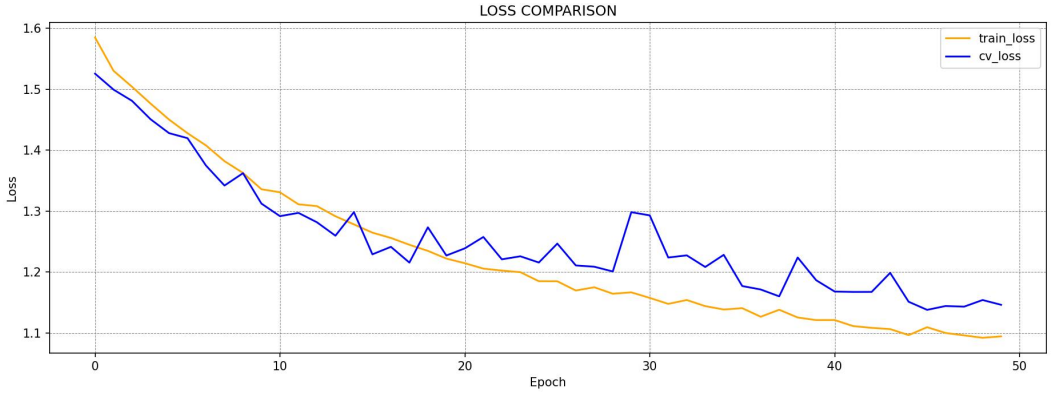
La rev02 della TinyVGG utilizzata è invece il seguente:

Tests Summary

	Train loss	Train Acc	CV loss	CV Acc	Epoch Train time	Epochs trained	Delta loss train / CV	Comment
01_TinyVGG16_TrivAug_0.001_32_Classes5	1,415	40,875%	1,394	42,383%	12s	50	0,021	Training got stuck and the parameters were not improving over the last epochs
02_TinyVGG16_TrivAug_0.01_32_Classes5	0,809	71,575%	1,354	49,609%	12s 500ms	50	0,545	
03_TinyVGG16rev02_TrivAug_0.005_32_Classes5	0,886	68,375%	1,182	53,32%	11s 500ms	50	0,296	
04_TinyVGGrev02_CustomDataAug.p50_0.005_32_Classes5	0,832	69,1%	0,959	62,598%	15s 500ms	100	0,127	
05_TinyVGGrev03_CustomDataAug_0.005_32_Classes5	1,112	57,15%	1,212	51,66%	15s 500ms	100	0,1	
06_TinyVGGrev02-b_CustomDataAug_0.005_32_Classes5	1,095	58,05%	1,146	54,883%	20s	50	0,051	Overfitting signs showed during the last epochs
07_TinyVGGrev02_CustomDataAug.p25_0.005_32_Classes5	0,769	71,25%	0,886	67,676%	13s 500ms	150	0,117	
08_TinyVGGrev02_CustomDataAug.p25_0.005_128_Classes5	0,935	64,941%	0,995	62,387%	15s 200ms	150	0,06	
09_TinyVGGrev02_CustomDataAug.p25_0.005_32_Classes5defined	0,84	68,675%	0,973	66,406%	15s	150	0,133	Uguale a modello 07 ma con trainig classes più facili e selezionate ad hoc

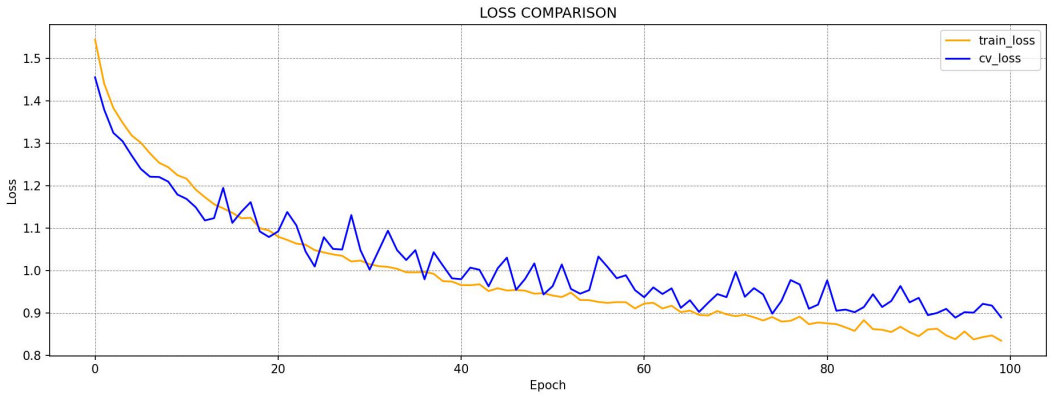
La generalizzazione del modello di cui sotto sembra la migliore, ma solo dopo 50 epochs il training è stato interrotto perché si rilevavano derive di overfitting

`06_TinyVGGrev02-b_CustomDataAug_0.005_32_Classes5`



Si è scelto di andare col modello nr.07 (riportato sotto) in quanto dopo 150 epochs è ancora piuttosto abile nel generalizzare, e arriva a una loss di CV che è la migliore fra quelle raggiunte sinora.

`07_TinyVGGrev02_CustomDataAug.p25_0.005_32_Classes5`



Una nota sul modello 09: è stato scelto un set di immagini ritenute ben distinguibili e con poca possibilità di essere confuse, ed è stato ripetuto il training del modello 07 con parametri identici. Il risultato è stato di ottenere un modello che performa in maniera simile a quello 07, ma che davanti a un test set scaricato da google immagini in effetti performa meglio con una accuracy che sfiora il 68%.