
ELABORAZIONE DIGITALE DI IMMAGINI PER L'ESTRAZIONE DI **INFORMAZIONI VISIVE.**

COMPUTER VISION


Francesco Pertile - Filippo Pruzzi – Lorenzo Vivarelli

CHE COS'È LA COMPUTER VISION?

Branca dell'intelligenza artificiale (AI) che studia e programma algoritmi che permettono ai computer di **replicare i processi dell'apparato visivo umano** e di rilevare e interpretare informazioni importanti attraverso un'**immagine digitale**, un video o altri input visivi.

Non solo può riconoscere oggetti, animali o persone ma può anche estrapolare informazioni utili, interpretare i dati ricavati, elaborarli e intraprendere azioni.

LA STORIA

- **1959** Esperimento: mostrarono ad un gatto una serie di immagini e scoprirono che esso reagiva prima ai bordi netti o alle linee.  L'elaborazione delle immagini deve iniziare con forme semplici.
- **1963** Fu sviluppata la prima tecnologia di scansione delle immagini al computer. Iniziano le ricerche sull'Intelligenza Artificiale.
- **1974** Tecnologia di riconoscimento ottico dei caratteri e tecnologia di riconoscimento intelligente dei caratteri (usando reti neurali).

LA STORIA

- **1982** Marr dimostra che la vista funziona in modo gerarchico e introduce degli algoritmi per rilevare angoli, bordi ecc..

Fukushima sviluppa una rete con degli strati convoluzionali in una rete neurale per riconoscere i modelli.

- **2001** Approfondimento degli studi sul riconoscimento di oggetti.

Vengono sviluppate le prime applicazioni di riconoscimento facciale in tempo reale.

- **2010** Le CNN (Convolutional Neural Network) vengono inserite in un programma di riconoscimento delle immagini.

Dal 2010 in poi il tasso di errore è sempre diminuito.

LE FASI FONDAMENTALI DELLA CV

➤ **ACQUISIZIONE DELL'IMMAGINE**

➤ **ELABORAZIONE DELL'IMMAGINE**

➤ **INTERPRETAZIONE DELL'IMMAGINE**

ACQUISIZIONE DELL'IMMAGINE

Tramite foto, video, o anche in tempo reale per **finalità di analisi**.

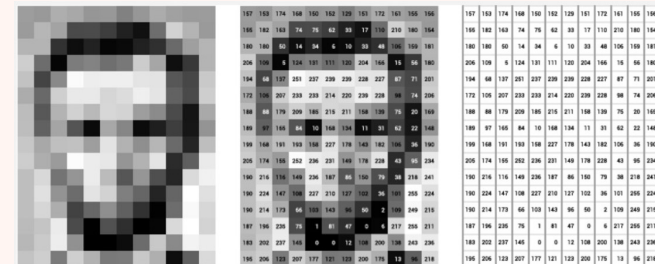
Gli algoritmi di CV operano su sequenze di bit.

Le immagini vengono salvate come matrici di pixel.

Immagini in scala di grigi



Il pixel assume valori da 0 a 255.



Immagini a colori, Servono **tre matrici** per immagine: una per il

rosso, una per il **verde** e una per il **blu** (RGB).

ELABORAZIONE DELL'IMMAGINE

L'elaborazione dell'immagine avviene attraverso dei modelli di **Deep Learning**, che comprende **tecniche di apprendimento autonomo** avanzate basate su reti neurali artificiali organizzate in livelli.

Questi modelli vengono **addestrati** precedentemente a riconoscere patterns ricorrenti all'interno delle immagini.

Questo meccanismo è analogo a quello dell'occhio umano, che riesce a distinguere varie forme concentrandosi su dettagli particolari.

TIPOLOGIA DI ANALISI

Per ottenere le informazioni dalle immagini i sistemi di CV possono essere basati su **tre tipologie di analisi** da utilizzare singolarmente o in combinazione:

Hand Crafted Features



L'algoritmo estrae quello che ritiene più **rilevante** all'interno di un'immagine o sequenza video

(colore; forma; area)

CV Features



L'algoritmo suddivide le immagini in parti più piccole per **confrontarle con il dataset** in modo da ottenere un'analisi più approfondita.

Data Driven Features



Un'analisi più avanzata che consente all'algoritmo di **riconoscere e classificare** le immagini senza la fase di estrazioni delle features (effettuata dalle reti neurali convoluzionali).

RETI NEURALI

Sono modelli matematici composti da **neuroni artificiali** che si ispirano al funzionamento biologico del cervello umano.

Vengono utilizzate per risolvere problemi ingegneristici di Intelligenza Artificiale.

Le **reti neurali cerebrali** sono formate invece da **neuroni biologici** interconnessi, che permettono a ciascun individuo di ragionare, fare calcoli in parallelo, riconoscere suoni, immagini, volti, imparare e agire...

RETI NEURALI

Sono formate da **tre strati** (che però possono coinvolgere migliaia di neuroni e decine di migliaia di connessioni):

Strato di input. Ha il compito di ricevere ed elaborare i segnali in ingresso adattandoli alle richieste dei neuroni della rete.

Strato nascosto (hidden). Ha in carica il processo di elaborazione vero e proprio (e può anche essere strutturato con più livelli di neuroni).

Strato di output. Vengono raccolti i risultati dell'elaborazione dello strato nascosto e vengono adattati alle richieste del successivo livello-blocco della rete neurale.

RETI NEURALI CONVOLUZIONALI

Sono **architetture di rete** per il Deep Learning che apprendono direttamente dai dati e sono alla base delle attività di riconoscimento delle immagini e della computer vision.

Grazie ai principi dell'algebra lineare, in particolare la moltiplicazione di matrici, riesce a riconoscere gli oggetti **identificando i modelli** all'interno dell'immagine.

Sono anche abbastanza efficaci nella classificazione di dati audio e segnali.

RETI NEURALI CONVOLUZIONALI

FUNZIONAMENTO E COSTITUZIONE

Una CNN può avere decine o centinaia di layer, ciascuno dei quali impara a rilevare feature diverse di un'immagine.

A ciascuna immagine di addestramento vengono applicati dei filtri a diverse

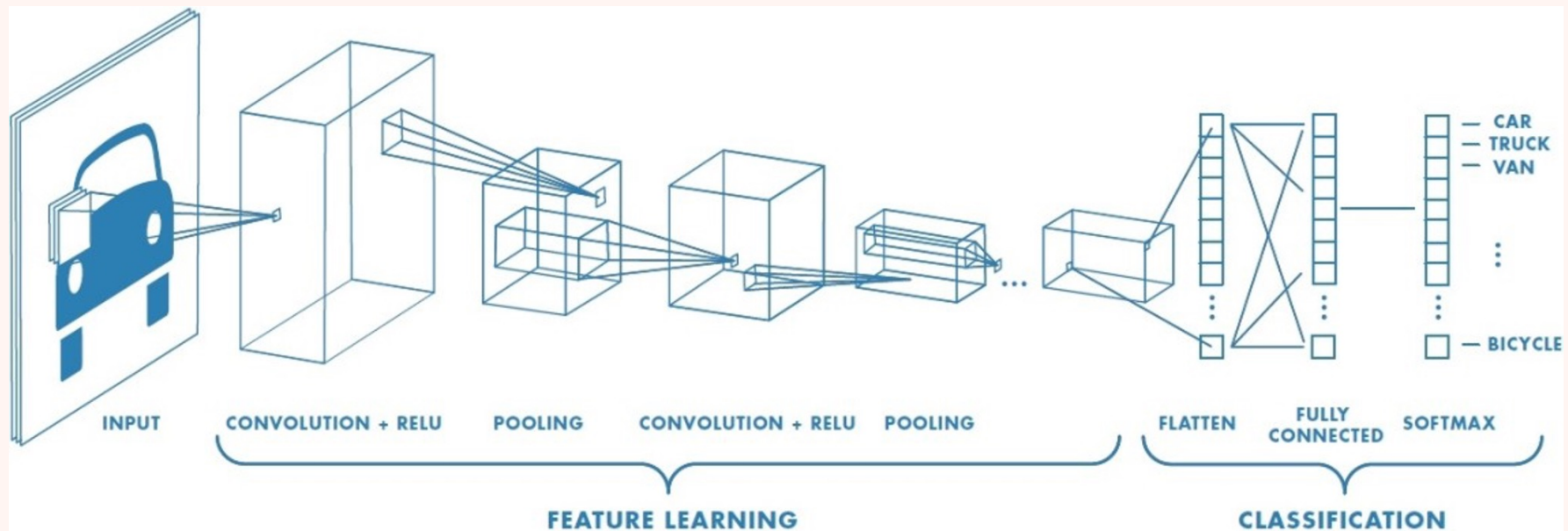
risoluzioni e l'output di ogni immagine convoluta viene utilizzato come input per il layer successivo.

Una CNN è costituita da un layer di input, un layer di output e da molti layer nascosti nel mezzo.

RETI NEURALI CONVOLUZIONALI

Contengono **tre tipi di livello** principali:

1. Livello convoluzionale;
2. Livello di pooling;
3. Livello completamente connesso (Fully-connected).



INTERPRETAZIONE DELL'IMMAGINE

La macchina identifica, prende e classifica **l'immagine elaborata** e se necessario intraprende un'azione.

Algoritmi:

Image Classification, Object Detection, Image Segmentation,

Face Recognition, Action Recognition, Visual Relationship Detection, Emotion Recognition,
Image Editing.

IMAGE GENERATION

Qui riportiamo un codice scritto da noi, il quale ci permette, accedendo alle librerie di torch, di andare a definire e addestrare la nostra deep convolutional generative adversarial network (DCGAN).

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import torchvision.datasets as datasets
from torch.utils.data import DataLoader
# Launch TensorBoard Session
from torch.utils.tensorboard import SummaryWriter
```

```
# Discriminator con strati convoluzionali
class Discriminator(nn.Module): # utilizziamo una rete neurale feedforward
    def __init__(self, img_channels): # img_dim = 784
        super(Discriminator, self).__init__() # ereditiamo dalla classe nn.Module
        self.disc = nn.Sequential( # utilizziamo un modello sequenziale
            nn.Conv2d(img_channels, 32, kernel_size=4, stride=2, padding=1), # Output: N x 32 x 14 x 14
            nn.LeakyReLU(0.2), # LeakyReLU è utilizzato per prevenire il problema di ReLU morto nei GAN
            nn.Conv2d(32, 64, kernel_size=4, stride=2, padding=1), # Output: N x 64 x 7 x 7
            nn.BatchNorm2d(64), # Batch normalization
            nn.LeakyReLU(0.2),
            nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1), # Output: N x 128 x 3 x 3
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2),
            nn.Conv2d(128, 1, kernel_size=3, stride=1, padding=0), # Output: N x 1 x 1 x 1
            nn.Sigmoid(), # Sigmoid per ottenere la probabilità dell'output tra 0 e 1
        )

    def forward(self, x):
        return self.disc(x).view(-1) # Output scalare per ogni immagine
```

```
# Generator con strati convoluzionali
class Generator(nn.Module): # utilizziamo una rete neurale feedforward
    def __init__(self, z_dim, img_channels): # z_dim = 64, img_dim = 784 -> 28x28x1
        super(Generator, self).__init__()
        self.gen = nn.Sequential(
            # Input: N x z_dim x 1 x 1
            nn.ConvTranspose2d(z_dim, 256, kernel_size=7, stride=1, padding=0), # Output: N x 256 x 7 x 7
            nn.BatchNorm2d(256),
            nn.ReLU(), # Funzione di attivazione
            nn.ConvTranspose2d(256, 128, kernel_size=4, stride=2, padding=1), # Output: N x 128 x 14 x 14
            nn.BatchNorm2d(128), # Batch normalization
            nn.ReLU(),
            nn.ConvTranspose2d(128, 64, kernel_size=4, stride=2, padding=1), # Output: N x 64 x 28 x 28
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.ConvTranspose2d(64, img_channels, kernel_size=3, stride=1, padding=1), # Output: N x img_channels x 28 x 28
            nn.Tanh(), # Output compreso tra -1 e 1
        )

    def forward(self, x):
        return self.gen(x)
```

[Click to collapse the range.](#)

```
# Hyperparameters
device = "cuda" if torch.cuda.is_available() else "cpu" # Verifica se è disponibile una GPU
lr = 3e-4 # Learning rate
z_dim = 64 # Dimensione del vettore latente (rumore)
image_channels = 1 # Canali di output per MNIST (grayscale)
batch_size = 128 # Dimensione del batch
num_epochs = 75 # Numero di epoche

disc = Discriminator(image_channels).to(device) # Discriminatore
gen = Generator(z_dim, image_channels).to(device) # Generatore
fixed_noise = torch.randn((batch_size, z_dim, 1, 1)).to(device) # Noise iniziale

# Trasformazioni per normalizzare il dataset
transforms = transforms.Compose([
    transforms.ToTensor(), # Converte l'immagine in un tensore
    transforms.Normalize((0.5,), (0.5,)), # Normalizza i valori tra -1 e 1
])

dataset = datasets.MNIST(root="dataset/", transform=transforms, download=True)
loader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
opt_disc = optim.Adam(disc.parameters(), lr=lr) # Ottimizzatore per il discriminatore
opt_gen = optim.Adam(gen.parameters(), lr=lr) # Ottimizzatore per il generatore
criterion = nn.BCELoss() # Loss per il discriminatore

# Tensorboard writers
writer_fake = SummaryWriter(f"runs/DCGAN_MNIST/fake")
writer_real = SummaryWriter(f"runs/DCGAN_MNIST/real")
step = 0
```

```
for epoch in range(num_epochs):
    for batch_idx, (real, _) in enumerate(loader):
        real = real.to(device) # Spostiamo le immagini reali sulla GPU
        batch_size = real.shape[0] # Dimensione del batch

        ### Train Discriminator: max log(D(x)) + log(1 - D(G(z)))
        noise = torch.randn(batch_size, z_dim, 1, 1).to(device) # Generiamo il rumore
        fake = gen(noise) # Generiamo le immagini fake
        disc_real = disc(real).view(-1) # Passiamo le immagini reali al discriminatore
        lossD_real = criterion(disc_real, torch.ones_like(disc_real)) # Calcoliamo la loss per le immagini reali
        disc_fake = disc(fake.detach()).view(-1) # Passiamo le immagini fake al discriminatore
        lossD_fake = criterion(disc_fake, torch.zeros_like(disc_fake)) # Calcoliamo la loss per le immagini fake
        lossD = (lossD_real + lossD_fake) / 2 # Calcoliamo la loss media per le immagini reali e fake
        disc.zero_grad() # Azzeriamo i gradienti
        lossD.backward() # Backpropagation
        opt_disc.step() # Aggiorniamo i pesi

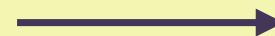
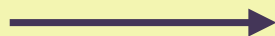
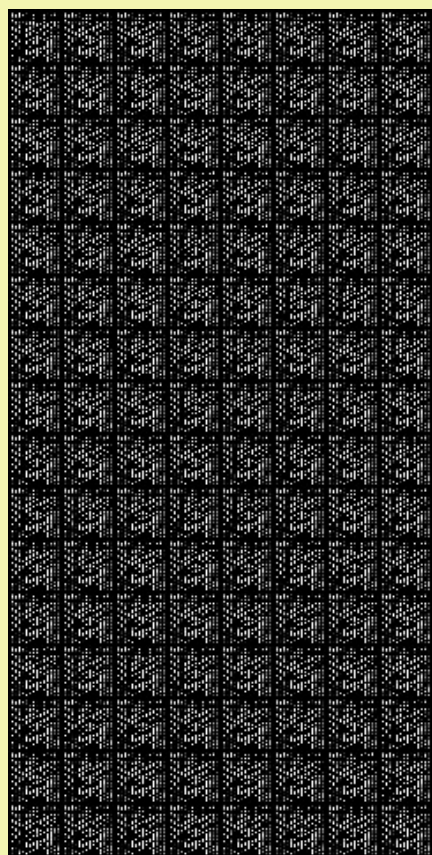
        ### Train Generator: min log(1 - D(G(z))) <-> max log(D(G(z)))
        output = disc(fake).view(-1) # Passiamo le immagini fake al discriminatore
        lossG = criterion(output, torch.ones_like(output)) # Calcoliamo la loss per le immagini fake
        gen.zero_grad() # Azzeriamo i gradienti
        lossG.backward() # Backpropagation
        opt_gen.step() # Aggiorniamo i pesi

    if batch_idx == 0:
        print( # Stampa delle loss
            f"Epoch [{epoch+1}/{num_epochs}] \
              Loss D: {lossD:.4f}, loss G: {lossG:.4f}"
        )

    with torch.no_grad(): # Disabilitiamo il calcolo dei gradienti
        fake = gen(fixed_noise).reshape(-1, 1, 28, 28) # Reshape delle immagini fake
        data = real.reshape(-1, 1, 28, 28) # Reshape delle immagini reali
        img_grid_fake = torchvision.utils.make_grid(fake, normalize=True) # Creiamo una griglia delle immagini fake
        img_grid_real = torchvision.utils.make_grid(data, normalize=True) # Creiamo una griglia delle immagini reali

        writer_fake.add_image("Mnist Fake Images", img_grid_fake, global_step=step) # Salviamo le immagini fake
        writer_real.add_image("Mnist Real Images", img_grid_real, global_step=step) # Salviamo le immagini reali
        step += 1
```


RISULTATI



CAMPI DI APPLICAZIONE

La computer vision è utilizzata in tutti i settori per cercare di migliorare l'esperienza del consumatore, ridurre i costi e aumentare la sicurezza.

Qui sotto riportiamo un esempio:

➤ **Tesla Vison**

TESLA VISION

È un sistema che si basa su telecamere che raccolgono informazioni visive, eliminando l'uso di radar.

Questo approccio sfrutta le stesse CNN per l'analisi visiva, ma con applicazioni e finalità completamente diverse rispetto al riconoscimento facciale.

I 3 punti del funzionamento dell'algoritmo di Tesla:

- Riconoscimento degli oggetti
- Previsione del movimento
- Decisioni in tempo reale



RISCONTRO NEGATIVO




Gender Shades

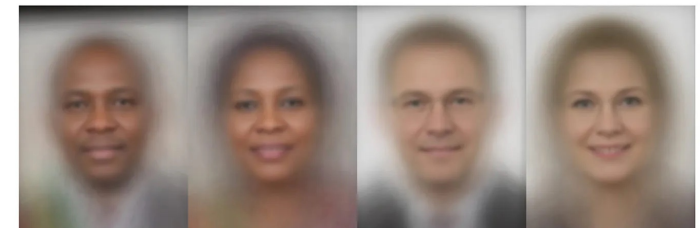
GENDER SHADES

Questo progetto ha evidenziato come gli algoritmi di computer vision, impiegati per sistemi di riconoscimento facciale, presentino gravi pregiudizi di genere e di razza.

Dimostrando che le prestazioni di questi sistemi variano notevolmente in base a questi due aspetti.

Poiché la CNN apprendono dai dataset su cui vengono addestrati, i risultati dello studio hanno mostrato che questi algoritmi erano sbilanciati (includevano una predominanza di uomini con la pelle chiara).

Gender Classifier	Darker Male	Darker Female	Lighter Male	Lighter Female	Largest Gap
 Microsoft	94.0% <div><div></div></div>	79.2% <div><div></div></div>	100% <div><div></div></div>	98.3% <div><div></div></div>	20.8% <div><div></div></div>
 FACE++	99.3% <div><div></div></div>	65.5% <div><div></div></div>	99.2% <div><div></div></div>	94.0% <div><div></div></div>	33.8% <div><div></div></div>
 IBM	88.0% <div><div></div></div>	65.3% <div><div></div></div>	99.7% <div><div></div></div>	92.9% <div><div></div></div>	34.4% <div><div></div></div>



BIBLIOGRAFIA

- <https://www.vgen.it/it/computer-vision-occhio-digitale/>
 - https://www.sas.com/it_it/insights/analytics/computer-vision.html#technical
 - <https://www.ibm.com/it-it/topics/computer-vision>
 - https://blog.osservatori.net/it_it/computer-vision-definizione-applicazioni
 - <https://www.ai4business.it/intelligenza-artificiale/deep-learning/reti-neurali/>
 - <https://www.ibm.com/it-it/topics/convolutional-neural-networks>
 - <https://it.mathworks.com/discovery/convolutional-neural-network.html>
 - <https://fritz.ai/computer-vision-at-tesla/>
 - <https://www.media.mit.edu/projects/gender-shades/overview/>
-