



Università degli Studi di Catania
Dipartimento di Ingegneria Elettrica Elettronica e Informatica
Corso di Laurea Magistrale in Ingegneria Informatica

COGNITIVE COMPUTING AND ARTIFICIAL INTELLIGENCE

Presentazione progetto finale

Studenti:

Alessandro Messina (O55000354)

Marco Pisasale (O55000348)

Anno Accademico 2018/2019



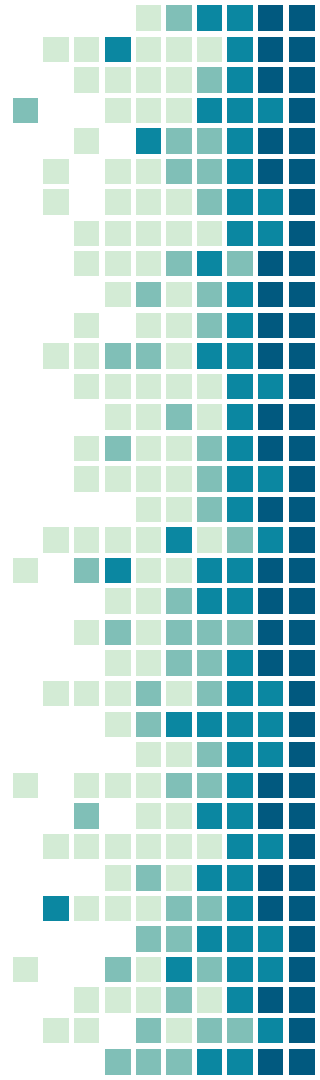
Introduzione

Chapter 1



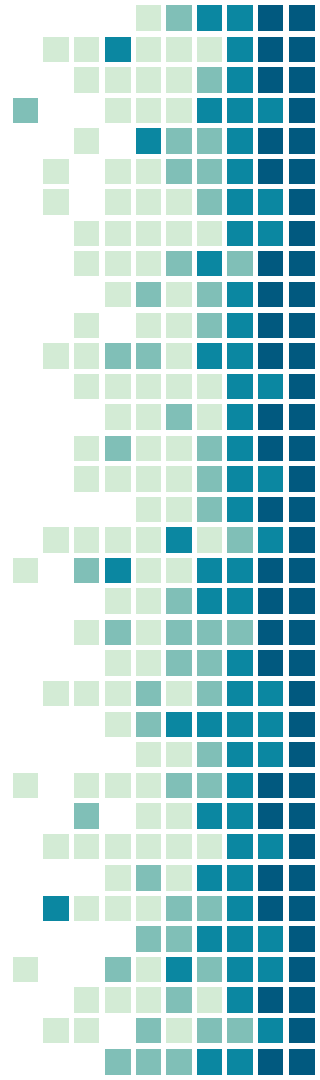
Obiettivi del progetto

- 1) Allenare il modello CycleGAN sul dataset di volti di persone bianche e nere.
- 2) Download del dataset di action recognition UCF-101.
- 3) Split del dataset UCF-101 in Utrain, Uval, Utest.
- 4) Allenare un classificatore sui volti.
- 5) Utilizzare un modello di face detection sul dataset UCF-101 e generare una nuova versione di Utrain (Utrain-inv) mediante l'utilizzo del classificatore e della CycleGAN.



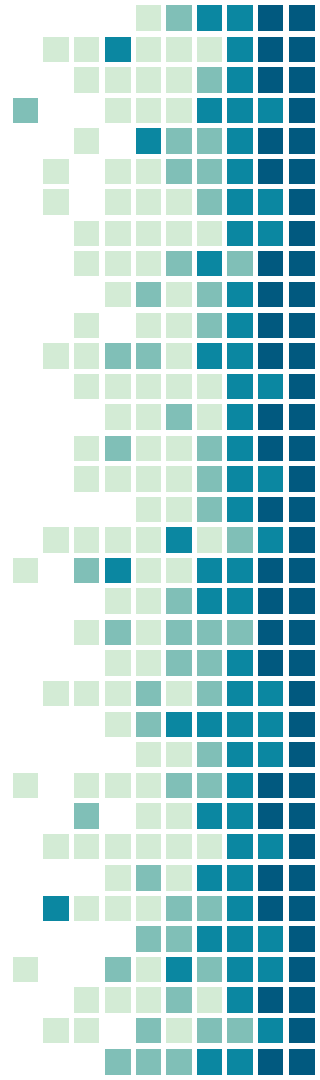
Obiettivi del progetto (continuo)

- 6) Allenare il classificatore video 3D-ResNet su Utrain, e verificare le prestazioni su Utest.
- 7) Allenare il classificatore video 3D-ResNet su Utrain-inv, e verificare le prestazioni su Utest.
- 8) Allenare il classificatore video 3D-ResNet sull'unione tra Utrain e Utrain-inv, e verificare le prestazioni su Utest.




Repository GitHub




- Il progetto realizzato è disponibile al seguente link:
<https://github.com/Taletex/AI-Projectwork>
- La repository contiene tutto il codice sorgente prodotto e utilizzato, la documentazione completa e vari screenshot, che evidenziano i progressi effettuati.



Repository GitHub



[Pull requests](#) [Issues](#) [Marketplace](#) [Explore](#)

[Taletex / AI-Projectwork](#)


[Watch](#) 0 [Star](#) 0 [Fork](#) 0

[Code](#) [Issues 0](#) [Pull requests 0](#) [Projects 0](#) [Wiki](#) [Security](#) [Insights](#)

Project work of Cognitive Computing and Artificial Intelligence course of University of Catania.

[32 commits](#) [1 branch](#) [0 releases](#) [1 contributor](#) [GPL-3.0](#)

[Branch: master](#) [New pull request](#) [Create new file](#) [Upload files](#) [Find File](#) [Clone or download](#)

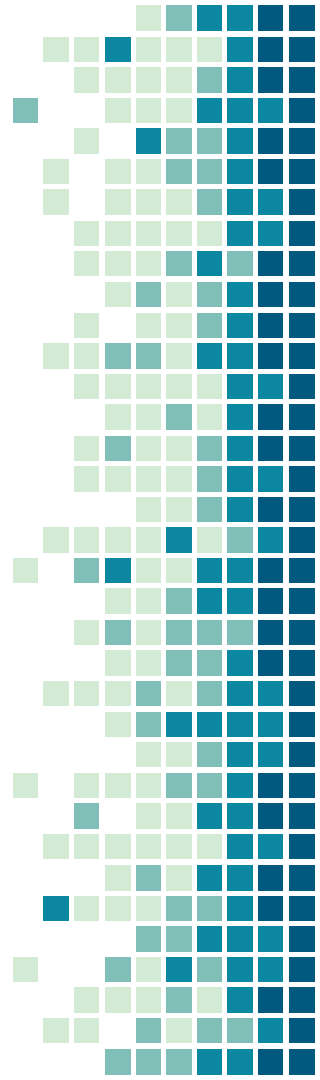
 **Taletex** Update .gitignore

Latest commit 11d467d 27 minutes ago

doc	Added documentation	2 days ago
progress	Updated scripts and progress	29 minutes ago
src	Updated scripts and progress	29 minutes ago
.gitignore	Update .gitignore	27 minutes ago
LICENSE	Initial commit	last month
README.md	Update README.md	last month

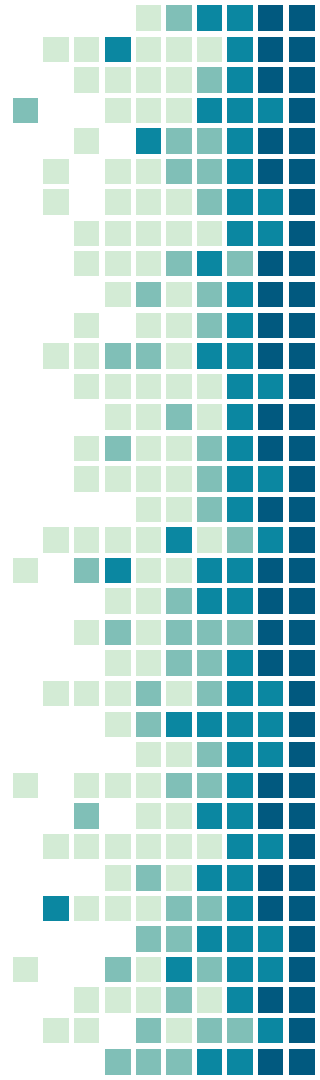
Contenuti repository GitHub

- **src:** codice sorgente prodotto e utilizzato, comprensivo degli script creati per la manipolazione dei dati generati nelle varie fasi del progetto.
- **doc:** cartella contenente la documentazione del progetto.
- **progress:** cartella contenente screenshot e file vari, relativi ai risultati ottenuti nelle varie fasi del progetto.



Configurazione Hardware utilizzata

- CPU: AMD Ryzen 5 2600x
- SCHEDA VIDEO: Gygabyte Geforce RTX 2070 Windforce
- RAM: 32GB ddr4 3200MHz
- SSD: Crucial MX500 500GB



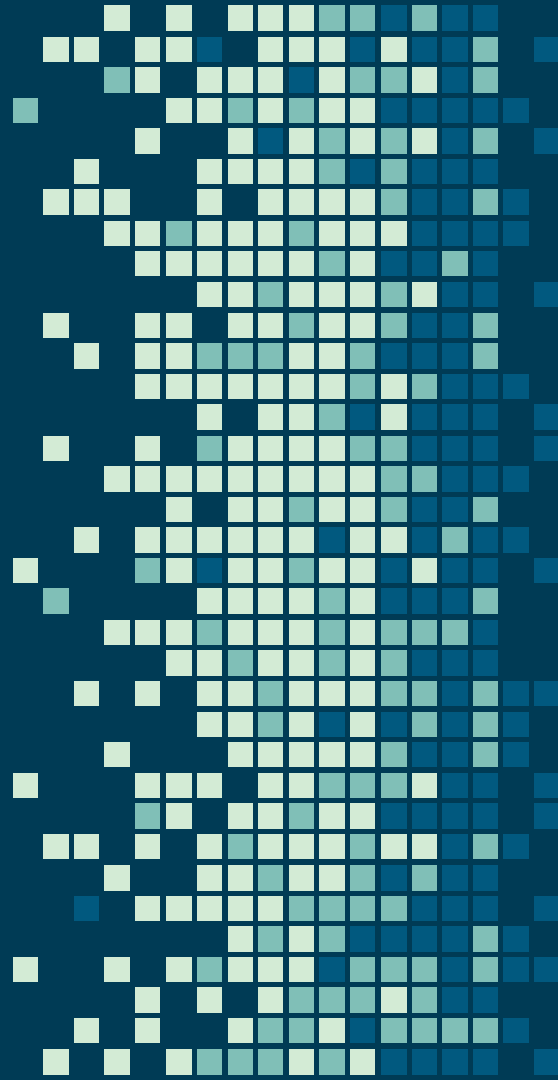
CycleGAN training

Chapter 2



“ Our goal is to learn a mapping $G : X \rightarrow Y$ such that the distribution of images from $G(X)$ is indistinguishable from the distribution Y using an adversarial loss. Because this mapping is highly under-constrained, we couple it with an inverse mapping $F : Y \rightarrow X$ and introduce a cycle consistency loss to enforce $F(G(X)) \approx X$ (and viceversa).

~ Abstract, CycleGAN paper



Funzionamento

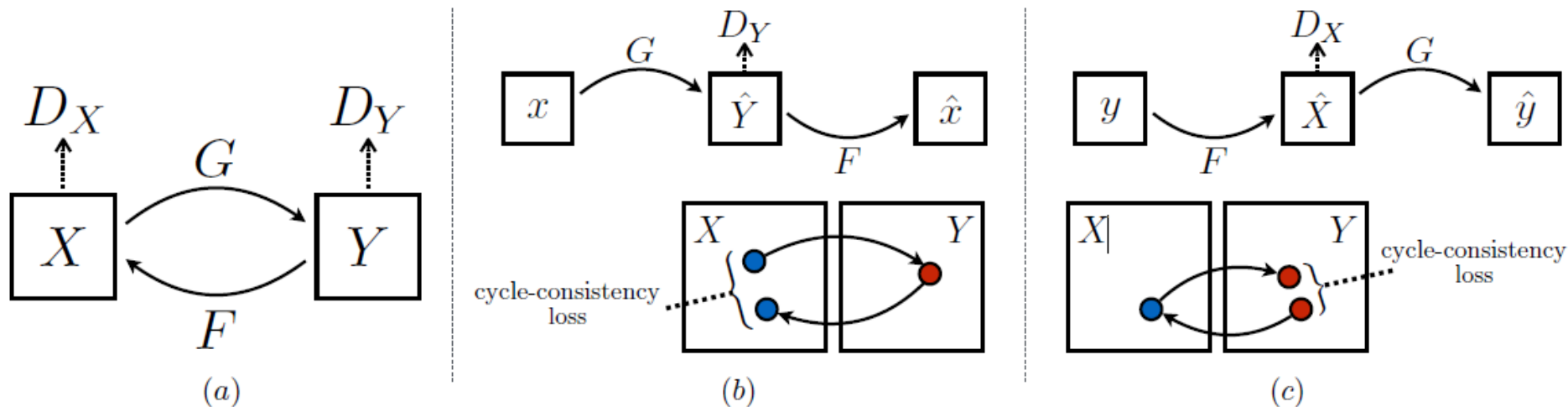
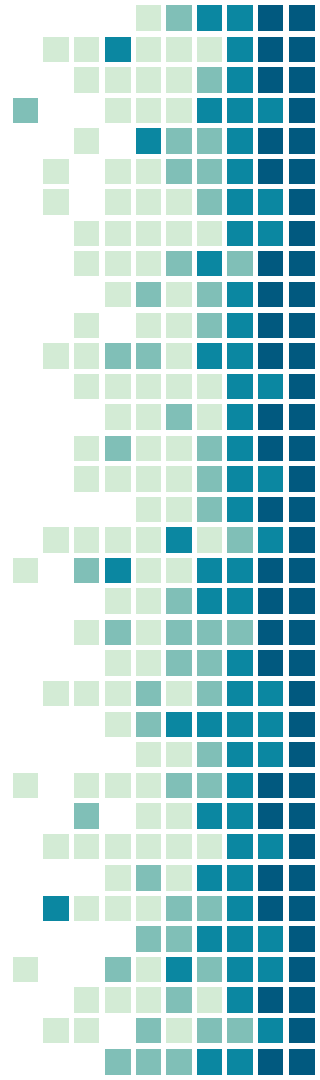


Figure 3: (a) Our model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators D_Y and D_X . D_Y encourages G to translate X into outputs indistinguishable from domain Y , and vice versa for D_X and F . To further regularize the mappings, we introduce two *cycle consistency losses* that capture the intuition that if we translate from one domain to the other and back again we should arrive at where we started: (b) forward cycle-consistency loss: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$, and (c) backward cycle-consistency loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$

Utilizzo

- Allenando la CycleGAN sul dataset di volti di persone bianche e nere abbiamo così generato due modelli, uno per trasformare volti bianchi in volti neri, e uno che applica la trasformazione opposta.
- Entrambi i modelli allenati in questa fase verranno utilizzati successivamente, sul dataset UCF-101 (in particolare su Utrain), per la generazione del dataset inverso (Utrain-inv).



Performance metrics

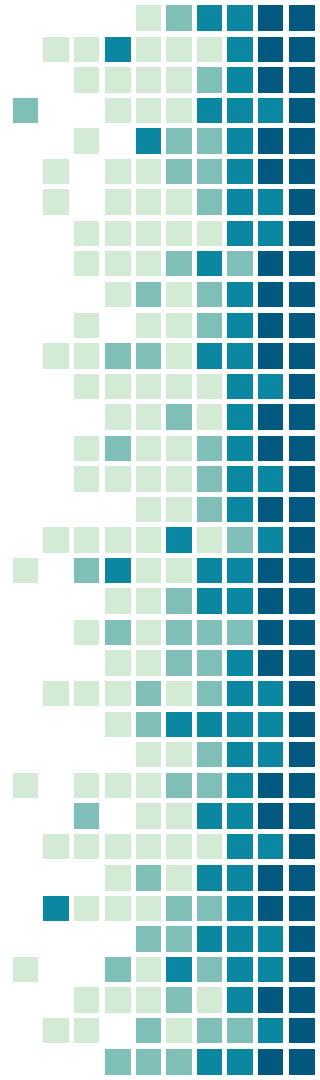
Trattandosi di modelli generativi, non sono disponibili performance metrics come l'accuracy, in quanto non stiamo lavorando con training data del tipo (X,Y) .

La rete fornisce l'andamento delle seguenti loss:

- Adversarial Loss della mapping function $G : X \rightarrow Y$ (trasformazione diretta) espressa come:

$$\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{\text{data}}(y)} [\log D_Y(y)] \\ + \mathbb{E}_{x \sim p_{\text{data}}(x)} [\log(1 - D_Y(G(x)))],$$

formula analoga per la mapping function $F : Y \rightarrow X$ (trasformazione inversa).



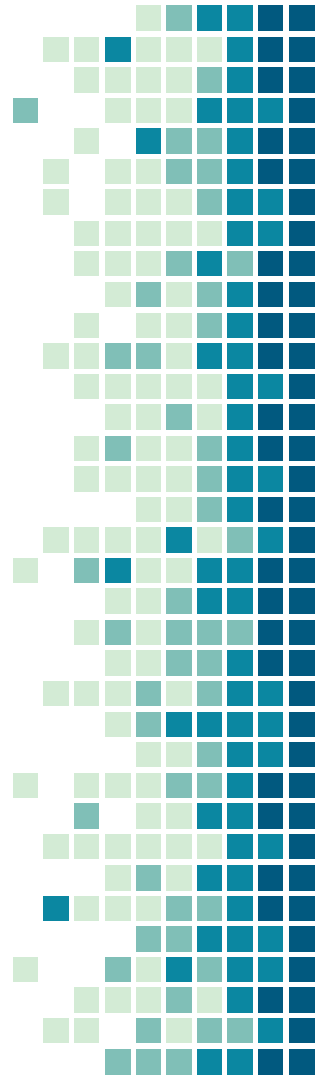
Performance metrics (continuo)

- Cycle consistency loss, sia forward ($x \rightarrow G(x) \rightarrow F(G(x)) \approx x$), che backward ($y \rightarrow F(y) \rightarrow G(F(y)) \approx y$), valutata come:

$$\begin{aligned}\mathcal{L}_{\text{cyc}}(G, F) = & \mathbb{E}_{x \sim p_{\text{data}}(x)} [\|F(G(x)) - x\|_1] \\ & + \mathbb{E}_{y \sim p_{\text{data}}(y)} [\|G(F(y)) - y\|_1].\end{aligned}$$

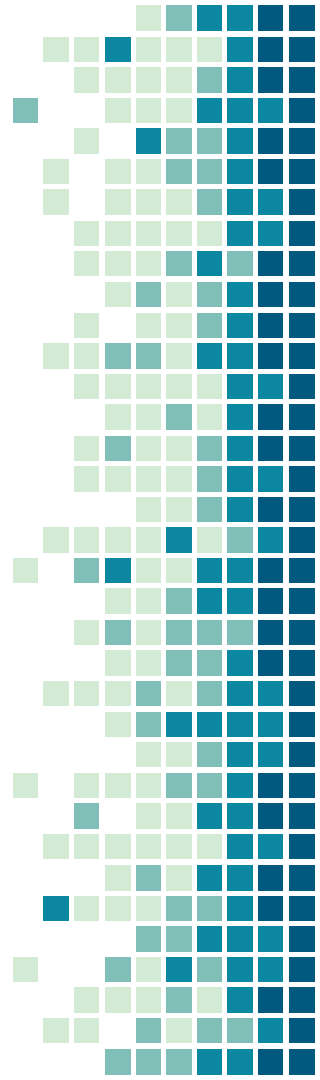
- Di conseguenza la full Objective function risulta essere:

$$\begin{aligned}\mathcal{L}(G, F, D_X, D_Y) = & \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) \\ & + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) \\ & + \lambda \mathcal{L}_{\text{cyc}}(G, F),\end{aligned}$$



CycleGAN Input/Outputs

- **Real_{A, B}**: immagine originale, appartenente alla classe A o B, utilizzata come input del modello.
- **Fake_{B, A}**: immagine generata a partire dall'originale, applicando la rispettiva trasformazione ($A \rightarrow B$ o $B \rightarrow A$).
- **Rec_{A, B}**: immagine ricostruita a partire dall'immagine fake, applicando la trasformazione inversa.
- **Idt_{B, A}**: immagine aggiuntiva, generata a partire dall'originale applicando la prima trasformazione, nella quale si introduce una loss addizionale (identity mapping loss) per regolarizzare il generatore in modo tale da mantenere la *color composition* originale. Utile in alcuni contesti applicativi (e.g. trasformazioni *painting* \rightarrow *photo*).



Esempi di trasformazione (1)



Real_A

Fake_B

Rec_A

Idt_B



Real_B

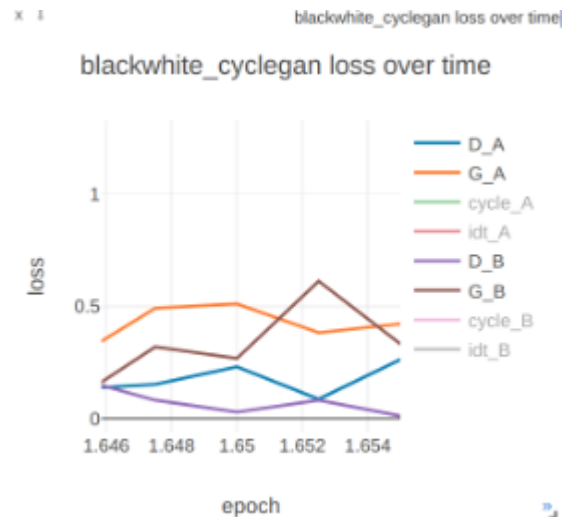
Fake_A

Rec_B

Idt_A

Classes:

- A = black
- B = white



Esempi di trasformazione (2)



Real_A

Fake_B

Rec_A

Idt_B



Real_B

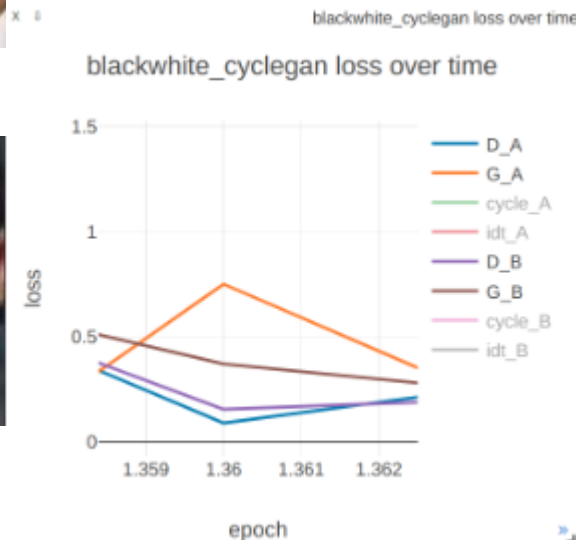
Fake_A

Rec_B

Idt_A

Classes:

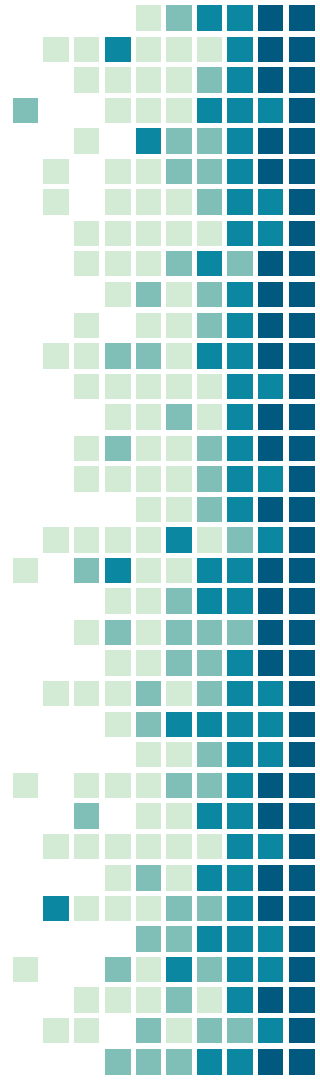
- A = black
- B = white



Analisi qualitativa dei risultati ottenuti

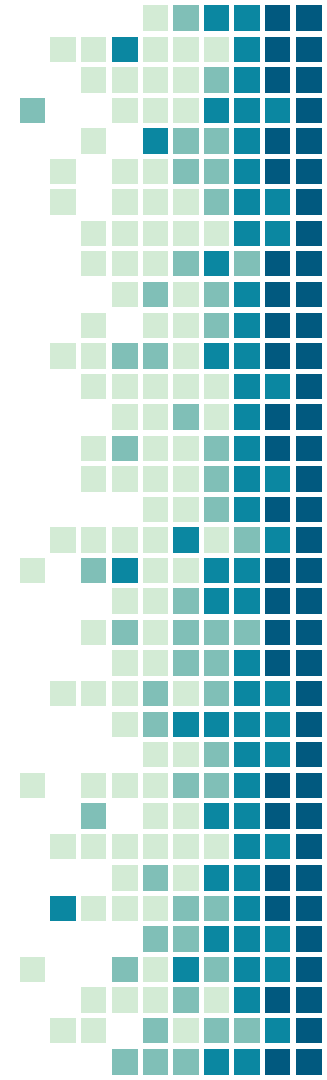
La rete fornisce come output l'andamento delle loss precedentemente viste, che mostrano le seguenti proprietà:

- Loss decrescenti col progredire delle epoche, evidenziando una migliore capacità della rete di effettuare le trasformazioni.
- Loss dei discriminatori di entrambe le classi tipicamente inferiori rispetto alle rispettive dei generatori, in linea col principio del maggiore potere discriminativo di una rete, rispetto a quello generativo, a parità di condizioni.
- Cycle consistency loss tipicamente più alte rispetto alle altre, ad evidenziare le maggior difficoltà della rete nell'effettuare entrambe le trasformazioni per ricostruire l'input originale.



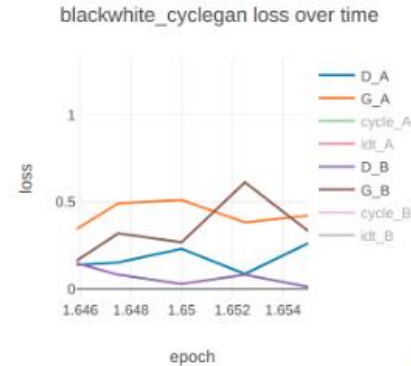
Note sui risultati

- Per questioni legate alla complessità computazionale di esecuzione della CycleGAN (e quindi di tempo) è stato possibile allenarla per un numero limitato di epoche, il che ha sicuramente influenzato negativamente l'accuratezza delle trasformazioni effettuate dai due modelli e quindi i risultati ottenuti, anche nelle fasi successive del progetto.
- Risultati migliori, in termini di efficacia della trasformazione applicata, sarebbero sicuramente stati possibili avendo avuto la possibilità di allenare la rete per un numero di epoche maggiore.



Analisi qualitativa degli errori

- In alcuni casi l'output risulta molto simile all'input, evidenziando una trasformazione insufficiente. Esempio:



- Ciò nonostante, le trasformazioni risultano comunque visivamente apprezzabili nella stragrande maggioranza dei casi, come da esempi precedentemente mostrati.

Face Classifier training

Chapter 3

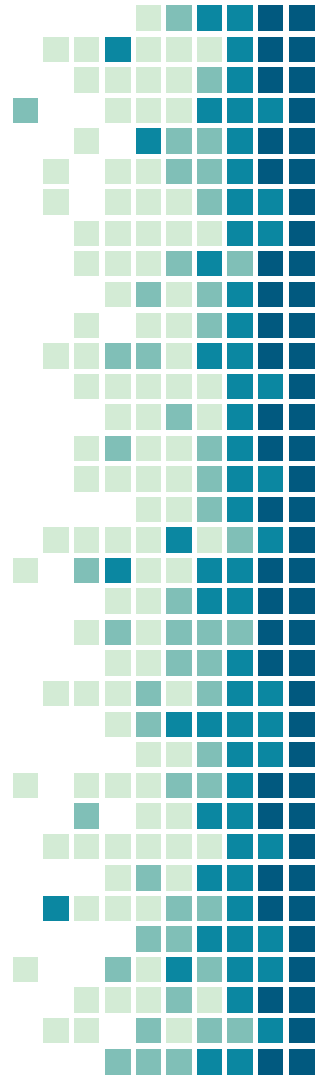


Modello utilizzato

Per effettuare la face classification sui volti (dataset black_white) è stata utilizzata una CNN costituita da:

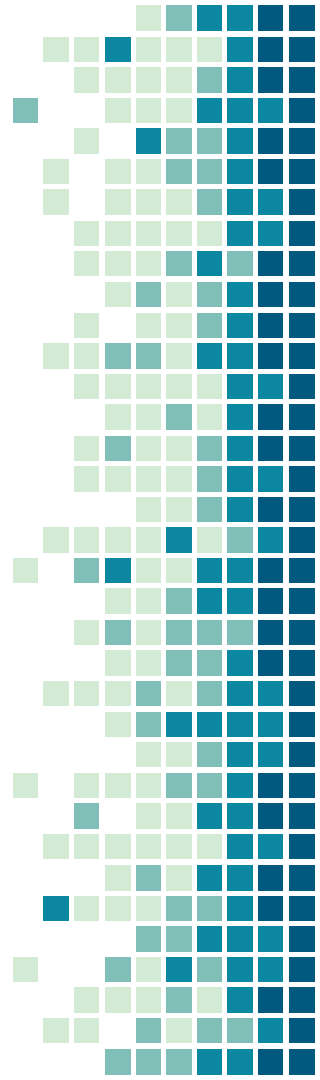
- 4 layer convoluzionali;
- 2 layer di pooling;
- 2 layer lineari.

La rete restituisce come output la classe di appartenenza (black o white) del volto analizzato.



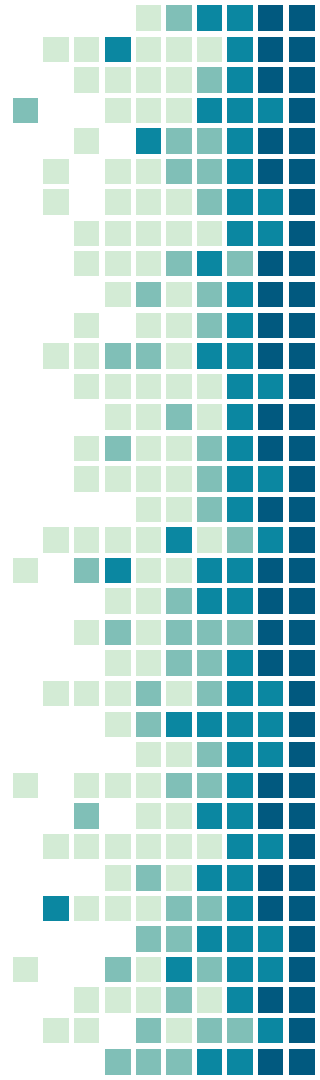
Architettura CNN

```
CNN(  
  (conv_layers): Sequential(  
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1))  
    (1): ReLU()  
    (2): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))  
    (3): ReLU()  
    (4): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1))  
    (5): ReLU()  
    (6): AvgPool2d(kernel_size=2, stride=2, padding=0)  
    (7): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1))  
    (8): ReLU()  
    (9): AvgPool2d(kernel_size=2, stride=2, padding=0)  
  )  
  (fc_layers): Sequential(  
    (0): Linear(in_features=4096, out_features=1024, bias=True)  
    (1): ReLU()  
    (2): Linear(in_features=1024, out_features=2, bias=True)  
  )  
)
```

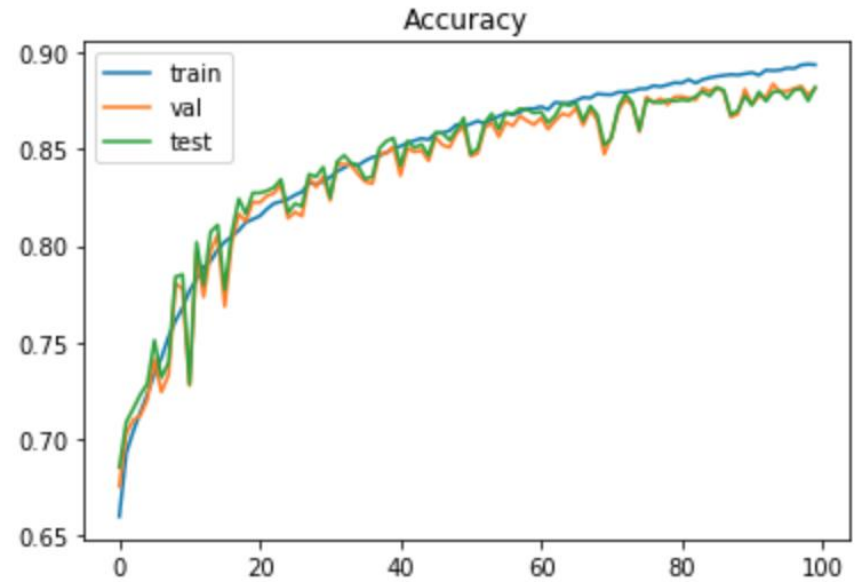
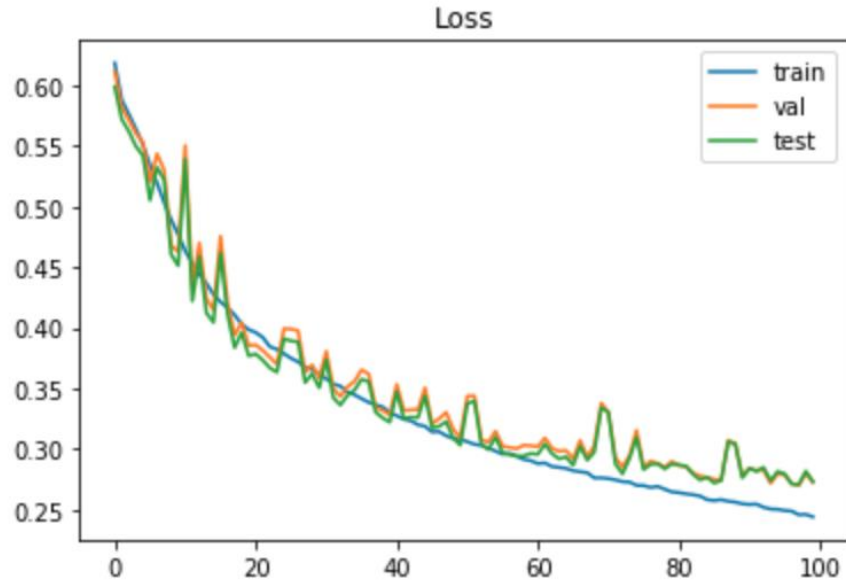


Training e risultati

- La rete è stata allenata sul dataset dei volti (black_white) per un totale di 100 epoche, con un learning rate $lr = 0,01$.
- Le migliori performance si sono registrate all'epoch 89, con una validation accuracy $VA = 90,99\%$, alla quale corrisponde una test accuracy $TeA = 90,82\%$.



Andamenti loss e accuracy



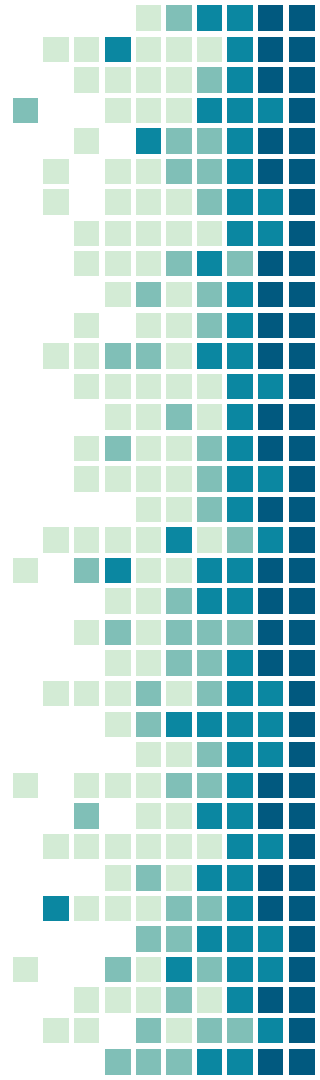
Face Detector & Utrain-inv generation

Chapter 4



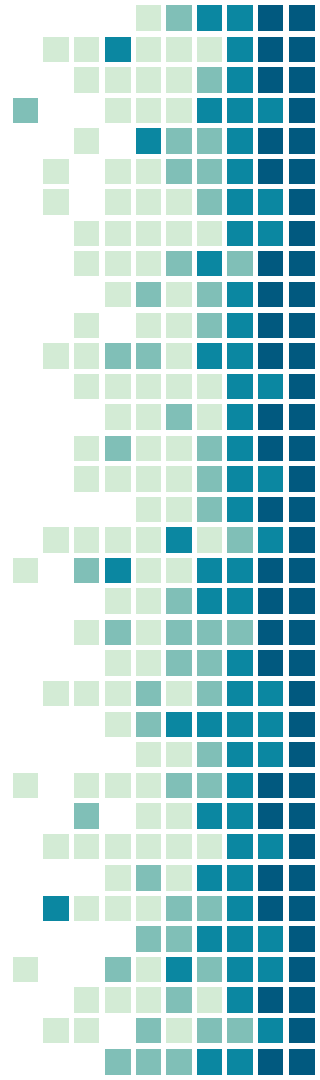
Obiettivi (punto 5)

- Utilizzo del DNN Face Detector in OpenCV per l'identificazione dei volti nei video del dataset UCF-101 (dataset di action recognition).
- Estrazione dei volti identificati dal Face Detector nelle frame dei video e classificazione utilizzando il classificatore (CNN) precedentemente allenato.
- Sulla base della classificazione del volto come bianco o nero, utilizzo del rispettivo modello della CycleGAN per la trasformazione nella classe opposta.



Obiettivi (punto 5) (continuo)

- Post-processing dell'output della CycleGAN: resize e sostituzione del volto originale per ogni frame di ogni video.
- Una volta trasformate tutte le frame del video nelle quali compaiono dei volti, ricostruzione di ogni video a partire dalle frame, utilizzando il software FFmpeg. I video così generati costituiranno il dataset Utrain-inv.



Implementazione

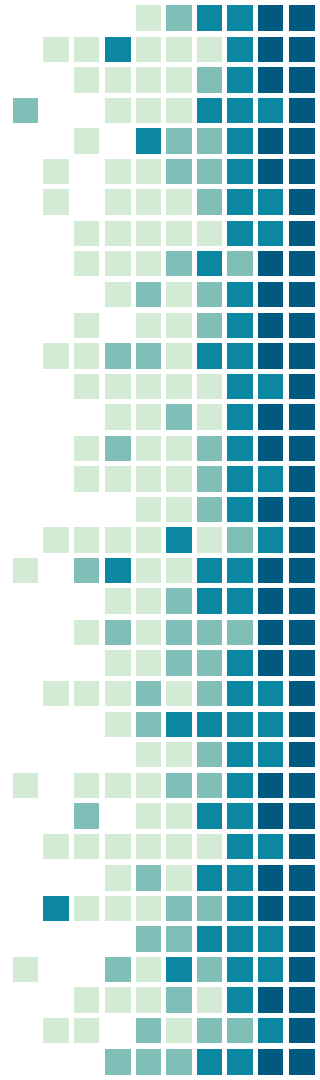
Per il raggiungimento degli obiettivi precedentemente menzionati sono stati integrati i codici del Face Detector, del Face Classifier e della CycleGAN, al fine di effettuare le seguenti operazioni:

- Face detection e salvataggio cropped image (temporanea).
- Processamento dell'immagine temporanea da parte del Face Classifier, per la determinazione della classe di appartenenza.
- Processamento dell'immagine temporanea da parte della CycleGAN e scelta dell'output (img_fake_A o img_fake_B) sulla base dell'output del Face Classifier.
- Rimozione dell'immagine temporanea.



Implementazione (continuo)

- Post-processing dell'output della CycleGAN: resize e sostituzione del volto originale con l'output della CycleGAN per ogni frame di ogni video.
- Salvataggio di ogni frame così processata nella directory corretta.
- Generazione di Utrain-inv: una folder per ogni video di ogni classe.
- Creazione di uno script per la ricostruzione dei video a partire dalle frame generate utilizzando il software FFmpeg.

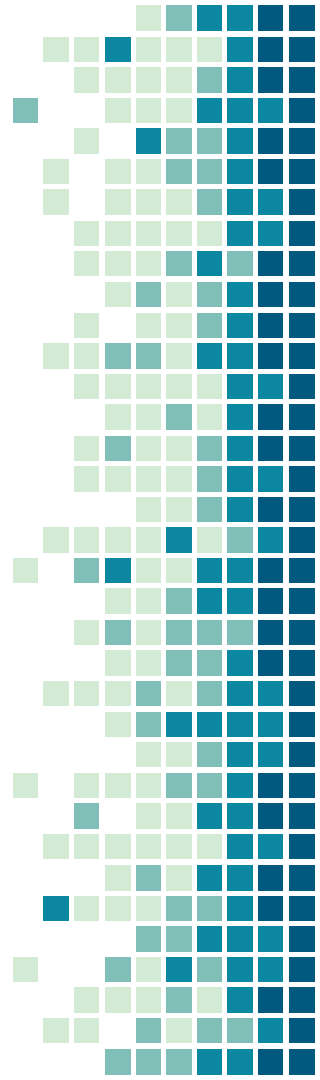


Source code 1: Detection part

```
def detectFaceOpenCVDnn(net, frame, index, frame_count, destDir):
    saveName = destDir + "/img" + str(index) + "_" + str(frame_count) + ".jpg"
    frameOpencvDnn = frame.copy()
    frameHeight = frameOpencvDnn.shape[0]
    frameWidth = frameOpencvDnn.shape[1]
    blob = cv2.dnn.blobFromImage(frameOpencvDnn, 1.0, (300, 300), [104, 117, 123], False, False)

    # convert the image from the current frame
    img = Image.fromarray(cv2.cvtColor(numpy.array(frame), cv2.COLOR_RGB2BGR))

    net.setInput(blob)
    detections = net.forward()
    bboxes = []
    for i in range(detections.shape[2]):
        confidence = detections[0, 0, i, 2]
        if confidence > conf_threshold:
            x1 = int(detections[0, 0, i, 3] * frameWidth)
            y1 = int(detections[0, 0, i, 4] * frameHeight)
            x2 = int(detections[0, 0, i, 5] * frameWidth)
            y2 = int(detections[0, 0, i, 6] * frameHeight)
            bboxes.append([x1, y1, x2, y2])
            cv2.rectangle(frameOpencvDnn, (x1, y1), (x2, y2), (0, 255, 0), int(round(frameHeight/150)), 8)
```



Source code 2: CycleGAN part

```
# crop and save the face image from the current frame
croppedImg = T.functional.crop(img, y1, x1, (y2-y1), (x2-x1))
croppedImg.save("./img.jpg", "JPEG", icc_profile=img.info.get('icc_profile'))
saveName = destDir + "/img" + str(index) + "_" + str(frame_count) + ".jpg"

# save in the CycleGAN test folders
croppedImg.save("../pytorch-CycleGAN-and-pix2pix/datasets/UCF-101/testA/img.jpg", "JPEG", icc_profile=img.info.get(
croppedImg.save("../pytorch-CycleGAN-and-pix2pix/datasets/UCF-101/testB/img.jpg", "JPEG", icc_profile=img.info.get(

# execute the CycleGAN
test_custom.executeCyclegan()

# delete the saved images
os.remove("../pytorch-CycleGAN-and-pix2pix/datasets/UCF-101/testA/img.jpg")
os.remove("../pytorch-CycleGAN-and-pix2pix/datasets/UCF-101/testB/img.jpg")
```



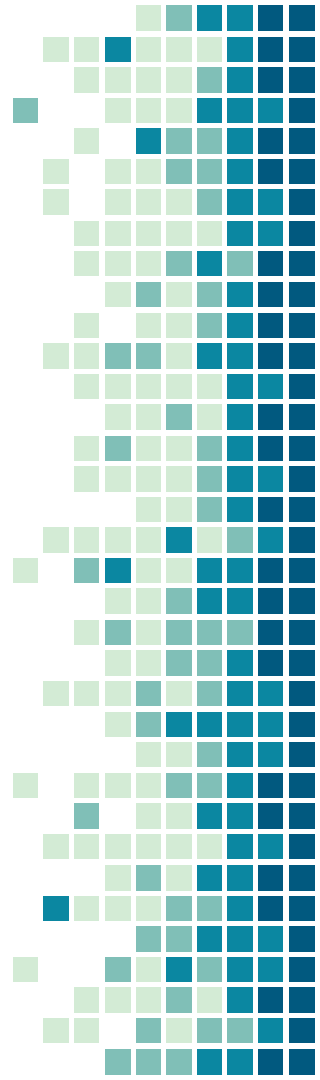
Source code 3: Face Classifier and post-processing part

```
# Classification and loading of the right cyclegan image
if(face_classification.face_classifier("./img.jpg") == 0):
    cycleGanImg = Image.open("./results/blackwhite_cyclegan/test_latest/images/img_fake_B.png")
else:
    cycleGanImg = Image.open("./results/blackwhite_cyclegan/test_latest/images/img_fake_A.png")

# paste the img in the frame (need to resize the img first)
cycleGanImg = cycleGanImg.resize(((x2-x1), (y2-y1))), PIL.Image.ANTIALIAS)
img.paste(cycleGanImg, (x1, y1))

# save the current frame
img.save(saveName, "JPEG", icc_profile=img.info.get('icc_profile')) # img0_123_S2

return frameOpencvDnn, bboxes
```



Esempio di trasformazione video



Frame originale

Frame trasformata

3D-ResNet

Video Classifier

Chapter 5



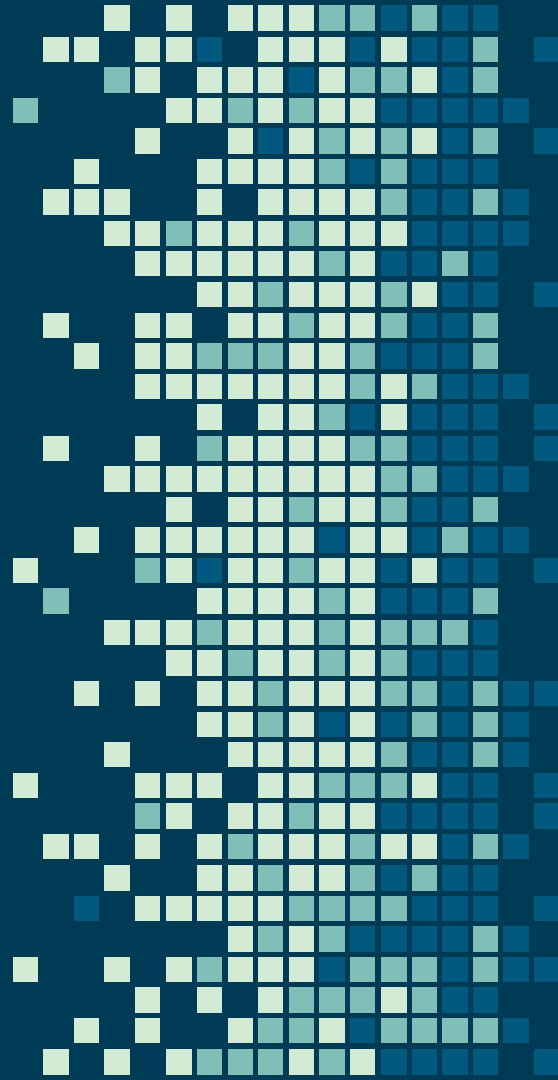
Network architecture

- Una 3D-ResNet è una 3D CNN basata su Residual Networks.
- Le Residual Networks sono una tra le architetture più potenti, che offrono le migliori performance in ambito di action recognition.
- La differenza tra una 3D-ResNet e la ResNet originale consiste nel numero differente di dimensioni dei convolutional kernels e del pooling.
- La peculiarità delle ResNet consiste nella presenza di shortcut connections nel modello.



“ *ResNets introduce shortcut connections that bypass a signal from one layer to the next. The connections pass through the gradient flows of networks from later layers to early layers, and ease the training of very deep networks. [...] The connections bypass a signal from the top of the block to the tail. ResNets are consist of multiple residual blocks.*

*~ Network architecture,
3D-ResNet paper*



Shortcut connections

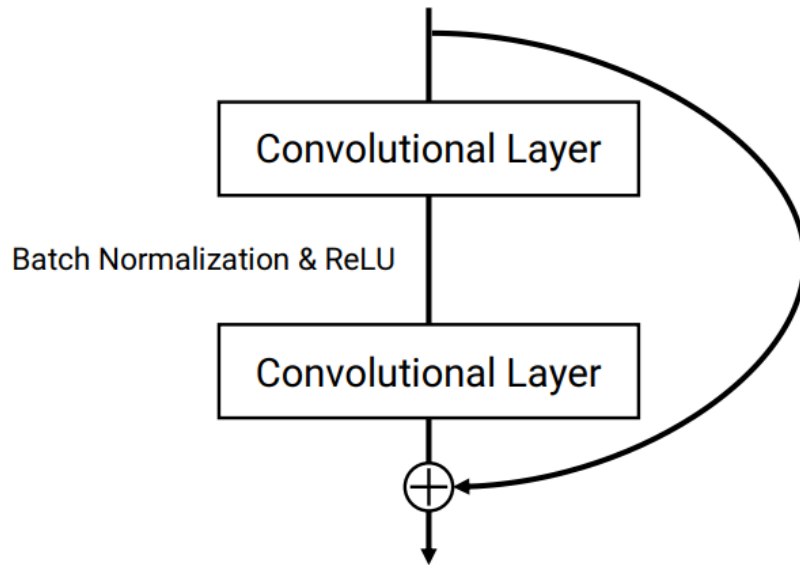
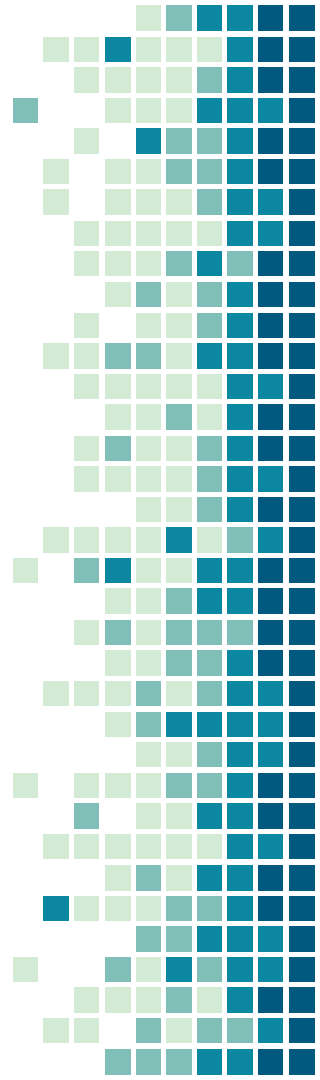


Figure 1: Residual block. Shortcut connections bypass a signal from the top of the block to the tail. Signals are summed at the tail.

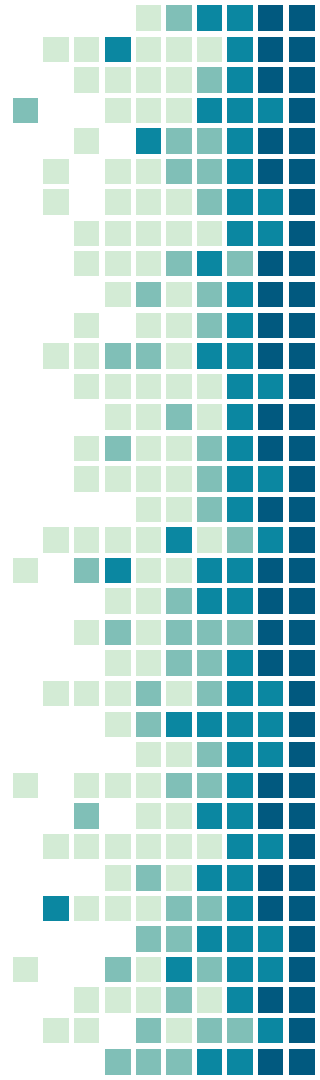
Obiettivi: Performance evaluation

- 6) Allenare il classificatore video 3D-ResNet su Utrain, e verificare le prestazioni su Utest.
- 7) Allenare il classificatore video 3D-ResNet su Utrain-inv, e verificare le prestazioni su Utest.
- 8) Allenare il classificatore video 3D-ResNet sull'unione tra Utrain e Utrain-inv, e verificare le prestazioni su Utest.



Model and training parameters

- Per la Video Classification è stata impiegata una 3D-ResNet con depth = 18 (resnet-18), allenata from scratch sul dataset UCF-101.
- La rete, per funzionare, necessita di ricevere in ingresso un ulteriore file (JSON), contenente alcune proprietà relative ai video da processare.



Pre-processing: JSON generation

- È stata svolta la seguente procedura di data preparation per la generazione del JSON, relativo al dataset UCF-101 da noi splittato in Utrain, Uval e Utest (punto 3):

- Convert from avi to jpg files using `utils/video_jpg_ucf101_hmdb51.py`

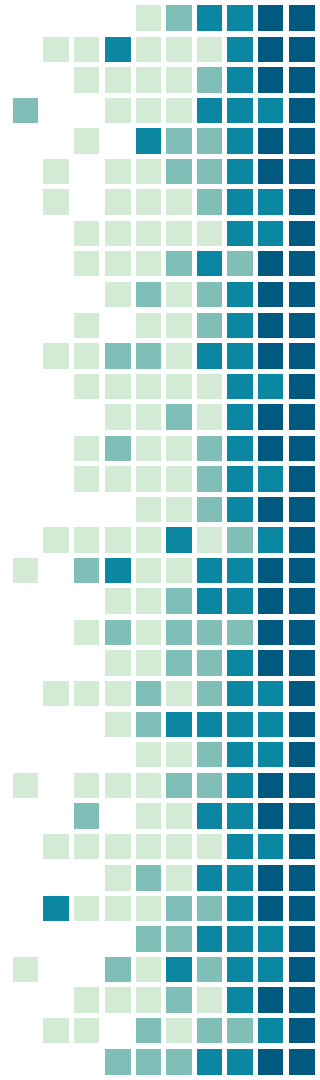
```
python utils/video_jpg_ucf101_hmdb51.py avi_video_directory jpg_video_directory
```

- Generate n_frames files using `utils/n_frames_ucf101_hmdb51.py`

```
python utils/n_frames_ucf101_hmdb51.py jpg_video_directory
```

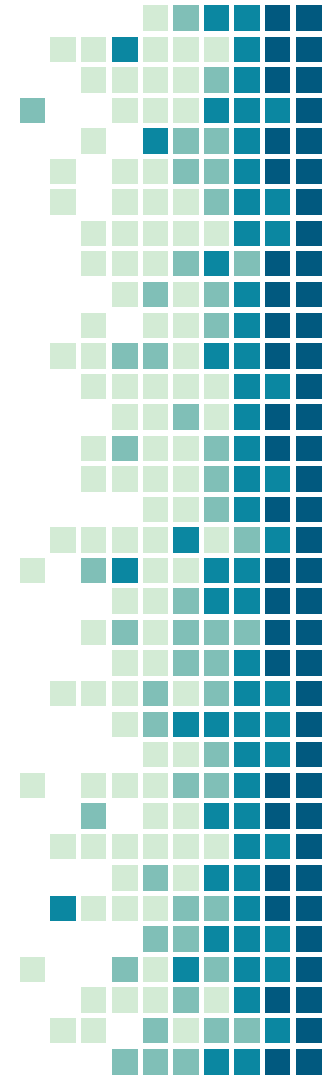
- Generate annotation file in json format similar to ActivityNet using `utils/ucf101_json.py`
 - `annotation_dir_path` includes `classInd.txt`, `trainlist0{1, 2, 3}.txt`, `testlist0{1, 2, 3}.txt`

```
python utils/ucf101_json.py annotation_dir_path
```



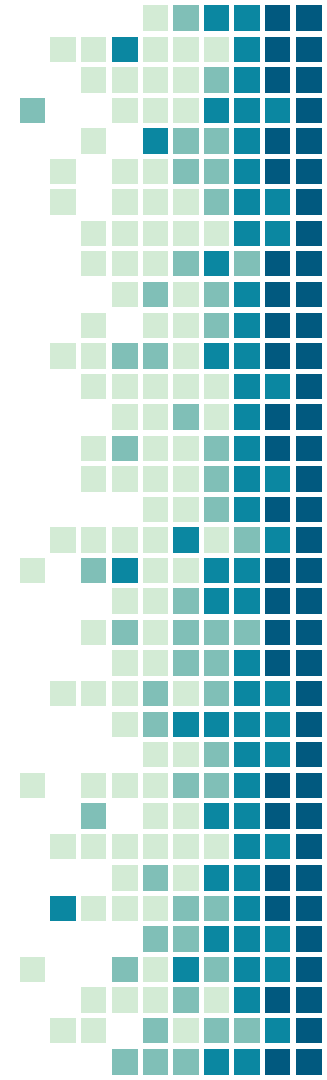
Pre-processing: TXT annotation

- La procedura di JSON generation appena vista prevede l'utilizzo di alcuni file di testo (in formato .txt) da passare come input, contenenti informazioni relative agli indici delle classi del dataset (nel file classInd.txt) e allo split dei dati precedentemente effettuato (nei file trainlist.txt e testlist.txt).
- A tale scopo è stato creato uno script apposito (file createTxtForAnnotation.py), che a partire dai dataset genera i corrispondenti file di testo.

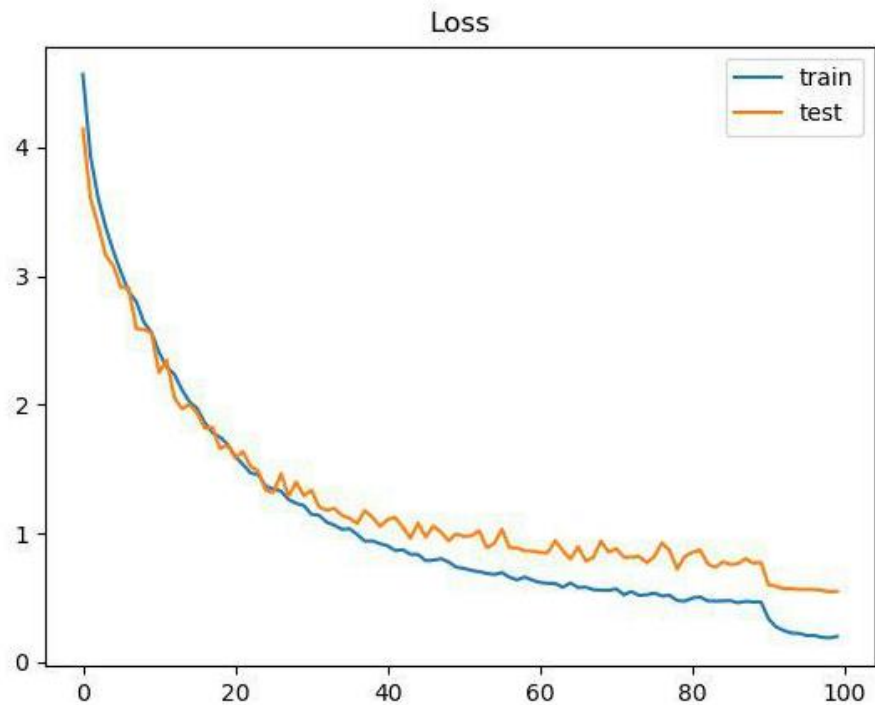
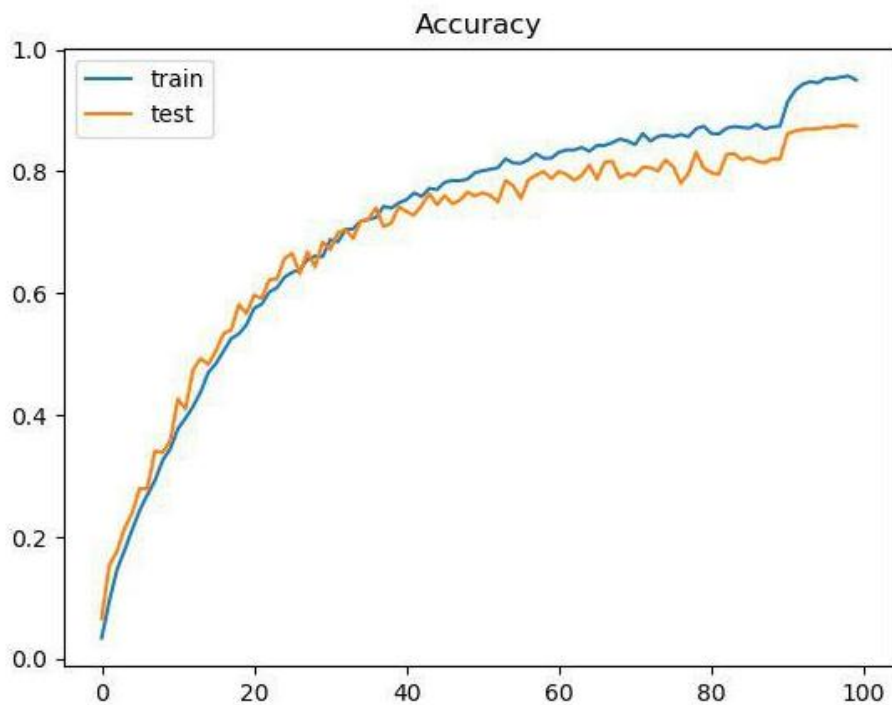


Punto 6: Training su Utrain e risultati

- La rete è stata allenata sul dataset Utrain per un totale di 100 epoche, con learning rate $lr = 0,01$ e batch size = 64.
- Le migliori performance si sono registrate all'epoch 97, con una test accuracy $TeA = 87,53\%$.



Punto 6: Loss e Accuracy



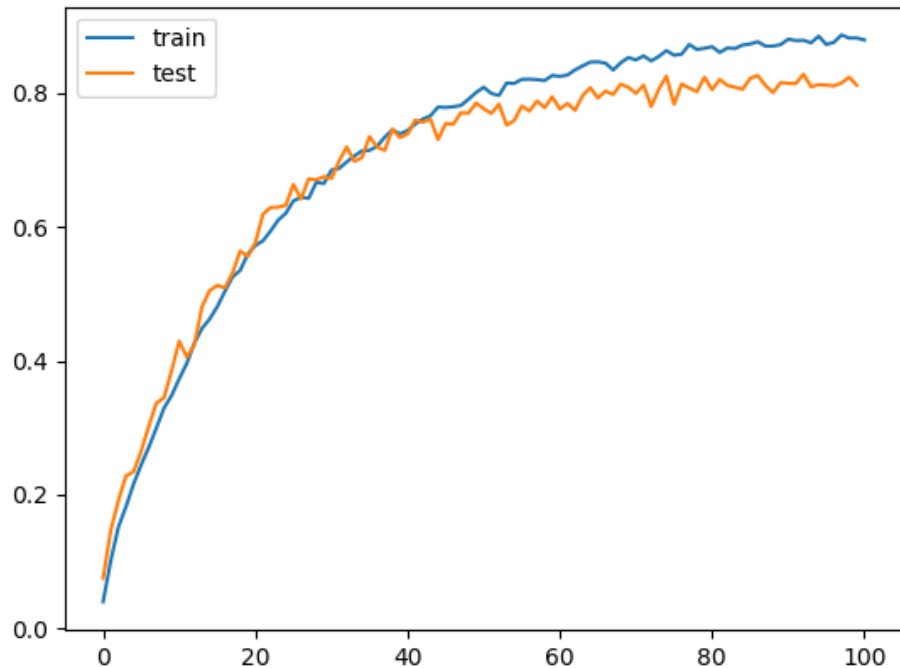
Punto 7: Training su Utrain-inv e risultati

- La rete è stata allenata sul dataset Utrain-inv per un totale di 100 epoche, con learning rate $lr = 0,01$ e batch size = 64.
- Le migliori performance si sono registrate all'epoch 92, con una test accuracy $TeA = 82,79\%$.

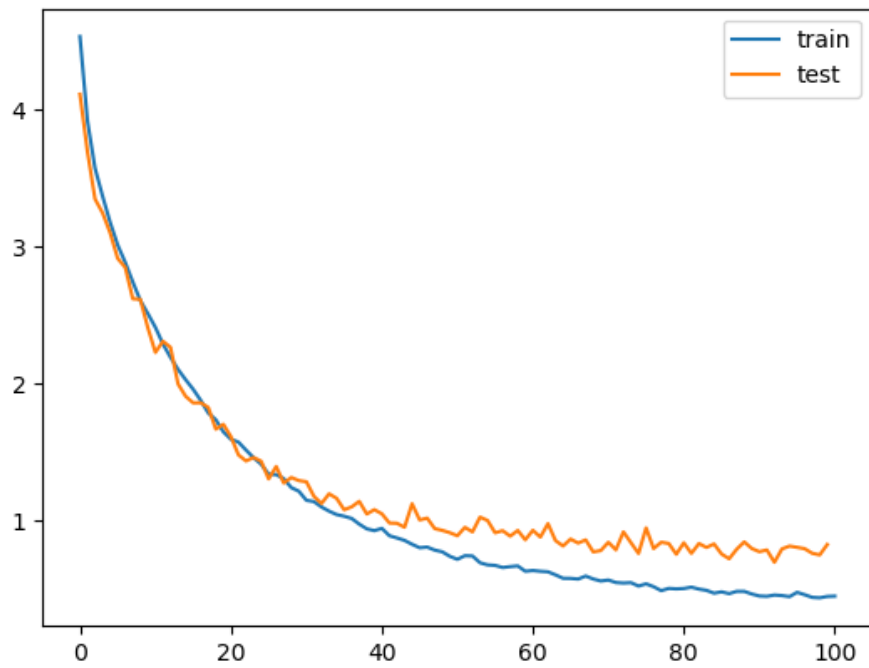


Punto 7: Loss e Accuracy

Accuracy

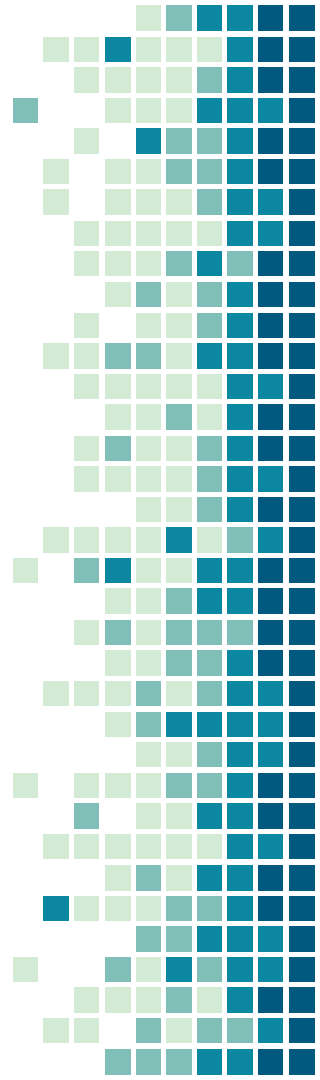


Loss

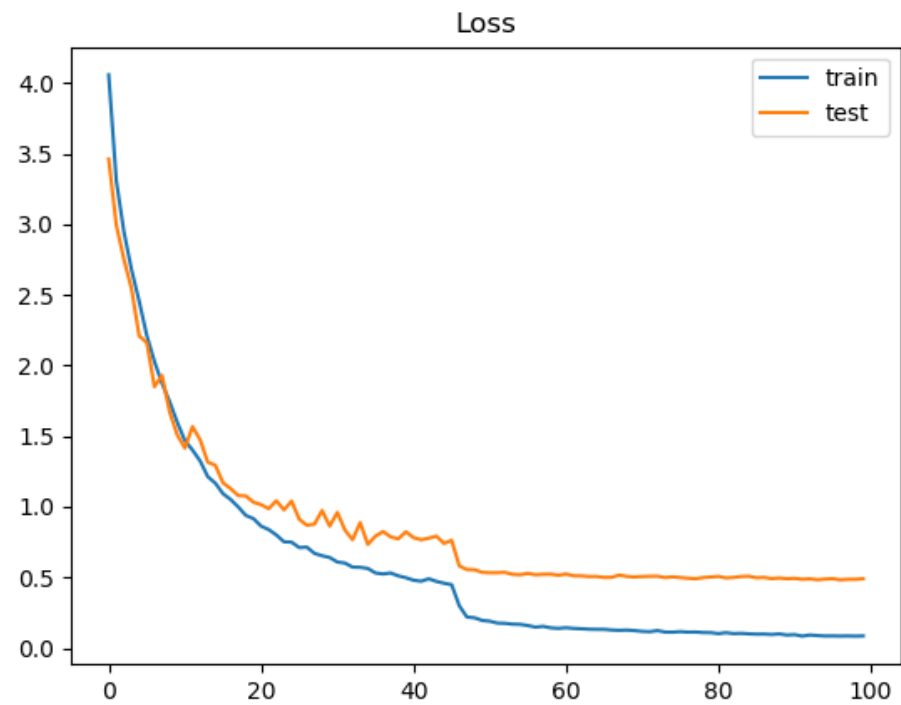
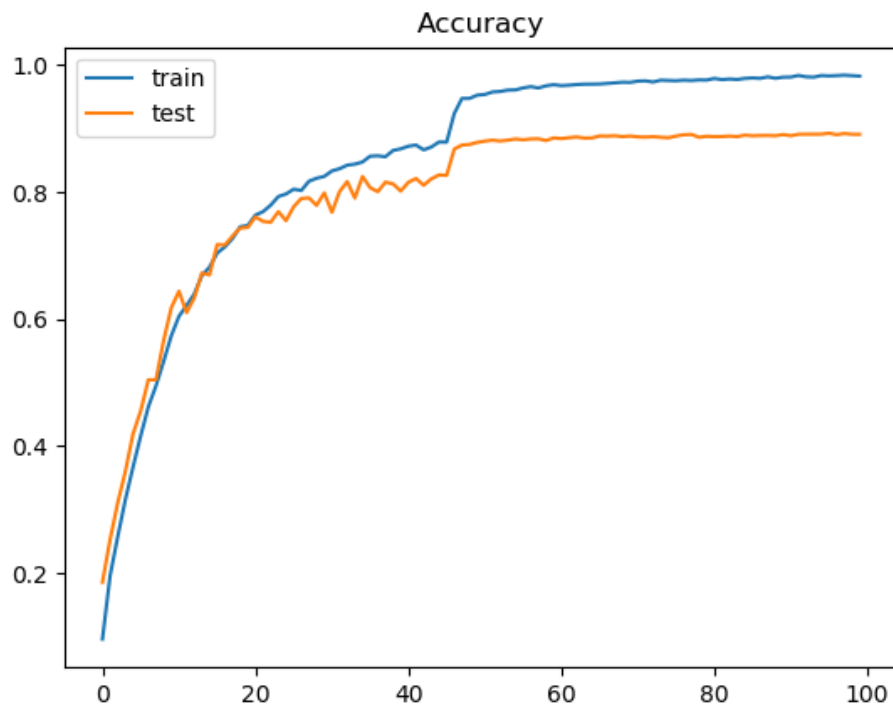


Punto 8: Training sull'unione di Utrain e Utrain-inv e risultati

- La rete è stata allenata sul dataset ottenuto dall'unione di Utrain e Utrain-inv per un totale di 100 epoche, con learning rate $lr = 0,01$ e batch size = 64.
- Le migliori performance si sono registrate all'epoch 95, con una test accuracy $TeA = 89,29\%$.

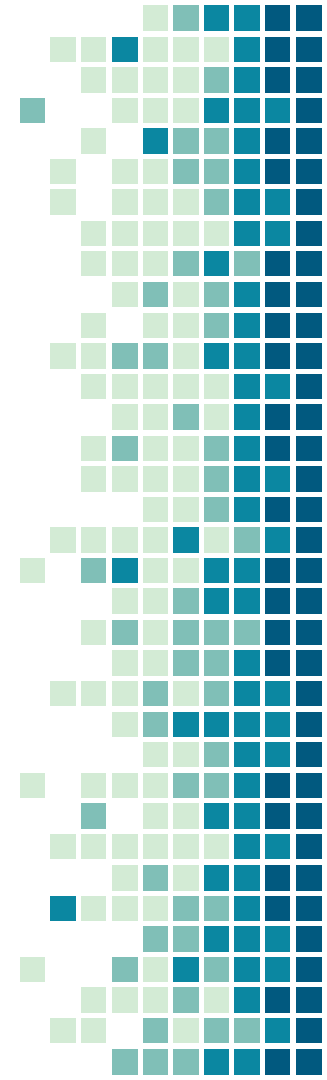


Punto 8: Loss e Accuracy



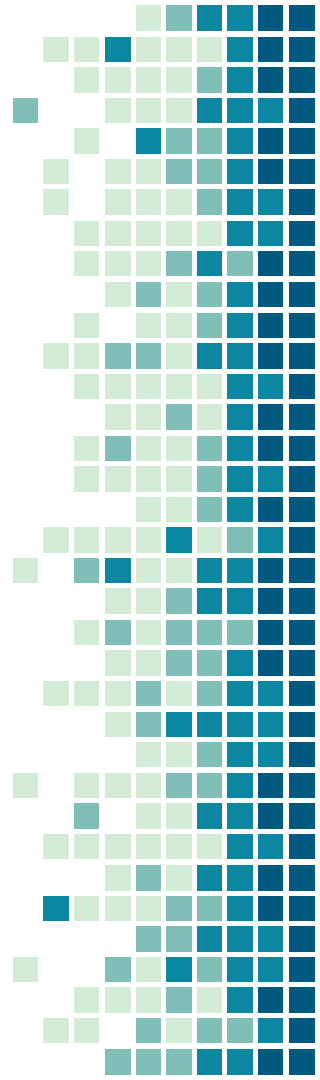
Riepilogo best performance

	Epoch	Test Acc. (%)
Utrain	97	87,53
Utrain-inv	92	82,79
Utrain + Utrain-inv	95	89,29



Note sui risultati

- Il dataset originale (Utrain) garantisce una migliore fase di training rispetto al dataset generato mediante l'utilizzo della CycleGAN (Utrain-inv), che permette al modello di raggiungere una più alta test accuracy.
- Le migliori performance in termini di accuracy si raggiungono utilizzando il dataset unione tra Utrain e Utrain-inv, anche se, in quest'ultimo caso, la presenza di dati che differiscono solo per il colore del volto negli split aumenta il rischio overfit, che si presenta prima (in termini di epoche) rispetto agli esperimenti precedenti (come da grafici precedentemente mostrati).



THANKS!

