

# AI e CV applicate **all'anatomia**

BARDINI - FAGARAZ - TBIBI - TOFFOLON

# Indice



---

**Introduzione alle  
reti neurali**



---

**AI applicata alla  
diagnosi di  
tumori**



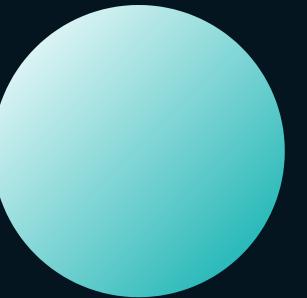
---

**AI applicata alla  
farmacogenomica**



---

**CV applicata  
all'analisi  
forense**



# Introduzione alle reti neurali

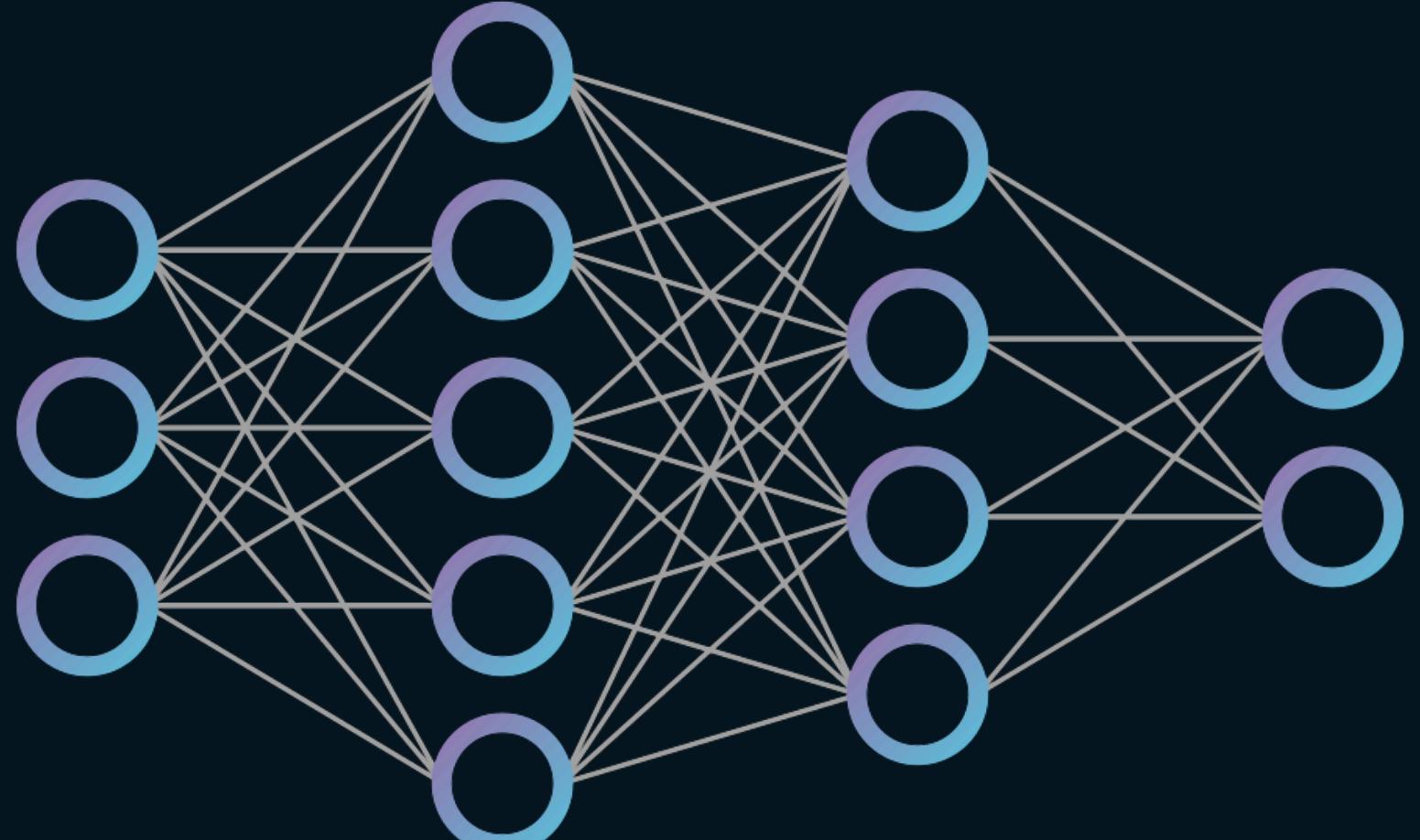
---

LUCA FAGARAZ



# Le reti neurali

Le reti neurali sono alla base dell'intelligenza artificiale. Queste ci permettono, dati degli input, di calcolare degli output in modo deterministico. La loro peculiarità è che possono essere allenate.

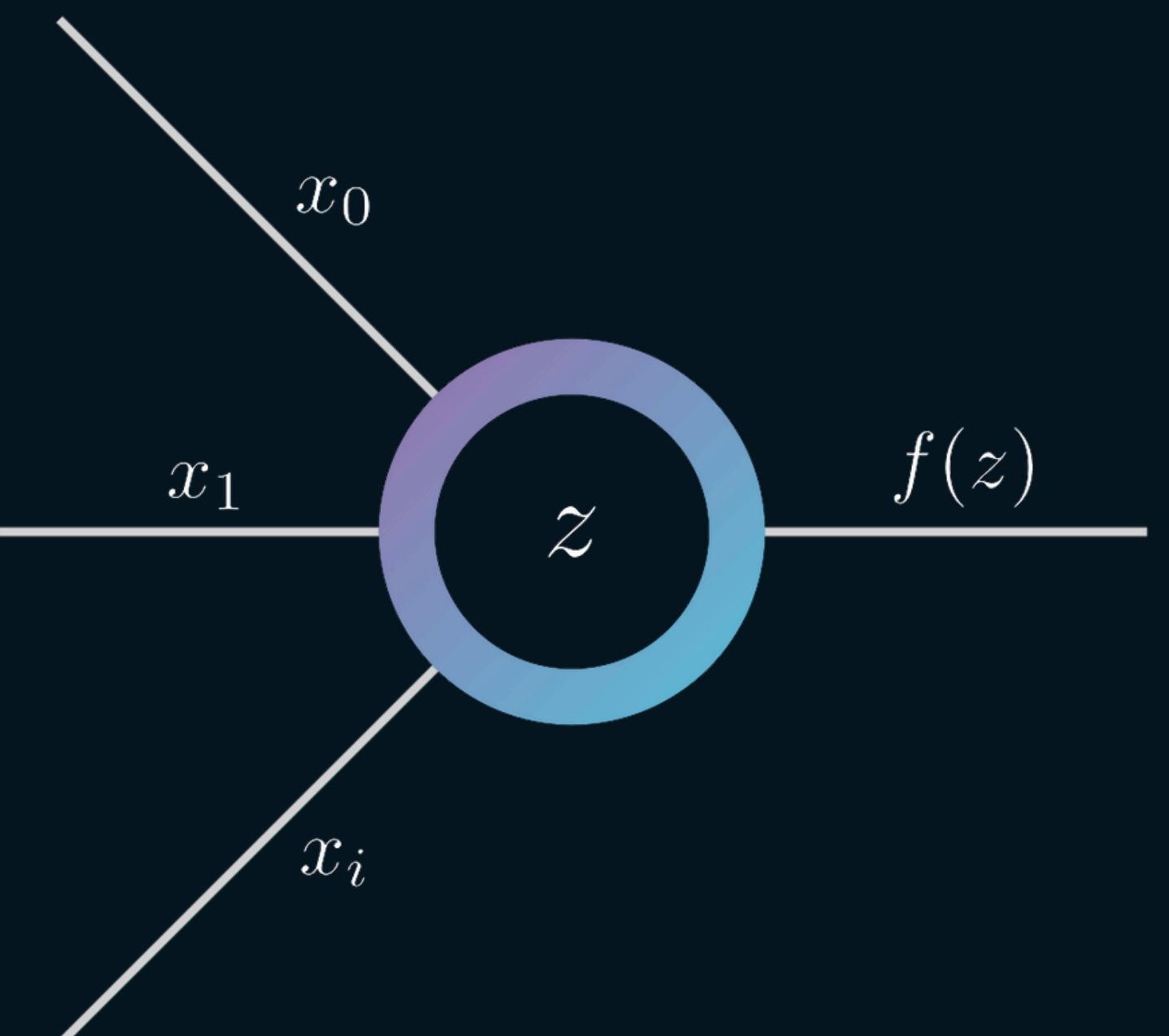


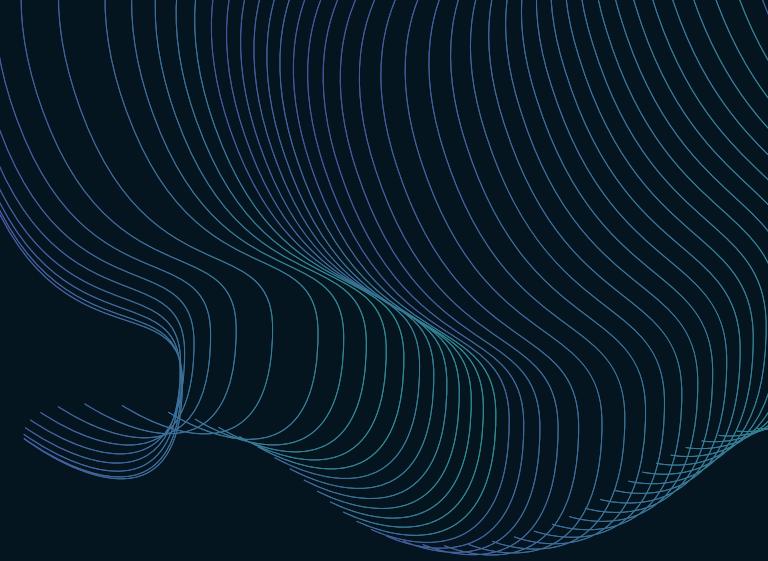


# Il neurone

Il neurone è la componente fondamentale di una rete neurale. Esso calcola uno stato in funzione degli input e in base a questo attiva più o meno i neuroni successivi.

$$z_j = \sum_i w_i a_i + b_j$$

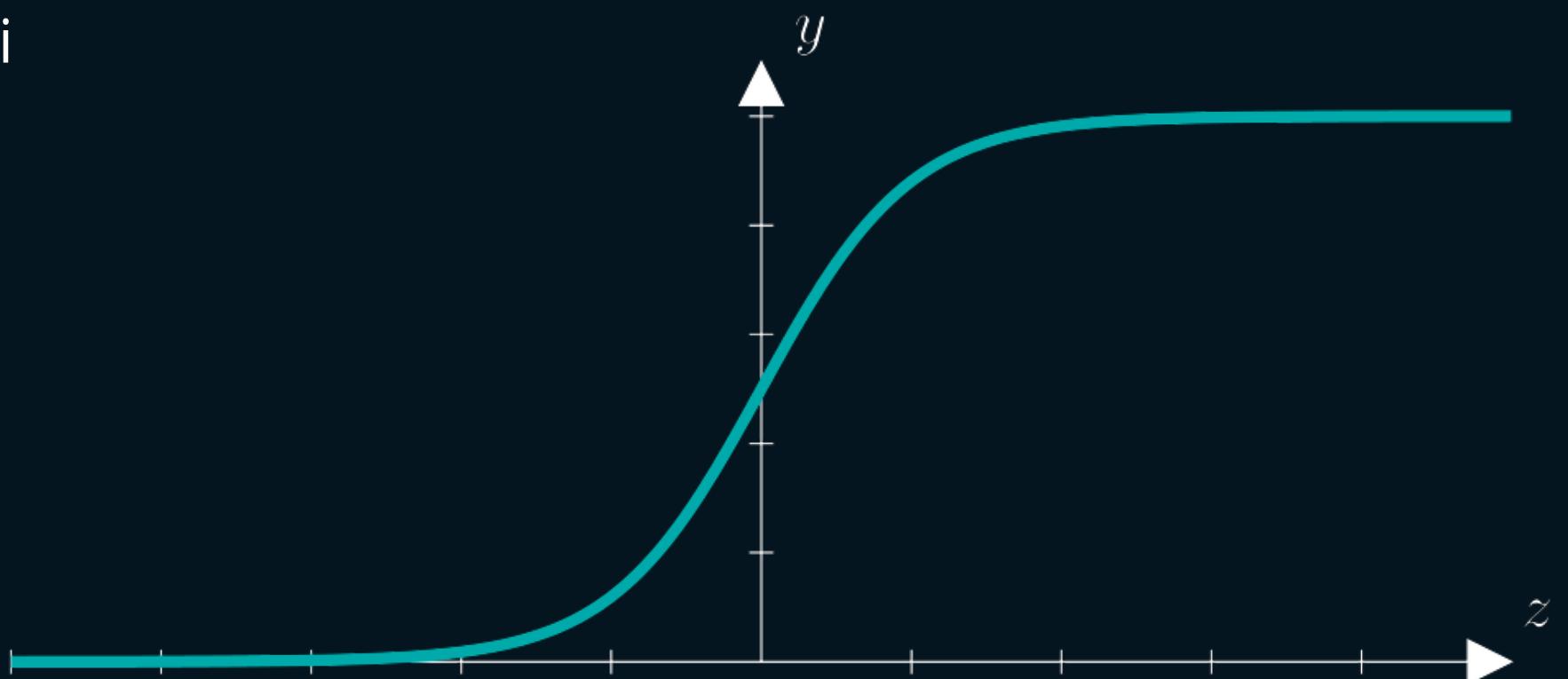


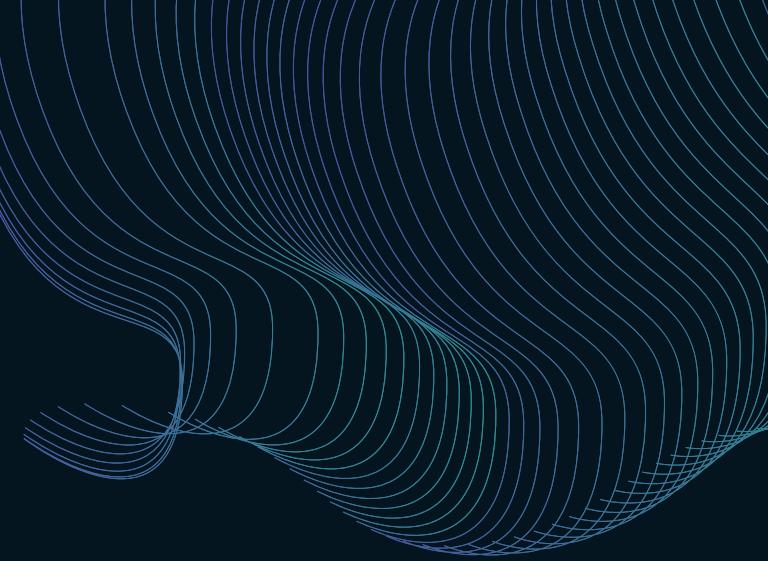


# Il neurone

Il neurone è la componente fondamentale di una rete neurale. Esso calcola uno stato in funzione degli input e in base a questo attiva più o meno i neuroni successivi.

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

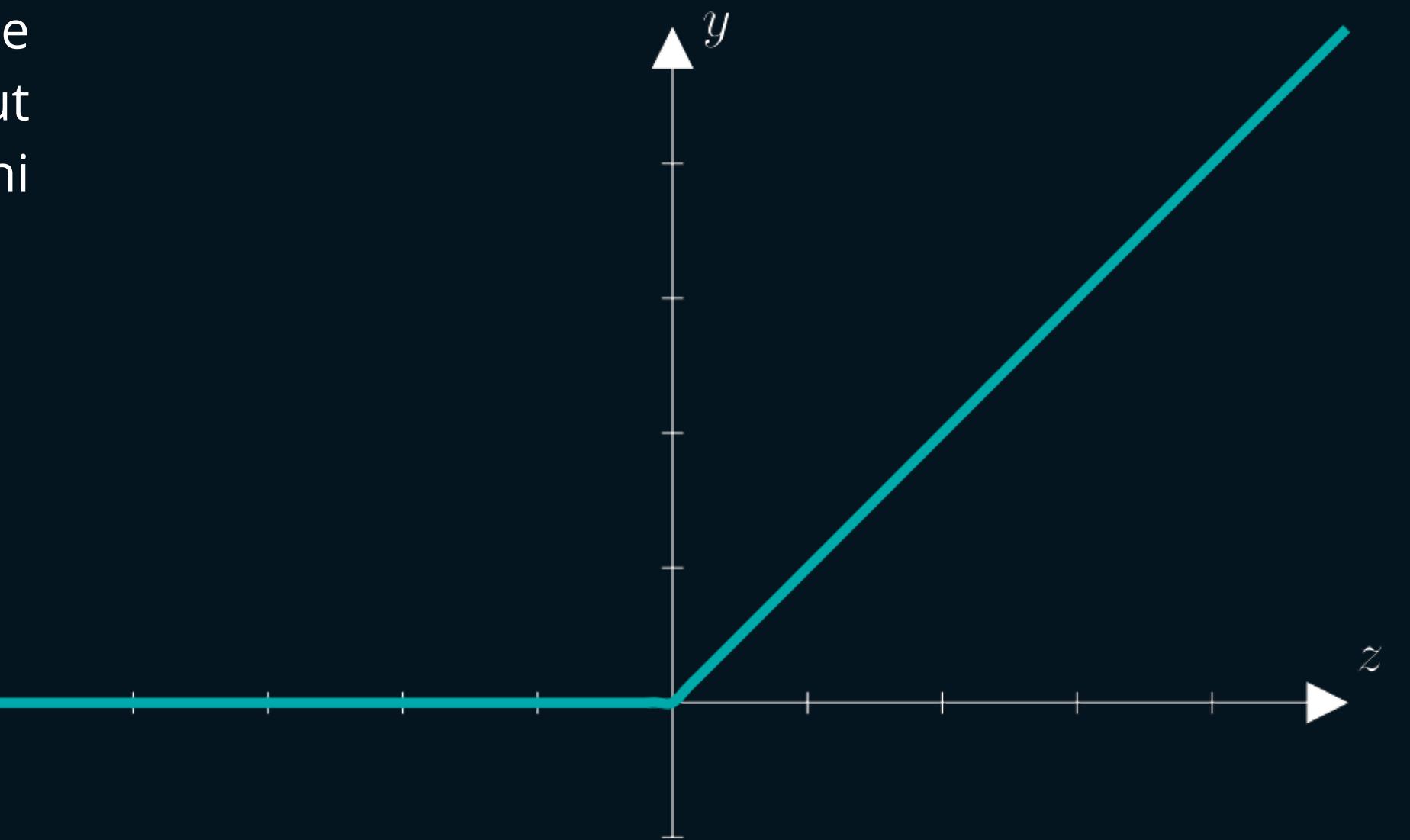




# Il neurone

Il neurone è la componente fondamentale di una rete neurale. Esso calcola uno stato in funzione degli input e in base a questo attiva più o meno i neuroni successivi.

$$\text{ReLU}(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$



# Gli strati e i collegamenti

Per determinare “quanto” un neurone dovrebbe attivare il successivo possiamo assegnare ad ogni collegamento un **“peso”**, ossia uno scalare.

$$x_j^{(i+1)} = \sigma(w_{jk}^{(i)} \cdot x_j^{(i)} + b_j^{(i)})$$



LUCA FAGARAZ

# Gli strati e i collegamenti

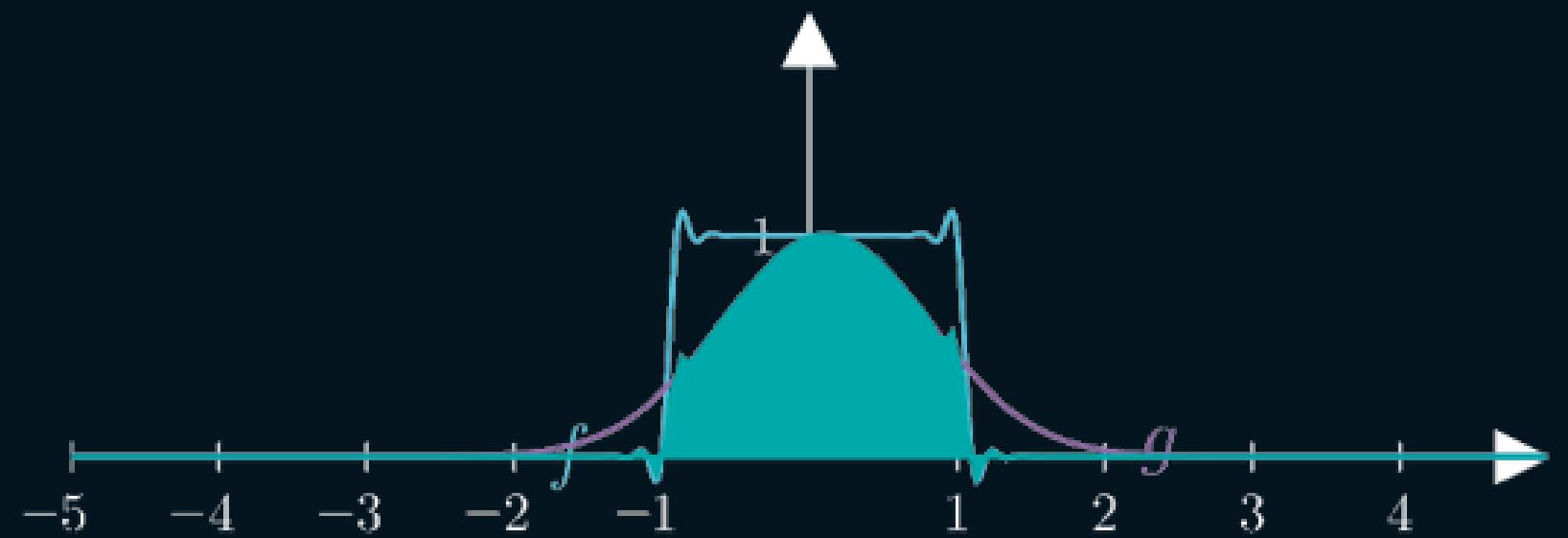
La rete sarà composta da:

- Uno strato di **input**
- Uno starto di **output**
- Più strati **intermedi** (hidden layers)



# Le convoluzioni

Le convoluzioni sono un'operazione fondamentale per i modelli di reti neurali moderni. Esse permettono di fornire una “misura” della correlazione tra due oggetti.



$$(f * g)(t) = \int_{-\infty}^{+\infty} f(\tau) \cdot g(t - \tau) d\tau$$

# La funzione costo

Per determinare quanto i pesi e i bias specifici della nostra rete siano “sbagliati”, possiamo definire una funzione “**costo**” che quantifichi la differenza tra l’output ottenuto e quello atteso.

$$C_0 = \left( a^{(n)} - y \right)^2$$

# La funzione costo

Per determinare quanto i pesi e i bias specifici della nostra rete siano “sbagliati”, possiamo definire una funzione “**costo**” che quantifichi la differenza tra l’output ottenuto e quello atteso.

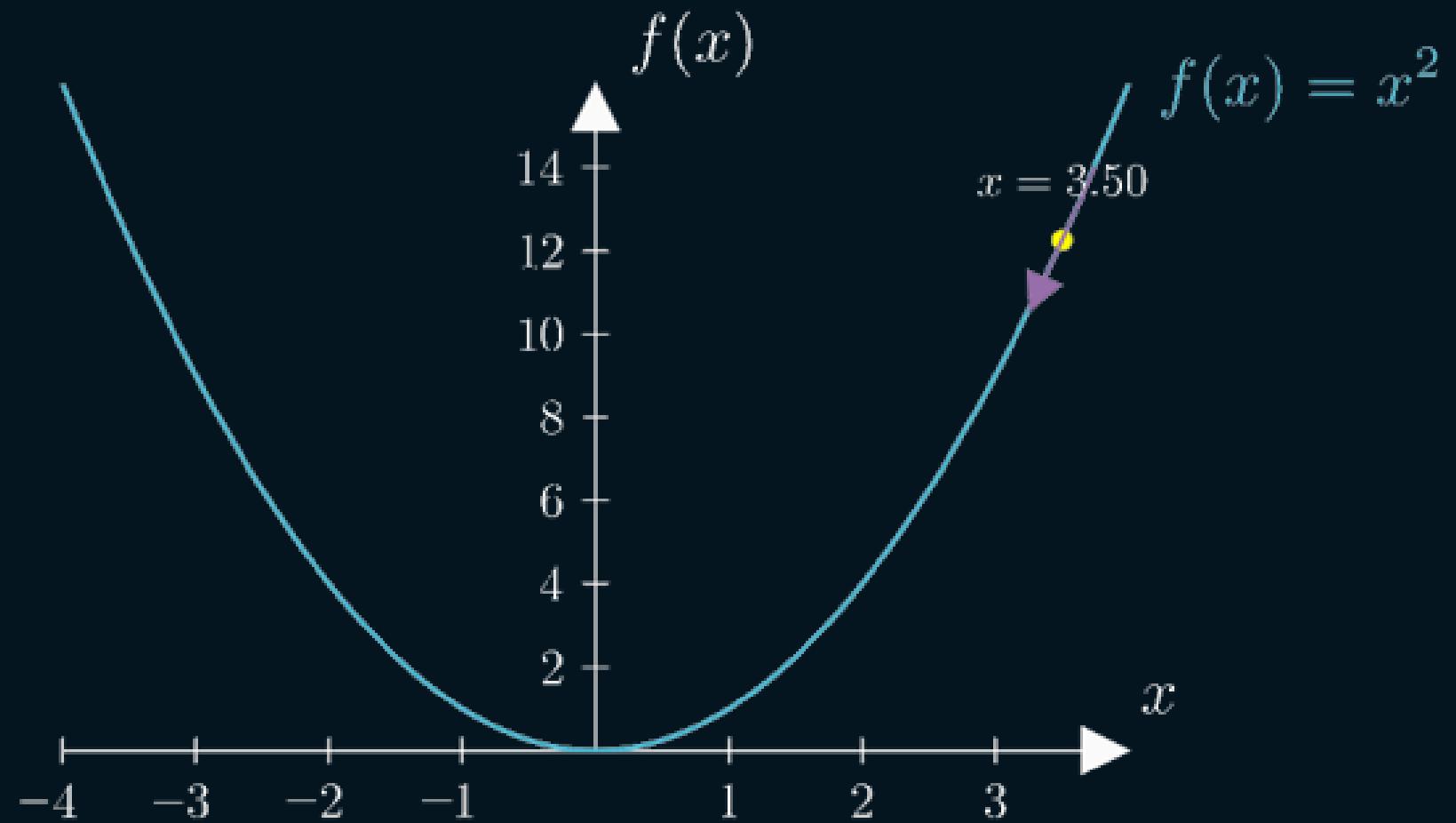
$$C = \frac{1}{n} \sum_{i=0}^n C_i$$



# Minimizzare il costo: SGD

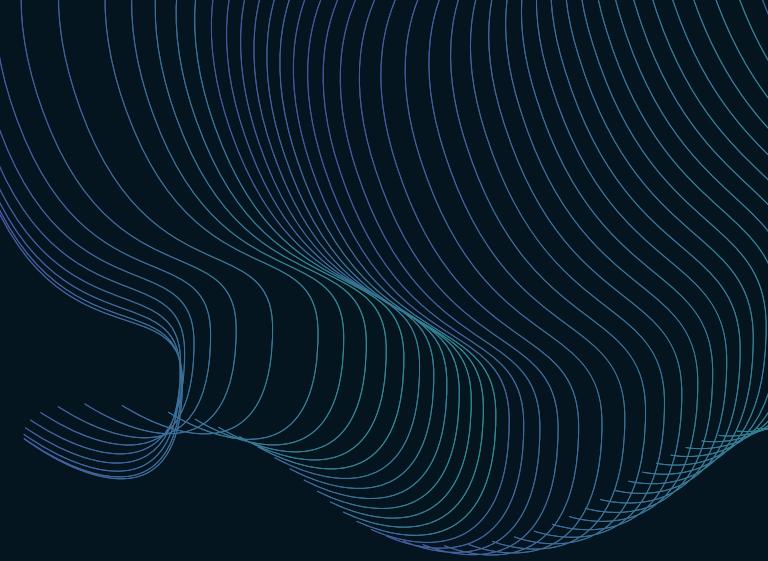
Successivamente, per rendere operativa la nostra rete ci basta minimizzare la funzione costo appena definita. Possiamo usare il metodo della **discesa del gradiente**.

$$\Delta x_i = -\eta \frac{\partial f}{\partial x_i}$$



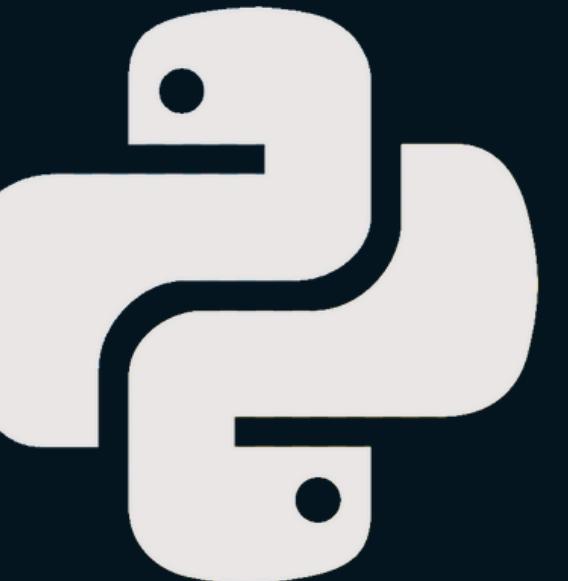
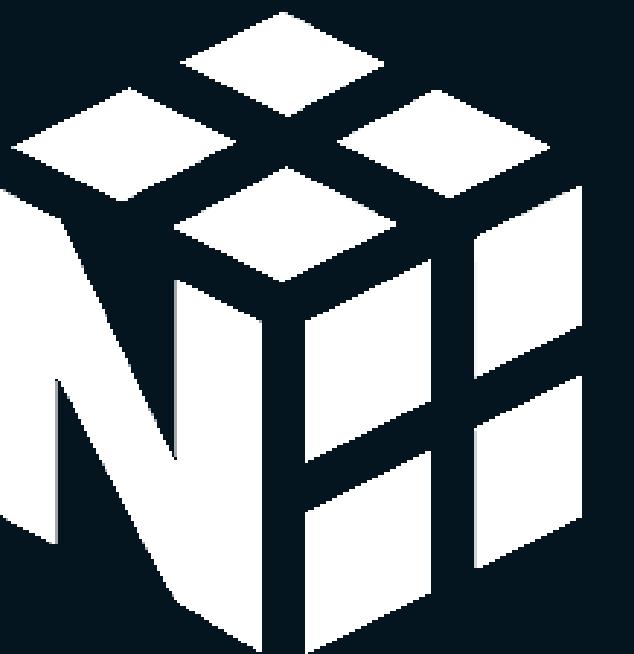


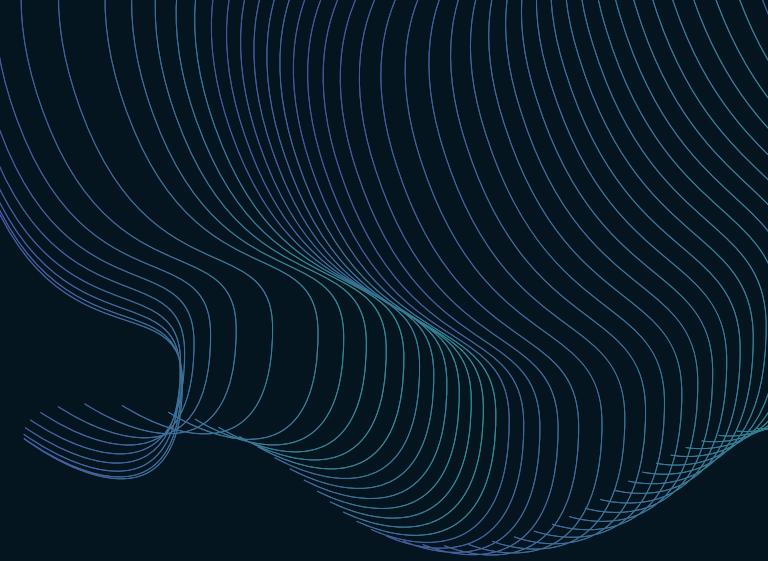
LUCA FAGARAZ



# Progetto pratico: Riconoscimento di cifre

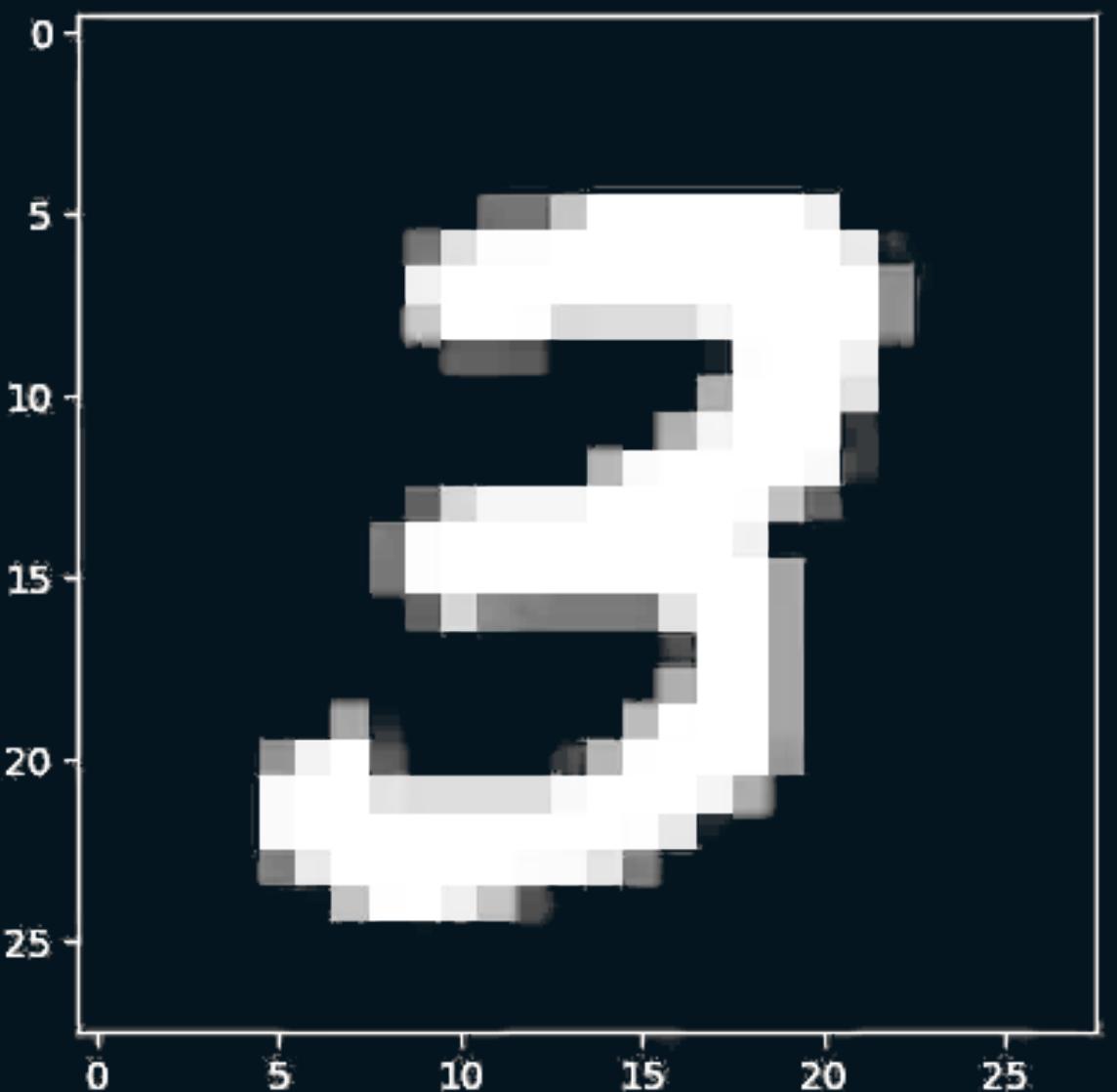
Il modello è stato realizzato in **Python** senza l'uso di librerie o framework che non fossero quelle per l'ottimizzazione di operazioni tra matrici e tensori (**numpy**).





# Il dataset: MNIST

Il dataset utilizzato è quello fornito dal **MNIST** (modified National Institute of Standards and Technology database). Esso contiene un totale di circa 60.000 immagini  $28 \times 28$  di cifre da 0 a 9 scritte a mano.

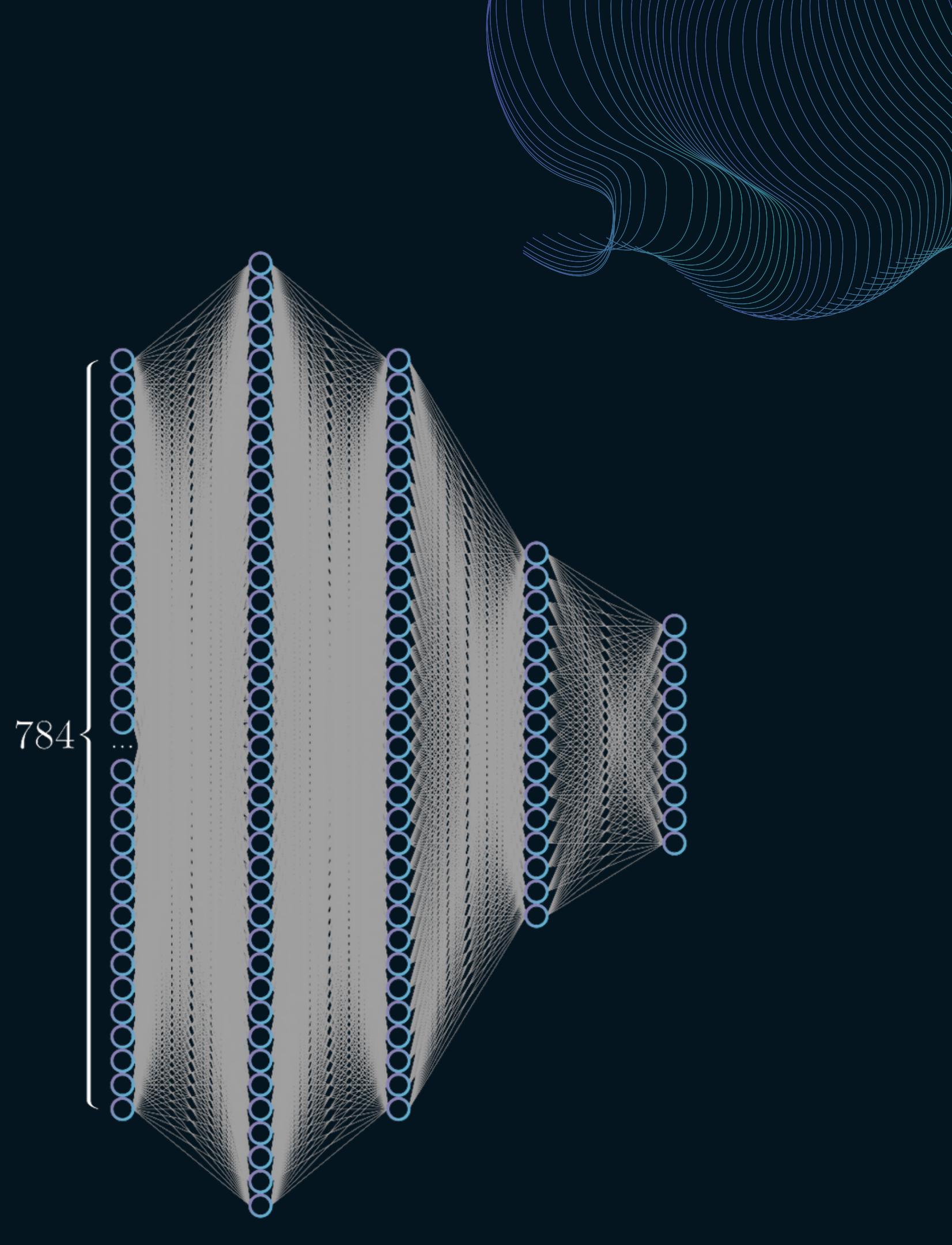




# Il nostro modello

Il modello sviluppato consiste in:

- Uno strato di input di **784** neuroni (uno per ogni pixel)
- Tre strati nascosti di **40, 32, e 16** neuroni
- Uno strato di output di **10** neuroni (uno per ogni cifra)



# Il nostro modello

```
...  
SIZE = (784, 40, 32, 16, 10)  
  
def σ(X):  
    return 1.0 / (1.0 + np.exp(-X))  
  
def y(self, a):  
    for b, w in zip(self.B, self.W):  
        a = σ((w @ a)+b)  
    return a
```

# Il nostro modello

```
def SGD(self, training_data, epochs, mini_batch_size, η, test_data=None):
    if test_data: n_test = len(test_data)
    n = len(training_data)
    for j in range(epochs):
        random.shuffle(training_data)
        mini_batches = [training_data[k:k+mini_batch_size] for k in range(0, n, mini_batch_size)]
        print(f"Learning rate η: {η}")
        for mini_batch in mini_batches:
            self.update_mini_batch(mini_batch, η)
        # η -= η/(epochs+1)
        if test_data:
            print(f"Epoch {j}: {self.evaluate(test_data)} / {n_test}")
        else:
            print(f"Epoch {j} complete")
```

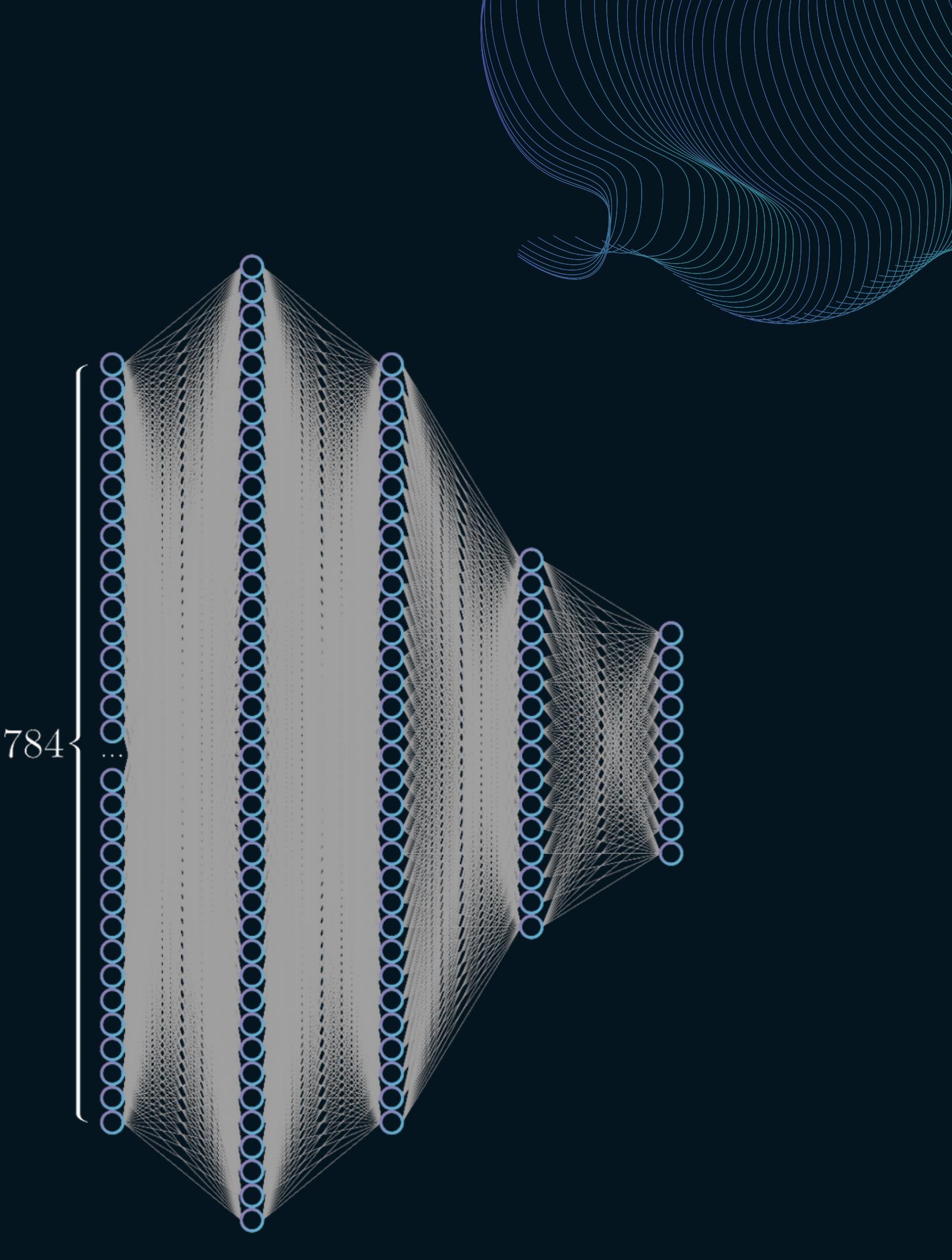


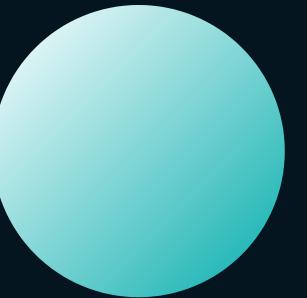
# L'addestramento

L'addestramento si è svolto nei seguenti passaggi:

- 75 epoch con campioni da 10 immagini e  $\eta = 0.063$
- 90 epoch con campioni da 10 immagini e  $\eta = 0.041$
- 150 epoch con campioni da 100 immagini e  $\eta = 0.041$

L'accuratezza raggiunta dal modello è del **92.27%**





# La diagnosi di malattie

---

TOMMASO BARDINI

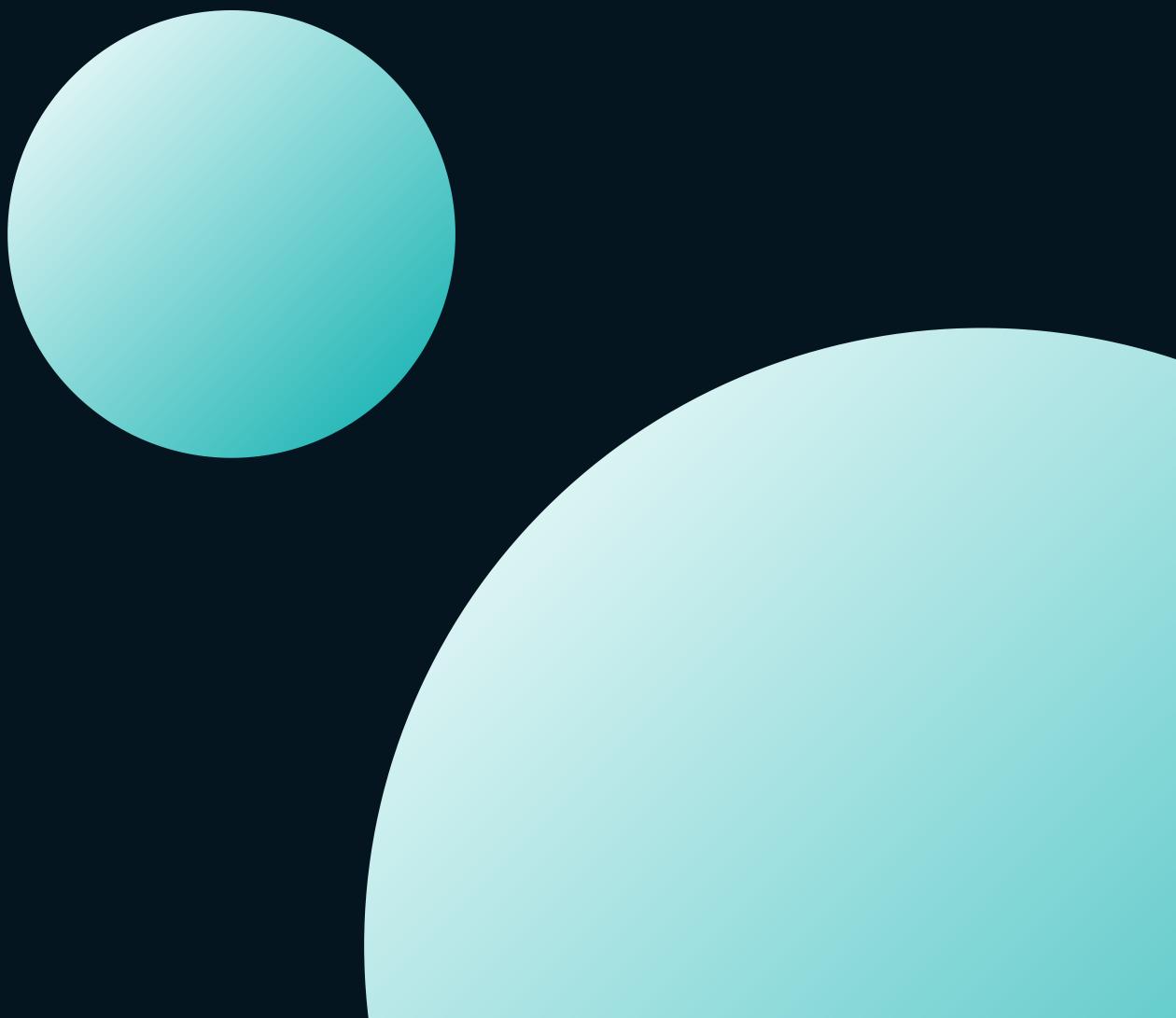




TOMMASO BARDINI

# AI applicata alla diagnosi

L'intelligenza artificiale applicata alla medicina permette di analizzare grandi quantità di dati clinici e supportare i medici nell'identificazione di patologie. Grazie ad algoritmi avanzati come il machine learning e il deep learning, l'IA diventa un valido alleato nella diagnosi, aiutando a riconoscere malattie anche complesse.

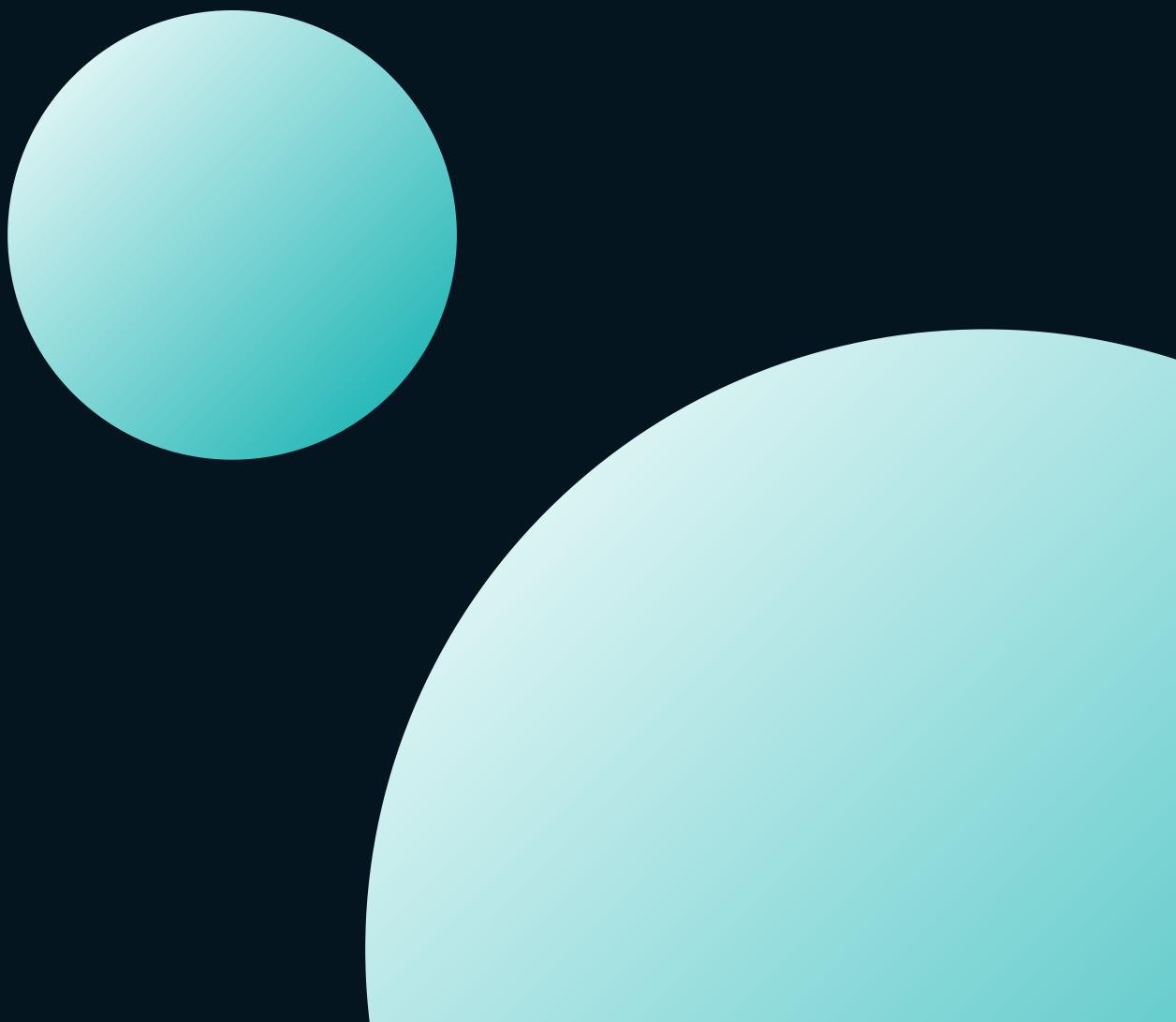




TOMMASO BARDINI

# Cosa può fare l'IA

L'IA è in grado di elaborare enormi quantità di dati medici in tempi molto rapidi, individuando schemi e anomalie che a volte sfuggono all'occhio umano. Può suggerire possibili diagnosi basate su milioni di casi precedenti e prevedere il rischio di malattie future grazie alle sue capacità di analisi predittiva.



# Vantaggi e svantaggi

## Vantaggi:

- Maggiore velocità e precisione
- Alleggerimento del lavoro dei medici

## Svantaggi:

- Possibilità di errore
- Mancanza di intuito e di visione globale

# Rischio di Bias

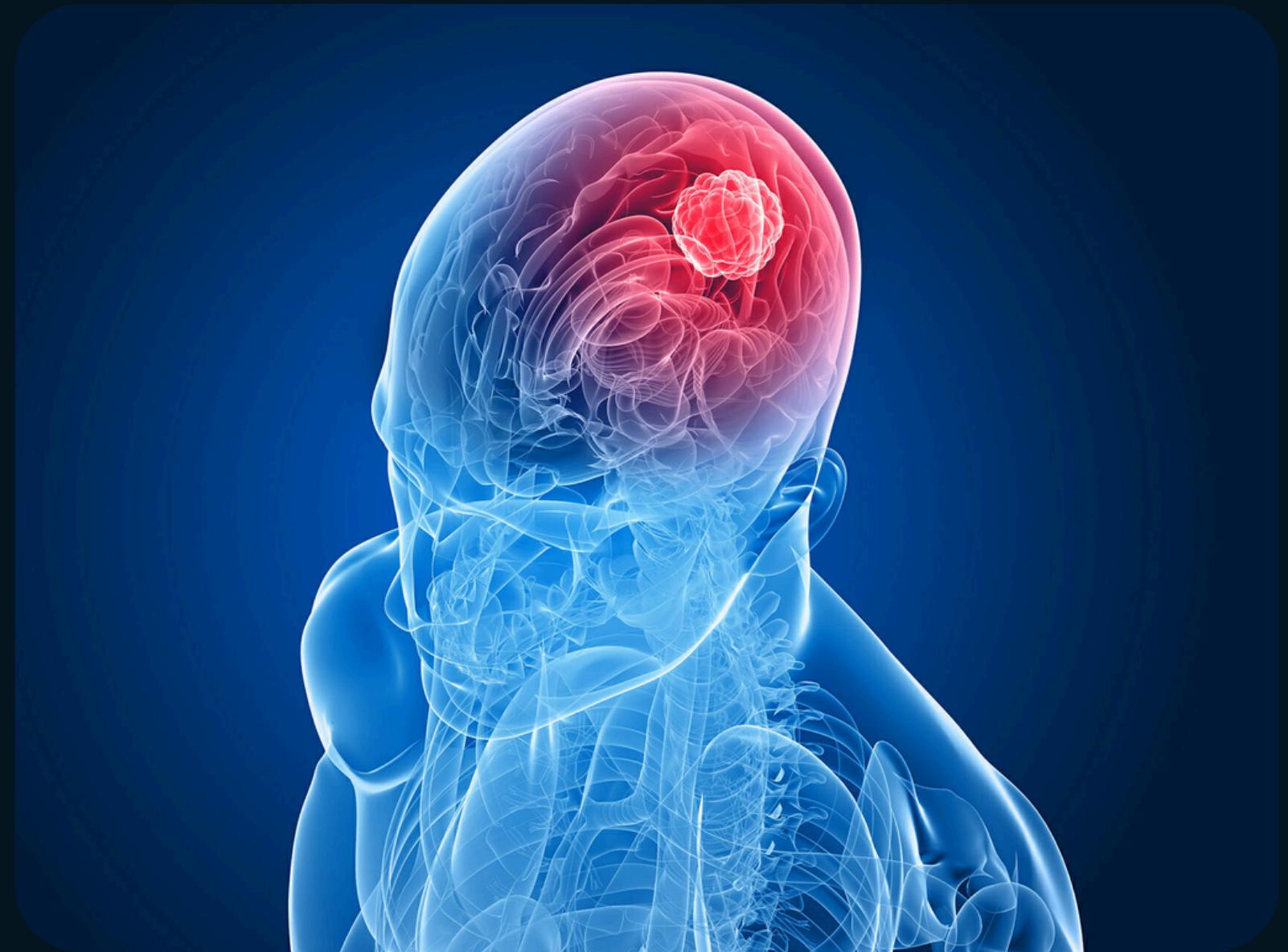
Un grande limite dell'IA è il rischio di bias, ovvero distorsioni nei risultati causate da dati non rappresentativi. Alcuni algoritmi faticano a diagnosticare correttamente su pazienti con pelle scura, donne o persone provenienti da contesti svantaggiati, portando a diagnosi meno accurate per certi gruppi.





# Il tumore cerebrale

I tumori al cervello sono masse anomale di cellule che si replicano in modo incontrollato. Possono originarsi direttamente nel cervello o arrivare da altri organi tramite metastasi. Le forme più comuni sono i gliomi, i meningiomi e i tumori pituari, ciascuno con caratteristiche e pericolosità diverse.

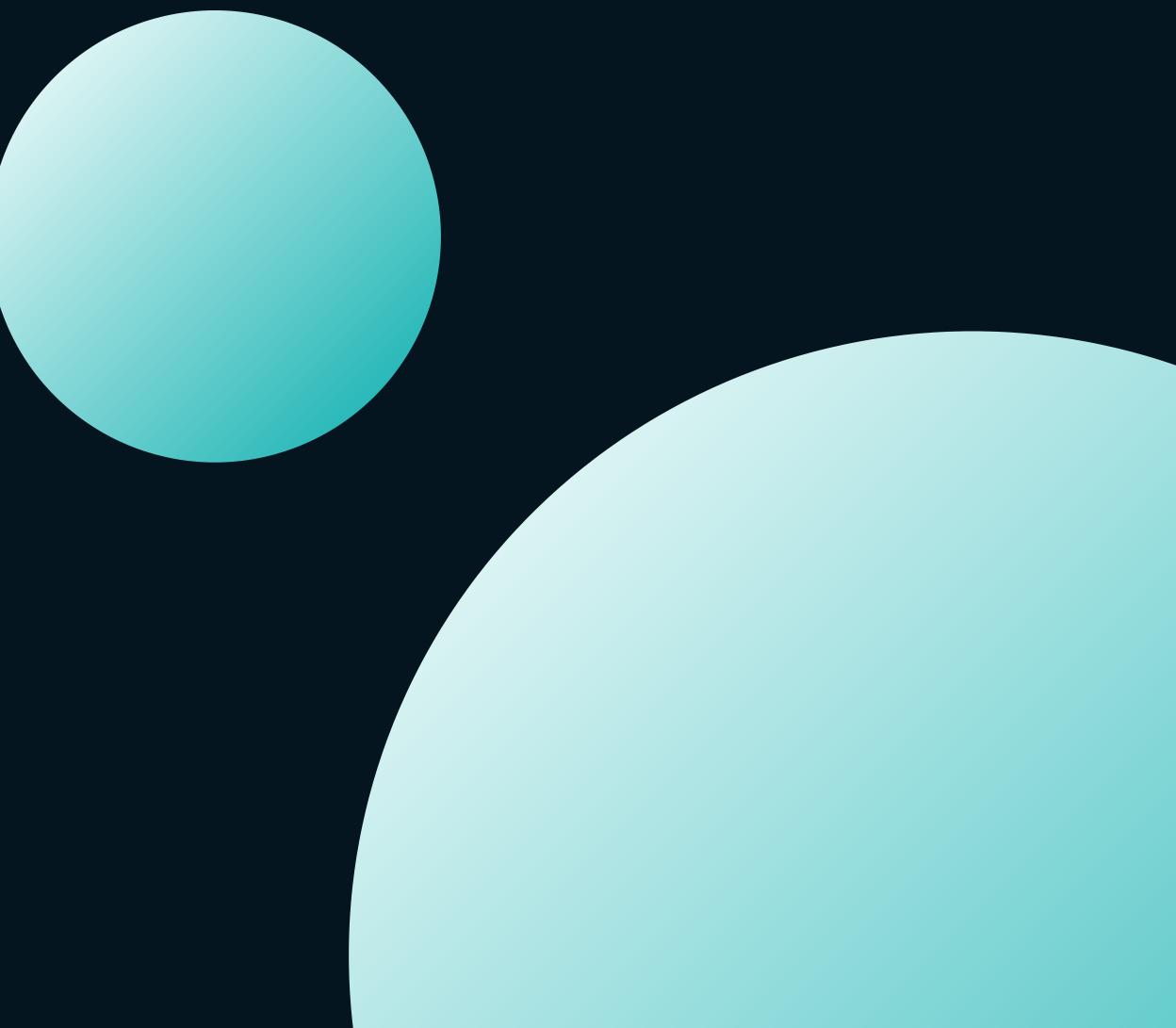


# Tipi principali di tumori

I **gliomi** si sviluppano dalle cellule gliali e diventano pericolosi nei gradi più alti.

I **meningiomi**, più frequenti nelle donne, sono spesso benigni e crescono lentamente.

I **tumori pituari** nascono nell'ipofisi e possono alterare la produzione ormonale, provocando sintomi evidenti e specifici.

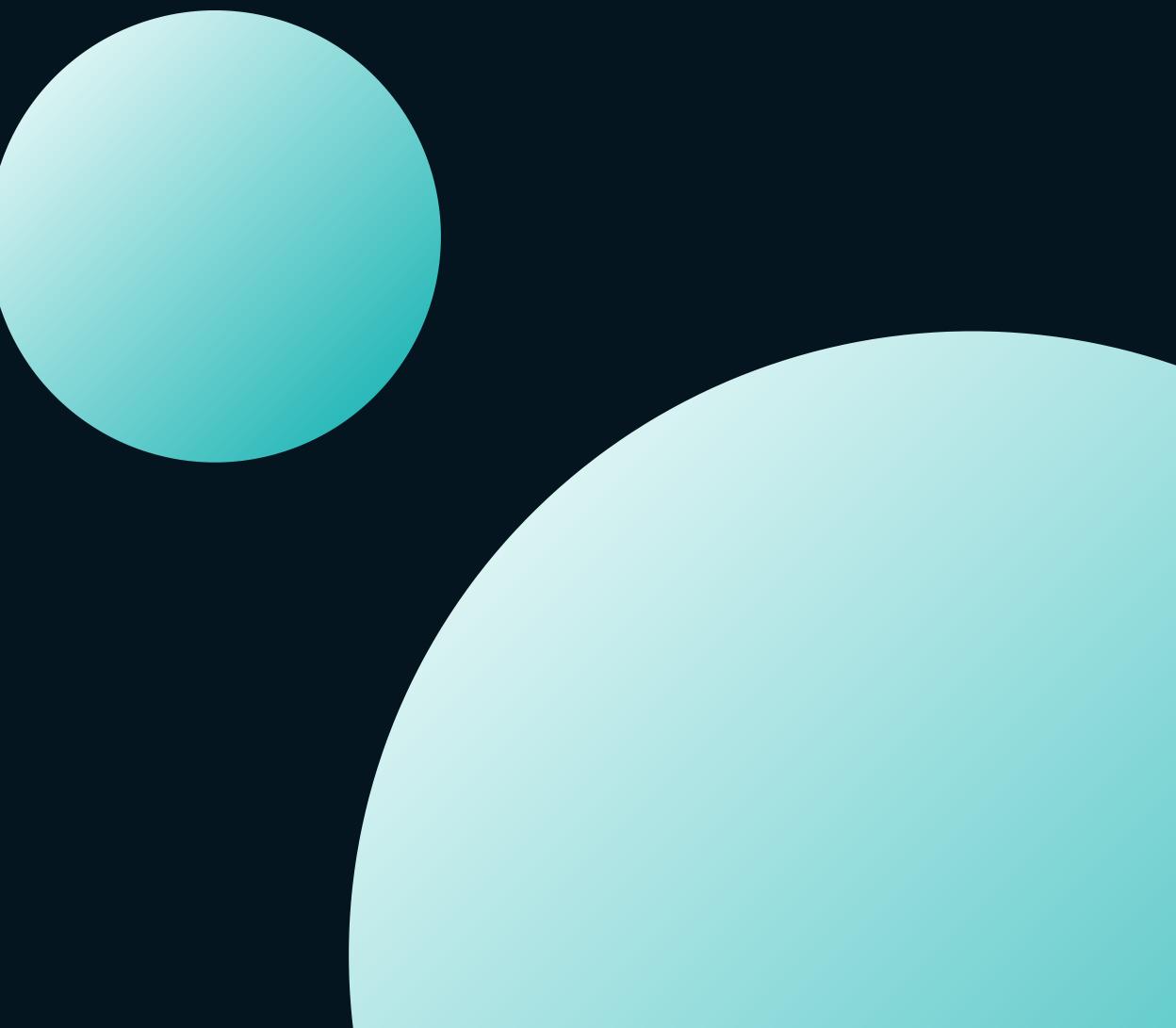




TOMMASO BARDINI

# Progetto pratico con IA

Come dataset Abbiamo utilizzato immagini radiologiche prese da Kaggle che mostrano cervelli sani e con vari tipi di tumore.





# Pytorch e la rete neurale

Pytorch è un framework sviluppato da Microsoft usato per l'ottimizzazione di operazioni algebriche su tensori basato Python. Il modello che abbiamo sviluppato consiste in 5 layer convoluzionali, ognuno passante attraverso una funzione di attivazione ReLU, e un ultimo layer lineare.



# La rete

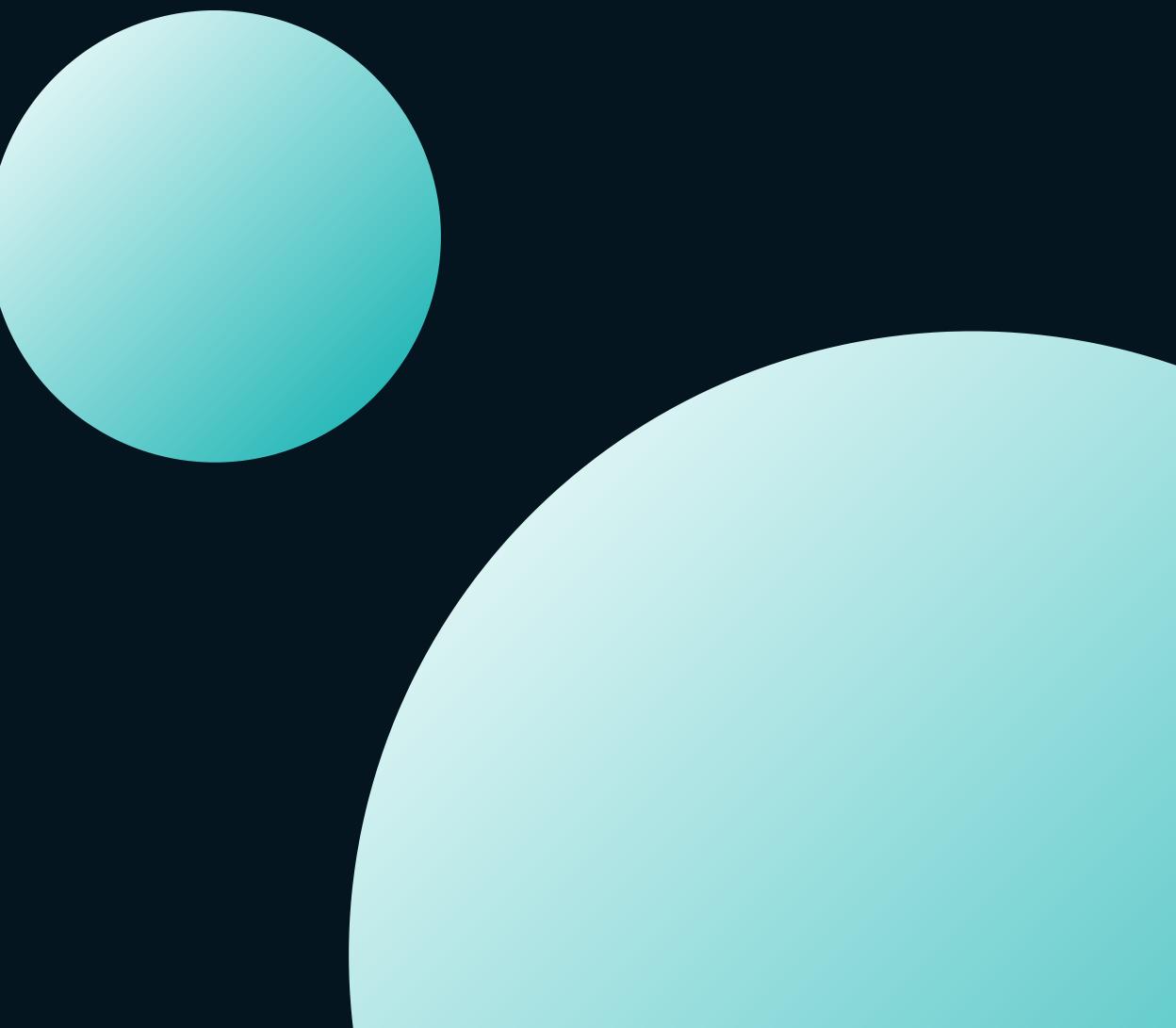
```
• • •

class BrainTumorCNN(nn.Module):
    def __init__(self, num_classes=4):
        super(BrainTumorCNN, self).__init__()
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.dropout = nn.Dropout(0.5)

        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)    # (B, 32, 224, 224)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)   # (B, 64, 112, 112)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1) # (B, 128, 56, 56)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=3, padding=1) # (B, 256, 28, 28)
        self.conv5 = nn.Conv2d(256, 256, kernel_size=3, padding=1) # (B, 256, 14, 14)

        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(256 * 7 * 7, 512)
        self.fc2 = nn.Linear(512, num_classes)
```

# Caratteristiche del network

- Learning rate:  $\eta = 0.0007$
  - Funzione costo: Cross Entropy Loss
  - Numero totale di epoch: 30
  - Numero di immagini per campione: 32
- 

# Caratteristiche

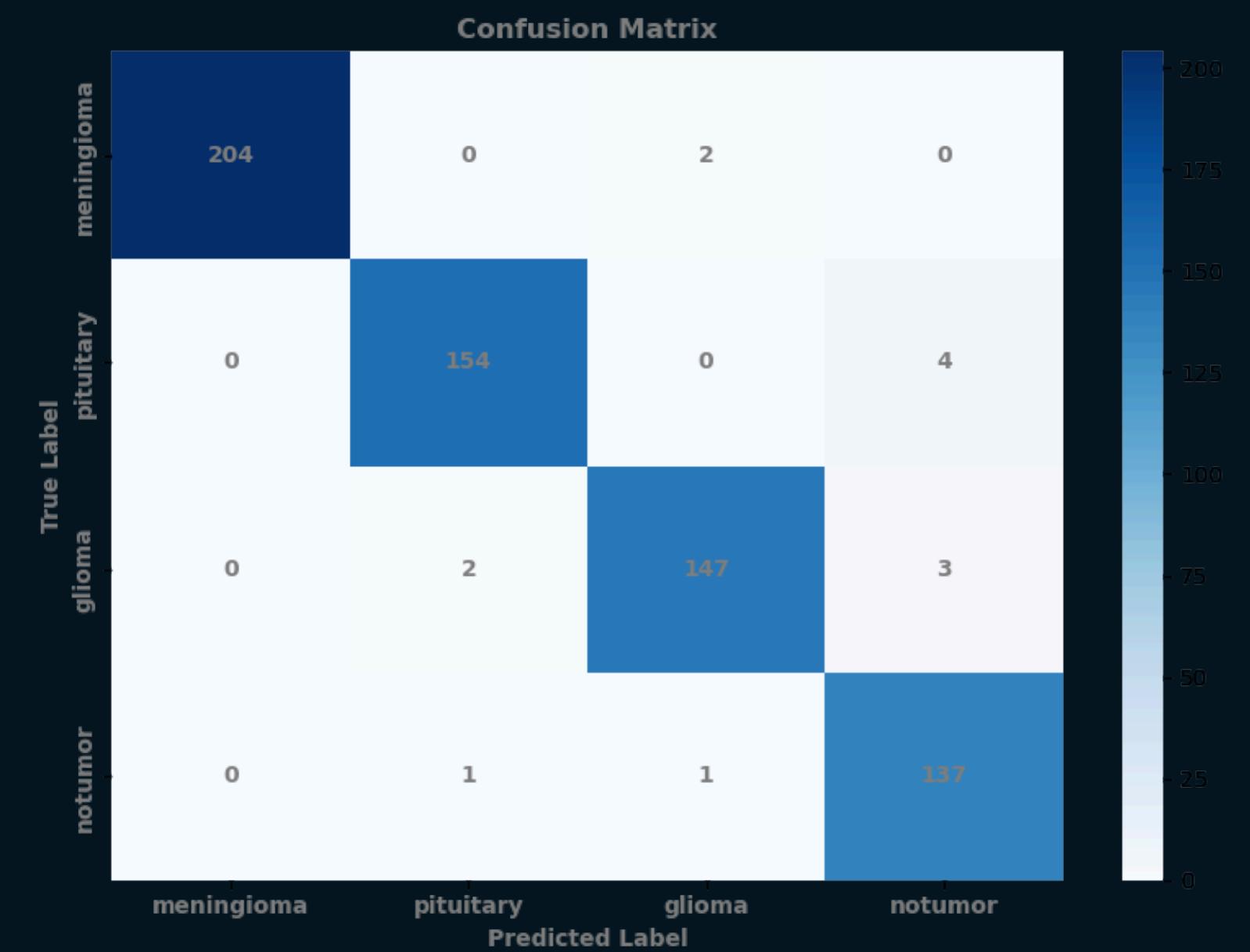
```
target_size = (224, 224)
batch_size = 32
num_classes = 4
label_map = {
    'notumor': 0,
    'glioma': 1,
    'meningioma': 2,
    'pituitary': 3
}

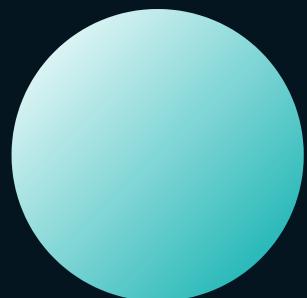
model = BrainTumorCNN(num_classes=4).to(device)

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.0007)
```

# Risultati del modello

Il modello addestrato ha raggiunto una precisione del 98%





# AI e farmacogenomica

---

CRISTIAN TOFFOLON



# Cosa è la farmacogenomica?

*"Branca della biologia che si occupa di studiare come le differenze genetiche delle persone possano influenzare la risposta ai farmaci"*

# Implementazione di AI

Strumenti come l'informatica e la **AI**, in grado di studiare ed elaborare enormi quantità di dati, possono essere sfruttati per la **personalizzazione di trattamenti** di oncologia, cardiologia e psichiatria.



# Raccolta ed uso dei dati genetici e clinici

Una **grande mole di dati** permette all'IA di elaborare modelli predittivi. L'analisi di grandi quantità di informazioni, provenienti da **test genetici e cartelle cliniche**, rende possibile sviluppare terapie personalizzate più efficaci e sicure.



## Banche dati farmacogenomiche

- **PharmGKB:** database che collega varianti genetiche con risposta ai farmaci
- **ClinVar:** annotazioni su varianti genomiche e il loro significato clinico

## Banche dati cliniche

- **Registri sanitari** nazionali e ospedalieri.
- Iniziative di **medicina personalizzata** (es. UK Biobank, All of Us negli USA)

# Personalizzazione con AI

La **AI** non si limita a "*vedere i sintomi*", ma incrocia **centinaia di variabili**: età, sesso, polimorfismi genetici, comorbidità, stile di vita, ecc.



# Personalizzazione con AI



Analisi dei **pattern comuni** per scoprire **correlazioni nascosti** tra geni, farmaci ed esiti terapeutici.



Ridurre il **tempo necessario** per identificare biomarcatori predittivi.



Aiuta a **standardizzare** la medicina personalizzata, rendendola **accessibile su larga scala**.



# Etica del progetto

L'etica in questo progetto non è un ostacolo, ma una **condizione fondamentale**. Solo rispettando la privacy, garantendo accesso equo e assicurando trasparenza e responsabilità, possiamo costruire una medicina personalizzata che sia davvero al **servizio di tutti**.



# Etica del progetto



I dati genetici sono **altamente sensibili**, è fondamentale garantire il consenso informato e applicare misure come **anonimizzazione e cifratura**, nel rispetto del GDPR.



Si rischia di ampliare le **disuguaglianze sanitarie**. Serve promuovere l'inclusione nei dati e nelle politiche che assicurino **pari accesso** alle innovazioni terapeutiche.



Chi risponde degli **errori dell'IA**? La trasparenza, la supervisione umana e l'uso di algoritmi interpretabili sono **condizioni etiche essenziali**.



# Algoritmo dimostrativo

**Farmacogenomica e Intelligenza Artificiale:  
Un modello dimostrativo in PyTorch**

*"Progetto dimostrativo di machine learning per la scelta del farmaco in base a codice genomico e diagnosi"*

# Introduzione al problema

L'obiettivo dell'algoritmo è trovare il farmaco più adatto in base a diagnosi e profilo genetico. Il processo è stato reso **dimostrativo** grazie a questi passaggi:

- Codifica **semplificata** del codice genomico e diagnosi
  - Generazione **sintetica** di dati con pattern logici
  - Addestramento di una rete neurale **semplice** che predice il farmaco ottimale
- 

# Architettura del progetto

Il progetto è organizzato in **moduli indipendenti e riutilizzabili**, ognuno con una responsabilità ben definita.

```
pharmagenomic_demo/
├── data_generation.py      # Generazione dataset simulato
├── encoding.py            # Funzioni di codifica input/output
└── model.py               # Rete neurale
└── run_train.py           # Script di training
```

Struttura del Progetto

# Parametri clinici e genomici considerati

Sono state prese in considerazione tre diverse categorie di **diagnosi**, le varianti genetiche di due **geni** e tre possibili **farmaci raccomandati**.



# Parametri clinici e genomici considerati

```
# Possibili diagnosi cliniche simulate
diagnoses = ["Depressione", "Ipertensione", "Asma"]

# Varianti genetiche per due geni farmacogenetici noti
genes = {
    # Poor, Intermediate, Extensive, Ultra metabolizer
    "CYP2D6": ["PM", "IM", "EM", "UM"],
    "CYP2C19": ["PM", "IM", "EM", "UM"]
}

# Farmaci simulati per la raccomandazione
drugs = ["Sertralina", "Nortriptilina", "Paroxetina"]
```

Parametri clinici e genomici presi in considerazione

# Generazione del Dataset

Questo modello usa un approccio di **apprendimento supervisionato**, apprende a generalizzare le relazioni tra input e output. Necessita di una **logica nascosta** durante la generazione dei dati per poi confrontare l'esito corretto alla predizione.



# Generazione del Dataset

```
# Funzione che genera un singolo campione paziente
def generate_sample():

    ...

    # Logica farmacogenomica fittizia
    if diagnosis == "Depressione":
        if g1 == "PM":
            drug = "Sertralina"
        elif g2 == "UM":
            drug = "Nortriptilina"
        else:
            drug = "Paroxetina"
    elif diagnosis == "Ipertensione":
        drug = "Paroxetina"
    else: # Asma
        drug = "Nortriptilina"

    ...

Logica di generazione
```

# Generazione del Dataset

```
{'diagnosis': 'Depressione', 'CYP2D6': 'PM', 'CYP2C19': 'IM', 'drug': 'Sertralina'}  
{'diagnosis': 'Asma', 'CYP2D6': 'UM', 'CYP2C19': 'UM', 'drug': 'Nortriptilina'}  
{'diagnosis': 'Ipertensione', 'CYP2D6': 'IM', 'CYP2C19': 'PM', 'drug': 'Paroxetina'}  
{'diagnosis': 'Depressione', 'CYP2D6': 'EM', 'CYP2C19': 'IM', 'drug': 'Paroxetina'}  
{'diagnosis': 'Ipertensione', 'CYP2D6': 'EM', 'CYP2C19': 'IM', 'drug': 'Paroxetina'}
```

Esempio Dataset

# Preparazione dei dati

È necessario, per addestrare un **modello ML**, convertire i dati delle diagnosi, dei genomi e dei farmaci in **vettori numerici**.

# Preparazione dei dati

```
# Dizionario per la codifica delle diagnosi
diagnosis2idx = {
    "Depressione": 0,
    "Ipertensione": 1,
    "Asma": 2
}

# Codifica per i metabolizzatori dei due geni
gene_encoding = {
    "PM": 0,
    "IM": 1,
    "EM": 2,
    "UM": 3
}

# Farmaci codificati
drug2idx = {
    "Sertralina": 0,
    "Nortriptilina": 1,
    "Paroxetina": 2
}
```

# Preparazione dei dati

```
def encode_sample(sample):
    """
    Converte un campione da formato testuale a numerico.
    Output: (input_vector, target_label), target_label può essere None se non presente.
    """
    diagnosis_code = diagnosis2idx[sample["diagnosis"]]
    cyp2d6_code = gene_encoding[sample["CYP2D6"]]
    cyp2c19_code = gene_encoding[sample["CYP2C19"]]

    input_vector = [diagnosis_code, cyp2d6_code, cyp2c19_code]

    drug_label = sample.get("drug")
    if drug_label in drug2idx:
        target = drug2idx[drug_label]
    else:
        target = None # nessun target se è solo input da predire

    return input_vector, target
```

# Costruzione del modello

Il cuore del progetto è una **rete neurale fully connected** costruita con PyTorch. Il modello impara ad associare profili clinici/genomici a un farmaco corretto, imitando la logica fittizia usata nella generazione dati.



# Costruzione del modello

```
class PharmaModel(nn.Module):
    def __init__(self, input_dim=3, hidden_dim=16, output_dim=3):
        super(PharmaModel, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

Costruzione del modello di Machine Learning

# Allenamento del modello

Nel **ciclo di addestramento**, ad ogni iterazione, il modello effettua una previsione, calcola l'errore rispetto alla risposta corretta e aggiorna i propri pesi per migliorare. La funzione *loss.backward()* propaga l'errore nei livelli della rete, mentre *optimizer.step()* modifica i pesi.



# Allenamento del modello

```
# 6. Training loop
epochs = 50
losses = []

for epoch in range(epochs):
    outputs = model(X_train)
    loss = criterion(outputs, y_train)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    losses.append(loss.item())

print(f"Epoch {epoch+1}/{epochs}, Loss: {loss.item():.4f}")
```

Loop di allenamento

# Impostazioni dell'allenamento

- Numero di pazienti generati:  **$n = 1000$**
  - Percentuale di dati usati per training/test: **80% train ; 20% test**
  - Numero di layer intermedi: **1 hidden layer**
  - Neuroni intermedi: **16**
  - Learning rate:  **$\eta = 0.01$**
  - Funzione costo: **Cross Entropy Loss**
  - Numero totale di epoch: **50**
- 

# Risultati dell'allenamento

Epoch 1/50, Loss: 1.1266

Epoch 10/50, Loss: 0.7957

Epoch 20/50, Loss: 0.6367

Epoch 30/50, Loss: 0.5203

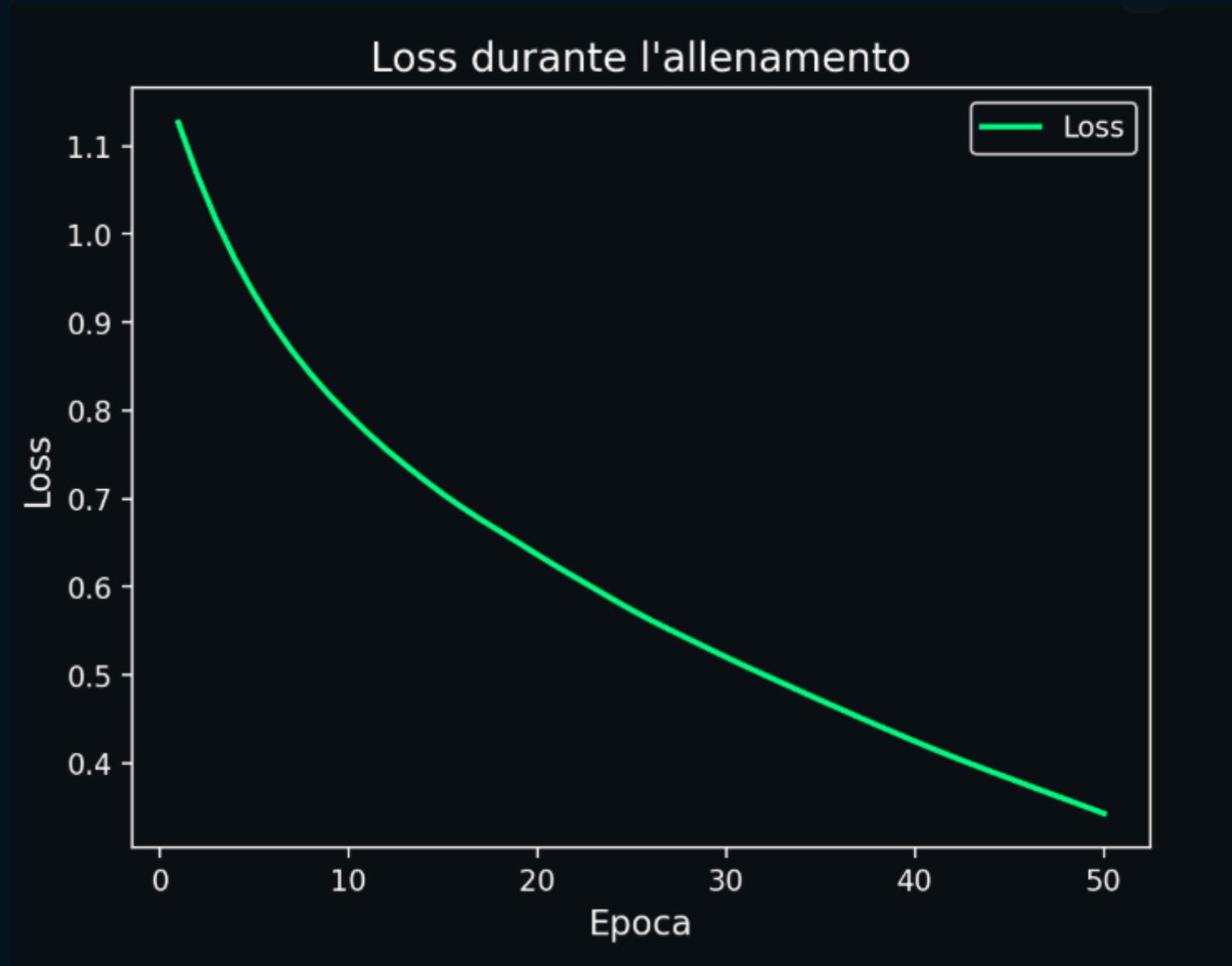
Epoch 40/50, Loss: 0.4248

Epoch 50/50, Loss: 0.3430

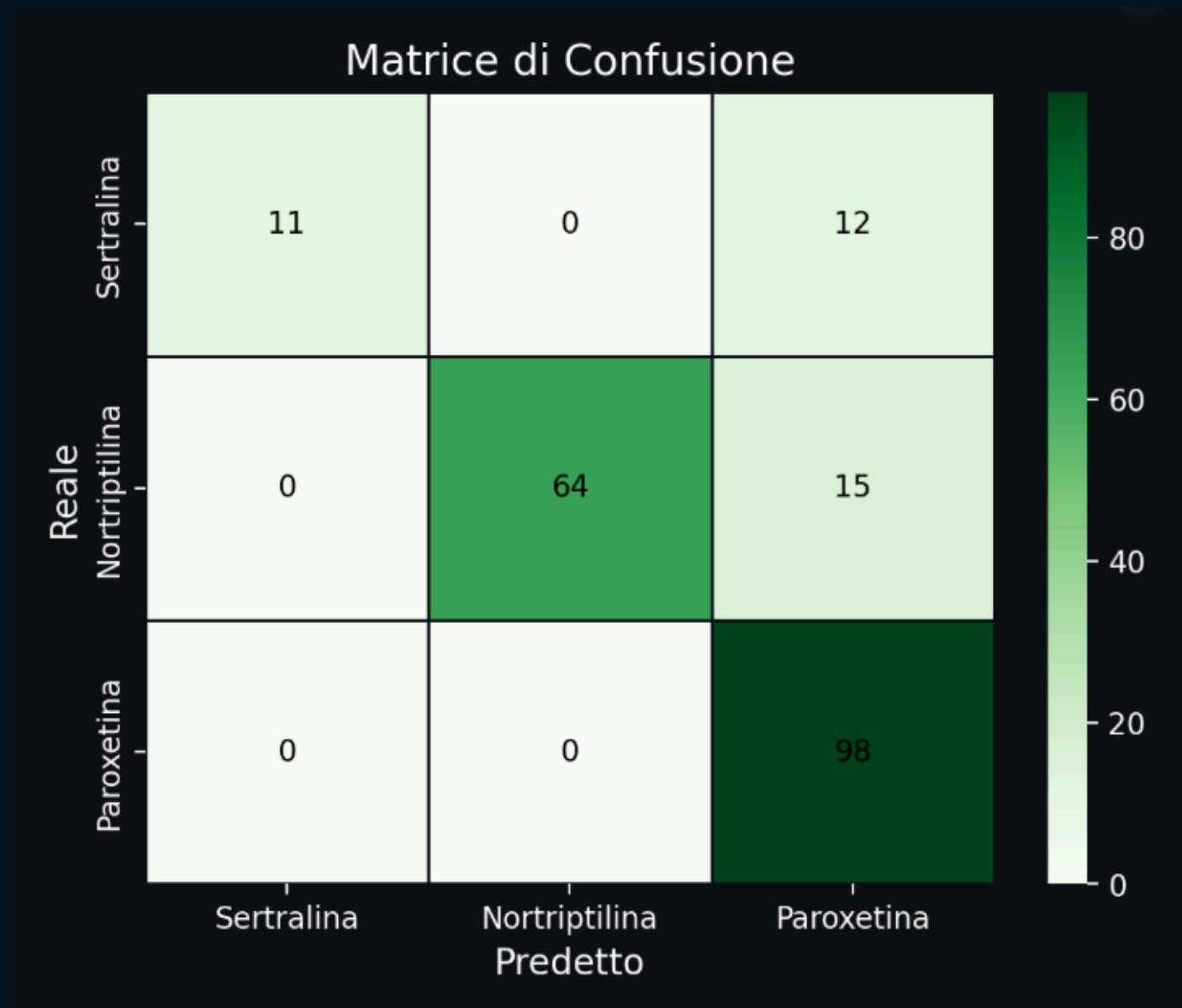
🔍 Accuratezza sul test set: 86.50%

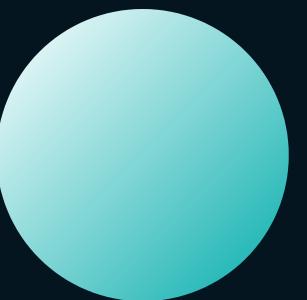
Valutazione test set

# Risultati dell'allenamento



# Risultati dell'allenamento

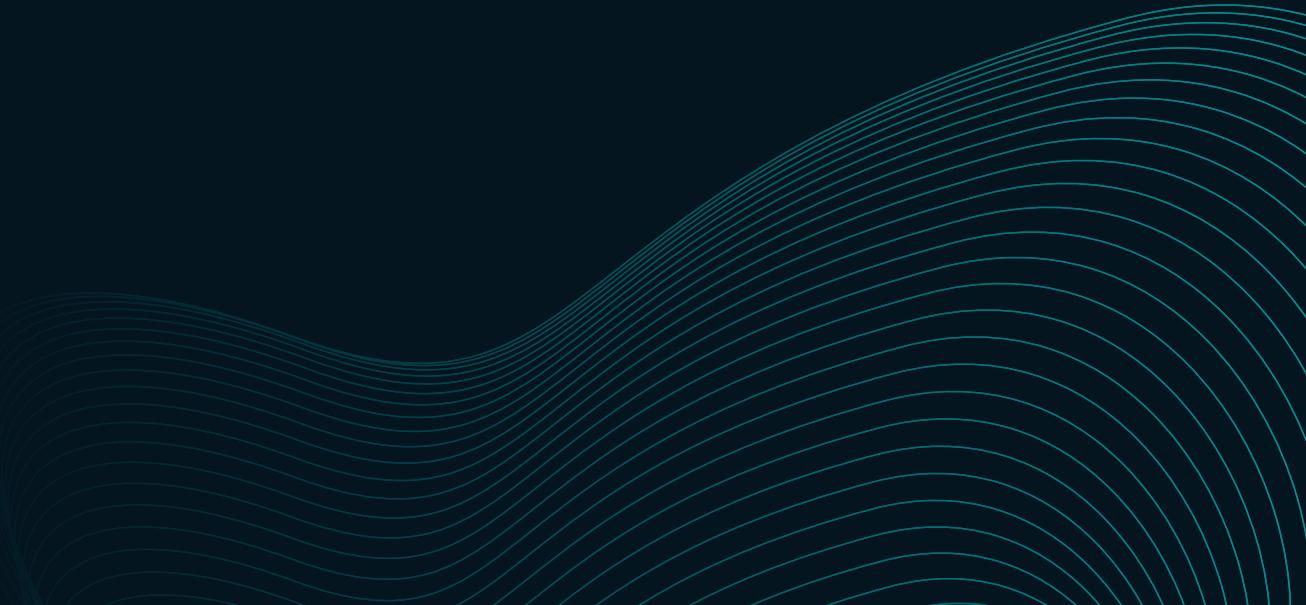




# CV e riconoscimento facciale

---

MANAL TBIBI

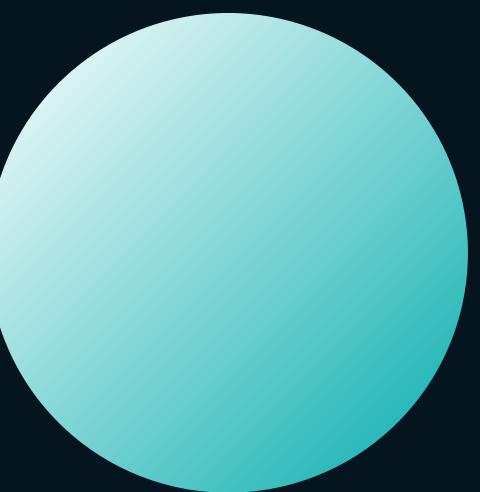




MANAL TBIBI

# L'informatica forense

Branca delle scienze forensi che si occupa dell'identificazione, acquisizione, conservazione, analisi e presentazione di prove digitali in modo legalmente valido e scientificamente rigoroso.

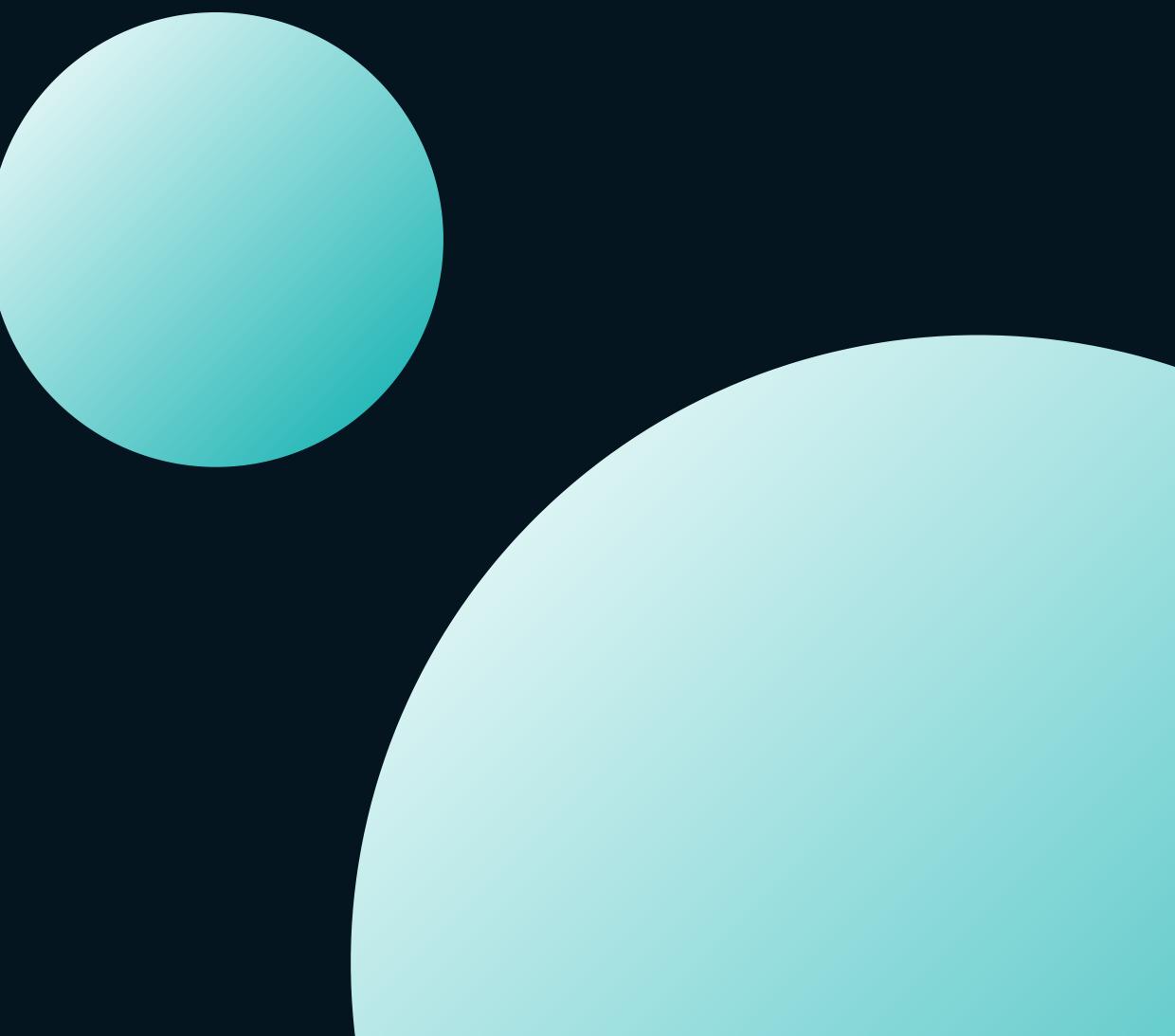




# La biometria forense

Branca dell'informatica forense che applica tecnologie biometriche per l'identificazione e la verifica di individui nell'ambito di indagini legali e processi giudiziari.

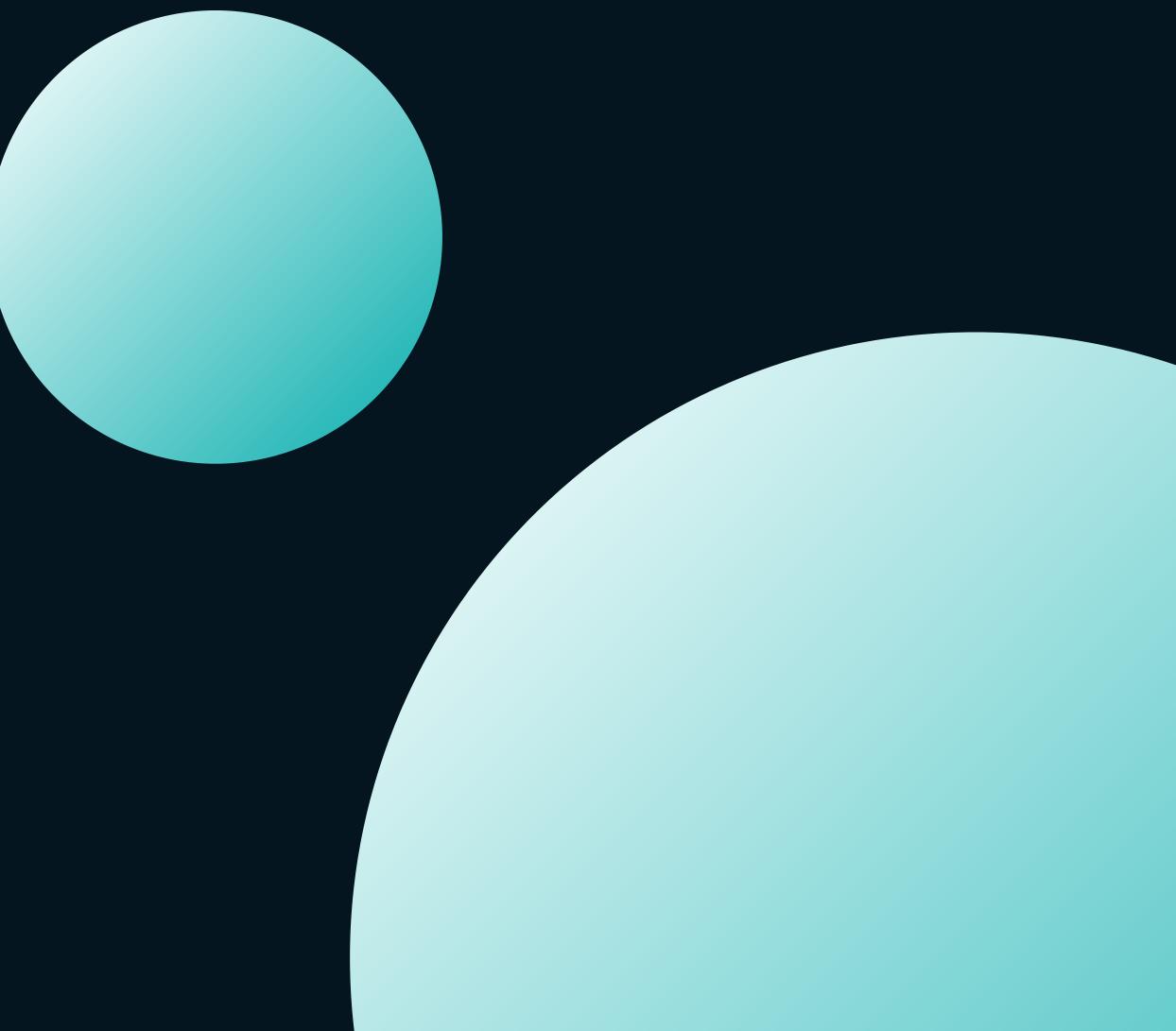
Utilizza caratteristiche biologiche uniche e misurabili per collegare individui a scene del crimine, identificare vittime o sospetti, e fornire prove scientifiche per l'amministrazione della giustizia.





# Il riconoscimento facciale

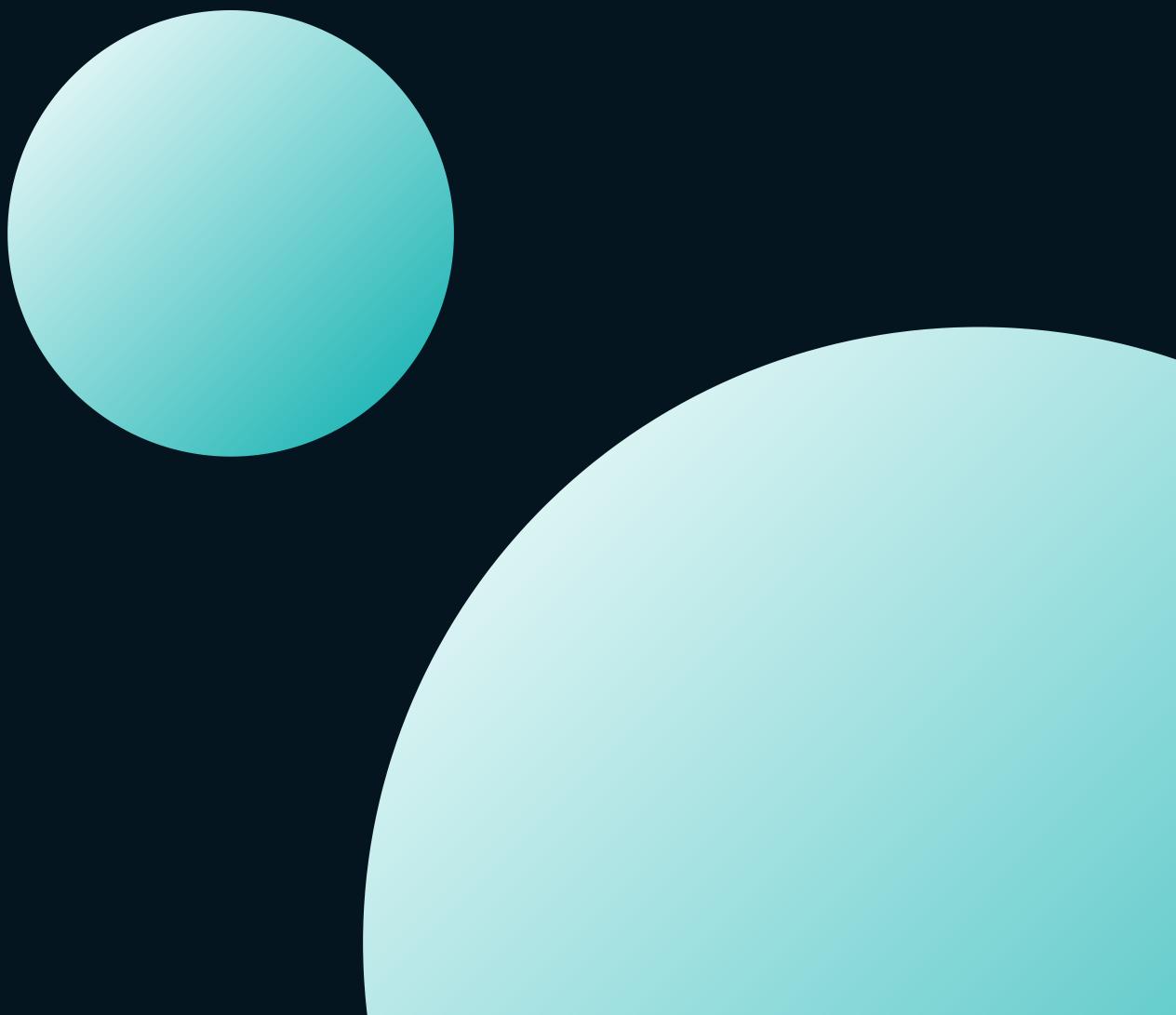
Tecnica di identificazione automatica che utilizza algoritmi di intelligenza artificiale per analizzare le caratteristiche uniche del volto umano e confrontarle con immagini presenti in database biometrici, con lo scopo di identificare o verificare l'identità di un individuo nell'ambito di indagini giudiziarie o procedimenti legali.



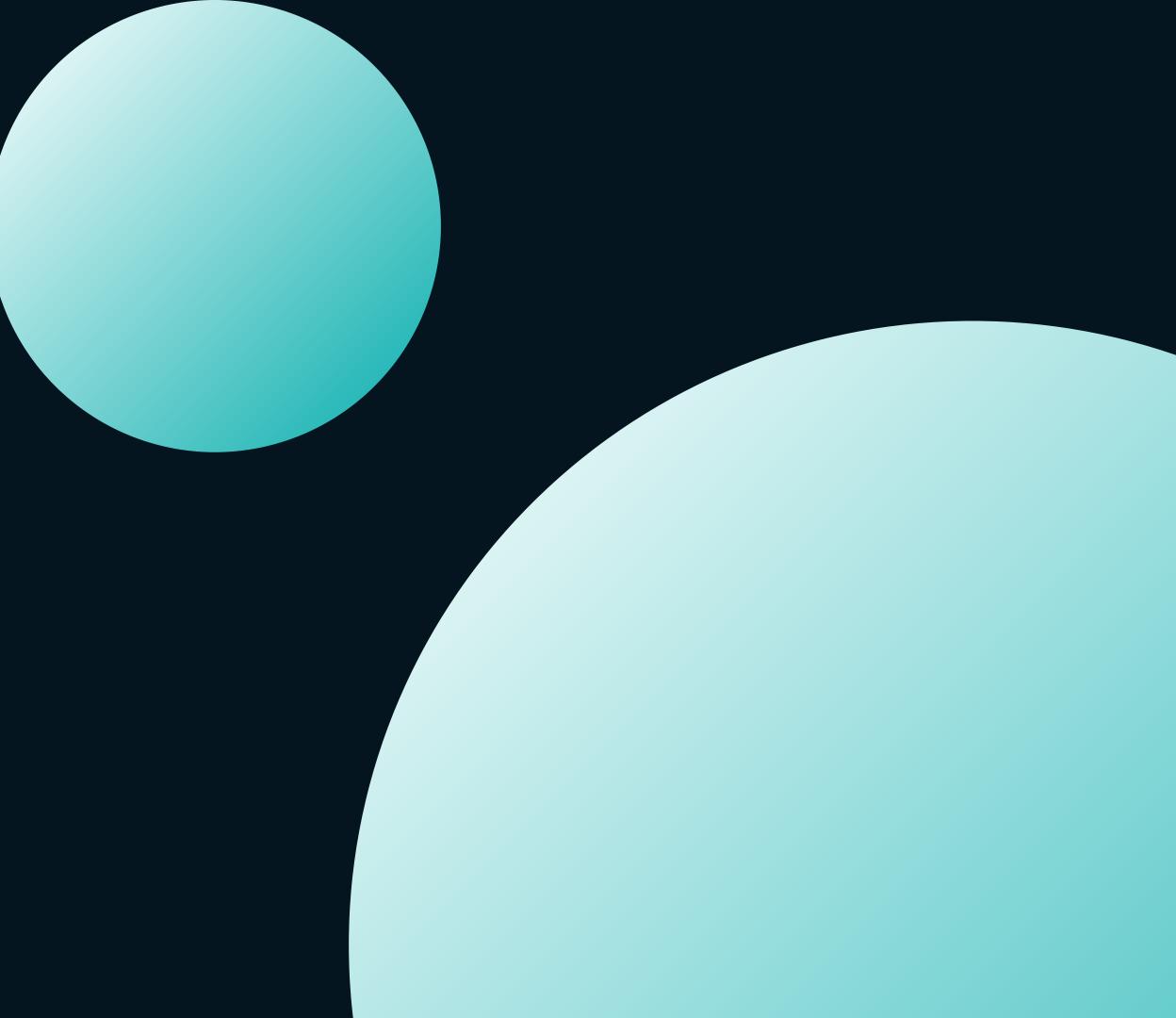


# Fasi del riconoscimento

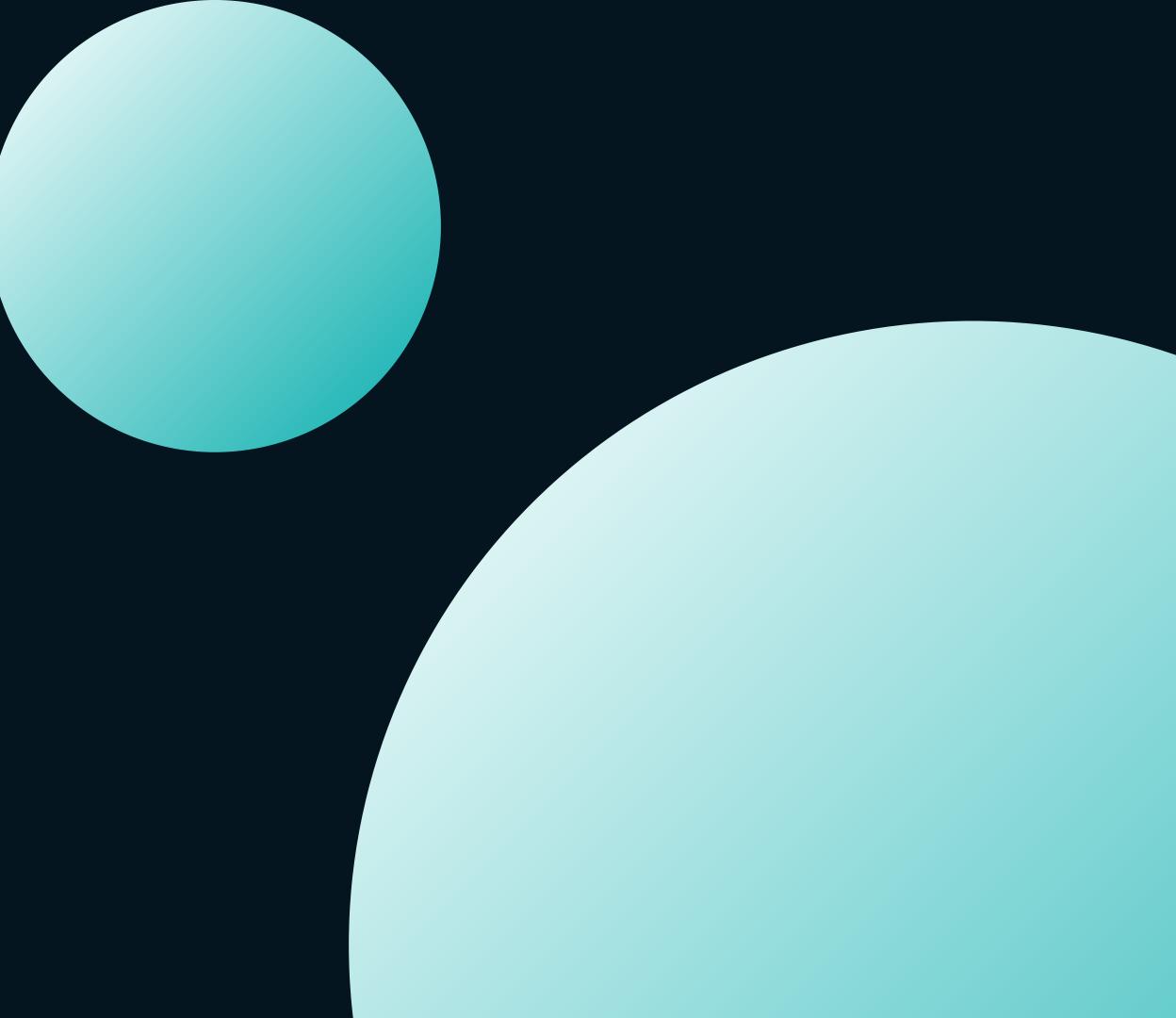
- Rilevamento del volto
- Allineamento del volto
- Estrazione delle caratteristiche
- Corrispondenza
  - Identificazione
  - Verifica



# Database pubblici

- LFW (Labeled Faces in the Wild)
  - CelebA (CelebFaces Attributes Dataset)
  - VGGFace / VGGFace2
- 

# Database forensi

- FBI NGI-IPS (Next Generation Identification - Interstate Photo System)
  - EURODAC (Unione Europea)
  - INTERPOL Facial Recognition System (IFRS)
- 



MANAL TBIBI

# Problematiche



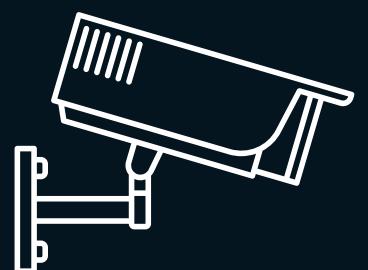
Problemi tecnici



Questioni legali ed etiche



Bias e discriminazioni



Privacy



MANAL TBIBI

# Progetto pratico

Riconoscimento del volto di un sospettato attraverso  
l'utilizzo di OpenCV e Deep Face.



# Omicidi della famiglia Watts

- Nel 2018, Shannan Watts e le due figlie Bella (4 anni) e Celeste (3 anni) spariscono misteriosamente da casa loro in Colorado.
- Chris Watts, inizialmente, dichiara alle autorità di non sapere dove siano e partecipa a ricerche e appelli pubblici per ritrovarle.
- Viene subito considerato persona di interesse dopo alcune incongruenze nei suoi racconti e comportamenti strani.



# Omicidi della famiglia Watts

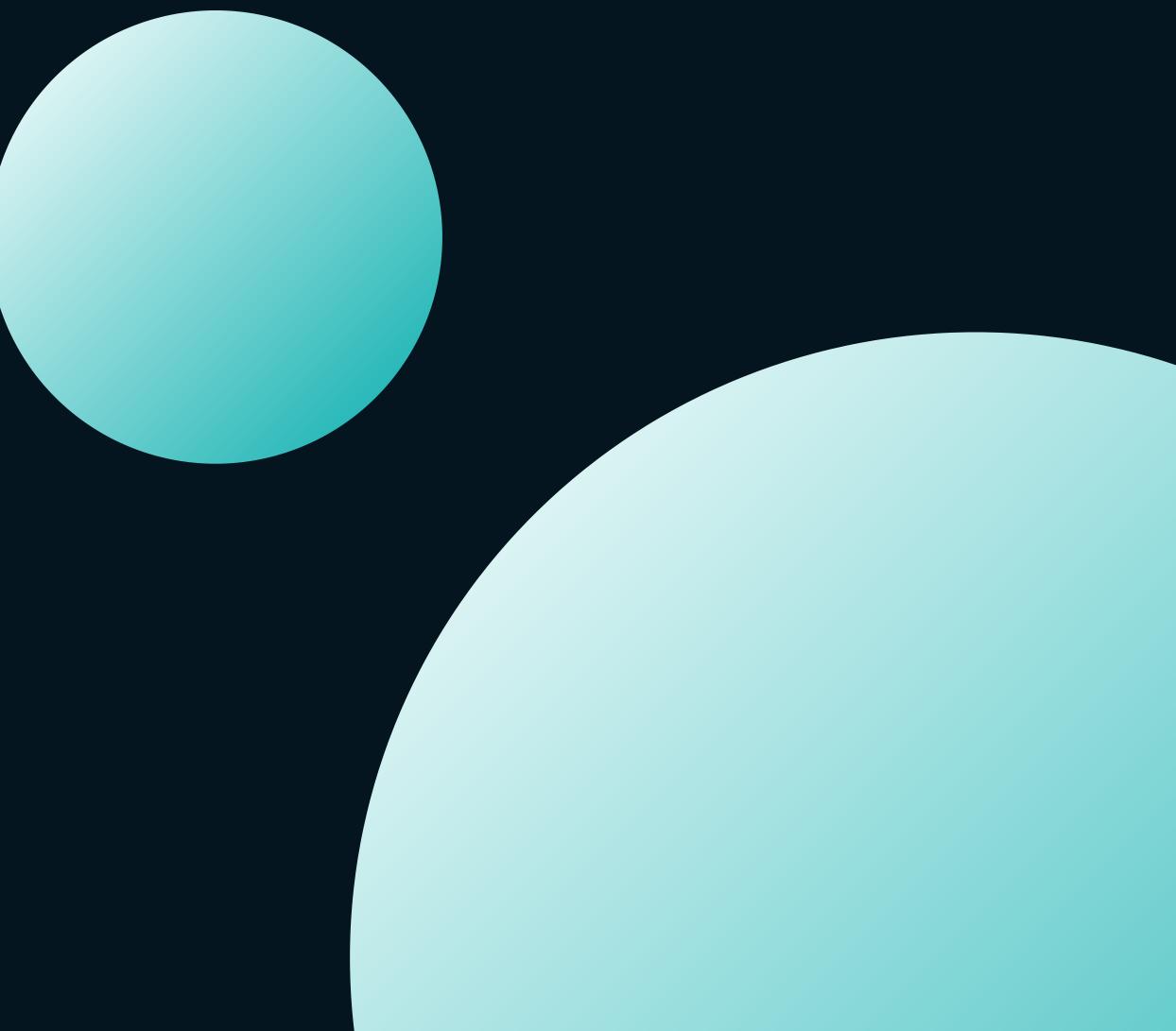
- Dopo alcuni giorni, Chris confessa di aver ucciso la moglie e le figlie in casa, ammettendo di averle soffocate.
- Le vittime sono state trovate sepolte in siti diversi, con Shanann in una fossa e le bambine in un serbatoio di petrolio vicino a casa.
- Nel novembre 2018, Chris Watts accetta un patteggiamento e viene condannato all'ergastolo senza possibilità di libertà condizionale.



MANAL TBIBI

# Computer Vision

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

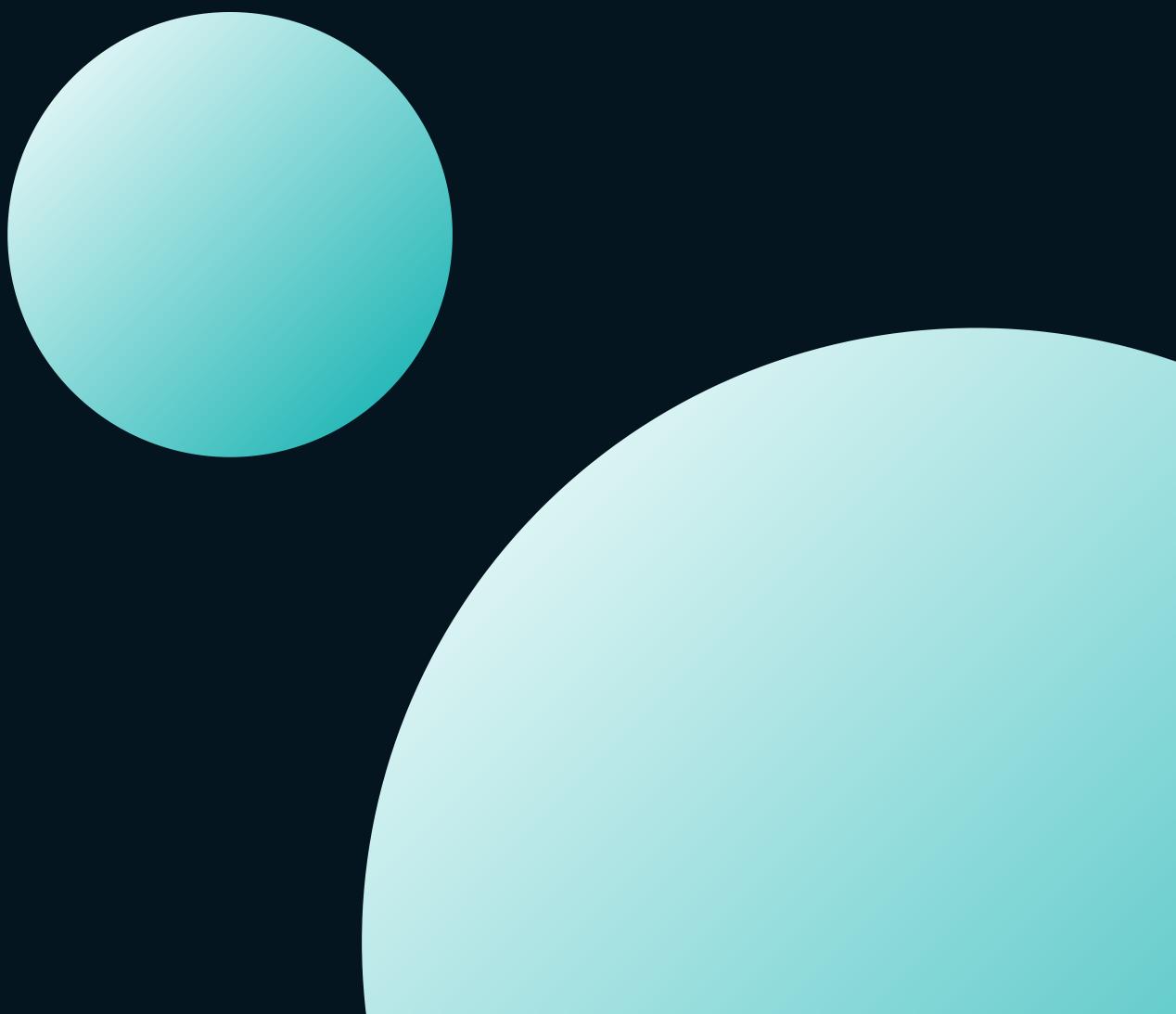




MANAL TBIBI

# OpenCV

Libreria open-source di strumenti software per elaborare immagini e video, usata per realizzare applicazioni di computer vision e machine learning.



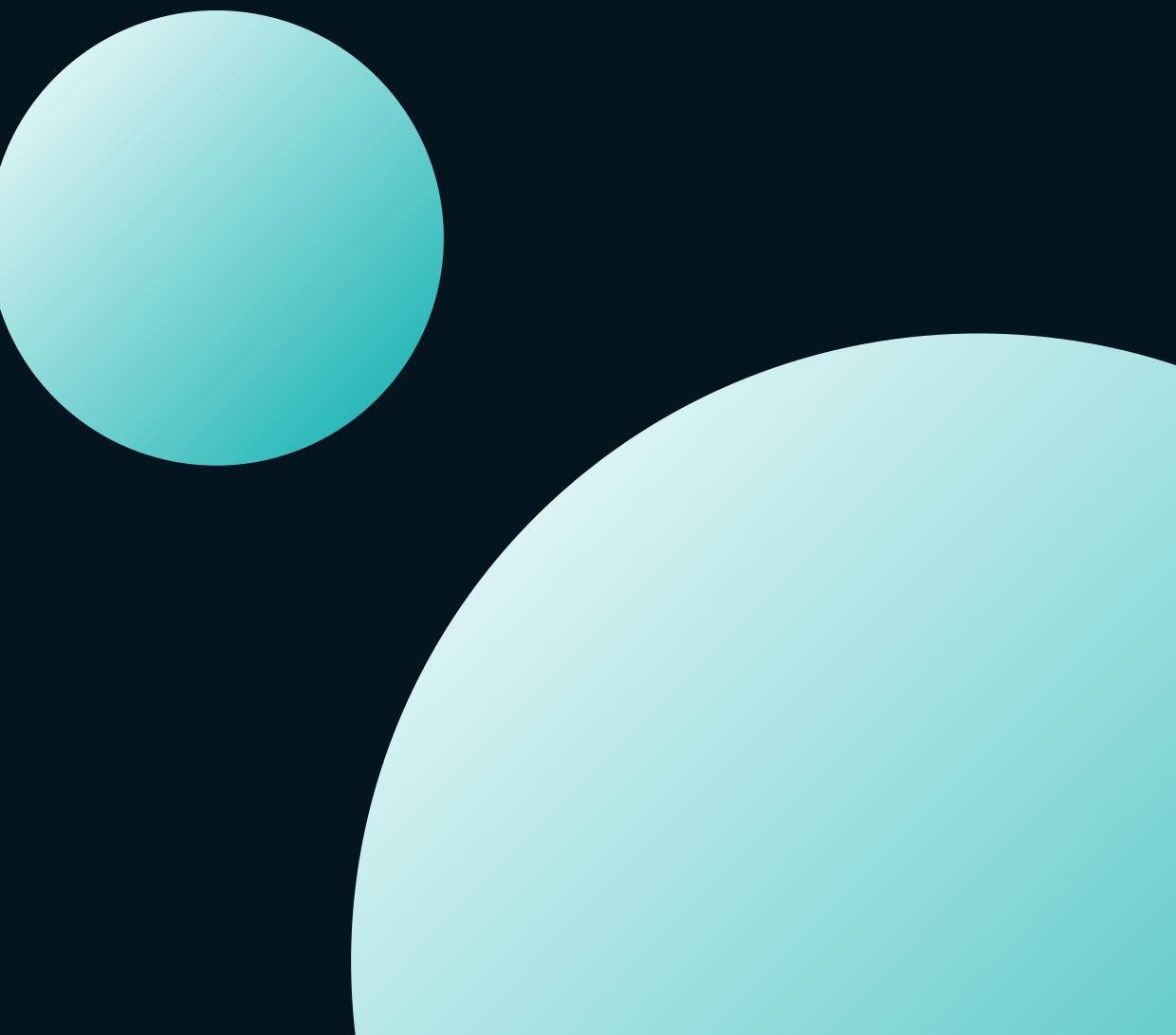


MANAL TBIBI

# DeepFace

Libreria Python open source per il riconoscimento facciale basata su modelli di deep learning pre-addestrati basati su Tensorflow.

Permette di rilevare, riconoscere e analizzare volti con alta accuratezza, supportando funzioni come verifica identità, stima di età, genere ed emozioni.

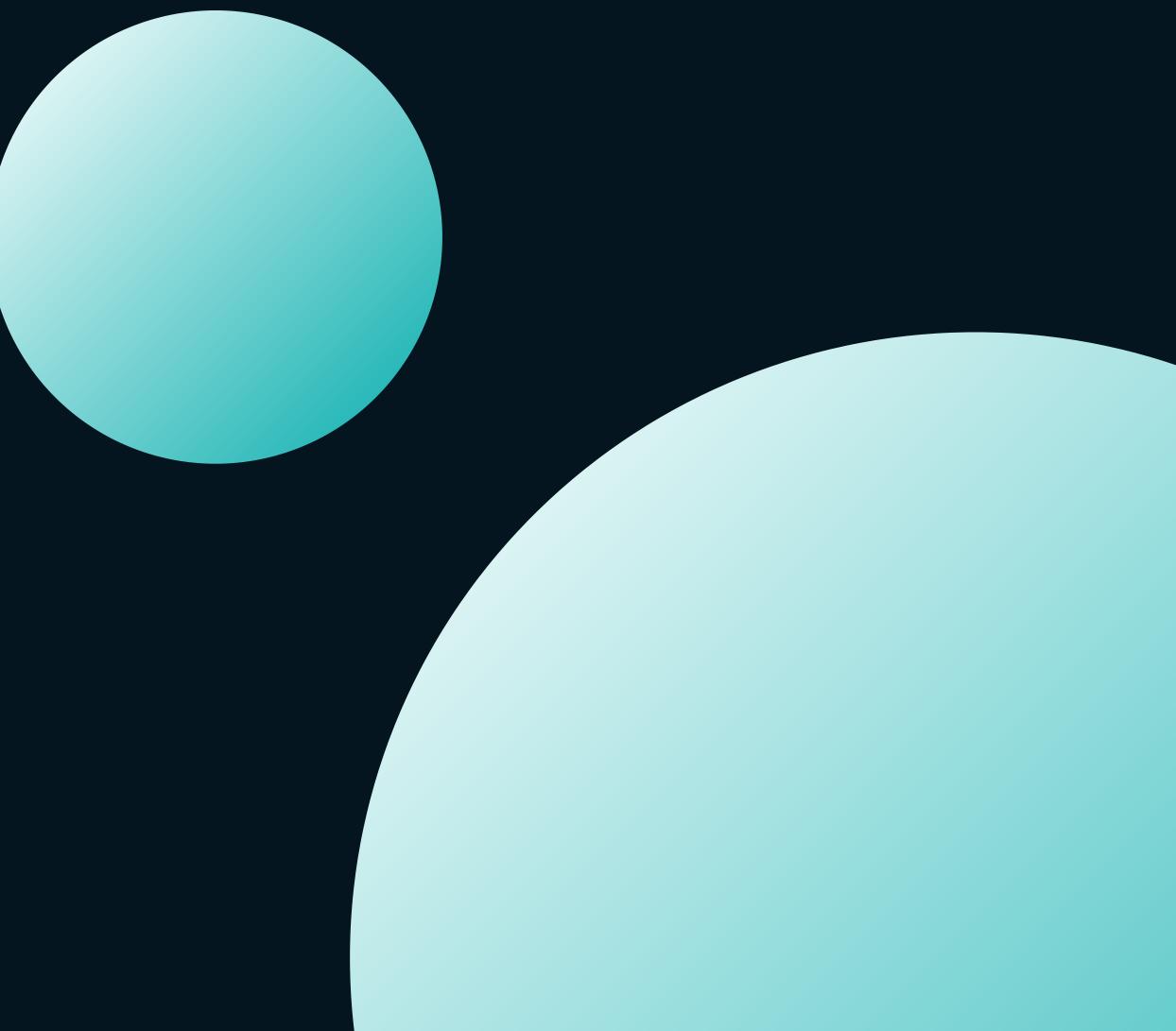




# Embedding

Metodo di rappresentazione dei dati che consente di codificare, attraverso l'uso di reti neurali convoluzionali, le immagini come vettori numerici in uno spazio multidimensionale.

In questo spazio, concetti simili sono rappresentati da punti vicini, mentre concetti diversi sono rappresentati da punti distanti.



# Controllo facciale

```
for (x, y, w, h) in faces:  
    face_crop = frame[y:y+h, x:x+w]  
    try:  
        result = DeepFace.verify(face_crop, reference_img, enforce_detection=False)  
        if result["verified"]:  
            face_box = (x, y, w, h)  
            break  
    except Exception as e:  
        print(f"\nVerification error: {e}")
```

# Individuazione grafica del volto

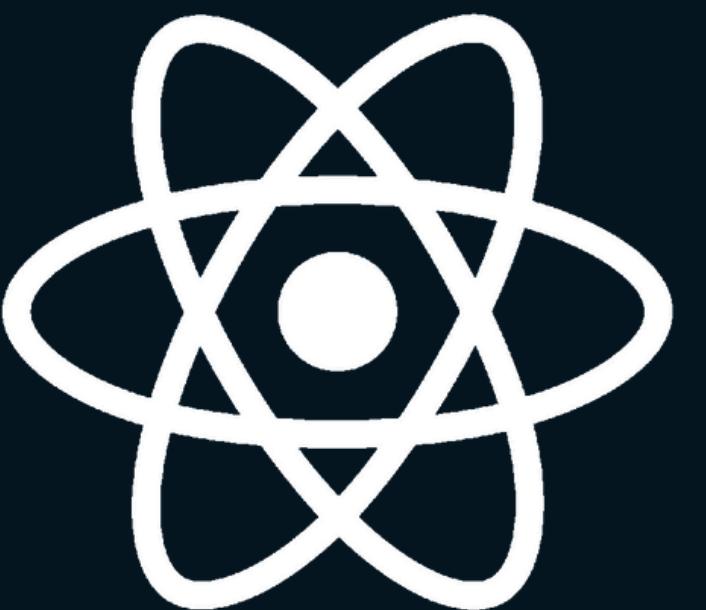
```
...
if face_box:
    x, y, w, h = face_box
    cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 16, 0), 2)

out.write(frame)
frame_count += 1
```



# Sviluppo di una dashboard

La dashboard è stata realizzata in **React** (per il front-end) e **Flask** (per il back-end). È stata usata Material UI come libreria grafica.



Grazie  
dell'attenzione

TOMMASO BARDINI - LUCA FAGARAZ  
MANAL TBIBI - CRISTIAN TOFFOLON