Introduzione al Laboratorio

Per usare un comando Linux in Colab, si mette ! davanti:

```
!echo Hello World
```

```
Hello World
```

Colab indenta normalmente con 2 spazi; se preferite 4 spazi, andate in Strumenti -> Impostazioni -> Editor

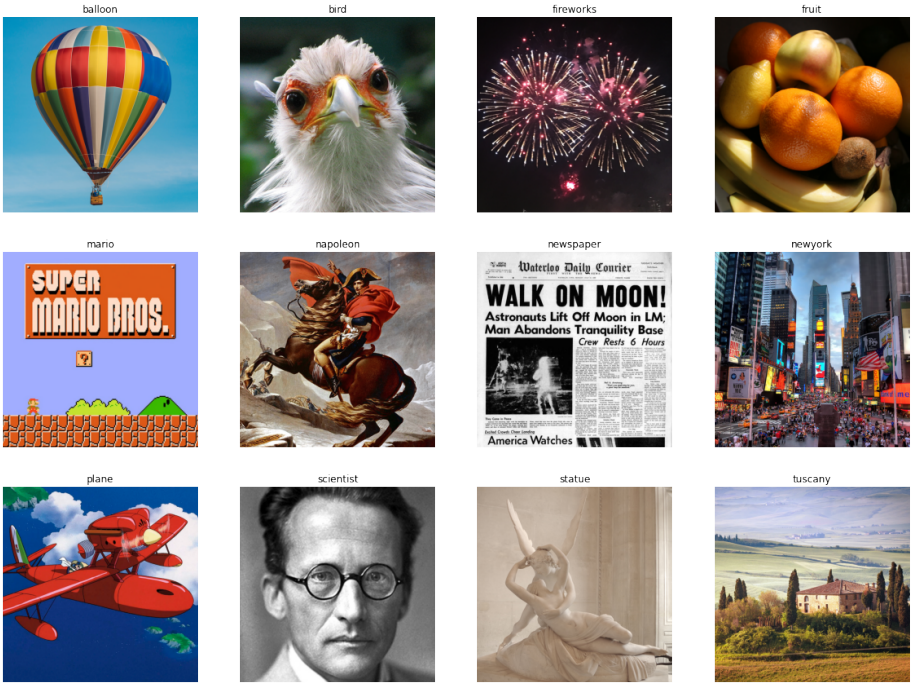
Durante il laboratorio useremo la repository samuelemarro/ml_intro, che contiene un paio di immagini di esempio e qualche funzione per scaricare e visualizzare immagini.

```
!git clone https://github.com/samuelemarro/ml_intro
```

```
fatal: destination path 'ml_intro' already exists and is not an empty directory.
```

```
from ml_intro import utils, samples

utils.plot_image(samples.bird)
```

```
utils.plot_images(samples.standard_images, titles=samples.image_names, columns=4)
```



Laboratorio 1 - Elaborazione Immagini

Tensori

In informatica i tensori (noti anche come array n-dimensionali o ndarray) sono una generalizzazione dei vettori e delle matrici. In particolare:

- Un vettore è un tensore di rango 1 (1-tensore), perché basta un'indice per identificare un elemento
- Una matrice è un tensore di rango 2 (2-tensore), perché servono due indici per identificare un elemento
- Uno scalare è un tensore di rango 0 (0-tensore), perché non servono indici per identificare un elemento (dato che è un unico elemento)

Notazione: [2, 5] vuol dire un 2-tensore le cui dimensioni sono 2 e 5, ovvero una matrice 2x5.

Immagini

Un'immagine a colori è essenzialmente un 3-tensore perché servono 3 indici: y, x e il colore (Red, Green, Blue). Quindi un'immagine HD 16:9 (che ha larghezza 1280 e altezza 720) avrà dimensioni [720, 1280, 3]. Nota che l'altezza viene prima. La coordinata (0, 0) corrisponde all'angolo in alto a sinistra.

Spazi di Colore

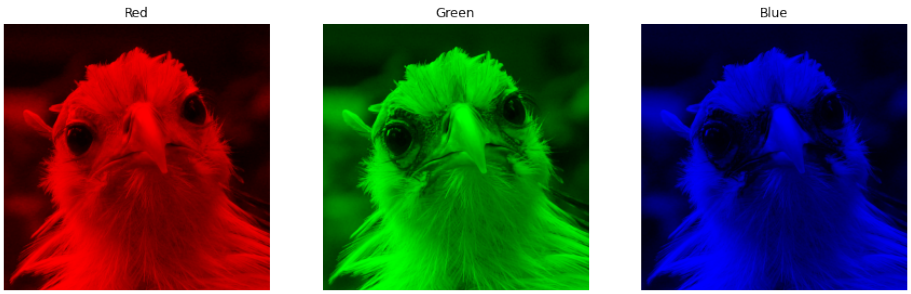
Il modo in cui si associa un colore a una serie di numeri viene detto *spazio di colore*. Il più comune è RGB (Red, Green e Blue), che viene usato per gli schermi, ma esistono anche RBGA (che aggiunge Alpha, il canale per la trasparenza), Grayscale (che usa un solo canale, la luminosità), CMYK (Cyan, Magenta, Yellow e Key, usato per le stampanti), HSV e molti altri.

RGB si basa sul fatto che sommando luce rossa, verde e blu si possono ottenere tutti i colori:

Se la nostra immagine è in RGB, possiamo applicare funzioni matematiche ai valori di rosso, verde e blu. Per esempio:

Isolare Canali

```
red = samples.bird.copy() # Cloniamo per evitare di modificare l'immagine originale
red[:, :, 1] = 0 # Imposta il canale 1 (Verde) a 0
red[:, :, 2] = 0 # Imposta il canale 2 (Blu) a 0

green = samples.bird.copy()
green[:, :, 0] = 0 # Imposta il canale 0 (Rosso) a 0
green[:, :, 2] = 0 # Imposta il canale 2 (Blu) a 0

blue = samples.bird.copy()
blue[:, :, 0] = 0 # Imposta il canale 0 (Rosso) a 0
blue[:, :, 1] = 0 # Imposta il canale 1 (Verde) a 0

utils.plot_images([red, green, blue], titles=['Red', 'Green', 'Blue'], columns=3)
```



Filtro rosso

```
import numpy as np

def red_filter(image):
    image = image.copy() # Per evitare di modificare l'immagine originale

    # Se aggiungessimo direttamente un numero a un'immagine a 8 bit, andremmo in overflow
```

```
    # Quindi convertiamo in interi a 16
    image = image.astype(np.uint16)

    image[:, :, 0] = image[:, :, 0] + 40 # Aggiungi 40 a [ogni y, ogni x, rosso]
    image = np.clip(image, 0, 255) # Se alcuni pixel sono > 255, impostali a 255
    return image.astype(np.uint8) # Riconverti a 8 bit

utils.plot_images([samples.bird, red_filter(samples.bird)])
```

Interpolazione di due immagini

```
def interpolate(image1, image2, coefficient): # 0 = tutto image1, 1 = tutto image2
    interpolated = image1 * (1 - coefficient) + image2 * coefficient
    # interpolated ora è un'immagine floating point, riconvertiamola in interi
    return interpolated.astype(np.uint8)

utils.plot_images([samples.bird, samples.tuscany, interpolate(samples.bird, samples.tuscany,
```

Schiarisci e scurisci

```
def darken(image):
    # Come prima, dobbiamo convertire in un formato che rappresenti correttamente i numeri
```

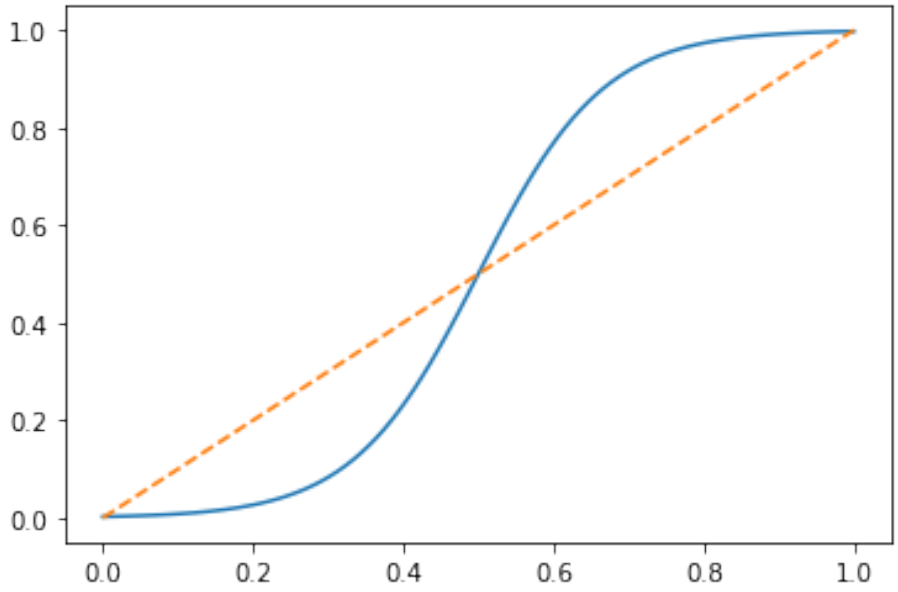
```
    image = image.astype(np.int16)
    image = image - 40 # Riduci rosso, verde e blu di 40
    return np.clip(image, 0, 255).astype(np.uint8) # Se alcuni valori sono scesi sotto zero
```

```
image = samples.bird
utils.plot_images([samples.bird, darken(samples.bird)])
```

Effetto Polaroid

Le Polaroid rendono i colori chiari più chiari e i colori scuri più scuri. Si può ottenere una buona approssimazione di questo effetto usando una variante della funzione logistica:

```
def scaled_logistic(x):
    return 1 / (1 + np.exp(-12 * x + 6))

x = np.linspace(0, 1, 100)
y = scaled_logistic(x)
import matplotlib.pyplot as plt
plt.plot(x, y)
plt.plot(x, x, '--') # Mostra la retta y = x per confronto
plt.show()
```




```
def polaroid(image):
    image = image.astype(np.float32) / 255 # Converti in range [0, 1]
    image = scaled_logistic(image)
    image = (image * 255).astype(np.uint8) # Riconverti in range [0, 255]
    return image
utils.plot_images([samples.bird, polaroid(samples.bird)])
```

Interpolando l'immagine originale con la versione Polaroid si può modificare l'intensità dell'effetto:

```
def parametric_polaroid(image, coefficient):
    return interpolate(image, polaroid(image), coefficient)
```

```
images = []
titles = []

for coefficient in np.linspace(0, 1, num=6):
    images.append(parametric_polaroid(samples.bird, coefficient))
    titles.append(str(coefficient))

utils.plot_images(images, titles=titles, columns=3)
```

Grayscale

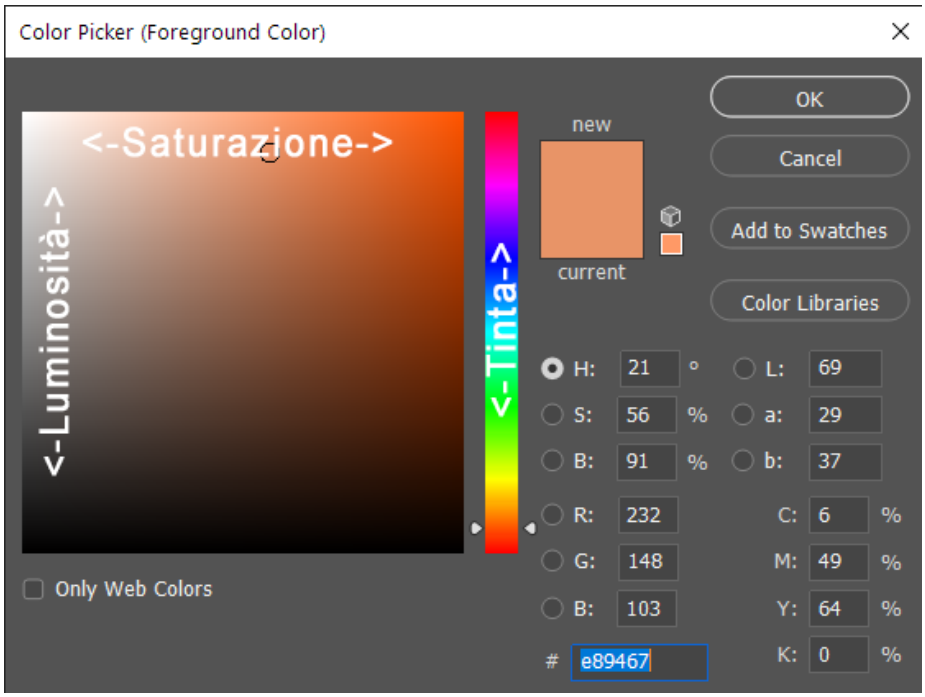
Grayscale è un formato di file che usa un solo canale: la luminosità. Per questa ragione, le dimensioni di un'immagine grayscale sono [altezza, larghezza], visto che non ha senso avere un indice extra per scegliere il canale.

```
import cv2 as cv

def to_grayscale(image):
    return cv.cvtColor(image, cv.COLOR_RGB2GRAY)

utils.plot_image(to_grayscale(samples.bird), space='gray')
```

HSV

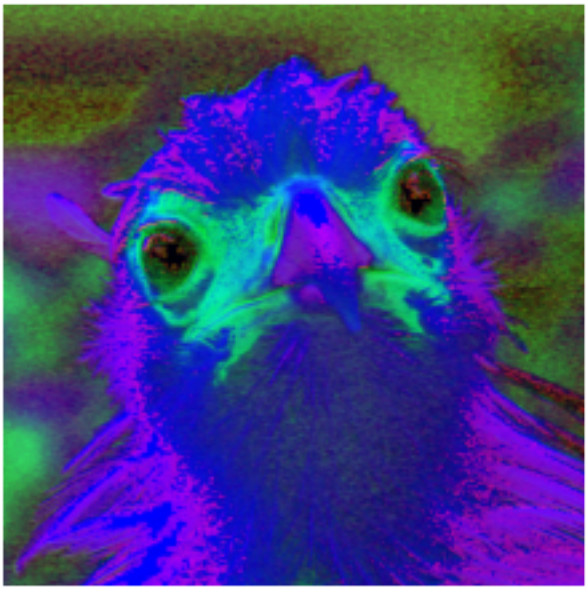
HSV è particolarmente comodo perché usa tre numeri molto intuitivi:

- Hue, ovvero la tinta tra i colori dell'arcobaleno (0 = rosso, 255 = violetto)
- Saturation, ovvero quanto i colori sono "vividi" (0 = colori sbiaditi, 255 = colori accesi)
- Value (nota anche come Brightness), ovvero la luminosità (0 = scuro, 255 = chiaro)



Per convertire un'immagine usiamo `cv.cvtColor`

```
def to_hsv(image):
    # Converti in HSV
    return cv.cvtColor(image, cv.COLOR_RGB2HSV)

def to_rgb(image):
    # Converti in RGB
    return cv.cvtColor(image, cv.COLOR_HSV2RGB)

# plot_image interpreta l'immagine come se fosse in RGB
utils.plot_image(to_hsv(samples.bird))
# Per visualizzare un'immagine HSV, si passa il parametro space
utils.plot_image(to_hsv(samples.bird), space='hsv') # Case-insensitive
```



Bianco e Nero

```
def hsv_grayscale(image):
  image = to_hsv(image)
  image[:, :, 1] = 0 # Impostiamo la saturazione a 0
```

```
  return to_rgb(image)

utils.plot_images([samples.bird, hsv_grayscale(samples.bird)])
```


Schiarisci e scurisci in HSV

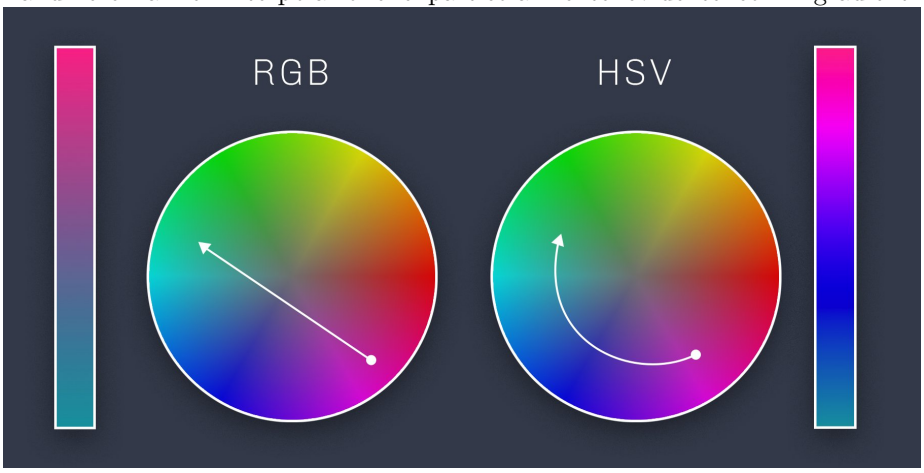
```
def hsv_darken(image):
  image = to_hsv(image)
  image = image.astype(np.int16) # Convertiamo in un formato che supporta numeri negativi
  image[:, :, 2] -= 40 # Togliamo 40 alla luminosità
  image = np.clip(image, 0, 255)
  image = image.astype(np.uint8)
  return to_rgb(image)

utils.plot_images([samples.bird, hsv_darken(samples.bird)])
```


Interpolazione in HSV

Interpolazioni in spazi diversi portano a risultati diversi:

```
def hsv_interpolate(image1, image2, coefficient):
    image1 = to_hsv(image1)
    image2 = to_hsv(image2)
    interpolated = interpolate(image1, image2, coefficient)
    return to_rgb(interpolated)

utils.plot_images([
    samples.bird,
    samples.tuscany,
    interpolate(samples.bird, samples.tuscany, 0.5),
    hsv_interpolate(samples.bird, samples.tuscany, 0.5)
  ],
  titles=['Image 1', 'Image 2', 'RGB interpolation', 'HSV interpolation'],
  columns=4
)
```

La differenza nell'interpolazione è particolarmente evidente con i gradienti:



Esercizio: Filtro Foto

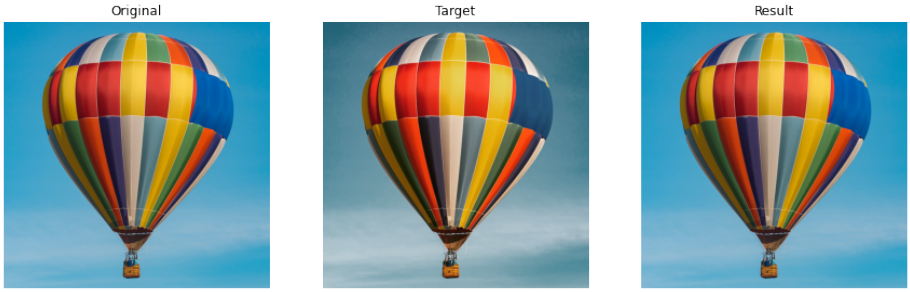
La maggior parte dei filtri sono formule applicate all'immagine in uno o più spazi di colore. Instagram ha 11 filtri di default:

```
utils.plot_images([samples.balloon] + samples.filter.images, titles=(['original'] + samples.
```

Scegliete un filtro e scrivete la funzione in modo che cv applichi lo stesso filtro (o che comunque esca molto simile).

```
def apply_filter(image):
    ...
    return image

reference_image = samples.filter.ludwig # Rimpiazza con il filtro scelto

transformed_image = apply_filter(samples.balloon)

distance = np.mean((reference_image - transformed_image) ** 2)
print('Distanza:', distance)

utils.plot_images([samples.balloon, reference_image, transformed_image], titles=['Original',
```

```
Distanza: 95.6607780456543
```



Disegnare su un'Immagine

Per disegnare un'immagine si può modificare direttamente i pixel:

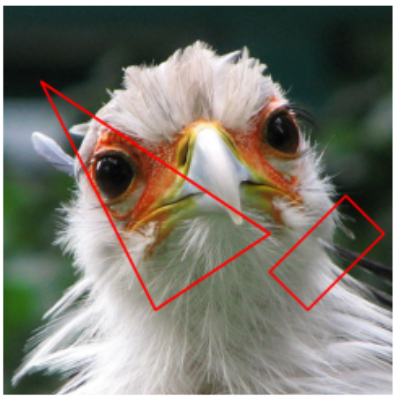
```
def draw_square(image, x, y, size):
    image = image.copy() # Così non modifichiamo l'immagine originale
    image[y:y+size, x:x+size] = [255, 255, 0] # Rosso massimo, verde massimo, niente blu =>
    return image

utils.plot_image(draw_square(samples.bird, 251, 251, 100))
```

```
def draw_line(image, start_x, start_y, length):
    for i in range(length):
        image = draw_square(image, start_x + i, start_y + i, 10)
    return image
utils.plot_image(draw_line(samples.bird, 251, 251, 300))
```

cv semplifica questo processo con diverse funzioni:

```
# Queste funzioni modificano l'immagine di partenza, quindi facciamo sempre copie

# cv.rectangle(immagine, (x_topleft, y_topleft), (x_bottomright, y_bottomright), colore, sp
square_image = cv.rectangle(samples.bird.copy(), (100, 300), (150, 350), [255, 0, 0], 5)

# cv.line(immagine, (x_topleft, y_topleft), (x_bottomright, y_bottomright), colore, spessor
line_image = cv.line(samples.bird.copy(), (100, 300), (500, 700), [255, 0, 0], 5)

# cv.circle(immagine, (centro_x, centro_y), raggio, colore, spessore)
circle_image = cv.circle(samples.bird.copy(), (500, 600), 100, [255, 0, 0], 5)

# cv.polylines vuole i punti di un poligono in maniera particolare (un array int32 di forma
points_1 = [(100, 200), (700, 600), (400, 800)]
points_1 = np.array(points_1, np.int32).reshape((len(points_1), 1, 2))

points_2 = [(900, 500), (1000, 600), (800, 800), (700, 700)]
points_2 = np.array(points_2, np.int32).reshape((len(points_2), 1, 2))

# cv.polylines(immagine, lista di poligoni, True se deve chiudere i poligoni, colore, spesso
polygon_image = cv.polylines(samples.bird.copy(),[points_1, points_2], True, (255, 0, 0), 5)

# cv.putText(immagine, testo, (posizione_x, posizione_y), font, fattore di scala, colore, sp
text_image = cv.putText(samples.bird.copy(), 'UniBo', (400, 700), cv.FONT_HERSHEY_SIMPLEX, 4
```

```
utils.plot_images([
    square_image,
    line_image,
    circle_image,
    polygon_image,
    text_image
])
```




UniBo

Sovraimporre immagini

Per sovrapporre un'immagine sopra un'altra basta rimpiazzare i pixel dell'immagine di partenza con quelli dell'immagine da sovrapporre:

```
utils.plot_image(samples.special.sunglasses_noalpha)

# Gli occhiali da sole hanno dimensione [960, 480]

def composite(background, foreground, x, y):
    composite_image = background.copy()
    # Ricorda che quando si indicizza si indicizza prima con y e poi con x
    composite_image[y:y+foreground.shape[0], x:x+foreground.shape[1]] = foreground
    return composite_image

sunglasses = samples.special.sunglasses_noalpha

# Ridimensiona gli occhiali
sunglasses = cv.resize(sunglasses, (720, 360))

composite_image = composite(samples.bird, sunglasses, 180, 200)

utils.plot_image(composite_image)
```

Se l'immagine è in RGBA, possiamo scegliere l'immagine degli occhiali quando alpha è alto e l'immagine dell'uccello quando alpha è basso

```
def smart_fusion(base_image, rgba_image):
    # Separa il canale alpha dagli altri
    alpha_channel = rgba_image[:, :, 3] / 255
    rgb_channels = rgba_image[:, :, :3]

    alpha_channel = alpha_channel.reshape(alpha_channel.shape[0], alpha_channel.shape[1], 1)

    composite = rgb_channels * alpha_channel + base_image * (1 - alpha_channel)
    return composite.astype(np.uint8)

def smart_composite(base_image, added_image, x, y):
    # Crea un'immagine completamente trasparente della stessa dimensione di base_image
    rgba_image = np.zeros([1024, 1024, 4])

    # Aggiungici gli occhiali da sole (con canale alpha)
    rgba_image = composite(rgba_image, added_image, x, y)

    # Fondi le due immagini
    return smart_fusion(base_image, rgba_image)

sunglasses_alpha = cv.resize(samples.special.sunglasses_alpha, (720, 360))

utils.plot_image(smart_composite(samples.bird, sunglasses_alpha, 180, 200))
```

Se non abbiamo il canale Alpha, possiamo comunque ”inventarcelo” dando 255
ai pixel da tenere e 0 ai pixel da rendere trasparenti

```
# Se Rosso, Verde e Blu sono tutti inferiori a 40, significa che è un pixel scuro
dark_pixels = np.logical_and(np.logical_and(sunglasses[:, :, 0] < 40, sunglasses[:, :, 1] <

# Convertiamo dark_pixels in interi invece che booleani
dark_pixels = dark_pixels.astype(np.uint8) # Ha valore 1 se il pixel è scuro, 0 se è chiaro
dark_pixels = dark_pixels * 255

# Dato che il canale è uno solo Matplotlib usa colori falsi (giallo = massimo, viola = mini
utils.plot_image(dark_pixels)
```

dark_pixels diventa il nostro canale Alpha:

```
# Crea un'immagine RGBA vuota della stessa dimensione degli occhiali
sunglasses_fakealpha = np.zeros([sunglasses.shape[0], sunglasses.shape[1], 4])

# Usa come canali RGB i canali RGB dell'immagine di partenza
sunglasses_fakealpha[:, :, 0:3] = sunglasses
sunglasses_fakealpha[:, :, 3] = dark_pixels
```

E quindi possiamo fare la composizione come con l'altra immagine:

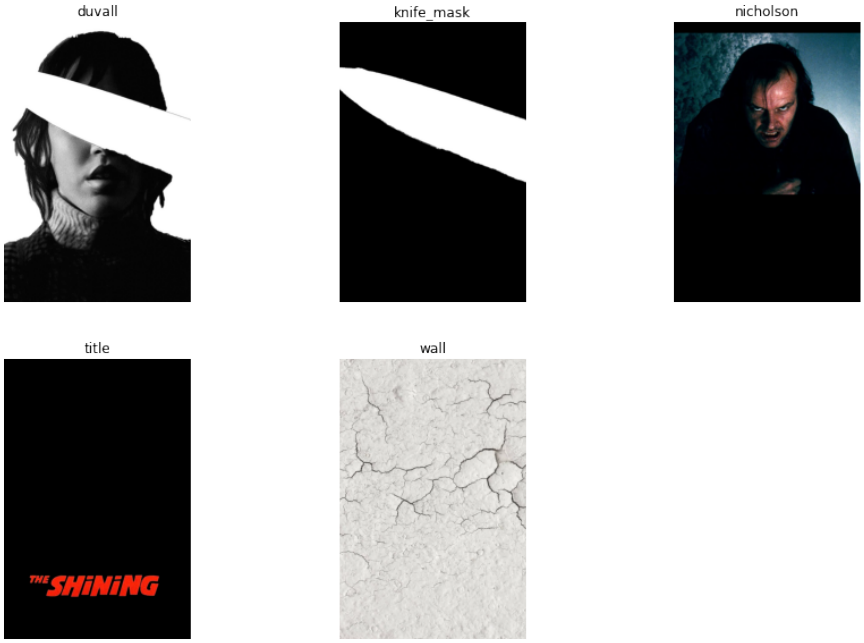
```
utils.plot_image(smart_composite(samples.bird, sunglasses_fakealpha, 180, 200))
```

Esercizio: Mascheramento

Realizzate la seguente immagine:

A partire dalle seguenti immagini:

```
utils.plot_images([
    samples.shining.duvall, samples.shining.knife_mask, samples.shining.nicholson, samples.s
    ], titles=['duvall', 'knife_mask', 'nicholson', 'title', 'wall'], columns=3)
print(samples.shining.duvall.shape)
```



Le immagini sono già allineate correttamente.

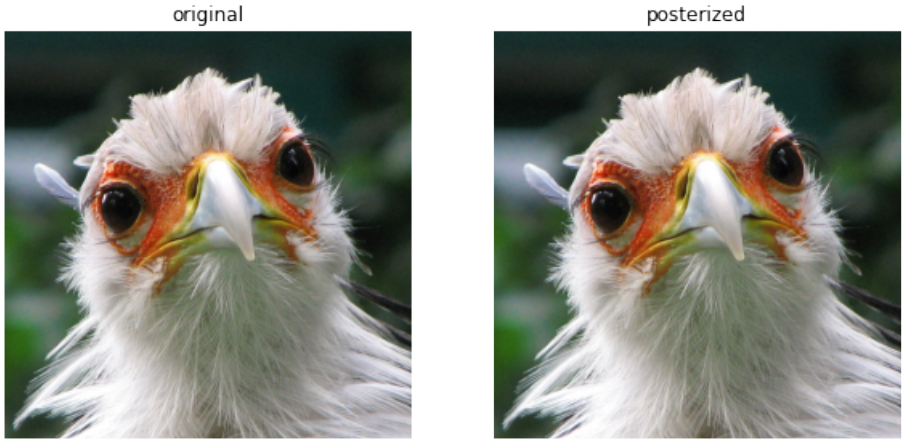
```
def shining():
    duvall_alpha = np.logical_or(np.logical_or(samples.shining.duvall[:, :, 0] < 250, sample
    utils.plot_image(duvall_alpha)
    duvall_rgba = np.zeros([samples.shining.duvall.shape[0], samples.shining.duvall.shape[1]
    duvall_blue = np.copy(samples.shining.duvall).astype(np.int16)
    duvall_blue[:, :, 0] -= 40
    duvall_blue[:, :, 1] -= 40
    duvall_blue = np.clip(duvall_blue, 0, 255)
    duvall_blue = duvall_blue.astype(np.uint8)
    duvall_rgba[:, :, 0:3] = duvall_blue
    duvall_rgba[:, :, 3] = duvall_alpha.astype(np.uint8) * 255

    fused = smart_fusion(samples.shining.nicholson, duvall_rgba)
    utils.plot_image(fused)

shining()
# utils.plot_image(shining(...))
```

Esercizio Extra: Effetto Poster

Photoshop permette di posterizzare un'immagine:

Scrivete una funzione che posterizzi (in maniera molto semplice) un'immagine.

```
def posterize(image):
    return image

utils.plot_images([samples.bird, posterize(samples.bird)], titles=['original', 'posterized']
```



```
!git clone https://github.com/samuelemarro/ml_intro
```

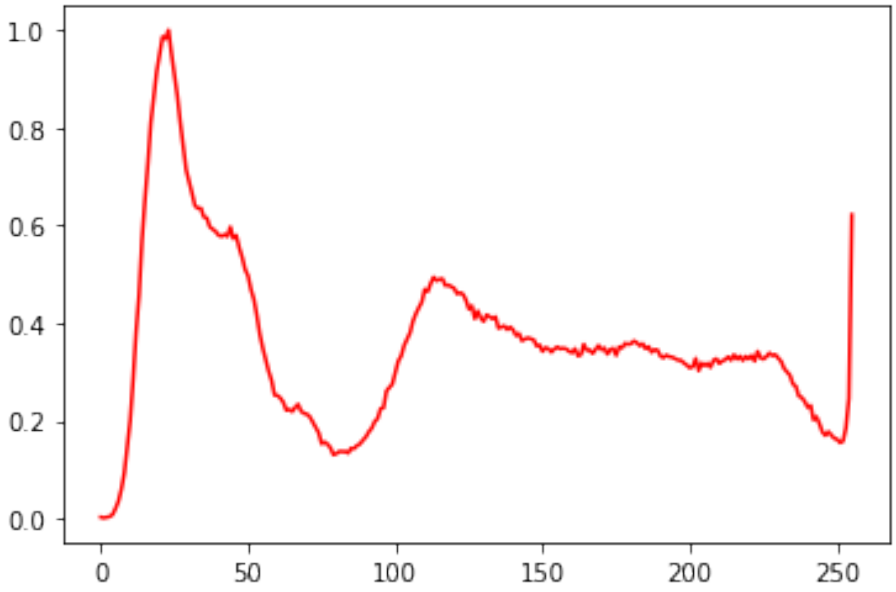
```
fatal: destination path 'ml_intro' already exists and is not an empty directory.
```

```
import cv2 as cv
import numpy as np
from ml_intro import samples, utils
```

Istogrammi

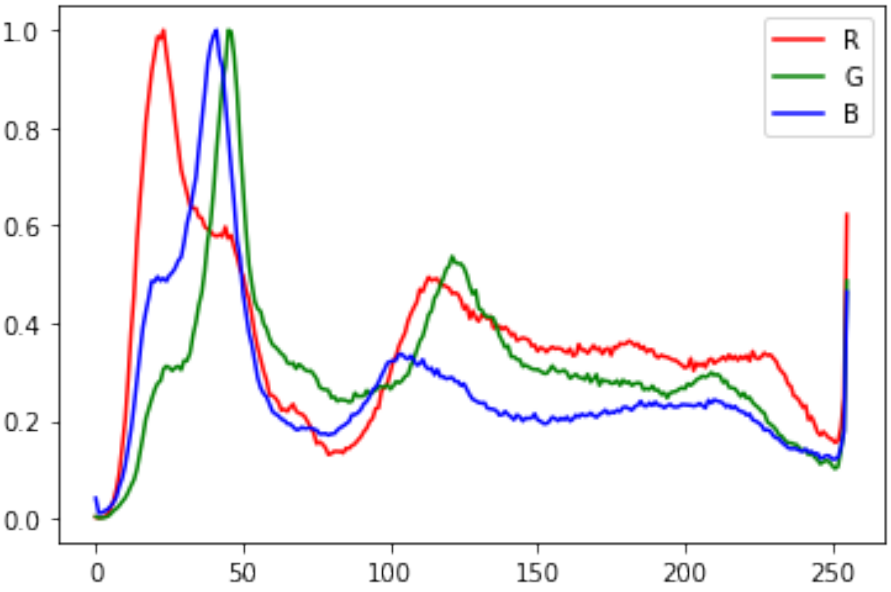
A partire da un canale è possibile realizzare un istogramma dei valori più comuni:

```
import matplotlib.pyplot as plt

# Sintassi: cv.calcHist(lista di immagini, lista di canali, maschera per calcolare solo in c
histogram_red = cv.calcHist([samples.bird], [0], None, [256], [0, 256])
# Normalizziamo l'istogramma mettendo il picco a 1
histogram_red /= np.max(histogram_red)

utils.plot_image(samples.bird[:, :, 0], space='gray')
plt.plot(histogram_red, color='r')
plt.show()
```



Usando più canali si può avere una visione più ampia dell'immagine:

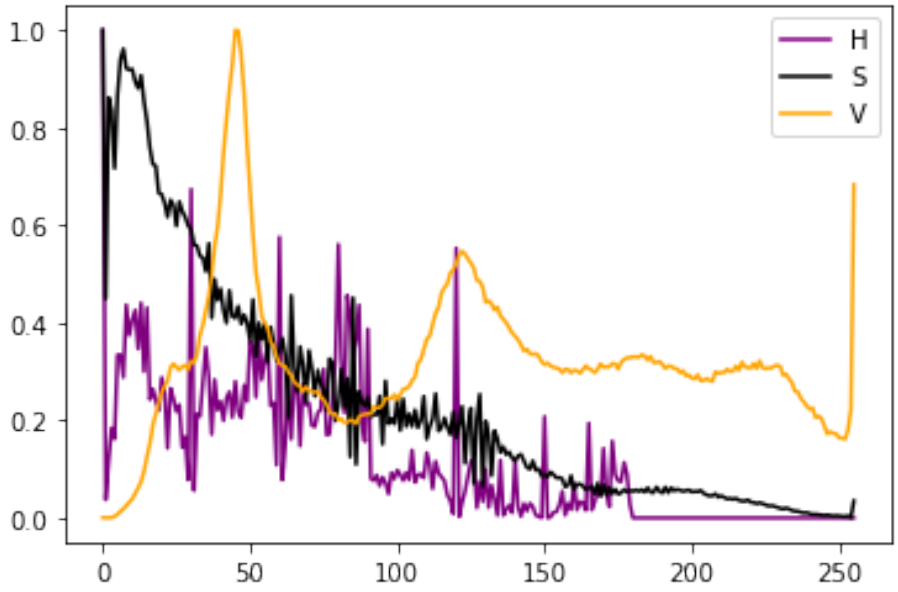
```
histogram_green = cv.calcHist([samples.bird], [1], None, [256], [0, 256])
histogram_green /= np.max(histogram_green)
histogram_blue = cv.calcHist([samples.bird], [2], None, [256], [0, 256])
histogram_blue /= np.max(histogram_blue)

utils.plot_image(samples.bird)
plt.plot(histogram_red, color='r')
plt.plot(histogram_green, color='g')
plt.plot(histogram_blue, color='b')
plt.legend(['R', 'G', 'B'])
plt.show()
```



RGB non ci dice però molte informazioni, proviamo con HSV:

```
bird_hsv = cv.cvtColor(samples.bird, cv.COLOR_RGB2HSV)

histogram_hue = cv.calcHist([bird_hsv], [0], None, [256], [0, 256])
histogram_hue = histogram_hue.astype(np.float32) / np.max(histogram_hue)
```

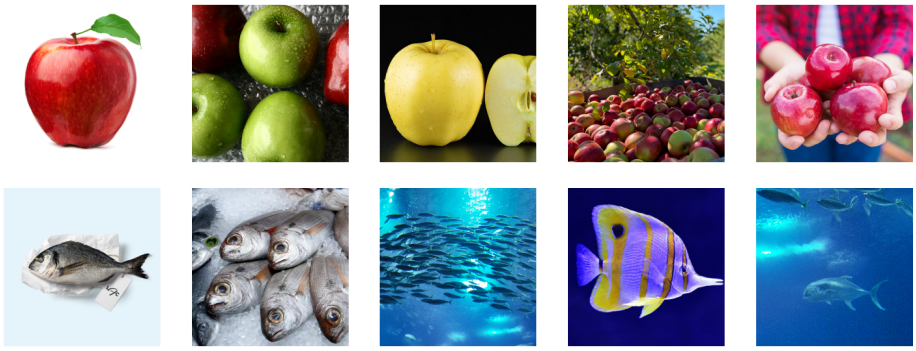
```
histogram_saturation = cv.calcHist([bird_hsv], [1], None, [256], [0, 256])
histogram_saturation /= np.max(histogram_saturation)
histogram_value = cv.calcHist([bird_hsv], [2], None, [256], [0, 256])
histogram_value /= np.max(histogram_value)

utils.plot_image(samples.bird)
plt.plot(histogram_hue, color='purple')
plt.plot(histogram_saturation, color='black')
plt.plot(histogram_value, color='orange')
plt.legend(['H', 'S', 'V'])
plt.show()
```

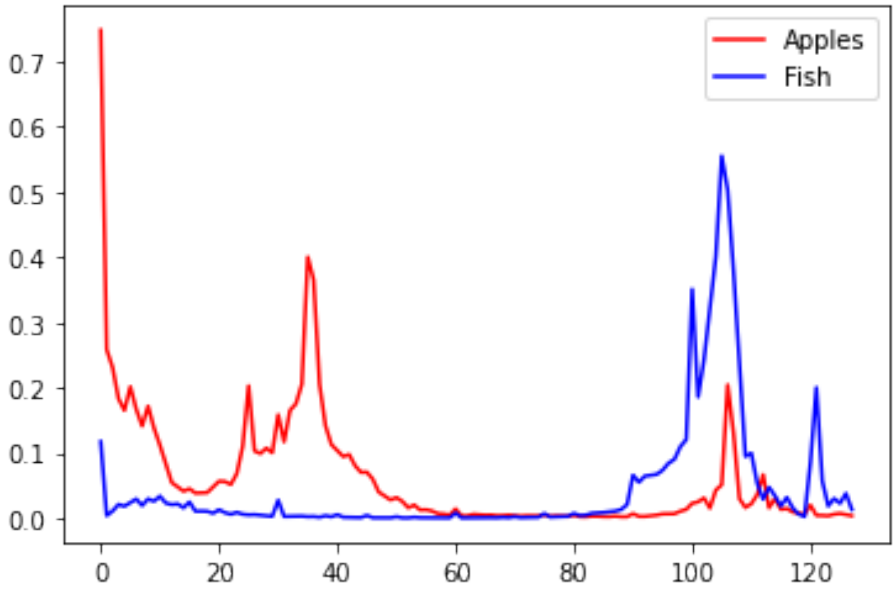


Calcolando gli istogrammi medi si possono ricavare statistiche utili:

```
utils.plot_images(samples.histogram.apples, columns=5)
utils.plot_images(samples.histogram.fish, columns=5)

apple_histogram_sum = np.zeros([128, 1], dtype=np.float32)
fish_histogram_sum = np.zeros([128, 1], dtype=np.float32)

for apple in samples.histogram.apples:
    # Usiamo il canale hue
    apple = cv.cvtColor(apple, cv.COLOR_RGB2HSV)
    apple_histogram = cv.calcHist([apple], [0], None, [128], [0, 128])
    apple_histogram /= np.max(apple_histogram)
    apple_histogram_sum += apple_histogram

for fish in samples.histogram.fish:
    # Usiamo il canale hue
    fish = cv.cvtColor(fish, cv.COLOR_RGB2HSV)
    fish_histogram = cv.calcHist([fish], [0], None, [128], [0, 128])
    fish_histogram /= np.max(fish_histogram)
    plt.show()
    fish_histogram_sum += fish_histogram

apple_histogram_mean = apple_histogram_sum / 5
fish_histogram_mean = fish_histogram_sum / 5
```

```
plt.plot(apple_histogram_mean, color='red')
plt.plot(fish_histogram_mean, color='blue')
plt.legend(['Apples', 'Fish'])
plt.show()
```

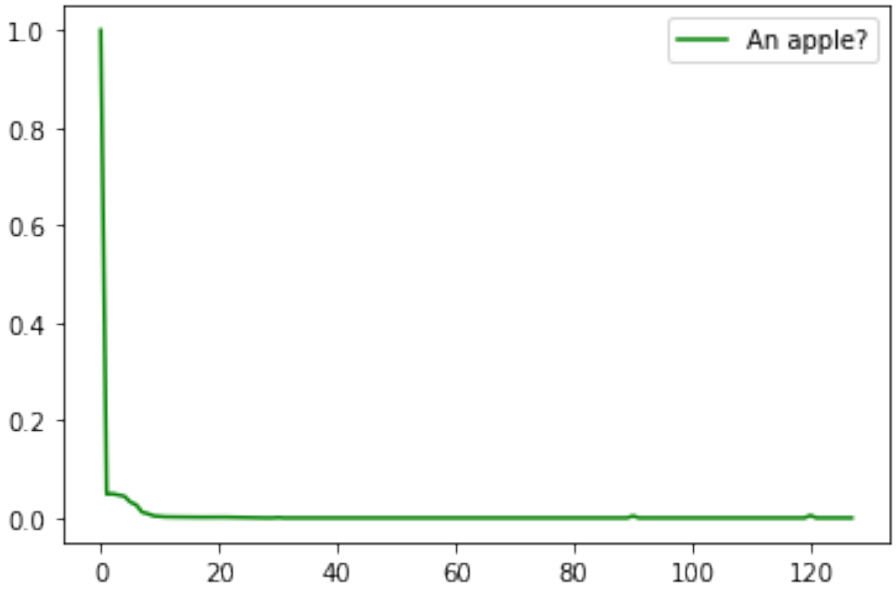


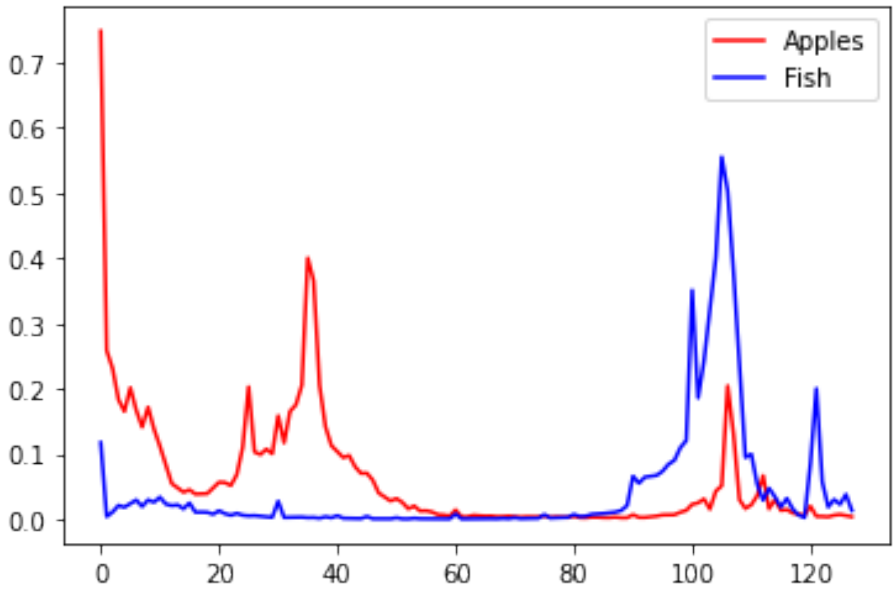


Possiamo quindi stabilire in maniera automatica se un'immagine è una mela o un pesce? Sì, confrontando i due istogrammi:

```
apple_test_hsv = cv.cvtColor(samples.histogram.apple_test, cv.COLOR_RGB2HSV)
apple_test_histogram = cv.calcHist([apple_test_hsv], [0], None, [128], [0, 128])
apple_test_histogram /= np.max(apple_test_histogram)

utils.plot_image(samples.histogram.apple_test)
```

```
plt.plot(apple_test_histogram, color='green')
plt.legend(['An apple?'])
plt.show()
plt.plot(apple_histogram_mean, color='red')
plt.plot(fish_histogram_mean, color='blue')
plt.legend(['Apples', 'Fish'])
plt.show()
```





Ci sono molti metodi per confrontare due istogrammi. Useremo la correlazione, che va da -1 (anticorrelati) a +1 (correlati):

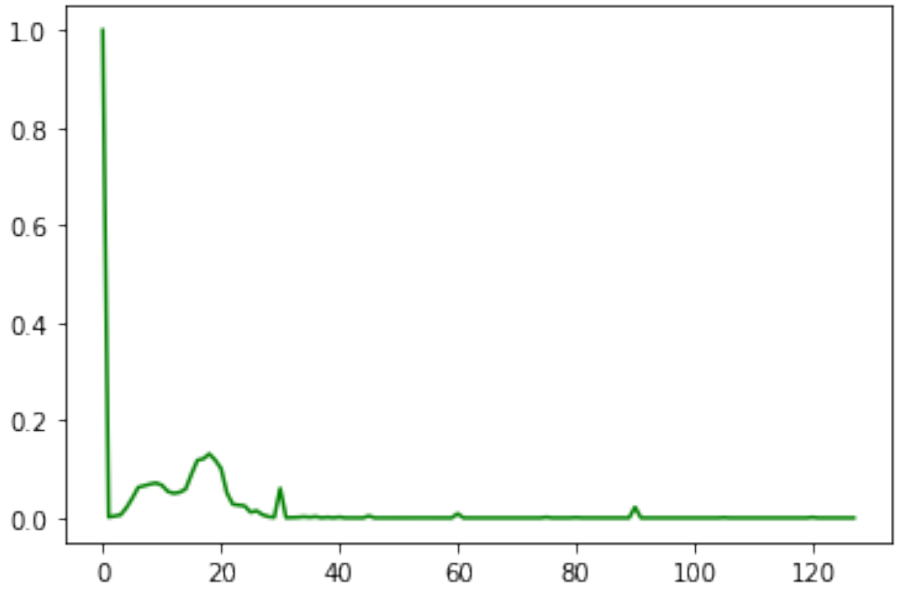
```
print(apple_test_histogram.dtype)
print(apple_histogram_mean.dtype)
apple_apple_correlation = cv.compareHist(apple_test_histogram, apple_histogram_mean, cv.HIST
```

```
apple_fish_correlation = cv.compareHist(apple_test_histogram, fish_histogram_mean, cv.HISTCN

print('Correlazione con l\'istogramma delle mele:', apple_apple_correlation)
print('Correlazione con l\'istogramma dei pesci:', apple_fish_correlation)
```

```
float32
float32
Correlazione con l'istogramma delle mele: 0.6663653480608641
Correlazione con l'istogramma dei pesci: 0.060968436966767826
```

Ovviamente non è sempre perfetto come metodo:

```
fish_incorrect_hsv = cv.cvtColor(samples.histogram.fish_incorrect, cv.COLOR_RGB2HSV)
fish_incorrect_histogram = cv.calcHist([fish_incorrect_hsv], [0], None, [128], [0, 128])
fish_incorrect_histogram /= np.max(fish_incorrect_histogram)

utils.plot_image(samples.histogram.fish_incorrect)

plt.plot(fish_incorrect_histogram, color='green')
plt.show()

fish_apple_correlation = cv.compareHist(fish_incorrect_histogram, apple_histogram_mean, cv.H
fish_fish_correlation = cv.compareHist(fish_incorrect_histogram, fish_histogram_mean, cv.HIS

print('Correlazione con l\'istogramma delle mele:', fish_apple_correlation)
print('Correlazione con l\'istogramma dei pesci:', fish_fish_correlation)
```




```
Correlazione con l'istogramma delle mele: 0.6406034023991304
Correlazione con l'istogramma dei pesci: 0.02897115501895503
```

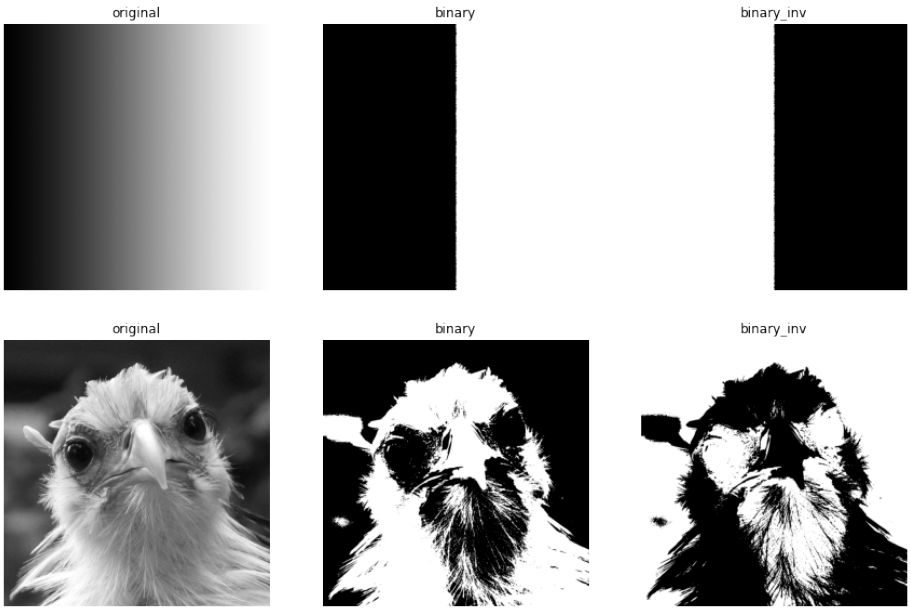
Thresholding

La volta scorsa abbiamo guardato i valori dei pixel per determinare se erano pixel chiari (quindi sfondo) o scuri (quindi l'oggetto d'interesse). Questo è un esempio di *thresholding*. Il thresholding può essere applicato solo a immagini a un canale, quindi è importante scegliere quale canale prendere.

cv2 offre diverse opzioni di thresholding:

```
def to_grayscale(image):
    return cv.cvtColor(image, cv.COLOR_RGB2GRAY)

# L'immagine è in bianco e nero, ma va comunque convertita in un solo canale
grayscale_gradient = to_grayscale(samples.special.bw_gradient)

# Sintassi: cv.threshold(immagine, valore di threshold, valore per indicare "true", tipo di
# Restituisce il valore usato come threshold e l'immagine con il threshold applicato

_, binary_image = cv.threshold(grayscale_gradient, 127, 255, cv.THRESH_BINARY) # Se è minore
_, binary_inv_image = cv.threshold(grayscale_gradient, 127, 255, cv.THRESH_BINARY_INV) # Se

utils.plot_images([
    grayscale_gradient,
```

```
    binary_image,
    binary_inv_image,
], titles=['original', 'binary', 'binary_inv'], columns=3, space='gray')

grayscale_bird = to_grayscale(samples.bird)

_, binary_image = cv.threshold(grayscale_bird, 127, 255, cv.THRESH_BINARY)
_, binary_inv_image = cv.threshold(grayscale_bird, 127, 255, cv.THRESH_BINARY_INV)

utils.plot_images([grayscale_bird, binary_image, binary_inv_image], titles=['original', 'bir
```

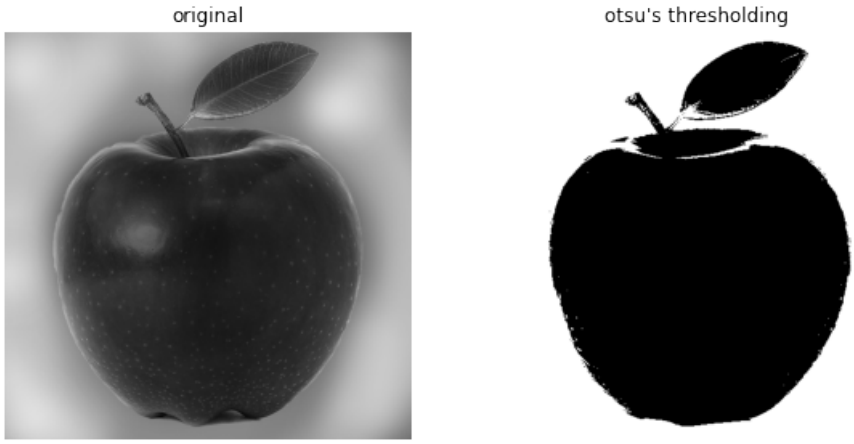


La maggior parte delle volte si usa `cv.THRESH_BINARY`.

Se non sappiamo il threshold da usare, si può usare la binarizzazione di Otsu:

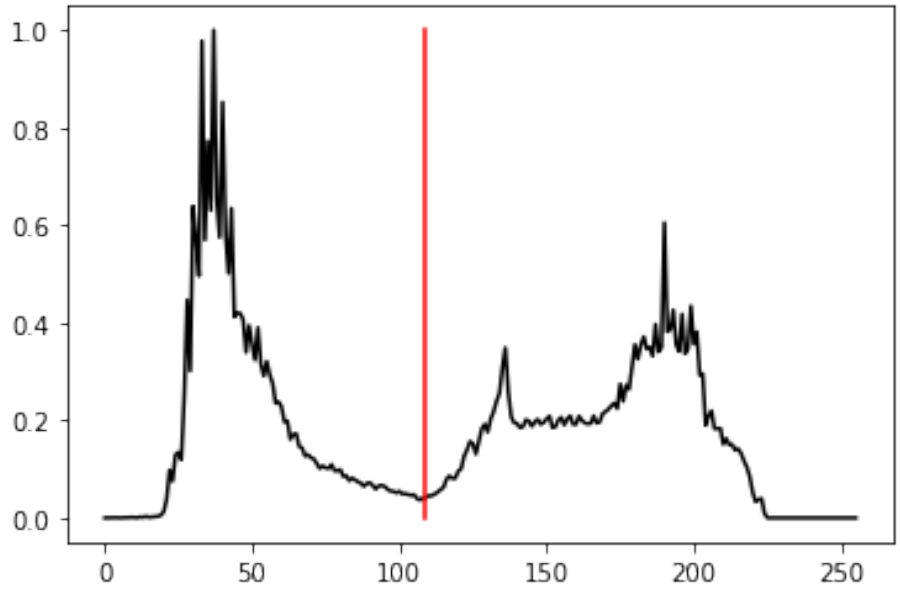
```
grayscale_apple = to_grayscale(samples.special.apple)

chosen_threshold, thresholded_apple = cv.threshold(grayscale_apple, 12345, 255, cv.THRESH_BI

print('Chosen threshold:', chosen_threshold)
utils.plot_images([grayscale_apple, thresholded_apple], titles=['original', 'otsu\'s thresho
```

```
Chosen threshold: 109.0
```



Curiosità tecnica: la binarizzazione di Otsu sceglie il threshold guardando l'istogramma e separandolo in due "distribuzioni".

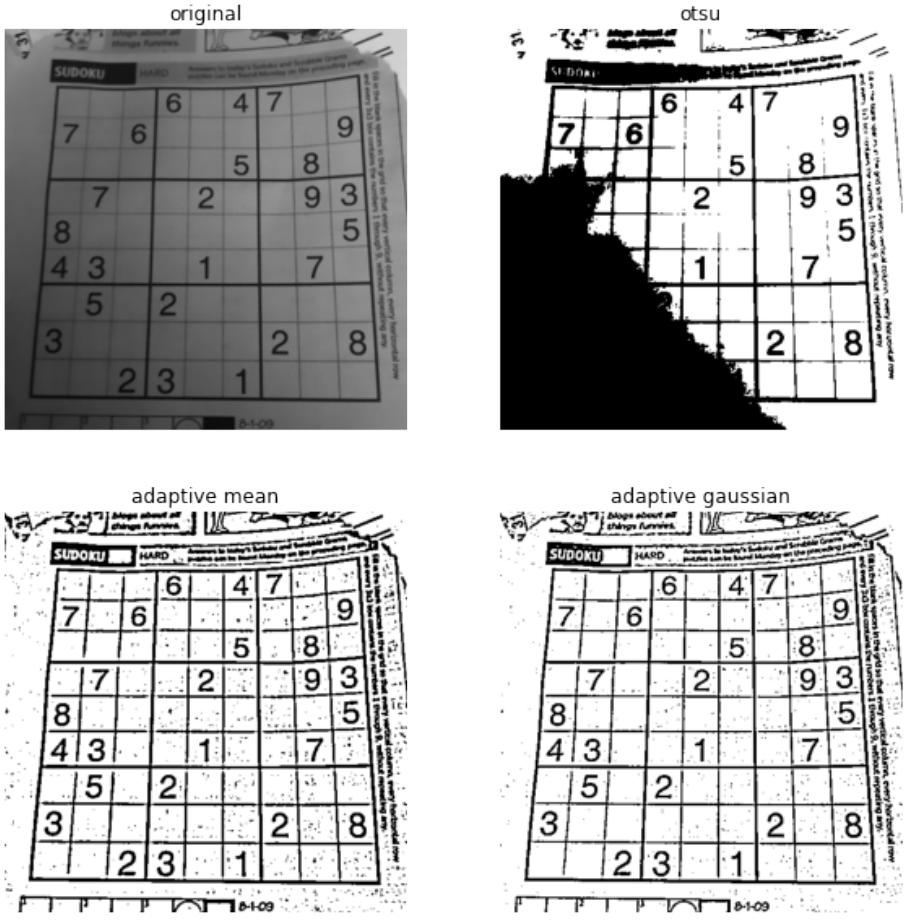
```
histogram_apple = cv.calcHist([samples.special.apple], [0], None, [256], [0, 256])
histogram_apple /= np.max(histogram_apple)
plt.plot(histogram_apple, color='black')
plt.plot([chosen_threshold, chosen_threshold], [0, 1], color='red')
plt.show()
```



Thresholding Adattivo

A volte un threshold è adeguato in certe zone ma non in altre: con l'adaptive thresholding, viene scelto un threshold locale in base ai pixel circostanti

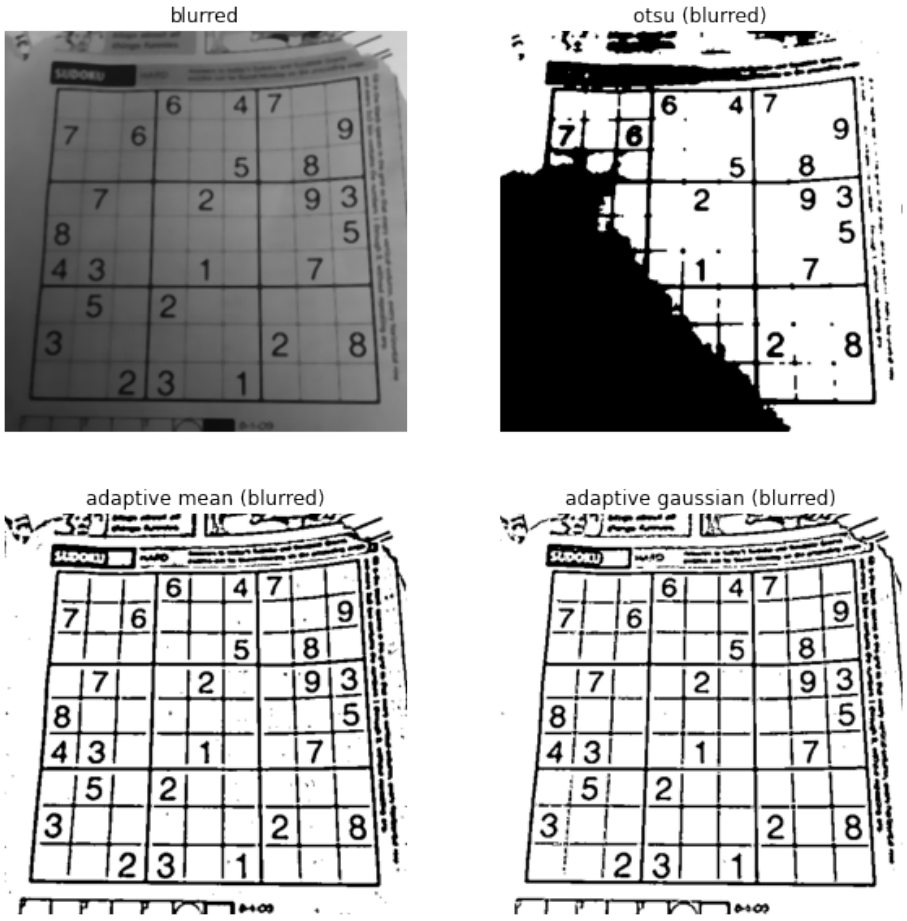
```
grayscale_sudoku = to_grayscale(samples.special.sudoku)
otsu_sudoku = cv.threshold(grayscale_sudoku, 12345, 255, cv.THRESH_OTSU)[1]

# Sintassi: cv.adaptiveThreshold(immagine, valore teorico massimo, tipo di scelta threshold,
adaptive_mean_sudoku = cv.adaptiveThreshold(grayscale_sudoku, 255, cv.ADAPTIVE_THRESH_MEAN_C
adaptive_gaussian_sudoku = cv.adaptiveThreshold(grayscale_sudoku, 255, cv.ADAPTIVE_THRESH_GA

utils.plot_images([grayscale_sudoku, otsu_sudoku, adaptive_mean_sudoku, adaptive_gaussian_su
```



Sfocare l'immagine può aiutare a "pulire" il risultato (anche con il thresholding normale):

```
grayscale_blur_sudoku = cv.medianBlur(grayscale_sudoku, 5)
```

```
otsu_blur_sudoku = cv.threshold(grayscale_blur_sudoku, 12345, 255, cv.THRESH_OTSU)[1]

# Sintassi: cv.adaptiveThreshold(immagine, valore teorico massimo, tipo di scelta threshold,
adaptive_mean_blur_sudoku = cv.adaptiveThreshold(grayscale_blur_sudoku, 255, cv.ADAPTIVE_THR
adaptive_gaussian_blur_sudoku = cv.adaptiveThreshold(grayscale_blur_sudoku, 255, cv.ADAPTIVE

utils.plot_images(
    [grayscale_blur_sudoku, otsu_blur_sudoku, adaptive_mean_blur_sudoku, adaptive_gaussian_b
    titles=['blurred', 'otsu (blurred)', 'adaptive mean (blurred)', 'adaptive gaussian (blur
    space='gray'
)
```



Oltre il Grayscale

Il thresholding non deve essere per forza applicato a immagini grayscale:

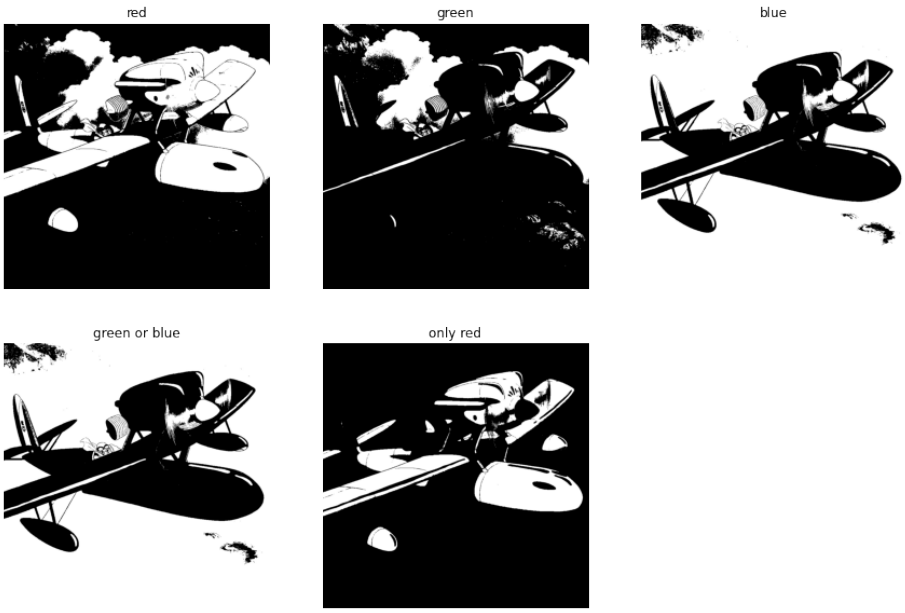
```
plane_red_channel = samples.plane[:, :, 0]

_, plane_red = cv.threshold(plane_red_channel, 205, 255, cv.THRESH_BINARY)

utils.plot_image(samples.plane)
utils.plot_image(plane_red, space='gray')
```

Nota che ha anche preso la nuvola perché il bianco ha un canale rosso molto alto.

Combinare Thresholding

Il risultato di un'operazione di thresholding è un tensore, quindi si può tranquillamente applicare funzioni matematiche. `np.maximum` prende il massimo di due tensori (e quindi fa una sorta di OR logico), mentre `np.minimum` prende il minimo di due tensori (e quindi fa una sorta di AND logico). Fare `255 - image` è una sorta di NOT logico.

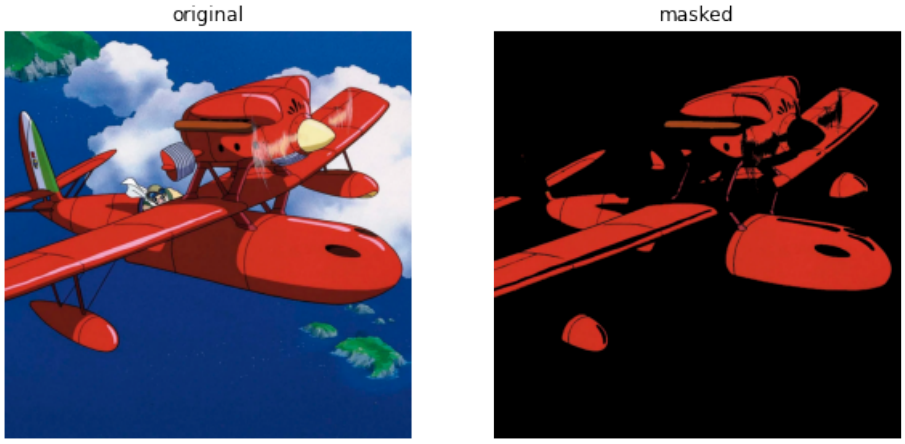
```
_, red_threshold = cv.threshold(samples.plane[:, :, 0], 125, 255, cv.THRESH_BINARY)
_, green_threshold = cv.threshold(samples.plane[:, :, 1], 140, 255, cv.THRESH_BINARY)
_, blue_threshold= cv.threshold(samples.plane[:, :, 2], 75, 255, cv.THRESH_BINARY)

green_or_blue = np.maximum(green_threshold, blue_threshold).astype(np.uint8)
only_red = np.minimum(red_threshold, 255 - green_or_blue).astype(np.uint8)

utils.plot_images([
    red_threshold,
    green_threshold,
    blue_threshold,
    green_or_blue,
    only_red
], titles=['red', 'green', 'blue', 'green or blue', 'only red'], columns=3, space='gray')
```



In alternativa, è possibile convertire le immagini in tensori booleani (es. facendo `np.equal(image, 255)`) e poi lavorarci sopra con operatori booleani bitwise (`&`, `|`, `~`).

Masking

```
def apply_mask(image, mask):
    return image * (np.equal(mask, 255)).astype(np.uint8).reshape(mask.shape + (1,))

utils.plot_image(only_red, space='gray')
utils.plot_images([samples.plane, apply_mask(samples.plane, only_red)], titles=['original',
```

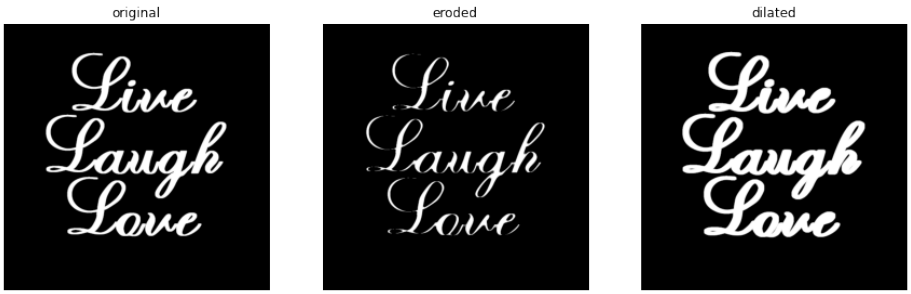


Erosione e Dilatazione

cv2 ha anche dei metodi per erodere e dilatare le immagini:

```
def erode(image, erosion_size):
    erosion_shape = cv.MORPH_ELLIPSE
    erosion_element = cv.getStructuringElement(erosion_shape, (2 * erosion_size + 1, 2 * ero
                                        (erosion_size, erosion_size))
    return cv.erode(image, erosion_element)

def dilate(image, dilation_size):
```

```
    dilation_shape = cv.MORPH_ELLIPSE

    dilation_element = cv.getStructuringElement(dilation_shape, (2 * dilation_size + 1, 2 *
    return cv.dilate(image, dilation_element)

text_image = to_grayscale(samples.special.boomer)
eroded = erode(text_image, 2)
dilated = dilate(text_image, 2)

utils.plot_images([text_image, eroded, dilated], titles=['original', 'eroded', 'dilated'], s
```



Erosione e dilatazione possono essere usati per espandere o restringere le mask.

Esercizio: Thresholding

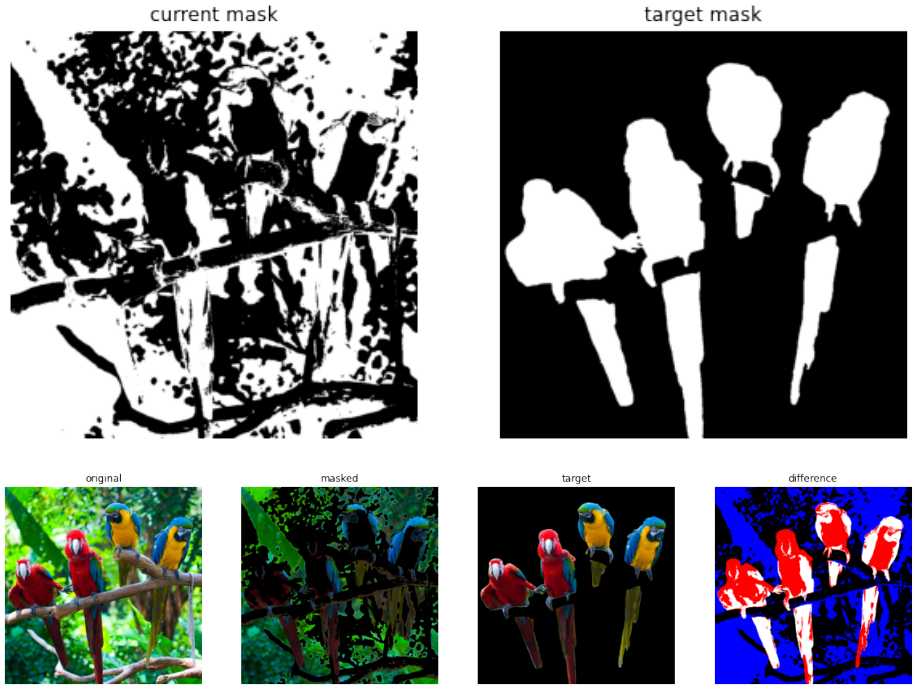
Scrivete una funzione che selezioni i pappagalli da questa immagine:

Questa è la maschera di ”soluzione” (fatta manualmente con Photoshop):

Il punteggio è calcolato come *area dell'intersezione tra le maschere / area dell'unione delle maschere*.

```
def find_mask(image):
    # Messa come esempio
    _, red_threshold = cv.threshold(image[:, :, 0], 75, 255, cv.THRESH_BINARY_INV)
    return red_threshold

def compute_difference_image(current_mask, target_mask):
    difference_image = np.zeros(current_mask.shape + (3,), dtype=np.uint8)
    boolean_current_mask = np.equal(current_mask, 255)
    boolean_target_mask = np.equal(target_mask, 255)
    # La parte corretta della maschera è bianca
    difference_image[boolean_current_mask & boolean_target_mask] = (255, 255, 255)
    # La parte che dovrebbe essere della maschera ma non lo è viene colorata di rosso
    difference_image[~boolean_current_mask & boolean_target_mask] = (255, 0, 0)
    # La parte che non dovrebbe essere della maschera ma lo è viene colorata di blu
    difference_image[boolean_current_mask & ~boolean_target_mask] = (0, 0, 255)
```

```
    return difference_image

def intersection_over_union(current_mask, target_mask):
    boolean_current_mask = np.equal(current_mask, 255)
    boolean_target_mask = np.equal(target_mask, 255)

    intersection = np.count_nonzero(boolean_current_mask & boolean_target_mask)
    union = np.count_nonzero(boolean_current_mask | boolean_target_mask)
    return intersection / union

original_parrots = samples.special.parrots
mask_parrots = find_mask(np.copy(original_parrots))
target_parrots = to_grayscale(samples.special.parrots_mask)
difference_image = compute_difference_image(mask_parrots, target_parrots)

utils.plot_images([mask_parrots, target_parrots], space='gray', titles=['current mask', 'tar

utils.plot_images(
    [original_parrots, apply_mask(original_parrots, mask_parrots), apply_mask(original_parro
    titles=['original', 'masked', 'target', 'difference'],
    columns=4
)
print('Punteggio: {:.2f}%'.format(intersection_over_union(mask_parrots, target_parrots) * 10
```

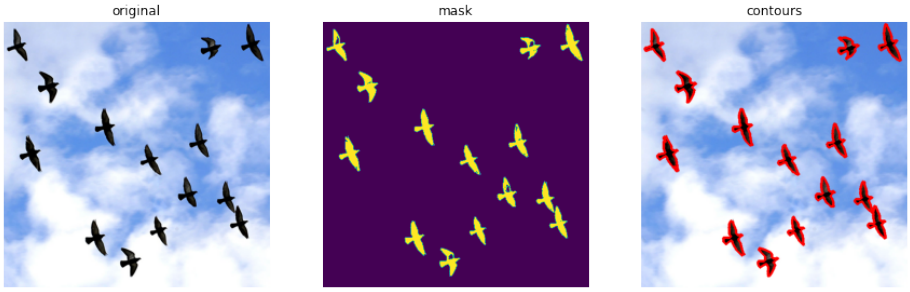


```
Punteggio: 19.08%
```

Rilevamento contorni

Una volta che abbiamo una maschera possiamo anche tracciare i contorni degli oggetti:

```
gray_birds = to_grayscale(samples.special.birds)
_, birds_mask = cv.threshold(gray_birds, 64, 255, cv.THRESH_BINARY_INV)
# Sintassi: cv.findContours(immagine, tipo di retrieval, tipo di approssimazione)
# Restituisce la lista dei contorni e la gerarchia tra contorni (chi contiene quali contorn
# Tipi comuni di retrieval:
# - RETR_EXTERNAL (solo i contorni esterni)
# - RETR_LIST (tutti i contorni)
# - RETR_CCOMP (contorni esterni + contorni dei buchi interni)
contours, _ = cv.findContours(birds_mask, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)

# Sintassi: cv.drawContours(immagine di partenza, contorni, indice del contorno o -1 per tu
image_with_contours = cv.drawContours(samples.special.birds.copy(), contours, -1, (255, 0, 

utils.plot_images([samples.special.birds, birds_mask, image_with_contours], titles=['origina

print('Numero di rondini:', len(contours))
```




```
Numero di rondini: 13
```

In alternativa si può anche disegnare una bounding box:

```
image_with_rects = samples.special.birds.copy()
for contour in contours:
    x, y, width, height = cv.boundingRect(contour)
    image_with_rects = cv.rectangle(image_with_rects.copy(), (x, y), (x+width, y+height), (2

utils.plot_image(image_with_rects)
```

Watershed

Spesso non si riesce a ottenere maschere esatte con il thresholding, oppure non si riesce a separare due oggetti diversi. Per fare maschere più avanzate si utilizza l'algoritmo Watershed.

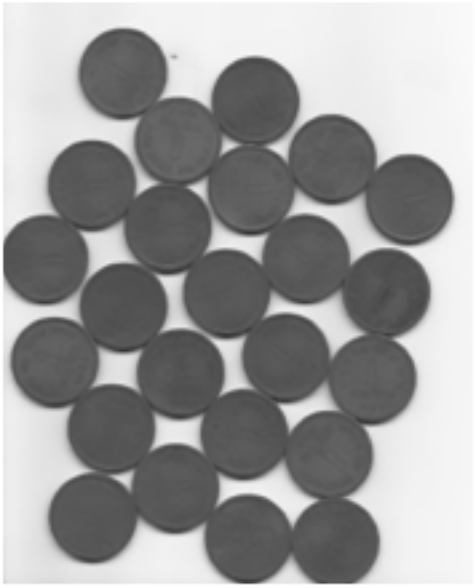
Watershed ha bisogno di tre informazioni:

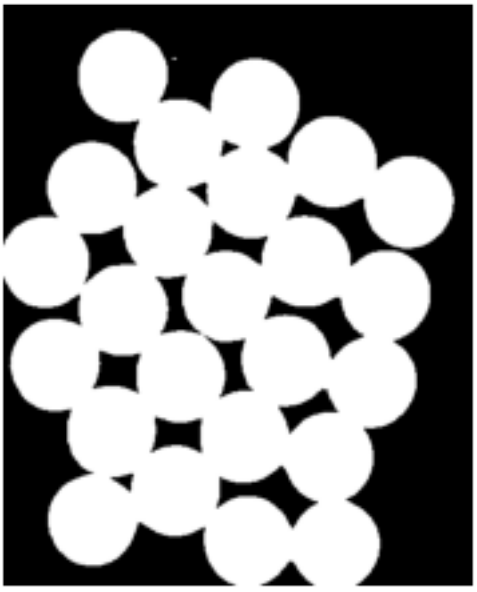
- Quali pixel appartengono sicuramente all'oggetto di interesse
- Quali pixel appartengono sicuramente allo sfondo
- Quali pixel appartengono a oggetti diversi tra di loro

```
utils.plot_image(samples.special.coins)
```

Prima di tutto separiamo l'oggetto di interesse dallo sfondo:

```
coins_gray = cv.cvtColor(samples.special.coins, cv.COLOR_RGB2GRAY)
utils.plot_image(coins_gray, space='gray')

_, thresholded_coins = cv.threshold(coins_gray, 12345, 255, cv.THRESH_BINARY_INV + cv.THRESH
utils.plot_image(thresholded_coins, space='gray')
```

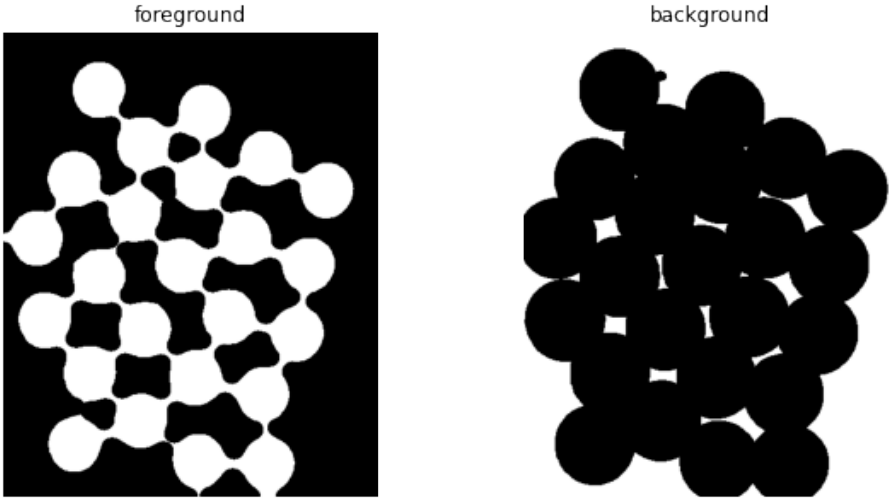




Se erodiamo la maschera otteniamo pixel che sono sicuramente monete, mentre se dilatiamo la maschera e la invertiamo otteniamo pixel che sono sicuramente sfondo:

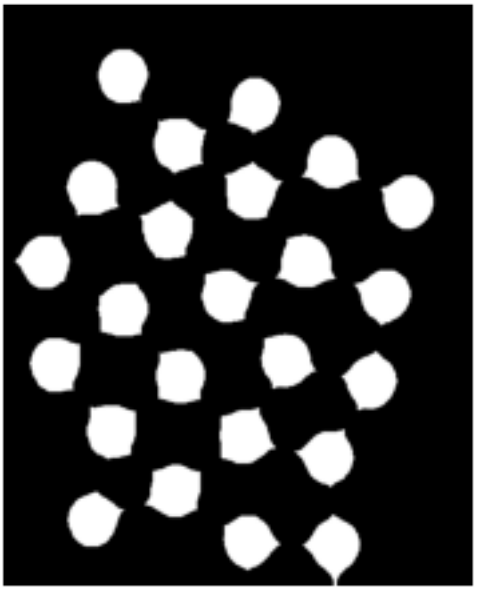
```
foreground = erode(thresholded_coins, 10)
background = 255 - dilate(thresholded_coins, 5)
```

```
utils.plot_image(samples.special.coins)
utils.plot_images([foreground, background], titles=['foreground', 'background'], space='gray
```



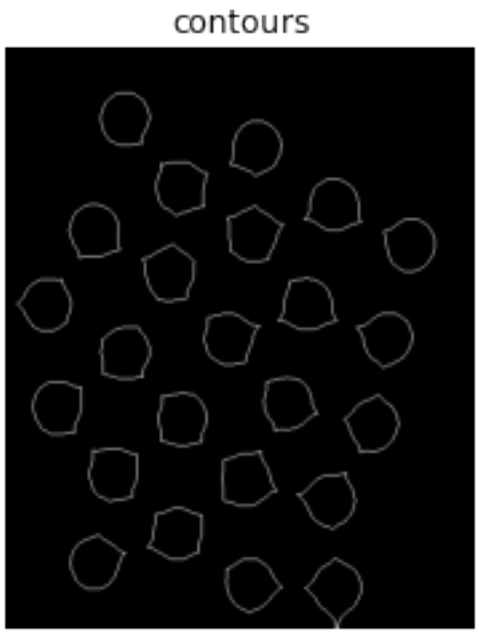
Per ottenere monete separate, erodiamo ancora di più:

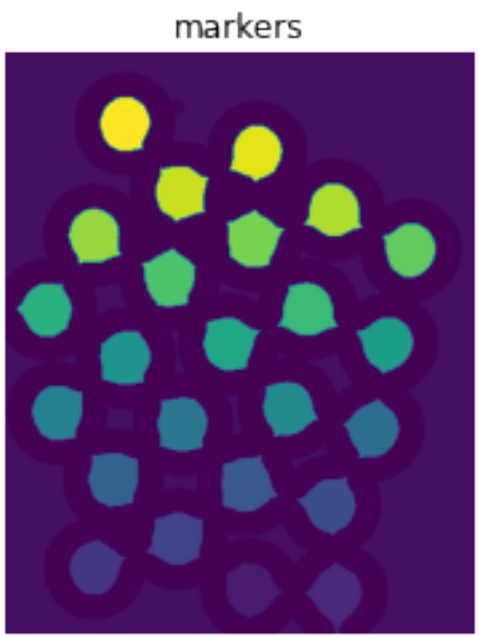
```
definitely_coins = erode(foreground, 7)

utils.plot_image(definitely_coins, space='gray')
```



Possiamo ora tracciare i contorni e ”riempirli” con numeri progressivi a partire da 2 (1 lo usiamo per lo sfondo):

```
markers = np.zeros([samples.special.coins.shape[0], samples.special.coins.shape[1]])
# Riempi di 1 lo sfondo
markers += (background / 255) * 1

contours, _ = cv.findContours(definitely_coins, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
drawn = np.zeros_like(samples.special.coins)
cv.drawContours(drawn, contours, -1, (255, 255, 255), 1)
utils.plot_image(drawn, title='contours')

i = 2
for contour in contours:
    markers = cv.drawContours(markers.copy(), [contour], -1, (i), cv.FILLED)
    i += 1

# Immagine a falsi colori
utils.plot_image(markers, title='markers')
```

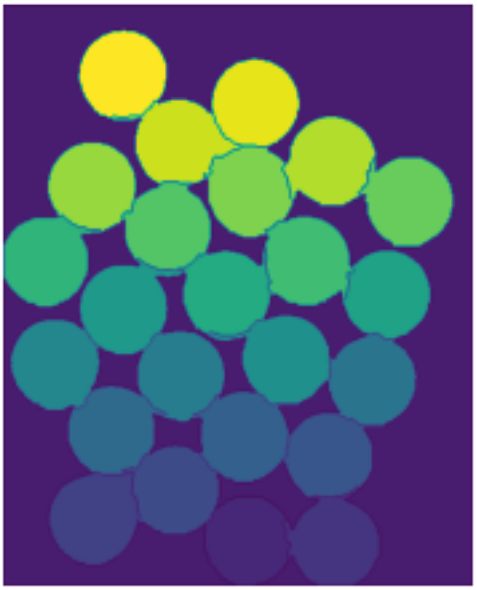




Infine, lanciamo l'algoritmo watershed:

```
found_markers = cv.watershed(samples.special.coins, markers.astype(np.int32))
```

```
utils.plot_image(found_markers)

# Il contorno ha valore -1
coins_with_contours = np.copy(samples.special.coins)
coins_with_contours[found_markers == -1] = [255, 0, 0]
utils.plot_image(coins_with_contours)
```



Elaborazione Video

Tutte le operazioni di cui abbiamo parlato funzionano anche con i video:

```
# Tutto questo blocco di codice serve solo per lavorare con la webcam in Colab, non serve s

# function to convert the JavaScript object into an OpenCV image
from IPython.display import display, Javascript, Image
from google.colab.output import eval_js
from base64 import b64decode, b64encode
import numpy as np
import PIL
import io
import html
import time

def js_to_image(js_reply):
  """
  Params:
          js_reply: JavaScript object containing image from webcam
  Returns:
          img: OpenCV BGR image
  """
  # decode base64 image
  image_bytes = b64decode(js_reply.split(',')[1])
```

```
  # convert bytes to numpy array
  jpg_as_np = np.frombuffer(image_bytes, dtype=np.uint8)
  # decode numpy array into OpenCV BGR image
  img = cv.imdecode(jpg_as_np, flags=1)
  img = cv.cvtColor(img, cv.COLOR_BGR2RGB)

  return img

# function to convert OpenCV Rectangle bounding box image into base64 byte string to be ove
def bbox_to_bytes(bbox_array):
  """
  Params:
          bbox_array: Numpy array (pixels) containing rectangle to overlay on video stream.
  Returns:
        bytes: Base64 image byte string
  """
  # convert array into PIL image
  bbox_PIL = PIL.Image.fromarray(bbox_array, 'RGBA')
  iobuf = io.BytesIO()
  # format bbox into png for return
  bbox_PIL.save(iobuf, format='png')
  # format return string
  bbox_bytes = '{}'.format((str(b64encode(iobuf.getvalue()), 'utf-8'))

  return bbox_bytes

# JavaScript to properly create our live video stream using our webcam as input
def video_stream():
  js = Javascript('''
    var video;
    var div = null;
    var stream;
    var captureCanvas;
    var imgElement;
    var labelElement;

    var pendingResolve = null;
    var shutdown = false;

    function removeDom() {
       stream.getVideoTracks()[0].stop();
       video.remove();
       div.remove();
       video = null;
       div = null;
       stream = null;
```

```
   imgElement = null;
   captureCanvas = null;
   labelElement = null;
}

function onAnimationFrame() {
  if (!shutdown) {
    window.requestAnimationFrame(onAnimationFrame);
  }
  if (pendingResolve) {
    var result = "";
    if (!shutdown) {
      captureCanvas.getContext('2d').drawImage(video, 0, 0, 640, 480);
      result = captureCanvas.toDataURL('image/jpeg', 0.8)
    }
    var lp = pendingResolve;
    pendingResolve = null;
    lp(result);
  }
}

async function createDom() {
  if (div !== null) {
    return stream;
  }

  div = document.createElement('div');
  div.style.border = '2px solid black';
  div.style.padding = '3px';
  div.style.width = '100%';
  div.style.maxWidth = '600px';
  document.body.appendChild(div);

  const modelOut = document.createElement('div');
  modelOut.innerHTML = "<span>Status:</span>";
  labelElement = document.createElement('span');
  labelElement.innerText = 'No data';
  labelElement.style.fontWeight = 'bold';
  modelOut.appendChild(labelElement);
  div.appendChild(modelOut);

  video = document.createElement('video');
  video.style.display = 'block';
  video.width = div.clientWidth - 6;
  video.setAttribute('playsinline', '');
  video.onclick = () => { shutdown = true; };
```

```
  stream = await navigator.mediaDevices.getUserMedia(
      {video: { facingMode: "environment"}});
  div.appendChild(video);

  imgElement = document.createElement('img');
  imgElement.style.position = 'absolute';
  imgElement.style.zIndex = 1;
  imgElement.onclick = () => { shutdown = true; };
  div.appendChild(imgElement);

  const instruction = document.createElement('div');
  instruction.innerHTML =
      '<span style="color: red; font-weight: bold;">' +
      'Clicca qui o sul video per fermare</span>';
  div.appendChild(instruction);
  instruction.onclick = () => { shutdown = true; };

  video.srcObject = stream;
  await video.play();

  captureCanvas = document.createElement('canvas');
  captureCanvas.width = 640; //video.videoWidth;
  captureCanvas.height = 480; //video.videoHeight;
  window.requestAnimationFrame(onAnimationFrame);

  return stream;
}
async function stream_frame(label, imgData) {
  if (shutdown) {
    removeDom();
    shutdown = false;
    return '';
  }

  var preCreate = Date.now();
  stream = await createDom();

  var preShow = Date.now();
  if (label != "") {
    labelElement.innerHTML = label;
  }

  if (imgData != "") {
    var videoRect = video.getClientRects()[0];
    imgElement.style.top = videoRect.top + "px";
    imgElement.style.left = (videoRect.left + videoRect.width) + "px";
```

```
        imgElement.style.width = videoRect.width + "px";
        imgElement.style.height = videoRect.height + "px";
        imgElement.src = imgData;
      }

      var preCapture = Date.now();
      var result = await new Promise(function(resolve, reject) {
        pendingResolve = resolve;
      });
      shutdown = false;

      return {'create': preShow - preCreate,
              'show': preCapture - preShow,
              'capture': Date.now() - preCapture,
              'img': result};
    }
    ''')

  display(js)

def video_frame(label, bbox):
  data = eval_js('stream_frame("{}", "{}")'.format(label, bbox))
  return data

def apply_transformation(func):
    video_stream()
    # label for video
    label_html = 'Capturing...'
    # initialze bounding box to empty
    bbox = ''
    count = 0
    while True:
        js_reply = video_frame(label_html, bbox)
        if not js_reply:
            break

        # convert JS response to OpenCV Image
        img = js_to_image(js_reply["img"])

        # create transparent overlay for bounding box
        bbox_array = np.zeros([480,640,4], dtype=np.uint8)

        # grayscale image for face detection
        transformed_image = func(img)
        bbox_array[:, :, :3] = transformed_image
```

```
        bbox_array[:,:,3] = (bbox_array.max(axis = 2) > 0 ).astype(int) * 255
        bbox_bytes = bbox_to_bytes(bbox_array)
        bbox = bbox_bytes

def red_tint(image):
    image = np.copy(image).astype(np.int16)
    image[:, :, 0] += 40
    image[:, :, 1] -= 40
    image[:, :, 2] -= 40
    return np.clip(image, 0, 255).astype(np.uint8)

def red_threshold(image):
    _, red_thresholded_image = cv.threshold(image[:, :, 0], 200, 255, cv.THRESH_BINARY + cv.
    _, green_thresholded_image = cv.threshold(image[:, :, 1], 100, 255, cv.THRESH_BINARY)
    _, blue_thresholded_image = cv.threshold(image[:, :, 2], 100, 255, cv.THRESH_BINARY)
    only_red = np.minimum(red_thresholded_image, 255 - green_thresholded_image, 255 - blue_t
    modified_image = np.copy(image)
    modified_image[:, :, 0] = only_red
    modified_image[:, :, 1] = only_red
    modified_image[:, :, 2] = only_red
    return modified_image

def red_bounding_box(image):
    _, red_thresholded_image = cv.threshold(image[:, :, 0], 200, 255, cv.THRESH_BINARY + cv.
    _, green_thresholded_image = cv.threshold(image[:, :, 1], 100, 255, cv.THRESH_BINARY)
    _, blue_thresholded_image = cv.threshold(image[:, :, 2], 100, 255, cv.THRESH_BINARY)
    only_red = np.minimum(red_thresholded_image, 255 - green_thresholded_image, 255 - blue_t
    contours, _ = cv.findContours(only_red, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
    # Ordina dal più grande al più piccolo
    contours = sorted(contours, key=lambda x: cv.contourArea(x), reverse=True)
    # Disegna un rettangolo intorno all'oggetto più grande
    if len(contours) == 0:
        return image
    x, y, width, height = cv.boundingRect(contours[0])
    image_with_rects = cv.rectangle(image.copy(), (x, y), (x+width, y+height), (0, 0, 255),
    return image_with_rects

#apply_transformation(red_tint)
#apply_transformation(red_threshold)
#apply_transformation(red_bounding_box)
```

Esercizio: Occhiali da Sole

Scrivete una funzione che disegna degli occhiali da sole addosso alla persona di una foto. Opzionalmente potete anche utilizzare la webcam.

```python
def draw_sunglasses(image, x, y, width, height):
    sunglasses_alpha = cv.resize(np.copy(samples.special.sunglasses_alpha), [width, height])
    actual_width = max(min(x + width, image.shape[1]) - x, 0)
    actual_height = max(min(y + height, image.shape[0]) - y, 0)
    if actual_width == 0 or actual_height == 0:
        return image
    sunglasses_alpha = sunglasses_alpha[:actual_height, :actual_width, :]
    image = np.copy(image)
    mask = sunglasses_alpha[:, :, 3] / 255
    mask = mask.reshape([sunglasses_alpha.shape[0], sunglasses_alpha.shape[1], 1])
    image[y:y + actual_height, x:x + actual_width, :3] = sunglasses_alpha[:, :, :3] * mask +

    return image

def sunglasses_on_face(image):
    return image

utils.plot_image(sunglasses_on_face(samples.bird))
# apply_transformation(sunglasses_on_face)
```

Esercizio Extra: Conteggio

Contate quante persone ci sono nella seguente immagine:

`utils.plot_image(samples.special.people)`

Cose non Viste

OpenCV ha molti strumenti e i tutorial della documentazione sono molto buoni. Alcuni consigliati:

- Creare immagini HDR
- Stimare la posizione 3D di un oggetto (utile per la realtà aumentata)
- Inpainting (il "Riempimento Intelligente" di Photoshop)

Altri strumenti utili non-neurali:

- AprilTag è un sistema per tracciare con alta qualità qualunque oggetto usando cubi di cartone e una webcam. Utile per applicazioni in robotica, tracciamento del corpo umano e anche VR
- Scikit-image contiene alcune tecniche avanzate per tracciare contorni, segmentare immagini e altri utilizzi di Computer Vision

```
!git clone https://github.com/samuelemarro/ml_intro
```

```
fatal: destination path 'ml_intro' already exists and is not an empty directory.
```

Caricare Dati

pandas è una libreria estremamente comoda per caricare dati tabulari (=tipo Excel). Se avete un file CSV (Comma Separated Values) formattato così:

```
Column1,Column2,Column3
foo,1,4.1
bar,4,9.3
...
```

Chiamando `pandas.read_csv(percorso)` pandas userà la prima riga per dare automaticamente un nome alle colonne. Per esempio, carichiamo `lucca_comics.csv`:

```
import pandas as pd
# names=['nome1', 'nome2', ...] è utile per quando la prima riga non contiene i nomi delle c
lucca_comics = pd.read_csv('./ml_intro/data/lucca_comics.csv')

print(lucca_comics)
```

```
    Year  Visitors
0   1993     29000
1   2002     50000
2   2003     52000
3   2004     50000
4   2006     84000
5   2007     90000
6   2008    130000
7   2009    140000
8   2010    135000
9   2011    155000
10  2012    180000
11  2013    200000
12  2014    240000
13  2015    220000
14  2016    270000
15  2017    243000
16  2018    251000
17  2019    270000
18  2021    100000
19  2022    271000
```

Alcuni metodi utili:

```python
lucca_comics_copy = lucca_comics.copy() # Restituisce una copia
print('Head:')
print(lucca_comics.head(5)) # Prime 5 righe
print('Tail:')
print(lucca_comics.tail(5)) # Ultime 5 righe
print('Shape:')
print(lucca_comics.shape) # Restituisce [numero righe, numero colonne]
print('len:')
print(len(lucca_comics)) # Restituisce il numero di righe
print('Columns:')
print(lucca_comics.columns) # Restituisce i nomi delle colonna
print('Drop (axis = 0):')
print(lucca_comics.drop([2, 4, 5], axis=0)) # Restituisce una copia della tabella senza le r
print('Drop (axis = 1):')
print(lucca_comics.drop('Visitors', axis=1)) # Restituisce una copia della tabella senza la
# Nota: si può passare inplace=True per modificare la tabella originale
```

```
Head:
   Year  Visitors
0  1993     29000
1  2002     50000
2  2003     52000
3  2004     50000
4  2006     84000
Tail:
    Year  Visitors
15  2017    243000
16  2018    251000
17  2019    270000
18  2021    100000
19  2022    271000
Shape:
(20, 2)
len:
20
Columns:
Index(['Year', 'Visitors'], dtype='object')
Drop (axis = 0):
    Year  Visitors
0   1993     29000
1   2002     50000
3   2004     50000
6   2008    130000
7   2009    140000
8   2010    135000
9   2011    155000
```

```
10  2012    180000
11  2013    200000
12  2014    240000
13  2015    220000
14  2016    270000
15  2017    243000
16  2018    251000
17  2019    270000
18  2021    100000
19  2022    271000
Drop (axis = 1):
    Year
0   1993
1   2002
2   2003
3   2004
4   2006
5   2007
6   2008
7   2009
8   2010
9   2011
10  2012
11  2013
12  2014
13  2015
14  2016
15  2017
16  2018
17  2019
18  2021
19  2022
```

Se proviamo a indicizzare direttamente, accetta solo nomi di colonne. Per scegliere una o più righe si usa `.iloc`:

```
print(lucca_comics.iloc[3:8]) # Scegli le righe con indice in [3, 8)
```

```
   Year  Visitors
3  2004     50000
4  2006     84000
5  2007     90000
6  2008    130000
7  2009    140000
```

NumPy lavora tranquillamente su colonne singole:

```
import numpy as np
```

```
print(np.mean(lucca_comics['Visitors']))
```

```
158000.0
```

Le colonne supportano la maggior parte delle operazioni di NumPy, ma NON sono tensori NumPy. Per renderli tensori NumPy usiamo `.to_numpy()`

```
lucca_comics['Visitors'].sum() # Funziona
# lucca_comics['Visitors'].reshape(1, -1) # Fallisce
lucca_comics['Visitors'].to_numpy().reshape(1, -1) # Funziona
```

```
array([[ 29000,  50000,  52000,  50000,  84000,  90000, 130000, 140000,
        135000, 155000, 180000, 200000, 240000, 220000, 270000, 243000,
        251000, 270000, 100000, 271000]])
```

Se vogliamo trattare tutta la tabella come un unico array NumPy, usiamo sempre `.to_numpy()`:

```
print(lucca_comics.to_numpy())
```

```
[[  1993  29000]
 [  2002  50000]
 [  2003  52000]
 [  2004  50000]
 [  2006  84000]
 [  2007  90000]
 [  2008 130000]
 [  2009 140000]
 [  2010 135000]
 [  2011 155000]
 [  2012 180000]
 [  2013 200000]
 [  2014 240000]
 [  2015 220000]
 [  2016 270000]
 [  2017 243000]
 [  2018 251000]
 [  2019 270000]
 [  2021 100000]
 [  2022 271000]]
```

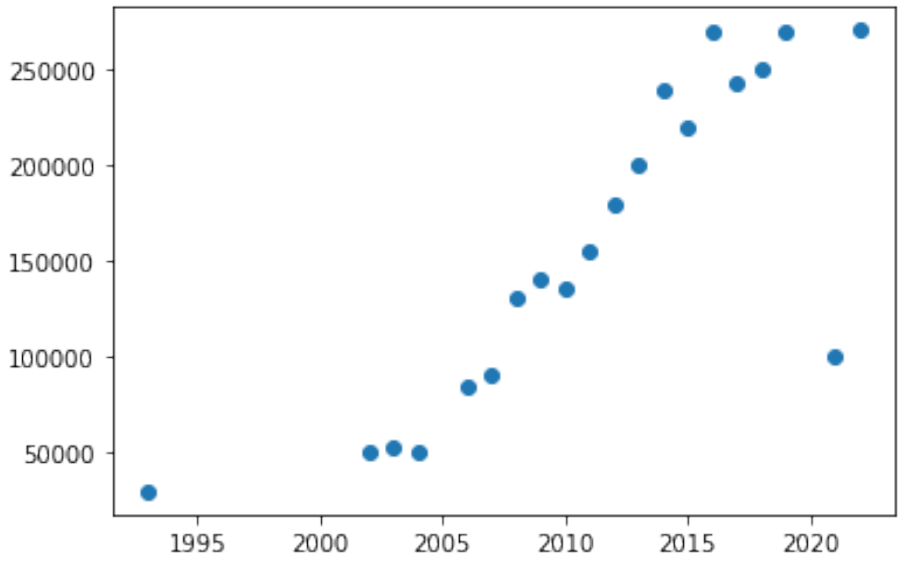
Pandas ha anche molti metodi per convertire le tabelle in cose utili:

- `to_csv()`
- `to_excel()`
- `to_html()`
- `to_latex()`
- `to_markdown()`

Preprocessing Dati

Prima di analizzare i dati conviene sempre farsi un'idea di come sono fatti. È quindi sempre utile guardare almeno a occhio i dati e fare un plot

```
import matplotlib.pyplot as plt

plt.scatter(lucca_comics['Year'], lucca_comics['Visitors'])
plt.show()
```



I problemi più comuni per i dati sono:

- Dati mancanti
- Outlier (dati "insoliti")
- Noise
- Problemi numerici
- Scale diverse di dati*
- Artefatti dovuti alla raccolta dati*

*= Vedremo dopo

Dati Mancanti

Problema: Mancano dei dati

Soluzione sbagliata: Inventarsi i dati

- Quando si inventano dati, si mettono anche inconsciamente delle ipotesi che potrebbero non essere vere

- Se inventate dati per una ricerca e non lo dite, è falso accademico. Tre casi famosi:
- Jan Hendrik Schön, che falsificò dati su superconduttori organici (documentario su YouTube)
- Victor Ninov, che falsificò dati sulla scoperta degli elementi con numero atomico 116 e 118 (documentario su YouTube)
- Andrew Wakefield, che falsificò i dati su un legame tra vaccini e autismo (le sue ricerche erano finanziate da un gruppo antivax)

Soluzione corretta: usare una tecnica che tollera l'assenza di dati (tutte le tecniche che vedremo possono essere usate con dati mancanti, però generalmente più un modello è semplice, meno dati richiede)

Outlier

Problema: Alcuni dati non hanno senso o hanno valori insoliti

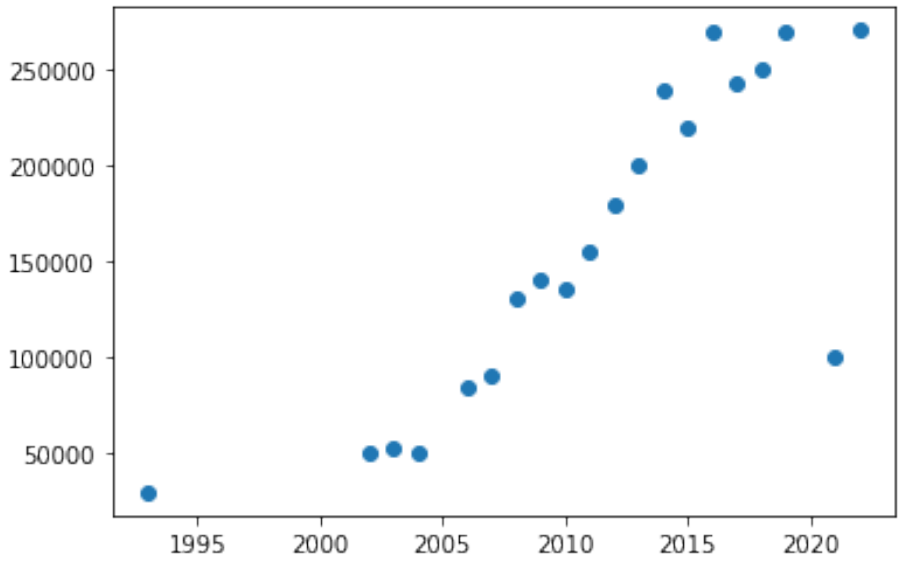
Soluzione sbagliata: fare finta di niente

- Gli outlier possono tranquillamente rendere un modello inutilizzabile
- Spesso gli outlier sono conseguenze di errori nella raccolta dati o eventi eccezionali
- Tenere gli outlier può essere una scelta sensata, ma deve essere una **scelta**

Soluzione corretta: Dire che ci sono outlier, e scegliere se o ignorarli o tenerli.

- Se c'è una buona spiegazione per un outlier (es. Covid), ben venga, possiamo ignorarlo
- "Non quadra con i dati, quindi è un outlier" non è una buona spiegazione!
- Se non c'è una buona spiegazione, si può comunque ignorarli, ma bisogna essere assolutamente certi che questi outlier non siano sintomo di qualcosa sotto che non avete capito
- La meccanica newtoniana coincide perfettamente con i dati sui periodi orbitali dei pianeti, eccetto per Mercurio, dove sbaglia leggermente
- Inizialmente si pensò un errore di misurazione o che ci fosse un pianeta non ancora scoperto che sballava i risultati
- In realtà la meccanica newtoniana era sbagliata: la soluzione fu inventare una nuova teoria della fisica, la relatività generale (pagina Wikipedia del problema di Mercurio)
- In ogni caso serve massima trasparenza: segnate sempre sui grafici gli outlier, dite esplicitamente se li avete ignorati o no, spiegate sempre perché li avete ignorati o no

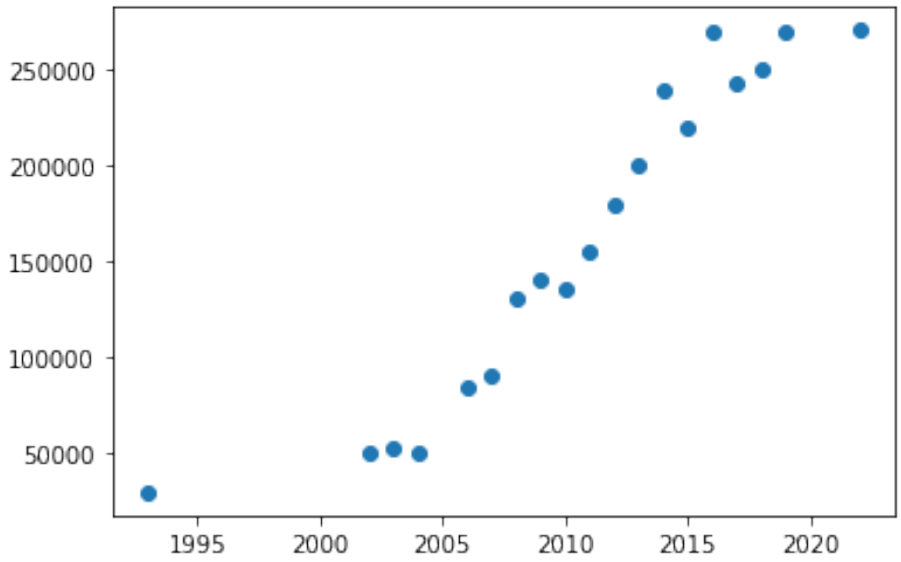
```
plt.scatter(lucca_comics['Year'], lucca_comics['Visitors'])
plt.show()
```



Senza il 2021:

```
# Il 2021 è la riga con indice 18
index_2021 = list(lucca_comics['Year']).index(2021)
print(index_2021)
# Drop con axis=0 toglie una riga
lucca_comics_no_2021 = lucca_comics.drop(lucca_comics.index[index_2021], axis=0)

plt.scatter(lucca_comics_no_2021['Year'], lucca_comics_no_2021['Visitors'])

plt.show()
```

```
18
```



scikit-learn è una libreria di Machine Learning specializzata nel Machine Learning "classico" (non reti neurali). scikit-learn ha anche molte tecniche di rilevazione automatica di outlier:

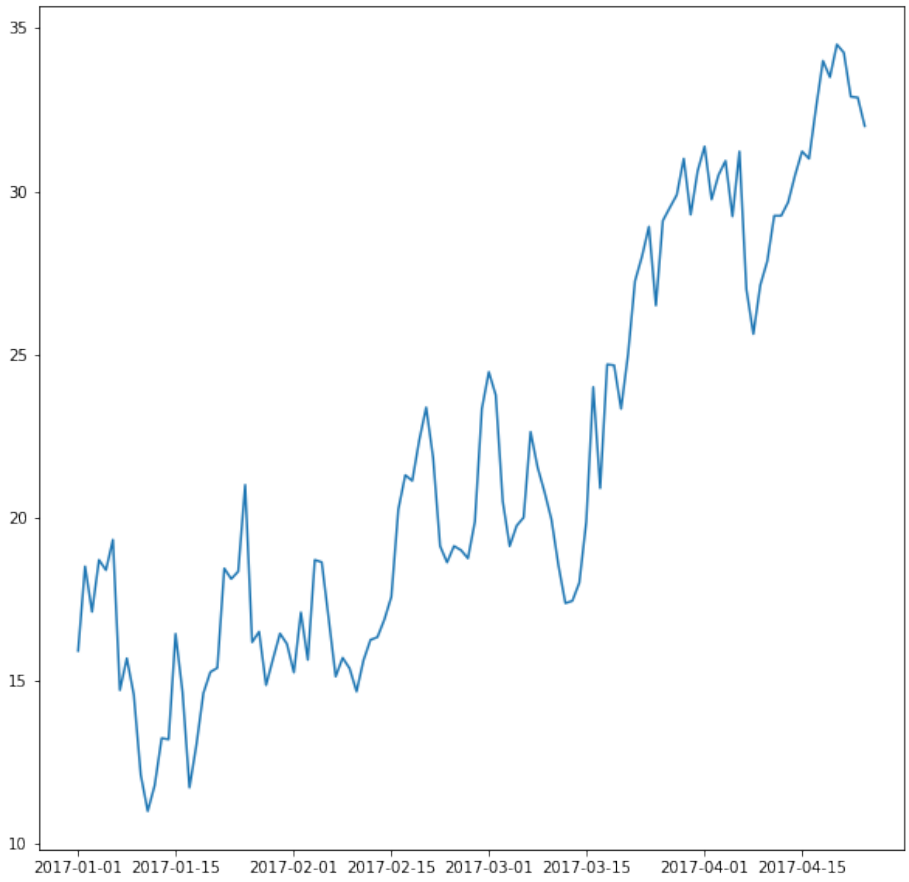
Alcuni esempi di come ragionano:

Importante: il rilevamento automatico di outlier è molto utile, ma bisogna sempre usarlo con cura per evitare i problemi di cui sopra

Noise

Problema: C'è del noise (disturbo di sottofondo) nella raccolta dati. La causa può essere la scarsa precisione dello strumento oppure il fatto che c'è una componente casuale nel fenomeno. Per esempio, i dati della temperatura hanno molto noise:

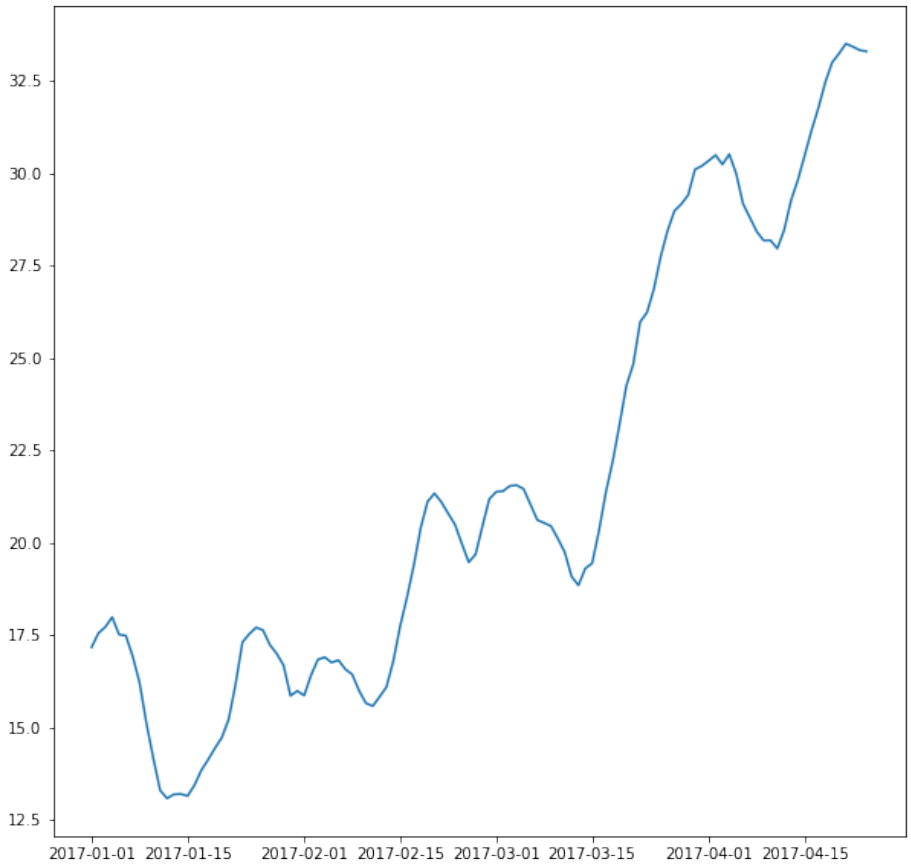
```
delhi_climate = pd.read_csv('./ml_intro/data/delhi_climate.csv')

plt.figure(figsize=(10, 10))
# Bisogna esplicitamente leggere le stringhe (che hanno formato tipo 2022-10-25) come date
delhi_dates = pd.to_datetime(delhi_climate['date'])
plt.plot(delhi_dates, delhi_climate['meantemp'])
plt.show()
```



Non c'è una soluzione perfetta per questi problemi. Per alcune analisi molto semplici si può "ammorbidire" i dati, per esempio calcolando la media mobile:

```
def moving_average(x, window_size):
    averages = []
    for i in range(len(x)):
        #  window/2  window/2
        # |------- i -------|

        window_start = int(i - window_size / 2)
        window_end = int(i + window_size / 2)

        # Se window_start o window_end vanno oltre i limiti, li clippiamo
        window_start = np.clip(window_start, 0, len(x))
        window_end = np.clip(window_end, 0, len(x))

        window_average = np.mean(x[window_start:window_end])
```

```
        averages.append(window_average)
    return averages

plt.figure(figsize=(10, 10))
plt.plot(delhi_dates, moving_average(delhi_climate['meantemp'], window_size=7))
plt.show()
```



Nella maggior parte dei casi, tuttavia, bisogna fare molta attenzione, perché il fatto che abbiamo fatto la media mobile ha introdotto un **artefatto** (= qualcosa nei dati che in realtà non c'è).

Se ora addestriamo un modello a prevedere la temperatura futura, penserà che la temperatura è qualcosa di "morbido", ma in realtà non è così. È un problema? Ciò dipende caso per caso

A volte è più semplice usare un modello che si sa essere resistente al noise. Come regola generale, più un modello è semplice e più resiste al noise.

Problemi Numerici

Problema: A volte i dati sono talmente grandi o talmente piccoli che non è possibile fare operazioni su di essi senza avere perdite di precisione o over/underflow.

Soluzione: Nel 99% dei casi si può riscalare i dati in modo che siano più facili da manipolare. A volte i problemi numerici non sono nei dati, ma nell'algoritmo: in quei casi si cerca di rimpiazzare l'algoritmo con qualcosa di più numericamente stabile.

Esercizio: Osservazioni dati

Fate ipotesi e identificate potenziali outlier nel seguente dataset:

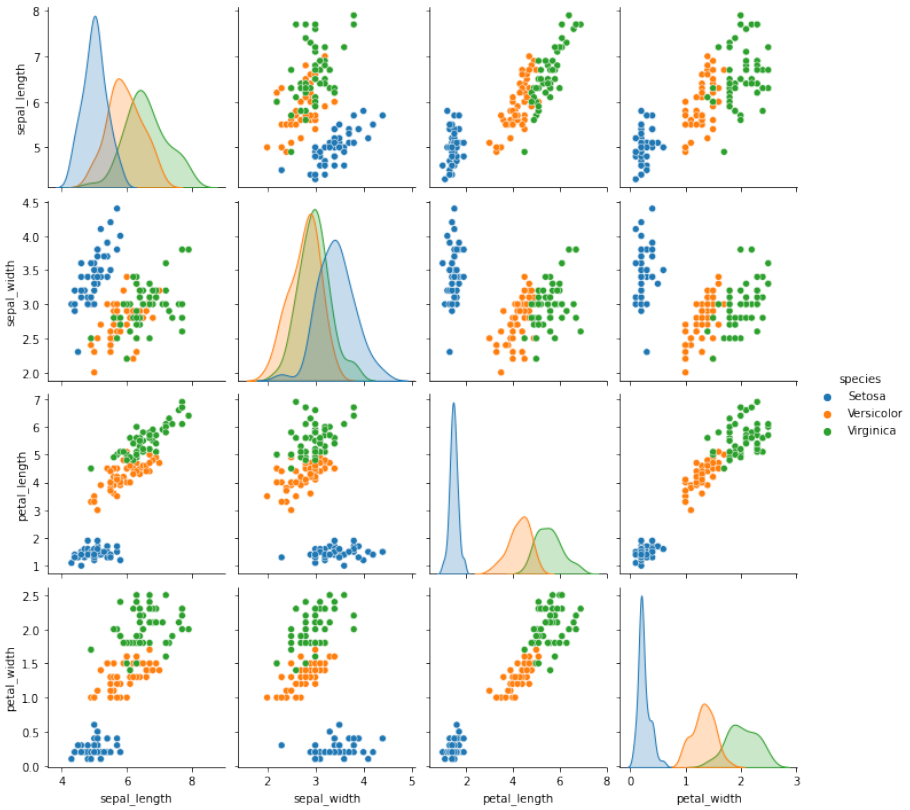
```
iris_dataset = pd.read_csv('./ml_intro/data/iris.csv')
print(iris_dataset.head())
```

```
   sepal_length  sepal_width  petal_length  petal_width species
0           5.1          3.5           1.4          0.2  Setosa
1           4.9          3.0           1.4          0.2  Setosa
2           4.7          3.2           1.3          0.2  Setosa
3           4.6          3.1           1.5          0.2  Setosa
4           5.0          3.6           1.4          0.2  Setosa
```

Il metodo pairplot può essere utile:

```
import seaborn as sns

print(iris_dataset['species'])
sns.pairplot(iris_dataset, hue='species')
```

```
0         Setosa
1         Setosa
2         Setosa
3         Setosa
4         Setosa
         ...
145    Virginica
146    Virginica
147    Virginica
148    Virginica
149    Virginica
Name: species, Length: 150, dtype: object

<seaborn.axisgrid.PairGrid at 0x7fd22ff966d0>
```



Classificazione

La classificazione può essere vista come l'imparare una funzione che prende in input le feature dei dati e restituisce la classe. Ha quindi qualche somiglianza con la regressione: infatti molti modelli di regressione possono essere usati anche per fare classificazione.

Lavoreremo con un dataset molto celebre, l'Iris Dataset. L'Iris Dataset contiene 150 righe e ha 5 feature:

- Lunghezza del sepalo
- Larghezza del sepalo
- Lunghezza del petalo
- Larghezza del petalo
- Specie di iris (Versicolor, Setosa, Virginica)

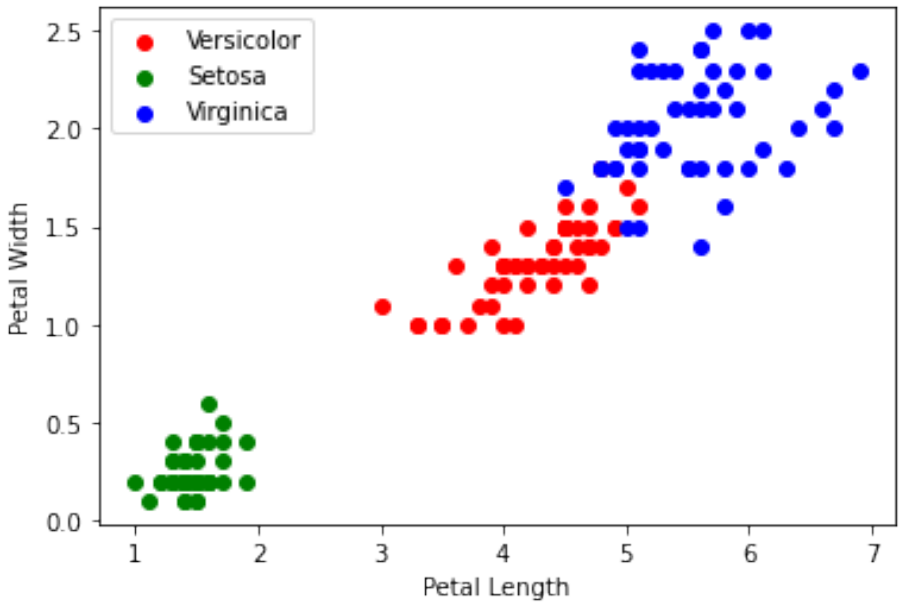
```
iris_dataset = pd.read_csv('./ml_intro/data/iris.csv')
print(iris_dataset.head(5))
```

```
   sepal_length  sepal_width  petal_length  petal_width species
```

```
0           5.1          3.5           1.4          0.2  Setosa
1           4.9          3.0           1.4          0.2  Setosa
2           4.7          3.2           1.3          0.2  Setosa
3           4.6          3.1           1.5          0.2  Setosa
4           5.0          3.6           1.4          0.2  Setosa
```

Mostriamo uno scatter plot della lunghezza e larghezza del petalo:

```
species_names = ['Versicolor', 'Setosa', 'Virginica']
colors = ['red', 'green', 'blue']
for species, color in zip(species_names, colors):
    matching_rows = iris_dataset.loc[iris_dataset['species'] == species]
    plt.scatter(matching_rows['petal_length'], matching_rows['petal_width'], color=color)
    plt.xlabel('Petal Length')
    plt.ylabel('Petal Width')
plt.legend(species_names)
plt.show()
```



La prima cosa da fare è dividere il dataset nel dataset di addestramento e di test. Useremo 80% del dataset per addestrare e 20% per testare

```
from sklearn.model_selection import train_test_split

no_species = iris_dataset.drop('species', axis=1)

# È pratica comune scrivere la X in maiuscolo e la y in minuscolo. La ragione è che X
# train_test_split(x, y, parametri)
```

```
# random_state imposta il seed, quindi useremo tutti lo stesso split
X_train, X_test, y_train, y_test = train_test_split(no_species, iris_dataset['species'], tra

# Per la classificazione binaria ci concentreremo su Versicolor (-1) vs Virginica (+1)
no_setosa = iris_dataset.loc[iris_dataset['species'] != 'Setosa']
species_class = np.array([1 if species == 'Virginica' else -1 for species in no_setosa['spec
#no_setosa['species'] = no_setosa['species'] == 'Virginica'
no_setosa_no_species = no_setosa.drop('species', axis=1)
X_train_no_setosa, X_test_no_setosa, y_train_no_setosa, y_test_no_setosa = train_test_split(
```

Classificazione Lineare

I classificatori lineari possono dividere in due classi, che corrispondono ai due semipiani dello spazio (o iperspazio, se siamo in più dimensioni).

Proviamo prima usando solo due feature (lunghezza e larghezza del petalo):

```
from sklearn.linear_model import Ridge

X_train_only_petal = X_train_no_setosa.drop(['sepal_length', 'sepal_width'], axis=1)
X_test_only_petal = X_test_no_setosa.drop(['sepal_length', 'sepal_width'], axis=1)

# Ridge è uguale a LinearRegression, ma fa classificazione
linear_iris_model = Ridge()
linear_iris_model.fit(X_train_only_petal, y_train_no_setosa)
```

```
Ridge()
```

Calcoliamo l'accuratezza sul dataset di test:

```
from sklearn.metrics import accuracy_score

predicted_scores = linear_iris_model.predict(X_test_only_petal)
print(predicted_scores)
# Score positivo significa +1, score negativo significa -1
# np.where(cond, a, b) restituisce a se cond è vera e b altrimenti
predicted_labels = np.where(predicted_scores > 0, 1, -1)
print(predicted_labels)
print(y_test_no_setosa)

# accuracy_score(vero, predetto)
accuracy = accuracy_score(y_test_no_setosa, predicted_labels)
print(f'Accuracy: {accuracy * 100}%')
```

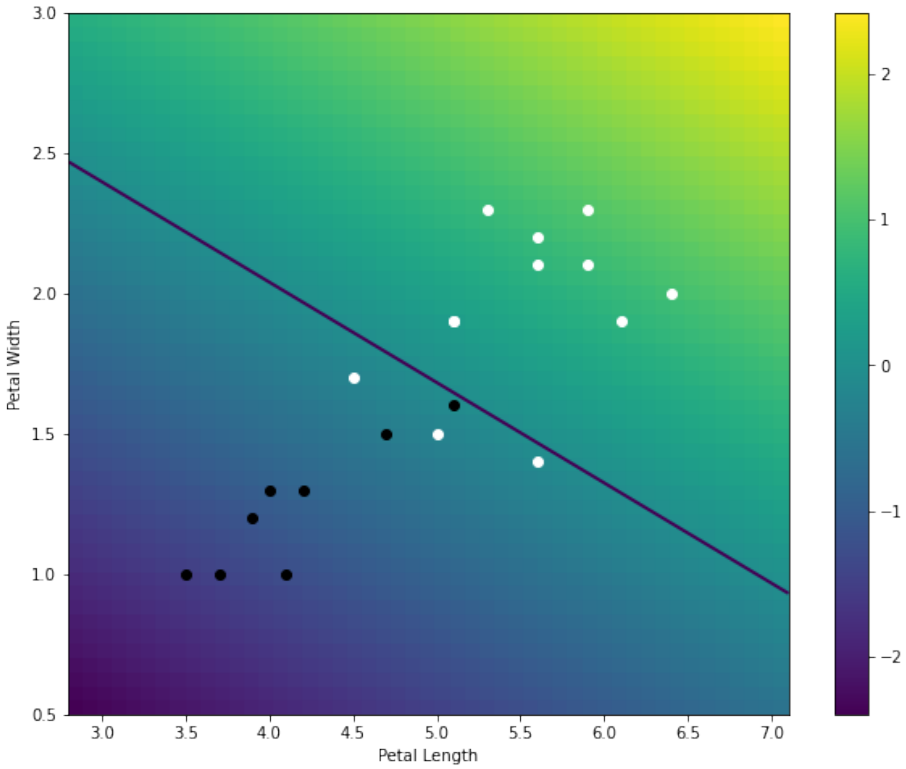
```
[ 0.74448217 -0.08225392 -0.05649279  0.99695319  1.14449963 -1.22282369
 -0.3525257   0.89202861 -0.22184     0.88312823  0.30886316 -0.90102964
 -0.19607887  0.900929    0.30886316 -1.06637686 -1.39707129 -0.81390584
  0.7702433  -1.48419509]
[ 1 -1 -1  1  1 -1 -1  1 -1  1  1 -1 -1  1  1 -1 -1 -1  1 -1]
```

```
[ 1  1 -1  1  1 -1 -1  1  1  1  1 -1  1  1  1 -1 -1 -1  1 -1]
Accuracy: 85.0%
```

Il vantaggio di usare solo due feature è che è facile da visualizzare:

```
# Potete saltare tutta questa parte di visualizzazione, è solo per fare grafici carini

# Genera una griglia di punti per la heatmap
length_range = np.linspace(X_train_only_petal['petal_length'].min() - 0.2, X_train_only_peta
width_range = np.linspace(X_train_only_petal['petal_width'].min() - 0.5, X_train_only_petal
visualization_X, visualization_Y = np.meshgrid(length_range, width_range)
visualization_X = visualization_X.flatten()
visualization_Y = visualization_Y.flatten()

data_for_model = np.concatenate([visualization_X.reshape(-1, 1), visualization_Y.reshape(-1,

visualization_Z = linear_iris_model.predict(data_for_model)

visualization_X = visualization_X.reshape(50, 50)
visualization_Y = visualization_Y.reshape(50, 50)
visualization_Z = visualization_Z.reshape(50, 50)

plt.figure(figsize=(10, 8))
im = plt.pcolor(visualization_X, visualization_Y, visualization_Z)
plt.colorbar(im)
plt.contour(visualization_X, visualization_Y, visualization_Z, [0], linewidths=2)
plt.scatter(X_test_only_petal['petal_length'][y_test_no_setosa == 1], X_test_only_petal['pet
plt.scatter(X_test_only_petal['petal_length'][y_test_no_setosa == -1], X_test_only_petal['pe
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not have val
  "X does not have valid feature names, but"
```



`coef_` ci dice quanto ciascuna feature contribuisce alla classe positiva

```
print(X_train_only_petal.columns)
print(linear_iris_model.coef_)
```

```
Index(['petal_length', 'petal_width'], dtype='object')
[0.435619   1.21785316]
```

Se invece di usare solo due feature ne usiamo di più, l'accuratezza sale:

```
linear_iris_model_whole = Ridge()
linear_iris_model_whole.fit(X_train_no_setosa, y_train_no_setosa)
```

```
Ridge()
```

```
predicted_scores = linear_iris_model_whole.predict(X_test_no_setosa)
print(predicted_scores)
# Score positivo significa +1, score negativo significa -1
# np.where(cond, a, b) restituisce a se cond è vera e b altrimenti
predicted_labels = np.where(predicted_scores > 0, 1, -1)
print(list(predicted_labels))
print(list(y_test_no_setosa))
accuracy = accuracy_score(y_test_no_setosa, predicted_labels)
```

```
print(f'Accuracy: {accuracy * 100}%')
```

```
[ 0.77206604  0.28767469  0.17407942  0.5189912   1.14318003 -1.21312121
 -0.62126738  1.10538002  0.19586406  0.84642794  0.61005552 -0.72964551
  0.30556548  0.88750274  0.61005552 -1.10403428 -1.27068392 -0.70702482
  0.98071626 -1.08270746]
[1, 1, 1, 1, 1, -1, -1, 1, 1, 1, 1, -1, 1, 1, 1, -1, -1, -1, 1, -1]
[1, 1, -1, 1, 1, -1, -1, 1, 1, 1, 1, -1, 1, 1, 1, -1, -1, -1, 1, -1]
Accuracy: 95.0%
```

```
print(X_train_no_setosa.columns)
print(linear_iris_model_whole.coef_)
```

```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width'], dtype='object')
[-0.30992417 -0.43313739  0.70120293  1.24663757]
```

Naive Bayes

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, y_train)
```

```
GaussianNB()
```

```
predicted_labels = gnb.predict(X_test)
print(list(predicted_labels))
print(list(y_test))
accuracy = accuracy_score(y_test, predicted_labels)
print(f'Accuracy: {accuracy * 100}%')
```

```
['Setosa', 'Versicolor', 'Versicolor', 'Setosa', 'Virginica', 'Versicolor', 'Virginica', 'Se
['Setosa', 'Versicolor', 'Versicolor', 'Setosa', 'Virginica', 'Versicolor', 'Virginica', 'Se
Accuracy: 96.66666666666667%
```

Regressione Logistica

Nonostante il nome, NON È UN TIPO DI REGRESSIONE! È un tipo di classificazione. Nella sua forma classica è binaria.

```
from sklearn.linear_model import LogisticRegression

logistic_model = LogisticRegression()
logistic_model.fit(X_train_no_setosa, y_train_no_setosa)
```

```
LogisticRegression()
```

```
predicted_labels = logistic_model.predict(X_test_no_setosa)
print(predicted_labels)
print(y_test_no_setosa)
```

```
accuracy = accuracy_score(y_test_no_setosa, predicted_labels)
print(f'Accuracy: {accuracy * 100}%')
```

```
[ 1  1  1  1  1 -1 -1  1 -1  1  1 -1 -1  1  1 -1 -1 -1  1 -1]
[ 1  1 -1  1  1 -1 -1  1  1  1  1 -1  1  1  1 -1 -1 -1  1 -1]
Accuracy: 85.0%
```

Sia la regressione logistica che la classificazione lineare possono essere estesi a varianti a classi multiple usando dei classificatori one-vs-rest:

Un classificatore one-vs-rest è un classificatore binario che dice se un input è una classe specifica (es. Setosa) oppure qualcos'altro (es. Versicolor o Virginica), senza però dire in quest'ultimo caso che classe specifica è.

La regressione logistica di scikit-learn supporta la classificazione a classi multiple senza dover fare modifiche:

```
logistic_model_multiclass = LogisticRegression()
logistic_model_multiclass.fit(X_train, y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWar
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

```
LogisticRegression()
```

```
predicted_labels = logistic_model_multiclass.predict(X_test)
print(list(predicted_labels))
print(list(y_test))
accuracy = accuracy_score(y_test, predicted_labels)
print(f'Accuracy: {accuracy * 100}%')
```

```
['Setosa', 'Versicolor', 'Versicolor', 'Setosa', 'Virginica', 'Versicolor', 'Virginica', 'Se
['Setosa', 'Versicolor', 'Versicolor', 'Setosa', 'Virginica', 'Versicolor', 'Virginica', 'Se
Accuracy: 96.66666666666667%
```

Decision Trees

I Decision Trees lavorano bene su più classi e possono gestire anche casi in cui le feature non sono linearmente separabili:

```
from sklearn.tree import DecisionTreeClassifier

tree_model = DecisionTreeClassifier()
tree_model.fit(X_train, y_train)
```

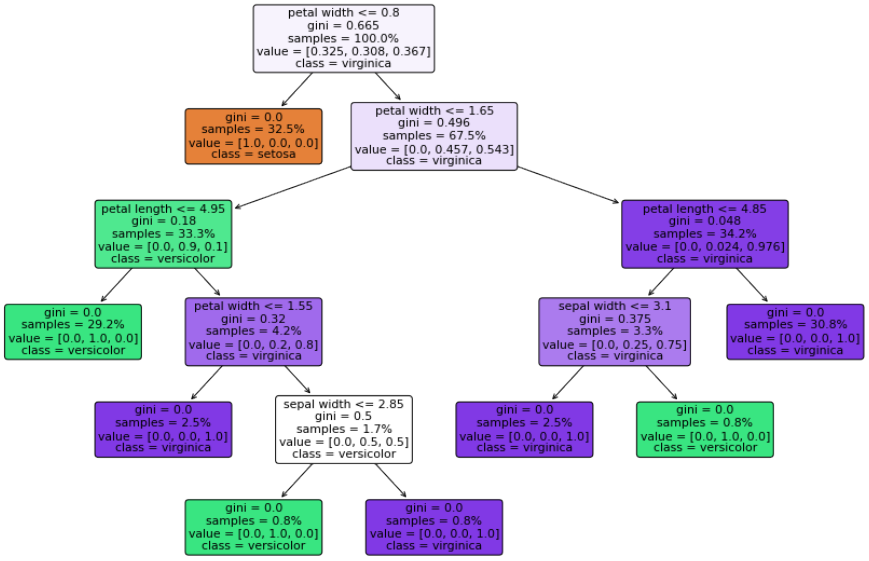
```
DecisionTreeClassifier()
```

```
predicted_labels = tree_model.predict(X_test)
print(list(predicted_labels))
print(list(y_test))
accuracy = accuracy_score(y_test, predicted_labels)
print(f'Accuracy: {accuracy * 100}%')
```

```
['Setosa', 'Versicolor', 'Versicolor', 'Setosa', 'Virginica', 'Versicolor', 'Virginica', 'Se
['Setosa', 'Versicolor', 'Versicolor', 'Setosa', 'Virginica', 'Versicolor', 'Virginica', 'Se
Accuracy: 96.66666666666667%
```

Inoltre, offrono anche una visualizzazione molto intuitiva di come ”ragionano”:

```
from sklearn.tree import plot_tree

plt.figure(figsize=(15, 10))

plot_tree(tree_model, proportion = True, rounded=True, filled=True, feature_names=['sepal le
pass
```



Esercizio: Classificazione

Il seguente dataset fornisce dati su come diversi fattori (età, etnia, livello di educazione...) si correlano all'essere ”ricco” (reddito >50k dollari) o no.

Fate uno split train/test 85/15 con seed 1 e addestrate un classificatore a vostra scelta sul dataset. Visualizzate poi i risultati come volete.

Per chi è masochista: usate un modello lineare o logistico.

```
rich_dataset = pd.read_csv('./ml_intro/data/rich.csv', index_col=False)
print(rich_dataset.head())
```

```
   age         workclass  fnlwgt  education  education-num  \
0   39         State-gov   77516  Bachelors             13
1   50  Self-emp-not-inc   83311  Bachelors             13
2   38           Private  215646    HS-grad              9
3   53           Private  234721       11th              7
4   28           Private  338409  Bachelors             13

       marital-status         occupation   relationship   race     sex  \
0       Never-married       Adm-clerical  Not-in-family  White    Male
1  Married-civ-spouse    Exec-managerial        Husband  White    Male
2            Divorced  Handlers-cleaners  Not-in-family  White    Male
3  Married-civ-spouse  Handlers-cleaners        Husband  Black    Male
4  Married-civ-spouse     Prof-specialty           Wife  Black  Female

   capital-gain  capital-loss  hours-per-week native-country
0          2174             0              40  United-States
1             0             0              13  United-States
2             0             0              40  United-States
3             0             0              40  United-States
4             0             0              40           Cuba
```

```
/usr/local/lib/python3.7/dist-packages/pandas/util/_decorators.py:311: ParserWarning: Lengt
  return func(*args, **kwargs)
```

Esercizio Extra: Classificazione Testo

Il seguente dataset contiene recensioni positive o negative di libri. La maggior parte dei modelli di Machine Learning richiede però di avere input numerici. Trovate un metodo semplice per distinguere le recensioni positive da quelle negative.

```
import json

with open('./ml_intro/data/positive-reviews.json') as f:
    positive_reviews = json.load(f)

with open('./ml_intro/data/negative-reviews.json') as f:
    negative_reviews = json.load(f)

print(positive_reviews[:2])
print(negative_reviews[:2])
```

```
['\nSphere by Michael Crichton is an excellant novel. This was certainly the hardest to put
```

["\nTHis book was horrible. If it was possible to rate it lower than one star i would have.

Preliminari

Da oggi usiamo le GPU. Per attivare la GPU bisogna andare in Runtime -> Cambia tipo di runtime e selezionare GPU.

Introduzione a Keras

Keras è una libreria per reti neurali mantenuta da Google. Oggi la useremo per addestrare una rete per FashionMNIST:

Caricare il dataset

```
import keras
from keras.datasets import fashion_mnist

import numpy as np

(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

# Quando abbiamo fatto laboratorio mi ero dimenticato di dividere per 255
x_train = x_train.astype(np.float32) / 255
y_train = y_train.astype(np.float32) / 255
x_test = x_test.astype(np.float32) / 255
y_test = y_test.astype(np.float32) / 255
```

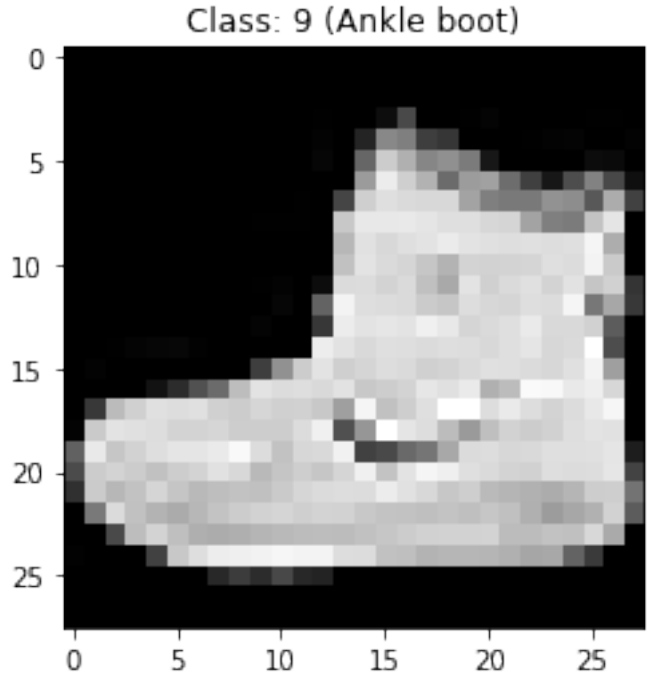
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labe
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-imag
26421880/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-label
5148/5148 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-image
4422102/4422102 [==============================] - 0s 0us/step
```

```
class_names = [
    'T-Shirt/Top',
    'Trouser',
    'Pullover',
    'Dress',
    'Coat',
    'Sandal',
    'Shirt',
    'Sneaker',
    'Bag',
    'Ankle boot'
]
```

```
print(x_train[0].shape)
```

```
(28, 28)
```

```
import matplotlib.pyplot as plt
index = 0
plt.imshow(x_train[index], cmap='gray')
plt.title(f'Class: {y_train[index]} ({class_names[y_train[index]]})')
plt.show()
```




```
!git clone https://github.com/samuelemarro/ml_intro
```

```
Cloning into 'ml_intro'...
remote: Enumerating objects: 247, done.ote: Counting objects: 100% (157/157), done.ote: Comp
```

```
import ml_intro.utils
```

```
ml_intro.utils.plot_images(x_train[:12], titles=y_train[:12], columns=3, space='gray')
```

Le classi sono numeriche, ma lasciarle così è un problema! Non è vero che un sandalo (classe 5) è a metà strada tra una maglietta (classe 0) e uno stivale (classe 9).

La soluzione è eseguire un **One-Hot Encoding**:

```
print('Prima:', y_train[0], '\n', y_train[1])
```

```
num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
# Un metodo alternativo è tf.one_hot, dove tf è la libreria tensorflow

print('Dopo: ', y_train[0], '\n', y_train[1])
```

```
Prima: 9
 0
Dopo:  [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

Creare la rete

```
import keras
from keras import layers
from keras import activations

network = keras.Sequential([
    layers.Flatten(),
    layers.Dense(100, activation=activations.tanh),
    layers.Dense(10, activation=activations.softmax)
])

# Prima di poter usare il modello dobbiamo dire a Keras la dimensione dei nostri input
# "None" vuol dire che il numero è ignoto/può cambiare (perché quante immagini alla volta gl
# può cambiare)
network.build((None, 28, 28))

network.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 100)               78500

 dense_1 (Dense)             (None, 10)                1010

=================================================================
Total params: 79,510
Trainable params: 79,510
Non-trainable params: 0
_________________________________________________________________
```

Addestrare la rete

```
network.compile(
    optimizer=keras.optimizers.SGD(learning_rate=1e-3), # Impostiamo SGD come ottimizzatore
    loss=keras.losses.CategoricalCrossentropy(),
    metrics=['accuracy']
)
```

```
history = network.fit(
    x_train,
    y_train,
    epochs=100, # Addestriamo per 100 epoche
    batch_size=128 # Usiamo una batch size di 128
)
```

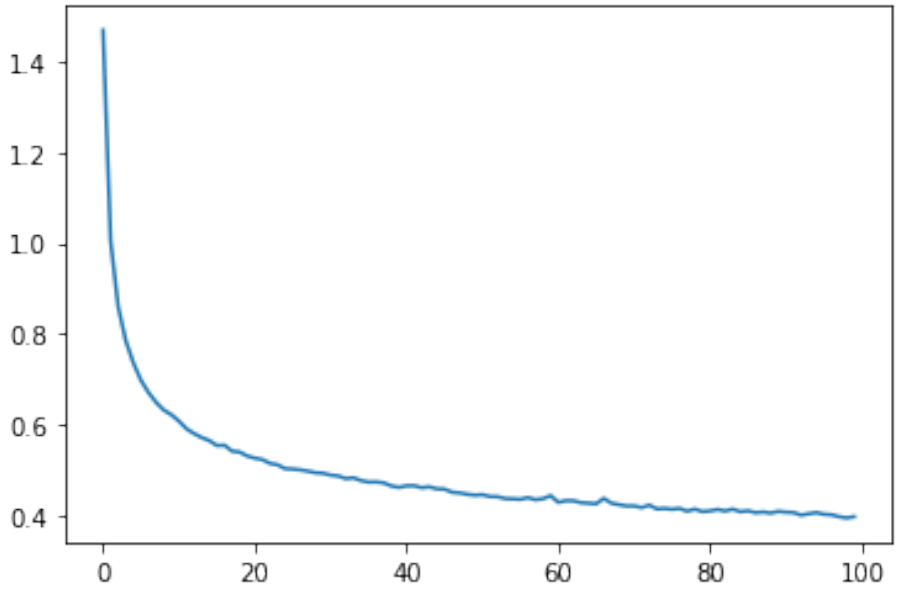
```
Epoch 1/100
375/375 [==============================] - 5s 3ms/step - loss: 1.4701 - accuracy: 0.5095
Epoch 2/100
375/375 [==============================] - 1s 2ms/step - loss: 1.0043 - accuracy: 0.6730
Epoch 3/100
375/375 [==============================] - 1s 2ms/step - loss: 0.8590 - accuracy: 0.7174
Epoch 4/100
375/375 [==============================] - 1s 2ms/step - loss: 0.7841 - accuracy: 0.7394
Epoch 5/100
375/375 [==============================] - 1s 2ms/step - loss: 0.7355 - accuracy: 0.7517
Epoch 6/100
375/375 [==============================] - 1s 2ms/step - loss: 0.6971 - accuracy: 0.7642
Epoch 7/100
375/375 [==============================] - 1s 2ms/step - loss: 0.6700 - accuracy: 0.7734
Epoch 8/100
375/375 [==============================] - 1s 2ms/step - loss: 0.6486 - accuracy: 0.7789
Epoch 9/100
375/375 [==============================] - 1s 2ms/step - loss: 0.6317 - accuracy: 0.7858
Epoch 10/100
375/375 [==============================] - 1s 2ms/step - loss: 0.6213 - accuracy: 0.7869
Epoch 11/100
375/375 [==============================] - 1s 2ms/step - loss: 0.6073 - accuracy: 0.7898
Epoch 12/100
375/375 [==============================] - 1s 2ms/step - loss: 0.5903 - accuracy: 0.7941
Epoch 13/100
375/375 [==============================] - 1s 2ms/step - loss: 0.5797 - accuracy: 0.7985
Epoch 14/100
375/375 [==============================] - 1s 2ms/step - loss: 0.5709 - accuracy: 0.8008
Epoch 15/100
375/375 [==============================] - 1s 2ms/step - loss: 0.5644 - accuracy: 0.8047
Epoch 16/100
375/375 [==============================] - 1s 2ms/step - loss: 0.5534 - accuracy: 0.8059
```

```
Epoch 86/100
375/375 [==============================] - 1s 2ms/step - loss: 0.4093 - accuracy: 0.8541
Epoch 87/100
375/375 [==============================] - 1s 2ms/step - loss: 0.4052 - accuracy: 0.8553
Epoch 88/100
375/375 [==============================] - 1s 2ms/step - loss: 0.4068 - accuracy: 0.8529
Epoch 89/100
375/375 [==============================] - 1s 2ms/step - loss: 0.4044 - accuracy: 0.8545
Epoch 90/100
375/375 [==============================] - 1s 3ms/step - loss: 0.4081 - accuracy: 0.8534
Epoch 91/100
375/375 [==============================] - 2s 6ms/step - loss: 0.4063 - accuracy: 0.8551
Epoch 92/100
375/375 [==============================] - 1s 2ms/step - loss: 0.4056 - accuracy: 0.8551
Epoch 93/100
375/375 [==============================] - 1s 2ms/step - loss: 0.3999 - accuracy: 0.8564
Epoch 94/100
375/375 [==============================] - 1s 2ms/step - loss: 0.4028 - accuracy: 0.8565
Epoch 95/100
375/375 [==============================] - 1s 2ms/step - loss: 0.4055 - accuracy: 0.8550
Epoch 96/100
375/375 [==============================] - 1s 2ms/step - loss: 0.4017 - accuracy: 0.8561
Epoch 97/100
375/375 [==============================] - 1s 2ms/step - loss: 0.4007 - accuracy: 0.8583
Epoch 98/100
375/375 [==============================] - 1s 2ms/step - loss: 0.3964 - accuracy: 0.8587
Epoch 99/100
375/375 [==============================] - 1s 2ms/step - loss: 0.3937 - accuracy: 0.8600
Epoch 100/100
375/375 [==============================] - 1s 2ms/step - loss: 0.3963 - accuracy: 0.8588
```

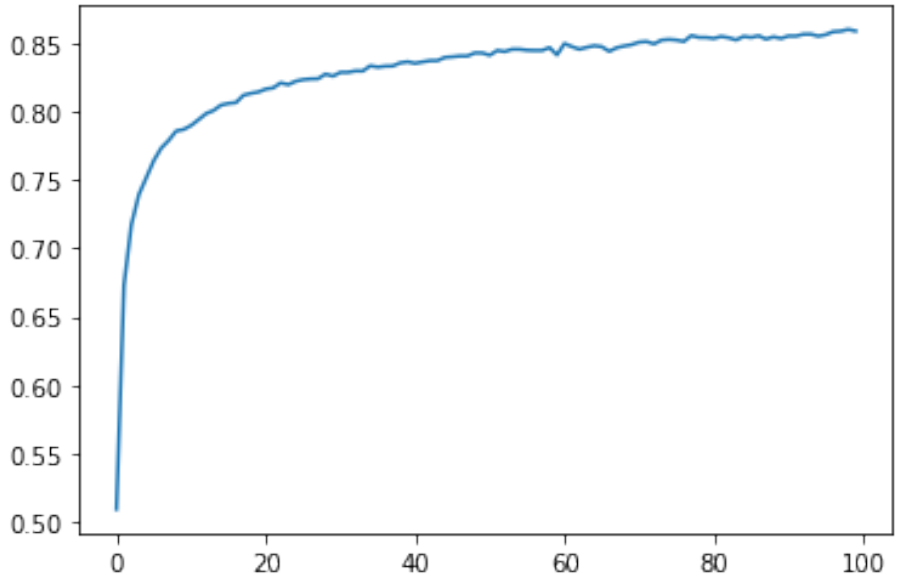
```
print(history.epoch)
print(history.history)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 2
{'loss': [1.470141887664795, 1.0042699575424194, 0.8589769005775452, 0.7841384410858154, 0.7
```

```
plt.plot(history.epoch, history.history['loss'])
plt.show()
```




```
plt.plot(history.epoch, history.history['accuracy'])
plt.show()
```



Nota: è possibile anche avere i grafici in tempo reale usando tensorboard.

Calcolare l'accuratezza di una rete

```
loss, accuracy = network.evaluate(x_test, y_test)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.4561 - accuracy: 0.8391
```

```
print('Loss:', loss)
print(f'Accuracy: {accuracy * 100}%')
```

```
Loss: 0.4561206102371216
Accuracy: 83.91000032424927%
```

Salvare e caricare modelli

Ci sono tre parti di un modello:

- I layer usati (che costituiscono l'**architettura**)
- I parametri che la rete ha appreso (i **pesi**)
- Le informazioni sull'addestramento in corso

Salvare tutto

```
# Suggerimento: se non vedete una nuova cartella tra i file, fate tasto destro sulle cartel
network.save('trained_models/fashion_mnist_model')
```

```
loaded_network = keras.models.load_model('trained_models/fashion_mnist_model')
```

```
loaded_network.evaluate(x_test, y_test)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.4561 - accuracy: 0.8391
```

```
[0.4561206102371216, 0.8391000032424927]
```

Salvare l'architettura

```
model_config = network.to_json()
```

```
print(model_config)
```

```
{"class_name": "Sequential", "config": {"name": "sequential", "layers": [{"class_name": "Inp
```

Il formato del config rende facile salvare come JSON:

```
import json

with open('trained_models/fashion_mnist_config.json', 'w') as f:
    json.dump(model_config, f)

with open('trained_models/fashion_mnist_config.json', 'r') as f:
    loaded_config = json.load(f)

untrained_network = keras.models.model_from_json(loaded_config)
```

```
# La rete non è addestrata! Dobbiamo perfino dire che loss e quali metriche vogliamo
untrained_network.compile(loss=keras.losses.CategoricalCrossentropy(), metrics=['accuracy'])
untrained_network.evaluate(x_test, y_test)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 3.6489 - accuracy: 0.1216
```

```
[3.6489155292510986, 0.12160000205039978]
```

Salvare i pesi

```
network.save_weights('trained_models/fashion_mnist_weights/weights')
```

```
untrained_network.load_weights('trained_models/fashion_mnist_weights/weights')
```

```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f49402733d0>
```

```
untrained_network.compile(loss=keras.losses.CategoricalCrossentropy(), metrics=['accuracy'])
untrained_network.evaluate(x_test, y_test)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.4561 - accuracy: 0.8391
```

```
[0.4561206102371216, 0.8391000032424927]
```

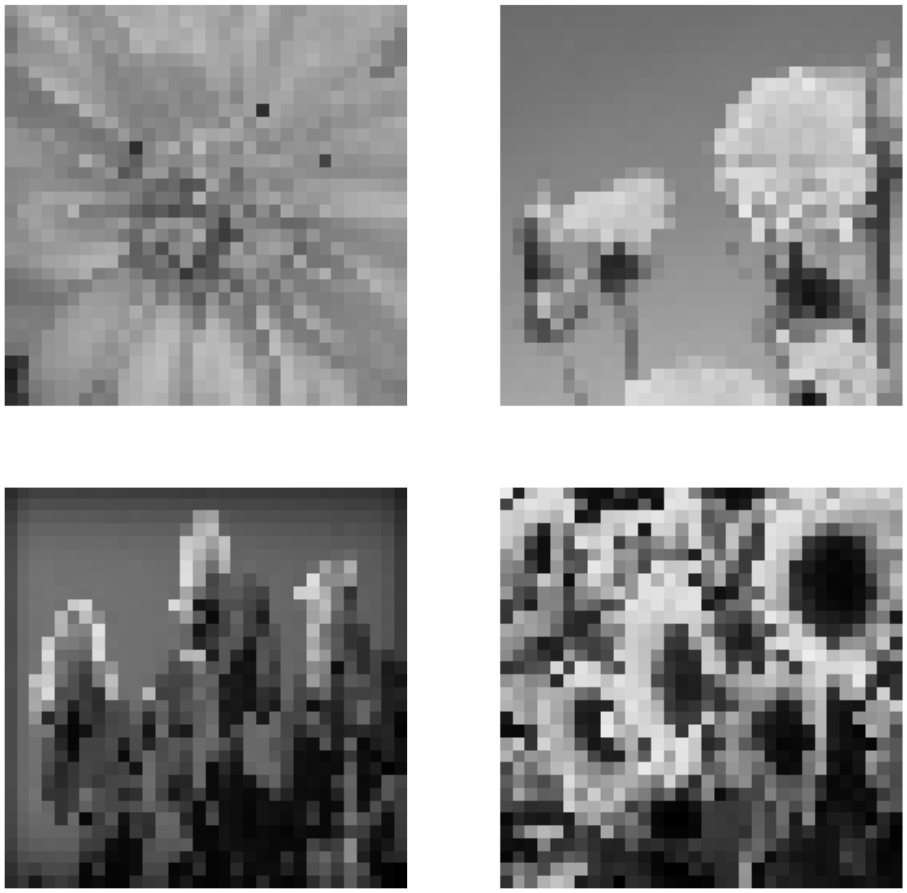
Esercizio: Fiori

Questo è un dataset di 4317 fiori appartenenti a cinque categorie:

- chamomile (camomilla)
- tulip (tulipano)
- rose (rosa)
- sunflower (girasole)
- dandelion (dente di leone)

```
import ml_intro.data

x_flowers, y_flowers = ml_intro.data.flowers_dataset()
print(x_flowers.shape)

ml_intro.utils.plot_images([x_flowers[0], x_flowers[1000], x_flowers[2005], x_flowers[3000]])
```

```
(4317, 32, 32)
```



Addestrate una rete neurale qualunque su questo dataset e salvatela.

```
from sklearn.model_selection import train_test_split
x_flowers_train, x_flowers_test, y_flowers_train, y_flowers_test = train_test_split(x_flowe
```

Overfitting

Sempre sul dataset dei fiori, proviamo ad addestrare una rete molto grande e per molto tempo:

```
big_network = keras.Sequential([
    layers.Flatten(),
    layers.Dense(100, activation=keras.activations.relu),
    layers.Dense(100, activation=keras.activations.relu),
    layers.Dense(100, activation=keras.activations.relu),
    layers.Dense(5, activation=keras.activations.softmax)
])
```

```
big_network.build((None, 32, 32))
big_network.summary()
```

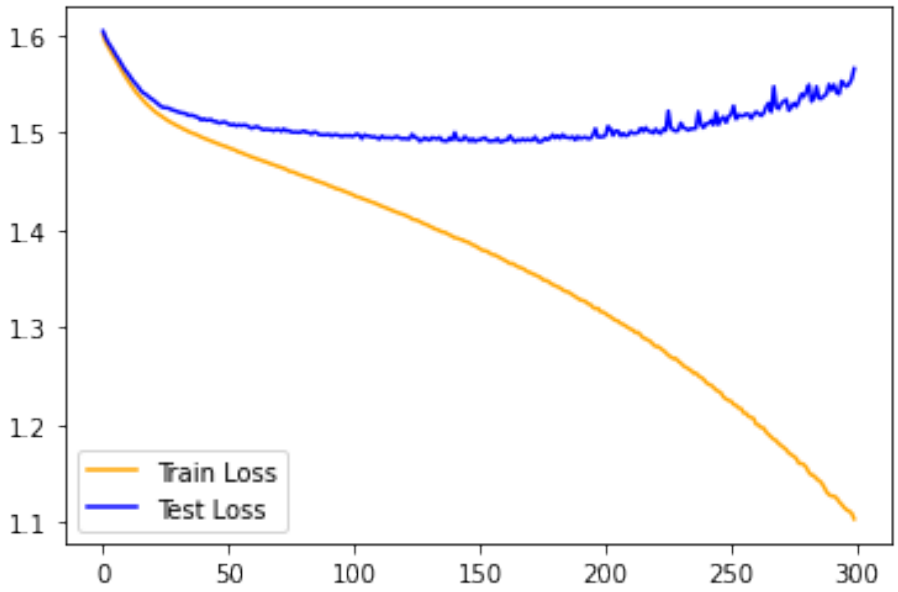
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_1 (Flatten)         (None, 1024)              0

 dense_2 (Dense)             (None, 100)               102500

 dense_3 (Dense)             (None, 100)               10100

 dense_4 (Dense)             (None, 100)               10100

 dense_5 (Dense)             (None, 5)                 505

=================================================================
Total params: 123,205
Trainable params: 123,205
Non-trainable params: 0
_________________________________________________________________
```

```
big_network.compile(
    optimizer=keras.optimizers.SGD(learning_rate=1e-3),
    loss=keras.losses.CategoricalCrossentropy(),
    metrics=['accuracy'],
)
```

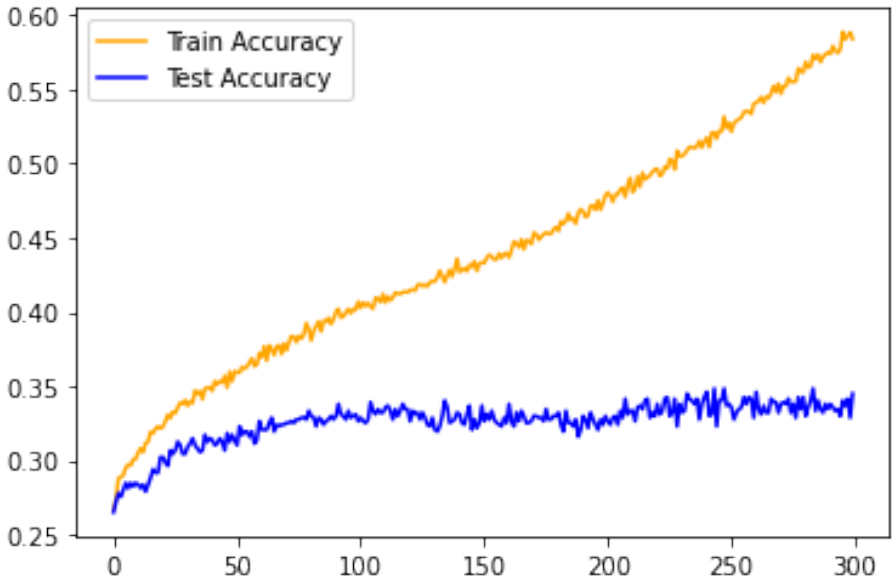
```
import tensorflow as tf

labels_flowers_train = keras.utils.to_categorical(y_flowers_train, 5)
labels_flowers_test = keras.utils.to_categorical(y_flowers_test, 5)

flowers_test_dataset = tf.data.Dataset.from_tensor_slices((x_flowers_test, labels_flowers_te

big_network_history = big_network.fit(
    x=x_flowers_train,
    y=labels_flowers_train,
    validation_data=flowers_test_dataset.batch(64),
    epochs=300
)
```

```
Epoch 1/300
108/108 [==============================] - 1s 4ms/step - loss: 1.6008 - accuracy: 0.2667 - v
Epoch 2/300
108/108 [==============================] - 0s 3ms/step - loss: 1.5928 - accuracy: 0.2734 - v
Epoch 3/300
```

```
plt.legend(['Train Loss', 'Test Loss'])
plt.show()
```




```
plt.plot(big_network_history.epoch, big_network_history.history['accuracy'], color='orange')
plt.plot(big_network_history.epoch, big_network_history.history['val_accuracy'], color='blue
plt.legend(['Train Accuracy', 'Test Accuracy'])
plt.show()
```



Dopo un certo numero di iterazioni l'accuratezza sul dataset di addestramento continua a salire, ma quella sul dataset di test si stabilizza e addirittura inizia a scendere. Si tratta di **overfitting**. L'overfitting avviene quando la rete, nel tentativo di massimizzare la propria accuratezza, inizia a imparare cose inutili ma che la aiutano a ottenere un'accuratezza maggiore.

Per esempio:

- Per coincidenza la maggior parte delle foto di gatti in un training set contiene gatti neri
- La rete decide che tutti gli animali neri sono gatti
- Quando la rete viene testata su un dataset diverso, vede un cane nero e decide che è un gatto

Ci sono strategie per evitare overfitting:

- Usare un modello più piccolo
- Ottenere più dati
- Usare early stopping
- Usare forme di regolarizzazione

Early Stopping

Il principio dell'early stopping è il seguente:

- Dividi il dataset di training in due sottodataset, uno di training "vero" e uno di *validazione*
- Addestra solo sul dataset di training e nel frattempo guarda l'accuratezza sul dataset di validazione

- Quando l'accuratezza di validazione inizia a scendere, significa che bisogna fermarsi

Q: Perché non possiamo usare direttamente il dataset di test per fare early stopping?

A: Perché nella vita reale il dataset di test non esiste! Raccogli tutti i dati possibili, fai training/validazione, addestri il miglior modello e lo consegni. Se i clienti chiamano perché il modello non va, significa che la rete non era abbastanza buona. Il dataset di test serve a simulare l'utilizzo nella vita reale del modello

Vantaggi dell'early stopping:

- Non richiede di indovinare quante epochs è il numero giusto di epochs

Svantaggi dell'early stopping:

- Toglie un po' di dati al training
- È solo una misura tampone: se la rete non riesce a ottenere una buona accuratezza sui dati che non ha già visto, l'early stopping non risolve il problema

```
x_flowers_train, x_flowers_val, labels_flowers_train, labels_flowers_val = train_test_split(
```

```
print(x_flowers_train.shape)
print(x_flowers_val.shape)
```

```
(2762, 32, 32)
(691, 32, 32)
```

```
# Tutto uguale alla rete di prima

big_network_2 = keras.Sequential([
    layers.Flatten(),
    layers.Dense(100, activation=keras.activations.relu),
    layers.Dense(100, activation=keras.activations.relu),
    layers.Dense(100, activation=keras.activations.relu),
    layers.Dense(5, activation=keras.activations.softmax)
])
big_network_2.build((None, 32, 32))
big_network_2.compile(
    optimizer=keras.optimizers.SGD(learning_rate=1e-3),
    loss=keras.losses.CategoricalCrossentropy(),
    metrics=['accuracy'],
)
```

```
flowers_val_dataset = tf.data.Dataset.from_tensor_slices((x_flowers_val, labels_flowers_val)
# monitor è cosa viene guardato (in questo caso la validation loss), patience è dopo
# quante epoch senza miglioramento l'early stopping si arrende
early_stopping = keras.callbacks.EarlyStopping(monitor='val_loss', patience=3)
```

```
big_network_2_history = big_network_2.fit(
    x=x_flowers_train,
    y=labels_flowers_train,
    validation_data=flowers_val_dataset.batch(64),
    epochs=300,
    callbacks=[early_stopping]
)
```

```
Epoch 1/300
87/87 [==============================] - 1s 5ms/step - loss: 1.6176 - accuracy: 0.2484 - val
Epoch 2/300
87/87 [==============================] - 0s 3ms/step - loss: 1.6040 - accuracy: 0.2614 - val
Epoch 3/300
87/87 [==============================] - 0s 3ms/step - loss: 1.5984 - accuracy: 0.2723 - val
Epoch 4/300
87/87 [==============================] - 0s 3ms/step - loss: 1.5943 - accuracy: 0.2744 - val
Epoch 5/300
87/87 [==============================] - 0s 3ms/step - loss: 1.5906 - accuracy: 0.2828 - val
Epoch 6/300
87/87 [==============================] - 0s 3ms/step - loss: 1.5872 - accuracy: 0.2900 - val
Epoch 7/300
87/87 [==============================] - 0s 3ms/step - loss: 1.5840 - accuracy: 0.2896 - val
Epoch 8/300
87/87 [==============================] - 0s 3ms/step - loss: 1.5810 - accuracy: 0.2980 - val
Epoch 9/300
87/87 [==============================] - 0s 3ms/step - loss: 1.5783 - accuracy: 0.3030 - val
Epoch 10/300
87/87 [==============================] - 0s 3ms/step - loss: 1.5757 - accuracy: 0.3001 - val
Epoch 11/300
87/87 [==============================] - 0s 3ms/step - loss: 1.5731 - accuracy: 0.3012 - val
Epoch 12/300
87/87 [==============================] - 0s 3ms/step - loss: 1.5708 - accuracy: 0.3045 - val
Epoch 13/300
87/87 [==============================] - 0s 3ms/step - loss: 1.5682 - accuracy: 0.3038 - val
Epoch 14/300
87/87 [==============================] - 0s 3ms/step - loss: 1.5658 - accuracy: 0.3081 - val
Epoch 15/300
87/87 [==============================] - 0s 3ms/step - loss: 1.5633 - accuracy: 0.3096 - val
Epoch 16/300
87/87 [==============================] - 0s 3ms/step - loss: 1.5610 - accuracy: 0.3106 - val
Epoch 17/300
87/87 [==============================] - 0s 3ms/step - loss: 1.5584 - accuracy: 0.3085 - val
Epoch 18/300
87/87 [==============================] - 0s 3ms/step - loss: 1.5560 - accuracy: 0.3085 - val
Epoch 19/300
```

La data augmentation consiste nell'applicare trasformazioni casuali ai dati in modo da ottenere dei dati extra:

Vantaggi della data augmentation:

- Permette di ottenere spesso un'accuratezza molto più alta
- Insegna alla rete che alcune cose "non importano": se nel dataset c'è sia l'immagine di un leone che la stessa immagine girata di 90°, la rete impara che la rotazione non importa per capire se un'immagine è un leone

Svantaggi della data augmentation:

- Se le augmentations non sono realistiche o corrette, la rete può imparare cose sbagliate (es. l'immagine di una M, girata di 180°, non è più l'immagine di una M, ma di una W)

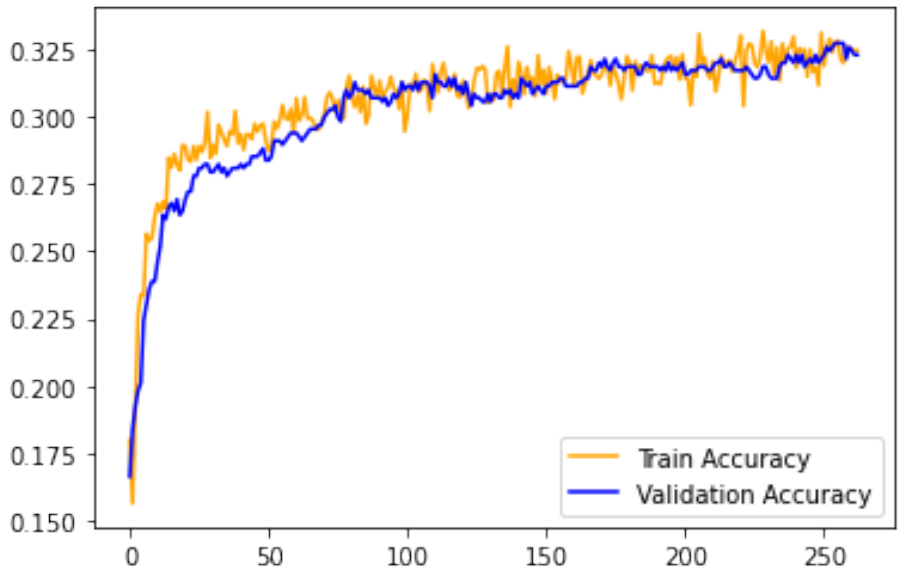
```
# Per prima cosa, trasformiamo i dati in un dataset di Tensorflow
flowers_train_dataset = tf.data.Dataset.from_tensor_slices((x_flowers_train, labels_flowers_

# Poi definiaimo le nostre data augmentation
# Attenzione all'ordine in cui le definiamo, è importante perché vengono fatte
# in successione!
data_augmentation = keras.Sequential([
    layers.Reshape((32, 32, 1), input_shape=(32, 32)), # La data augmentation di Keras funz
    layers.RandomFlip('horizontal_and_vertical'),
    layers.RandomRotation(0.2, fill_mode='reflect'), # Ruota casualmente fino al 20%; Refle
    layers.Reshape((32, 32)) # Ritorniamo alla dimensione classica
])

# Notate il .batch(100): la data augmentation di Keras non funziona su dataset non divisi i
flowers_train_dataset_augmented = flowers_train_dataset.batch(100).map(lambda x, y: (data_au

# Sempre tutto uguale

big_network_3 = keras.Sequential([
    layers.Flatten(),
    layers.Dense(100, activation=keras.activations.relu),
    layers.Dense(100, activation=keras.activations.relu),
    layers.Dense(100, activation=keras.activations.relu),
    layers.Dense(5, activation=keras.activations.softmax)
])
big_network_3.build((None, 32, 32))
big_network_3.compile(
    optimizer=keras.optimizers.SGD(learning_rate=1e-3),
    loss=keras.losses.CategoricalCrossentropy(),
    metrics=['accuracy'],
)

big_network_3_history = big_network_3.fit(
```

```
    flowers_train_dataset_augmented, # Notate che non bisogna più passare x= e y= separatame
    validation_data=flowers_val_dataset.batch(64),
    epochs=300,
    callbacks=[early_stopping] # Teniamo comunque l'early stopping
)
```

```
Epoch 1/300
28/28 [==============================] - 2s 50ms/step - loss: 1.6749 - accuracy: 0.1799 - va
Epoch 2/300
28/28 [==============================] - 1s 38ms/step - loss: 1.6450 - accuracy: 0.1564 - va
Epoch 3/300
28/28 [==============================] - 1s 39ms/step - loss: 1.6245 - accuracy: 0.1901 - va
Epoch 4/300
28/28 [==============================] - 1s 38ms/step - loss: 1.6142 - accuracy: 0.2270 - va
Epoch 5/300
28/28 [==============================] - 1s 39ms/step - loss: 1.6082 - accuracy: 0.2339 - va
Epoch 6/300
28/28 [==============================] - 1s 40ms/step - loss: 1.6037 - accuracy: 0.2335 - va
Epoch 7/300
28/28 [==============================] - 1s 39ms/step - loss: 1.5970 - accuracy: 0.2563 - va
Epoch 8/300
28/28 [==============================] - 1s 39ms/step - loss: 1.5985 - accuracy: 0.2538 - va
Epoch 9/300
28/28 [==============================] - 1s 44ms/step - loss: 1.5951 - accuracy: 0.2549 - va
Epoch 10/300
28/28 [==============================] - 2s 72ms/step - loss: 1.5917 - accuracy: 0.2629 - va
Epoch 11/300
28/28 [==============================] - 1s 40ms/step - loss: 1.5901 - accuracy: 0.2676 - va
Epoch 12/300
28/28 [==============================] - 1s 40ms/step - loss: 1.5899 - accuracy: 0.2647 - va
Epoch 13/300
28/28 [==============================] - 1s 41ms/step - loss: 1.5887 - accuracy: 0.2683 - va
Epoch 14/300
28/28 [==============================] - 1s 40ms/step - loss: 1.5862 - accuracy: 0.2657 - va
Epoch 15/300
28/28 [==============================] - 1s 41ms/step - loss: 1.5825 - accuracy: 0.2846 - va
Epoch 16/300
28/28 [==============================] - 1s 39ms/step - loss: 1.5850 - accuracy: 0.2810 - va
Epoch 17/300
28/28 [==============================] - 1s 39ms/step - loss: 1.5820 - accuracy: 0.2860 - va
Epoch 18/300
28/28 [==============================] - 1s 40ms/step - loss: 1.5797 - accuracy: 0.2820 - va
Epoch 19/300
28/28 [==============================] - 1s 40ms/step - loss: 1.5779 - accuracy: 0.2799 - va
Epoch 20/300
28/28 [==============================] - 1s 39ms/step - loss: 1.5787 - accuracy: 0.2893 - va
```

```
Epoch 251/300
28/28 [==============================] - 1s 41ms/step - loss: 1.5101 - accuracy: 0.3186 - va
Epoch 252/300
28/28 [==============================] - 1s 41ms/step - loss: 1.5114 - accuracy: 0.3255 - va
Epoch 253/300
28/28 [==============================] - 1s 41ms/step - loss: 1.5091 - accuracy: 0.3287 - va
Epoch 254/300
28/28 [==============================] - 1s 40ms/step - loss: 1.5114 - accuracy: 0.3237 - va
Epoch 255/300
28/28 [==============================] - 1s 40ms/step - loss: 1.5125 - accuracy: 0.3277 - va
Epoch 256/300
28/28 [==============================] - 1s 40ms/step - loss: 1.5109 - accuracy: 0.3280 - va
Epoch 257/300
28/28 [==============================] - 1s 39ms/step - loss: 1.5100 - accuracy: 0.3211 - va
Epoch 258/300
28/28 [==============================] - 1s 40ms/step - loss: 1.5125 - accuracy: 0.3197 - va
Epoch 259/300
28/28 [==============================] - 1s 40ms/step - loss: 1.5090 - accuracy: 0.3259 - va
Epoch 260/300
28/28 [==============================] - 1s 40ms/step - loss: 1.5094 - accuracy: 0.3251 - va
Epoch 261/300
28/28 [==============================] - 1s 39ms/step - loss: 1.5068 - accuracy: 0.3251 - va
Epoch 262/300
28/28 [==============================] - 1s 41ms/step - loss: 1.5116 - accuracy: 0.3240 - va
Epoch 263/300
28/28 [==============================] - 1s 42ms/step - loss: 1.5095 - accuracy: 0.3244 - va
```

```
plt.plot(big_network_3_history.epoch, big_network_3_history.history['accuracy'], color='oran
plt.plot(big_network_3_history.epoch, big_network_3_history.history['val_accuracy'], color='
plt.legend(['Train Accuracy', 'Validation Accuracy'])
plt.show()
```



Dropout

Dropout è una tecnica di regolarizzazione dove si spengono casualmente alcuni neuroni della rete. Ciò fa sì che la rete non si "ossessioni" con certe caratteristiche specifiche, ma invece debba guardare le cose da un punto di vista più generale.

Vantaggi del dropout:

- Concettualmente semplice
- Fa quello che deve fare

Svantaggi del dropout:

- Altre tecniche di regolarizzazione (es. L1 e L2, vedi questo link) normalmente funzionano un po' meglio (ma dipende)

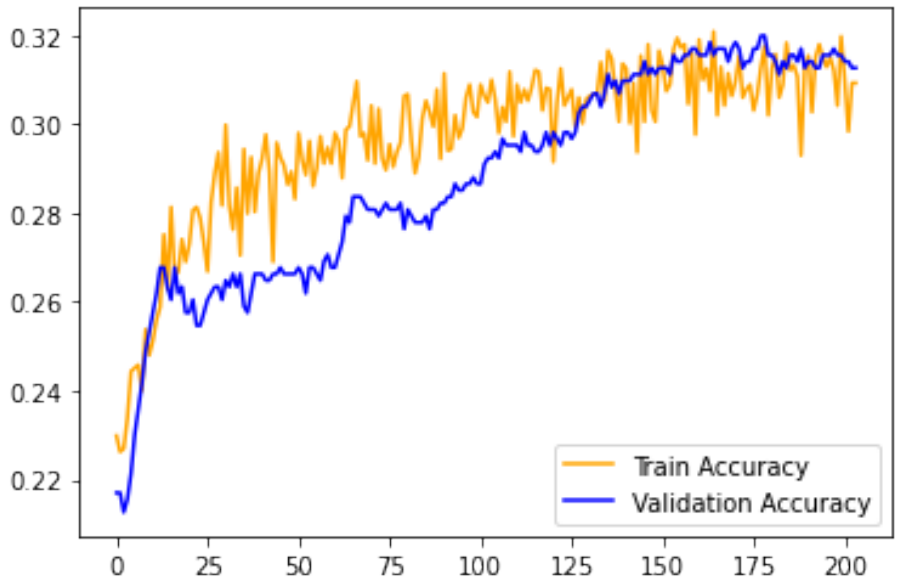
```
# Questa volta è diverso!

big_network_4 = keras.Sequential([
    layers.Flatten(),
    layers.Dense(100, activation=keras.activations.relu),
    layers.Dropout(0.1), # Spegni il 10% dei neuroni a caso
    layers.Dense(100, activation=keras.activations.relu),
    layers.Dense(100, activation=keras.activations.relu),
    layers.Dense(5, activation=keras.activations.softmax)
])
big_network_4.build((None, 32, 32))
```

```
big_network_4.compile(
    optimizer=keras.optimizers.SGD(learning_rate=1e-3),
    loss=keras.losses.CategoricalCrossentropy(),
    metrics=['accuracy'],
)

big_network_4_history = big_network_4.fit(
    flowers_train_dataset_augmented, # Teniamo la data augmentation
    validation_data=flowers_val_dataset.batch(64),
    epochs=300,
    callbacks=[early_stopping] # Teniamo l'early stopping
)
```

```
Epoch 1/300
28/28 [==============================] - 2s 43ms/step - loss: 1.6115 - accuracy: 0.2299 - va
Epoch 2/300
28/28 [==============================] - 1s 37ms/step - loss: 1.6058 - accuracy: 0.2263 - va
Epoch 3/300
28/28 [==============================] - 1s 40ms/step - loss: 1.5990 - accuracy: 0.2270 - va
Epoch 4/300
28/28 [==============================] - 2s 60ms/step - loss: 1.5990 - accuracy: 0.2339 - va
Epoch 5/300
28/28 [==============================] - 1s 38ms/step - loss: 1.5981 - accuracy: 0.2444 - va
Epoch 6/300
28/28 [==============================] - 1s 38ms/step - loss: 1.5934 - accuracy: 0.2451 - va
Epoch 7/300
28/28 [==============================] - 1s 39ms/step - loss: 1.5951 - accuracy: 0.2458 - va
Epoch 8/300
28/28 [==============================] - 1s 39ms/step - loss: 1.5962 - accuracy: 0.2400 - va
Epoch 9/300
28/28 [==============================] - 1s 39ms/step - loss: 1.5935 - accuracy: 0.2538 - va
Epoch 10/300
28/28 [==============================] - 1s 38ms/step - loss: 1.5921 - accuracy: 0.2480 - va
Epoch 11/300
28/28 [==============================] - 1s 39ms/step - loss: 1.5946 - accuracy: 0.2513 - va
Epoch 12/300
28/28 [==============================] - 1s 39ms/step - loss: 1.5909 - accuracy: 0.2560 - va
Epoch 13/300
28/28 [==============================] - 1s 40ms/step - loss: 1.5891 - accuracy: 0.2589 - va
Epoch 14/300
28/28 [==============================] - 1s 40ms/step - loss: 1.5834 - accuracy: 0.2752 - va
Epoch 15/300
28/28 [==============================] - 1s 40ms/step - loss: 1.5866 - accuracy: 0.2629 - va
Epoch 16/300
28/28 [==============================] - 1s 40ms/step - loss: 1.5842 - accuracy: 0.2813 - va
Epoch 17/300
28/28 [==============================] - 1s 39ms/step - loss: 1.5863 - accuracy: 0.2657 - va
```

```
Epoch 202/300
28/28 [==============================] - 1s 39ms/step - loss: 1.5293 - accuracy: 0.2983 - va
Epoch 203/300
28/28 [==============================] - 1s 39ms/step - loss: 1.5298 - accuracy: 0.3092 - va
Epoch 204/300
28/28 [==============================] - 1s 39ms/step - loss: 1.5315 - accuracy: 0.3092 - va
```

```
plt.plot(big_network_4_history.epoch, big_network_4_history.history['accuracy'], color='oran
plt.plot(big_network_4_history.epoch, big_network_4_history.history['val_accuracy'], color='
plt.legend(['Train Accuracy', 'Validation Accuracy'])
plt.show()
```



Importante

La regolarizzazione e l'early stopping non sono dei sostituti per dati di buona qualità e una rete ben pensata. Anche la data augmentation, per quanto estremamente utile, non può fare miracoli da sola. A volte bisogna accettare che il problema è o la rete in sé o i dati scarsi.

Il dataset e la rete che ho scelto per questo laboratorio sono intenzionalmente scarsi: il dataset è troppo piccolo per usarlo da solo e le reti neurali classiche non si usano più da anni per lavorare con le immagini.

Con Asperti vedrete le reti convoluzionali, che hanno accuratezze su immagini molto più alte di qualunque rete classica perché si comportano in maniera simile alla corteccia visiva umana.

Con me vedrete il transfer learning, ovvero come usare dati (e reti) "irrilevanti"

in modo da addestrare una rete come se aveste molti più dati.

Hyperparameter Tuning

Quando si addestra una rete bisogna fare diverse scelte su come deve avvenire l'addestramento:

- Qual è il learning rate dell'ottimizzatore?
- Quale ottimizzatore uso?
- Come configuro l'early stopping?
- Quale batch size uso?

...

L'insieme di tutte queste scelte viene chiamato l'insieme degli iperparametri ("iper" perché i parametri normali sono quelli che impara la rete).

A volte anche la struttura della rete viene considerata in senso lato come un iperparametro.

Il processo di scegliere gli iperparametri viene chiamato *hyperparameter tuning*.

Tecniche di Hyperparameter Tuning

Le due tecniche più comuni di hyperparameter tuning sono:

- *Grid search*: provare tutte le combinazioni

```
for learning_rate in [1e-5, 1e-4, 1e-3, 1e-2...]:
    for batch_size in [32, 64, 128...]:
        for early_stopping_patience in [1, 2, 5...]:
            for ...
                [lancia la rete]
```

- Ricerca manuale (chiamata scherzosamente Grad Student Descent):
- Lancia l'addestramento
- Prova a modificare un po' gli iperparametri andando a ispirazione
- Rilancia l'addestramento

Esistono alcune tecniche migliori di hyperparameter tuning:

- Bayesian hyperparameter tuning
- Algoritmi evolutivi
- Meta-Learning (che è il campo di "imparare a imparare")

però in pratica vengono usate poco.

Consigli per il Grad Student Descent

Quando devo fare hyperparameter tuning, normalmente io guardo i seguenti iperparametri (in ordine da più importante a meno importante).

Disclaimer: l'hyperparameter tuning è molto vicino alla "magia nera". La lista è completamente soggettiva.

- L'architettura della rete
- Il tipo di ottimizzatore (SGD, Adam...)
- Il learning rate dell'ottimizzatore
- In certi casi è anche possibile impostare come cambia il learning rate nel corso del tempo (viene chiamata la learning rate schedule)
- Gli iperparametri delle regolarizzazioni
- Gli altri iperparametri dell'ottimizzatore (es. il momentum per SGD)
- Gli iperparametri della data augmentation
- La batch size (anche se molti le danno più importanza)
- Il resto degli iperparametri

Extra: Normalization

Un piccolo trucco per ottenere un po' di accuratezza extra consiste nel *normalizzare* gli input (fare in modo che abbiano media 0 e deviazione standard 1). Per farlo si prende l'input e si fa: `input_normalizzato = (input - media) / deviazione standard`.

Keras supporta questo comportamento con il layer `Normalization`:

```
# axis indica su quali dimensioni vengono calcolate media e standard deviation
normalization_layer = layers.Normalization(axis=None)

# Notate che normalizza solo sui dati del training
normalization_layer.adapt(x_flowers_train)

# Il layer può ora essere inserito all'interno di un Sequential
```

IMPORTANTE: La media e la deviazione standard vanno sempre calcolati sul dataset di training. MAI sul dataset di validazione, il dataset di test, oppure il dataset originale. Fare ciò è un ERRORE GRAVE (si chiama data leakage).

Esercizio: Extended MNIST

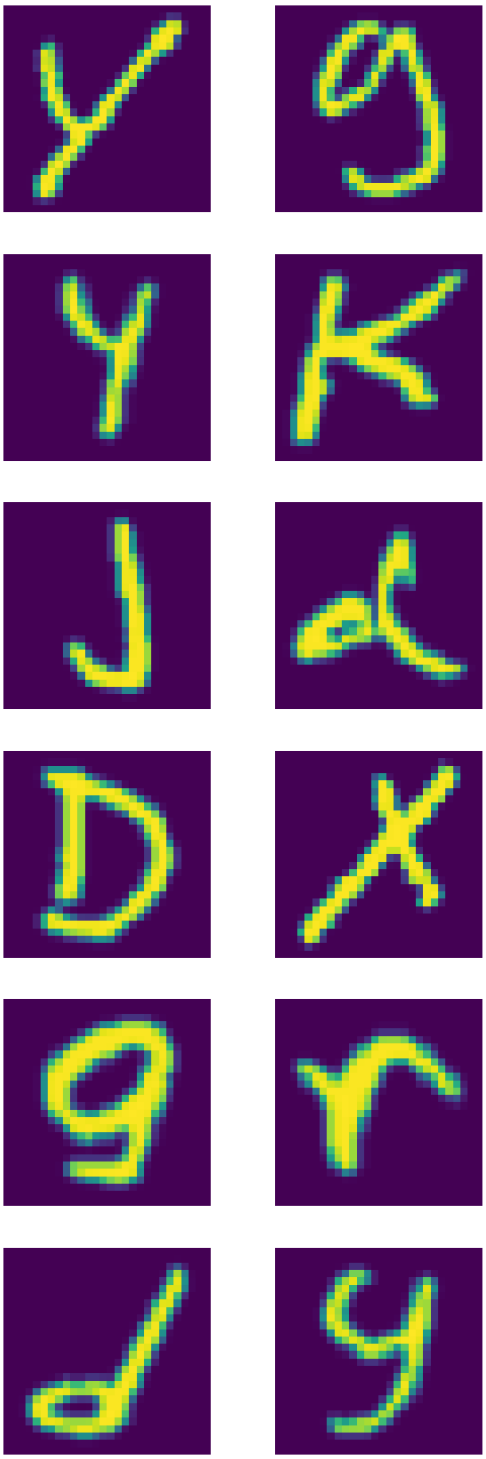
Il dataset Extended MNIST (EMNIST) contiene diversi dataset, tra cui il dataset EMNIST Letters. Contiene circa 800k immagini 28x28 delle 26 lettere dell'alfabeto.

La difficoltà è che il dataset è talmente grande che non è fattibile caricarlo tutto in un colpo. Per fortuna, Tensorflow Datasets si occupa di tutta la parte di caricamento e Keras lo supporta di default

```
import tensorflow_datasets as tfds

letters_dataset = tfds.load('emnist/letters')
```

```
import tensorflow as tf

emnist_train_dataset, emnist_test_dataset = letters_dataset['train'], letters_dataset['test'

# Questo fa sì che il dataset abbia la tipica struttura dove ogni elemento è costituito da
def basic_preparation(element):
    image = tf.reshape(element['image'], (28, 28))
    image = tf.transpose(image)
    label = tf.one_hot(element['label'] - 1, 26)
    print(label)

    return image, label


emnist_train_dataset = emnist_train_dataset.map(basic_preparation)
emnist_test_dataset = emnist_test_dataset.map(basic_preparation)

# Il dataset è così grande che non è possibile trattarlo come un tensore:
# È possibile prendere piccoli sottoinsiemi usando .take() e convertirli in array NumPy
example_letters = list(emnist_train_dataset.take(12))
example_images = [element[0].numpy() for element in example_letters]
ml_intro.utils.plot_images(example_images)
```

```
Tensor("one_hot:0", shape=(26,), dtype=float32)
Tensor("one_hot:0", shape=(26,), dtype=float32)
```



Create e addestrate una rete neurale per classificare le lettere. Qualunque tecnica e qualunque layer (anche quelli non visti) sono ammessi.

Consiglio per chi vuole farsi male: provate a vedere se riuscite a far funzionare il layer Conv2D (serve prima un layer reshape per aggiungere una dimensione extra al fondo)

```
emnist_model = keras.Sequential([layers.Flatten(), layers.Dense(26)])
emnist_model.build((None, 28, 28))
emnist_model.compile(
    optimizer=keras.optimizers.SGD(learning_rate=1e-3),
    loss=keras.losses.CategoricalCrossentropy()
)
# Ricordatevi di suddividere in batch!
emnist_model.fit(emnist_train_dataset.batch(100), epochs=1)
```

```
888/888 [==============================] - 4s 4ms/step - loss: 7.9561
```

```
<keras.callbacks.History at 0x7f48b14ea290>
```

Preliminari

```
!git clone https://github.com/samuelemarro/ml_intro
```

```
Cloning into 'ml_intro'...
remote: Enumerating objects: 267, done.ote: Counting objects: 100% (10/10), done.ote: Compre
```

```
!mkdir papu_dataset
!curl -L -o papu_dataset/papaya_pumpkin_images.npy https://www.dropbox.com/s/p9qsdb34kxopiiy
!curl -L -o papu_dataset/papaya_pumpkin_labels.npy https://www.dropbox.com/s/x0s7xuvjnbe42f8
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   132    0   132    0     0    349      0 --:--:-- --:--:-- --:--:--   349
100   363  100   363    0     0    516      0 --:--:-- --:--:-- --:--:--   516
100  102M  100  102M    0     0  36.7M      0  0:00:02  0:00:02 --:--:-- 56.0M
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   132    0   132    0     0    421      0 --:--:-- --:--:-- --:--:--   421
100   363  100   363    0     0    579      0 --:--:-- --:--:-- --:--:--   579
100  1728  100  1728    0     0   1956      0 --:--:-- --:--:-- --:--:--  1956
```

Qualche Trucco Extra per Progettare una Rete

- ReLU e le sue varianti sono delle ottime funzioni di attivazione per reti molto profonde
- Spesso conviene fare lo "shuffling" del dataset per evitare apprendimenti insoliti (per esempio se nel dataset ci sono tutti gatti, poi tutti cani, poi tutti canarini, la rete inizierà a imparare solo su gatti, poi su cani, poi su canarini e così si ricomincia, invece di imparare tutto insieme)

Transfer Learning

Il transfer learning è un insieme di tecniche per trasferire conoscenze acquisite da un modello a un altro.

L'approccio più comune al transfer learning è:

- Prendi un modello pre-addestrato
- Rimpiazza gli ultimi layer con dei nuovi layer
- Addestra i nuovi layer
- Fai finetuning di tutto il modello

Vantaggi del Transfer Learning:

- Enormemente più veloce del training normale
- Solitamente la rete ottenuta è più accurata rispetto a se fosse stata addestrata normalmente (soprattutto se la rete di partenza è stata addestrata su molti più dati)

- Se si hanno molti pochi dati, il transfer learning è essenzialmente obbligatorio

Svantaggi del Transfer Learning:

- Se la rete di partenza è stata addestrata su un task troppo diverso potrebbe non essere adatta (abbastanza raro)
- La dimensione delle immagini che vuole la rete potrebbe non quadrare con la dimensione delle immagini nel dataset
- Nota: se stai trasferendo solo le convoluzioni, quelle funzionano con qualunque dimensione delle immagini! (nel limite della ragione)
- Se la rete di partenza aveva dei problemi, ora ce li hai anche te
- Sei costretto/a a sottostare alle volontà di chi fornisce la rete

Lavoriamo su un dataset contenente 200 immagini di papaya e 200 immagini di zucche. Sono immagini 150x150.

```
import ml_intro.data
import ml_intro.utils as utils
import numpy as np
# papu = papaya pumpkin
papu_images = ml_intro.data.load_dataset_file('./papu_dataset/papaya_pumpkin_images.npy')
papu_labels = ml_intro.data.load_dataset_file('./papu_dataset/papaya_pumpkin_labels.npy')
```

```
utils.plot_images([papu_images[0].astype(np.uint8), papu_images[-1].astype(np.uint8)], title
```


```
from sklearn.model_selection import train_test_split

papu_images_train, papu_images_test, papu_labels_train, papu_labels_test = train_test_split(
print(len(papu_images_train))
```

```
320
```

```
import tensorflow as tf
```

```
papu_train_dataset = tf.data.Dataset.from_tensor_slices((papu_images_train, papu_labels_trai
papu_test_dataset = tf.data.Dataset.from_tensor_slices((papu_images_test, papu_labels_test))
```

Addestrare Senza Transfer Learning

```
import keras
import keras.layers as layers

standard_model = keras.Sequential([
    layers.Conv2D(8, (2, 2)), # 150x150x8
    layers.ReLU(),
    layers.Conv2D(16, (2, 2)), # 150x150x16
    layers.ReLU(),
    layers.MaxPooling2D((2, 2)), # 75x75x16
    layers.Conv2D(32, (2, 2), padding='same'), # 75x75x32 | padding='same' aggiunge degli ze
    layers.ReLU(),
    layers.MaxPooling2D((2, 2)), # 37x37x32
    layers.Conv2D(32, (2, 2), padding='same'), # 37x37x32
    layers.ReLU(),
    layers.MaxPooling2D((2, 2)), # 17x17x32
    layers.Conv2D(32, (2, 2), padding='same'), #17x17x32
    layers.MaxPooling2D((2, 2)), # 8x8x32
    layers.Flatten(), # 8 * 8 * 32 = 2048
    layers.Dense(100),
    layers.ReLU(),
    layers.Dense(1)
])

standard_model.build((None, 150, 150, 3))
standard_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 149, 149, 8)       104

 re_lu (ReLU)                (None, 149, 149, 8)       0

 conv2d_1 (Conv2D)           (None, 148, 148, 16)      528

 re_lu_1 (ReLU)              (None, 148, 148, 16)      0

 max_pooling2d (MaxPooling2D  (None, 74, 74, 16)       0
 )
```

```
 conv2d_2 (Conv2D)           (None, 74, 74, 32)        2080

 re_lu_2 (ReLU)              (None, 74, 74, 32)        0

 max_pooling2d_1 (MaxPooling  (None, 37, 37, 32)       0
 2D)

 conv2d_3 (Conv2D)           (None, 37, 37, 32)        4128

 re_lu_3 (ReLU)              (None, 37, 37, 32)        0

 max_pooling2d_2 (MaxPooling  (None, 18, 18, 32)       0
 2D)

 conv2d_4 (Conv2D)           (None, 18, 18, 32)        4128

 max_pooling2d_3 (MaxPooling  (None, 9, 9, 32)         0
 2D)

 flatten (Flatten)           (None, 2592)              0

 dense (Dense)               (None, 100)               259300

 re_lu_4 (ReLU)              (None, 100)               0

 dense_1 (Dense)             (None, 1)                 101

=================================================================
Total params: 270,369
Trainable params: 270,369
Non-trainable params: 0
_________________________________________________________________
```

```
standard_model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=1e-3),
    loss=keras.losses.BinaryCrossentropy(), # Notate che qui usiamo l'entropia incrociata b
    metrics=['accuracy']
)

standard_model.fit(
    papu_train_dataset,
    epochs=20
)
```

```
Epoch 1/20
4/4 [==============================] - 10s 125ms/step - loss: 8.1042 - accuracy: 0.4688
```

```
Epoch 2/20
4/4 [==============================] - 0s 52ms/step - loss: 7.9582 - accuracy: 0.4781
Epoch 3/20
4/4 [==============================] - 0s 53ms/step - loss: 7.9582 - accuracy: 0.4781
Epoch 4/20
4/4 [==============================] - 0s 53ms/step - loss: 7.9582 - accuracy: 0.4781
Epoch 5/20
4/4 [==============================] - 0s 54ms/step - loss: 7.9582 - accuracy: 0.4781
Epoch 6/20
4/4 [==============================] - 0s 54ms/step - loss: 7.9582 - accuracy: 0.4781
Epoch 7/20
4/4 [==============================] - 0s 53ms/step - loss: 7.9582 - accuracy: 0.4781
Epoch 8/20
4/4 [==============================] - 0s 53ms/step - loss: 7.9582 - accuracy: 0.4781
Epoch 9/20
4/4 [==============================] - 0s 54ms/step - loss: 7.9582 - accuracy: 0.4781
Epoch 10/20
4/4 [==============================] - 0s 53ms/step - loss: 7.9582 - accuracy: 0.4781
Epoch 11/20
4/4 [==============================] - 0s 53ms/step - loss: 7.9582 - accuracy: 0.4781
Epoch 12/20
4/4 [==============================] - 0s 53ms/step - loss: 7.9582 - accuracy: 0.4781
Epoch 13/20
4/4 [==============================] - 0s 55ms/step - loss: 7.9582 - accuracy: 0.4781
Epoch 14/20
4/4 [==============================] - 0s 54ms/step - loss: 7.9582 - accuracy: 0.4781
Epoch 15/20
4/4 [==============================] - 0s 53ms/step - loss: 7.9582 - accuracy: 0.4781
Epoch 16/20
4/4 [==============================] - 0s 54ms/step - loss: 7.9582 - accuracy: 0.4781
Epoch 17/20
4/4 [==============================] - 0s 53ms/step - loss: 7.9582 - accuracy: 0.4781
Epoch 18/20
4/4 [==============================] - 0s 52ms/step - loss: 7.9582 - accuracy: 0.4781
Epoch 19/20
4/4 [==============================] - 0s 54ms/step - loss: 7.9582 - accuracy: 0.4781
Epoch 20/20
4/4 [==============================] - 0s 54ms/step - loss: 7.9582 - accuracy: 0.4781

<keras.callbacks.History at 0x7fbf3cbdb410>
```

```
loss, accuracy = standard_model.evaluate(papu_test_dataset)
print('Loss:', loss)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

```
1/1 [==============================] - 0s 418ms/step - loss: 6.2903 - accuracy: 0.5875
Loss: 6.290310859680176
```

```
Accuracy: 58.75%
```

Addestrare con Transfer Learning

Al giorno d'oggi il transfer learning è supportato per molte architetture pre-trained. Molti modelli sono divisi in "feature extractor" ed "head", ed è quindi possibile caricarli separatemanete. In particolare, Keras supporta una decina di modelli (+ varianti) addestrati su ImageNet (lista).

```
import keras.applications as applications

base_model = applications.Xception(
    weights='imagenet',  # Carica i pesi per ImageNet
    input_shape=(150, 150, 3),
    include_top=False)  # Non caricare la parte finale

# Non vogliamo addestrare la parte base
# Curiosità: una parte di rete che è stata resa non-addestrabile viene detta "frozen" (conge
base_model.trainable = False

print(base_model.output_shape)
```

```
(None, 5, 5, 2048)
```

Una volta caricato il modello base, possiamo aggiungerci una head nostra:

```
full_model = keras.Sequential([
    base_model, # 5x5x2048 sono un po' tanti per mettere subito un Dense (sarebbero 5 * 5 *
    layers.GlobalAveragePooling2D(), # Modo molto semplice per ridurre la dimensione: fai la
    # layers.Flatten() # Notate che non serve il Flatten (visto che abbiamo già fatto la me
    layers.Dense(1)
])

full_model.build((None, 150, 150, 3))
full_model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 xception (Functional)       (None, 5, 5, 2048)        20861480

 global_average_pooling2d_1   (None, 2048)             0
 (GlobalAveragePooling2D)

 dense_3 (Dense)             (None, 1)                 2049

=================================================================
```

```
Total params: 20,863,529
Trainable params: 2,049
Non-trainable params: 20,861,480
_________________________________________________________________
```

Addestramento dei Layer Nuovi

```
# Il modello base si aspetta un preprocessing specifico (riscala a [-1, 1])
preprocess_function = keras.applications.xception.preprocess_input

papu_train_dataset_preprocessed = papu_train_dataset.map(lambda x, y: (preprocess_function(x
papu_test_dataset_preprocessed = papu_test_dataset.map(lambda x, y: (preprocess_function(x),
```

```
full_model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=1e-3),
    loss=keras.losses.BinaryCrossentropy(), # Notate che qui usiamo l'entropia incrociata b
    metrics=['accuracy']
)

full_model.fit(
    papu_train_dataset_preprocessed,
    epochs=10
)
```

```
Epoch 1/10
4/4 [==============================] - 3s 133ms/step - loss: 3.0026 - accuracy: 0.6062
Epoch 2/10
4/4 [==============================] - 1s 137ms/step - loss: 0.3257 - accuracy: 0.9344
Epoch 3/10
4/4 [==============================] - 1s 137ms/step - loss: 0.0444 - accuracy: 0.9812
Epoch 4/10
4/4 [==============================] - 1s 134ms/step - loss: 0.0292 - accuracy: 0.9812
Epoch 5/10
4/4 [==============================] - 1s 139ms/step - loss: 0.0280 - accuracy: 0.9844
Epoch 6/10
4/4 [==============================] - 1s 138ms/step - loss: 0.0223 - accuracy: 0.9937
Epoch 7/10
4/4 [==============================] - 1s 136ms/step - loss: 0.0164 - accuracy: 0.9937
Epoch 8/10
4/4 [==============================] - 1s 138ms/step - loss: 0.0127 - accuracy: 0.9937
Epoch 9/10
4/4 [==============================] - 1s 135ms/step - loss: 0.0104 - accuracy: 0.9969
Epoch 10/10
4/4 [==============================] - 1s 141ms/step - loss: 0.0089 - accuracy: 1.0000

<keras.callbacks.History at 0x7fbe8a40be50>
```

```
loss, accuracy = full_model.evaluate(papu_test_dataset_preprocessed)
```

```
print('Loss:', loss)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

```
1/1 [==============================] - 1s 951ms/step - loss: 0.0251 - accuracy: 0.9875
Loss: 0.025052014738321304
Accuracy: 98.75%
```

Finetuning

Spesso conviene fare qualche step di addestramento dove si addestra l'intera rete, in modo che le due parti possano imparare a "lavorare insieme".

ATTENZIONE! Rete molto grande + molti pochi dati = rischio enorme di overfitting.

```
# Scongeliamo il modello base
base_model.trainable = True

full_model.compile(
    optimizer=keras.optimizers.SGD(learning_rate=1e-5), # Notate il learning rate di 1e-5 i
    loss=keras.losses.BinaryCrossentropy(),
    metrics=['accuracy']
)

full_model.fit(
    papu_train_dataset_preprocessed,
    epochs=5 # Solo 5 epoche
)
```

```
Epoch 1/5
4/4 [==============================] - 12s 1s/step - loss: 0.6909 - accuracy: 0.8094
Epoch 2/5
4/4 [==============================] - 3s 641ms/step - loss: 0.6523 - accuracy: 0.8094
Epoch 3/5
4/4 [==============================] - 3s 634ms/step - loss: 0.6330 - accuracy: 0.8094
Epoch 4/5
4/4 [==============================] - 3s 636ms/step - loss: 0.6306 - accuracy: 0.8094
Epoch 5/5
4/4 [==============================] - 3s 626ms/step - loss: 0.6285 - accuracy: 0.8094
```

```
<keras.callbacks.History at 0x7fbdd3567450>
```

```
loss, accuracy = full_model.evaluate(papu_test_dataset_preprocessed)
print('Loss:', loss)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

```
1/1 [==============================] - 1s 946ms/step - loss: 0.0341 - accuracy: 0.9750
Loss: 0.03410574048757553
Accuracy: 97.50%
```

Quando il modello non è già diviso in feature extractor ed head, bisogna farlo manualmente. Ciò dipende molto dal

Esercizio: Transfer Learning

Addestrate con transfer learning una rete per il seguente dataset, contenente immagini di fagioli o broccoli.

```
!mkdir bebo_dataset
!curl -L -o bebo_dataset/bean_broccoli_images.npy https://www.dropbox.com/s/5tsg14ahhqqd5az/
!curl -L -o bebo_dataset/bean_broccoli_labels.npy https://www.dropbox.com/s/dtm6d1v3q01hv8b/
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   131    0   131    0     0    299      0 --:--:-- --:--:-- --:--:--   299
100   363  100   363    0     0    410      0 --:--:-- --:--:-- --:--:--     0
100  102M  100  102M    0     0  40.6M      0  0:00:02  0:00:02 --:--:--  104M
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   131    0   131    0     0    421      0 --:--:-- --:--:-- --:--:--   421
100   363  100   363    0     0    592      0 --:--:-- --:--:-- --:--:--   592
100  1728  100  1728    0     0   1786      0 --:--:-- --:--:-- --:--:--     0
```

```
bebo_images = ml_intro.data.load_dataset_file('./bebo_dataset/bean_broccoli_images.npy')
bebo_labels = ml_intro.data.load_dataset_file('./bebo_dataset/bean_broccoli_labels.npy')
```

Autoencoders

Un autoencoder è un'architettura che prende un input e restituisce l'input stesso.

Nella versione classica, la rete è progettata per "comprimere" i dati e decomprimerli. Non richiede labels, ed è per questo un tipo di architettura usata nell'*apprendimento non supervisionato*.

È importante che i dati compressi (chiamati *encoding*) siano più piccoli dell'immagine iniziale; altrimenti, la rete imparerà semplicemente a calcolare l'identità.

Un autoencoder si divide in due parti:

- Encoder
- Decoder

Preparare il Dataset

```
import keras.datasets.mnist as mnist

(mnist_images_train, mnist_labels_train), (mnist_images_test, mnist_labels_test) = mnist.loa
mnist_images_train = mnist_images_train.astype(np.float32) / 255
```

```
mnist_images_test = mnist_images_test.astype(np.float32) / 255

mnist_dataset_train = tf.data.Dataset.from_tensor_slices((mnist_images_train, mnist_images_t
mnist_dataset_test = tf.data.Dataset.from_tensor_slices((mnist_images_test, mnist_images_tes
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

Encoder e Decoder

```
encoder = keras.Sequential([
    layers.Reshape((28, 28, 1)),
    layers.Conv2D(8, (3, 3), padding='same'), # 28x28x8
    layers.ReLU(),
    layers.MaxPool2D((2, 2)),
    layers.Conv2D(8, (3, 3), padding='same'), # 14x14x8
    layers.ReLU(),
    layers.MaxPool2D((2, 2)), # 7x7x8
    layers.Flatten(), # 7 * 7 * 8 = 392
    layers.Dense(50) # 50
])

encoder.build((None, 28, 28))
encoder.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 reshape (Reshape)           (None, 28, 28, 1)         0

 conv2d_13 (Conv2D)          (None, 28, 28, 8)         80

 re_lu_5 (ReLU)              (None, 28, 28, 8)         0

 max_pooling2d_4 (MaxPooling  (None, 14, 14, 8)        0
 2D)

 conv2d_14 (Conv2D)          (None, 14, 14, 8)         584

 re_lu_6 (ReLU)              (None, 14, 14, 8)         0

 max_pooling2d_5 (MaxPooling  (None, 7, 7, 8)          0
 2D)

 flatten_1 (Flatten)         (None, 392)               0
```

```
 dense_4 (Dense)             (None, 50)                19650

=================================================================
Total params: 20,314
Trainable params: 20,314
Non-trainable params: 0
_________________________________________________________________
```

```
# Notate che il decoder che ho fatto è più o meno il contrario dell'encoder (anche se non de
# Il max pooling è un'operazione non invertibile

decoder = keras.Sequential([
    layers.Dense(392), # 392
    layers.ReLU(),
    layers.Reshape((7, 7, 8)), # 7x7x8 | È il contrario di Flatten()
    layers.Conv2DTranspose(8, (3, 3), strides=2, padding='same'), # 14x8x8
    layers.ReLU(),
    layers.Conv2DTranspose(1, (3, 3), strides=2, padding='same'), # 28x28x1
    layers.Reshape((28, 28)) # 28x28
])

decoder.build((None, 50))
decoder.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_5 (Dense)             (None, 392)               19992

 re_lu_7 (ReLU)              (None, 392)               0

 reshape_1 (Reshape)         (None, 7, 7, 8)           0

 conv2d_transpose (Conv2DTra  (None, 14, 14, 8)        584
 nspose)

 re_lu_8 (ReLU)              (None, 14, 14, 8)         0

 conv2d_transpose_1 (Conv2DT  (None, 28, 28, 1)        73
 ranspose)

 reshape_2 (Reshape)         (None, 28, 28)            0

=================================================================
Total params: 20,649
Trainable params: 20,649
```

```
Non-trainable params: 0
_________________________________________________________________
```

```
autoencoder = keras.Sequential([
    encoder,
    decoder
])

autoencoder.build((None, 28, 28))
autoencoder.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential_3 (Sequential)   (None, 50)                20314

 sequential_4 (Sequential)   (None, 28, 28)            20649

=================================================================
Total params: 40,963
Trainable params: 40,963
Non-trainable params: 0
_________________________________________________________________
```

Addestrare un Autoencoder

Che loss si usa per un autoencoder? In altre parole, come si misura se l'autoencoder ha ricostruito bene l'immagine?

Normalmente si calcola il MSE (Mean Squared Error) o il MAE (Mean Absolute Error):

Per semplicità oggi useremo il MAE.

```
autoencoder.compile(
    optimizer=keras.optimizers.Adam(learning_rate=1e-3),
    loss=keras.losses.MeanAbsoluteError(),
    metrics=['mae']
)

autoencoder.fit(
    mnist_dataset_train,
    epochs=20
)
```

```
Epoch 1/20
600/600 [==============================] - 3s 4ms/step - loss: 0.0778 - mae: 0.0778
Epoch 2/20
```

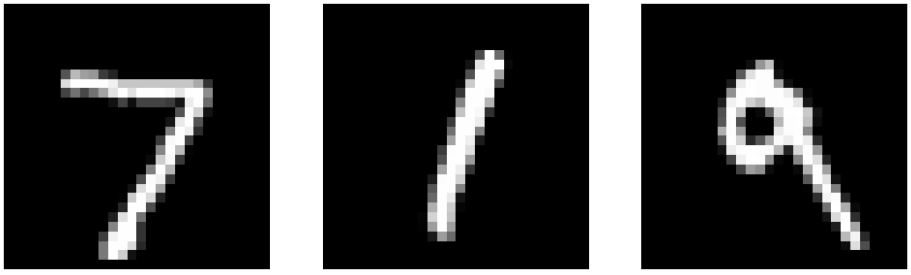
```
600/600 [==============================] - 3s 4ms/step - loss: 0.0454 - mae: 0.0454
Epoch 3/20
600/600 [==============================] - 3s 4ms/step - loss: 0.0389 - mae: 0.0389
Epoch 4/20
600/600 [==============================] - 3s 4ms/step - loss: 0.0354 - mae: 0.0354
Epoch 5/20
600/600 [==============================] - 3s 4ms/step - loss: 0.0330 - mae: 0.0330
Epoch 6/20
600/600 [==============================] - 3s 4ms/step - loss: 0.0311 - mae: 0.0311
Epoch 7/20
600/600 [==============================] - 3s 4ms/step - loss: 0.0300 - mae: 0.0300
Epoch 8/20
600/600 [==============================] - 3s 4ms/step - loss: 0.0293 - mae: 0.0293
Epoch 9/20
600/600 [==============================] - 3s 4ms/step - loss: 0.0287 - mae: 0.0287
Epoch 10/20
600/600 [==============================] - 3s 4ms/step - loss: 0.0282 - mae: 0.0282
Epoch 11/20
600/600 [==============================] - 3s 4ms/step - loss: 0.0279 - mae: 0.0279
Epoch 12/20
600/600 [==============================] - 3s 4ms/step - loss: 0.0275 - mae: 0.0275
Epoch 13/20
600/600 [==============================] - 3s 4ms/step - loss: 0.0273 - mae: 0.0273
Epoch 14/20
600/600 [==============================] - 3s 4ms/step - loss: 0.0271 - mae: 0.0271
Epoch 15/20
600/600 [==============================] - 3s 4ms/step - loss: 0.0268 - mae: 0.0268
Epoch 16/20
600/600 [==============================] - 3s 4ms/step - loss: 0.0266 - mae: 0.0266
Epoch 17/20
600/600 [==============================] - 3s 4ms/step - loss: 0.0263 - mae: 0.0263
Epoch 18/20
600/600 [==============================] - 3s 4ms/step - loss: 0.0261 - mae: 0.0261
Epoch 19/20
600/600 [==============================] - 3s 4ms/step - loss: 0.0259 - mae: 0.0259
Epoch 20/20
600/600 [==============================] - 3s 4ms/step - loss: 0.0257 - mae: 0.0257

<keras.callbacks.History at 0x7fbc4cf5a350>
```

```
loss, _ = autoencoder.evaluate(mnist_dataset_test)
print('Loss:', loss)
```

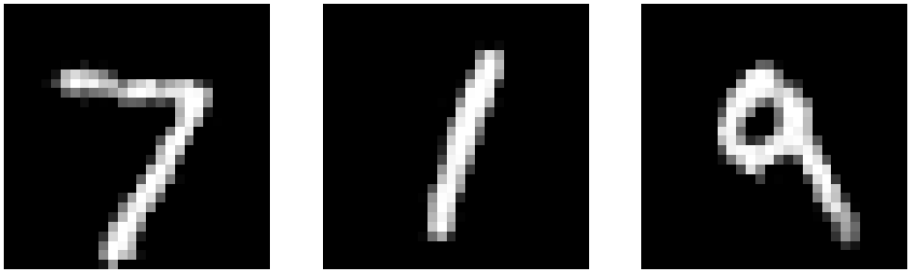
```
WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_test_function.<locals>.
```

```
100/100 [==============================] - 1s 3ms/step - loss: 0.0258 - mae: 0.0258
Loss: 0.025804683566093445
```

```
example_images = np.array([mnist_images_test[0], mnist_images_test[5], mnist_images_test[7]]

utils.plot_images(example_images, space='gray', columns=3)

reconstructed_images = autoencoder.predict(example_images)
utils.plot_images(reconstructed_images, space='gray', columns=3)
```



```
1/1 [==============================] - 0s 196ms/step

WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0
```



Una Nota sui Conv2DTranspose

I layer ConvTranspose (chiamati anche Deconvolution) sono il contrario delle convoluzioni. I parametri passati sono i parametri della Convoluzione originale.

Notate come, con una stride maggiore di 1, una convoluzione riduce la dimensione dell'input:

Di conseguenza, fare la convoluzione al contrario aumenterà la dimensione dell'output.

Un Caso Estremo: Autoencoder con Encoding di Dimensione 2

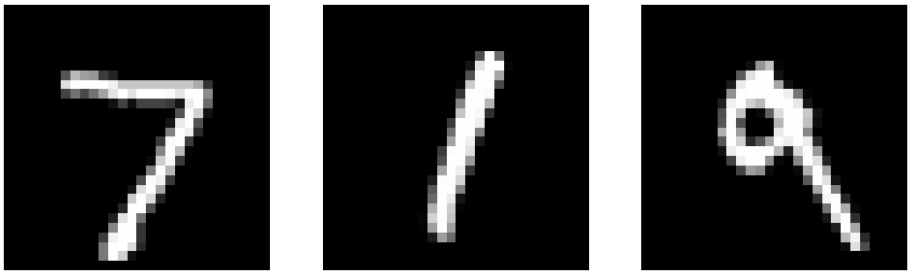
Se facciamo un autoencoder con un encoding di dimensione 2, possiamo visualizzare l'encoding su un grafico.

```
encoder_2 = keras.Sequential([
    layers.Reshape((28, 28, 1)),
    layers.Conv2D(8, (3, 3), padding='same'), # 28x28x8
    layers.ReLU(),
    layers.MaxPool2D((2, 2)),
    layers.Conv2D(8, (3, 3), padding='same'), # 14x14x8
    layers.ReLU(),
    layers.MaxPool2D((2, 2)), # 7x7x8
    layers.Flatten(), # 7 * 7 * 8 = 392
    layers.Dense(2) # 2
])

encoder_2.build((None, 28, 28))

decoder_2 = keras.Sequential([
    layers.Dense(392), # 392
    layers.ReLU(),
    layers.Reshape((7, 7, 8)), # 7x7x8 | È il contrario di Flatten()
    layers.Conv2DTranspose(8, (3, 3), strides=2, padding='same'), # 14x8x8
    layers.ReLU(),
    layers.Conv2DTranspose(1, (3, 3), strides=2, padding='same'), # 28x28x1
    layers.Reshape((28, 28)) # 28x28
])

decoder_2.build((None, 2))

autoencoder_2 = keras.Sequential([
    encoder_2,
    decoder_2
])

autoencoder_2.build((None, 28, 28))

autoencoder_2.compile(
    optimizer=keras.optimizers.Adam(learning_rate=1e-3),
    loss=keras.losses.MeanAbsoluteError(),
    metrics=['mae']
)

autoencoder_2.fit(
    mnist_dataset_train,
    epochs=20
)
```

```
Epoch 1/20
600/600 [==============================] - 3s 4ms/step - loss: 0.1166 - mae: 0.1166
Epoch 2/20
```

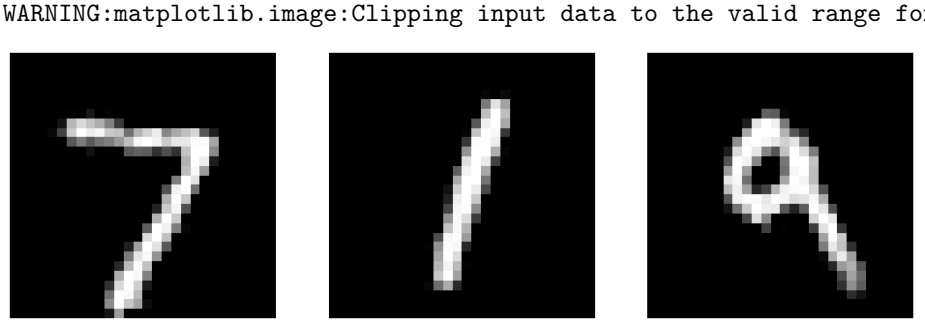
```
600/600 [==============================] - 3s 4ms/step - loss: 0.1062 - mae: 0.1062
Epoch 3/20
600/600 [==============================] - 2s 4ms/step - loss: 0.1042 - mae: 0.1042
Epoch 4/20
600/600 [==============================] - 3s 4ms/step - loss: 0.1029 - mae: 0.1029
Epoch 5/20
600/600 [==============================] - 3s 4ms/step - loss: 0.1019 - mae: 0.1019
Epoch 6/20
600/600 [==============================] - 3s 4ms/step - loss: 0.1011 - mae: 0.1011
Epoch 7/20
600/600 [==============================] - 3s 4ms/step - loss: 0.1006 - mae: 0.1006
Epoch 8/20
600/600 [==============================] - 3s 4ms/step - loss: 0.1002 - mae: 0.1002
Epoch 9/20
600/600 [==============================] - 3s 4ms/step - loss: 0.0998 - mae: 0.0998
Epoch 10/20
600/600 [==============================] - 3s 4ms/step - loss: 0.0995 - mae: 0.0995
Epoch 11/20
600/600 [==============================] - 3s 4ms/step - loss: 0.0992 - mae: 0.0992
Epoch 12/20
600/600 [==============================] - 3s 4ms/step - loss: 0.0990 - mae: 0.0990
Epoch 13/20
600/600 [==============================] - 3s 4ms/step - loss: 0.0987 - mae: 0.0987
Epoch 14/20
600/600 [==============================] - 3s 4ms/step - loss: 0.0986 - mae: 0.0986
Epoch 15/20
600/600 [==============================] - 3s 4ms/step - loss: 0.0984 - mae: 0.0984
Epoch 16/20
600/600 [==============================] - 3s 4ms/step - loss: 0.0982 - mae: 0.0982
Epoch 17/20
600/600 [==============================] - 3s 4ms/step - loss: 0.0981 - mae: 0.0981
Epoch 18/20
600/600 [==============================] - 3s 4ms/step - loss: 0.0979 - mae: 0.0979
Epoch 19/20
600/600 [==============================] - 2s 4ms/step - loss: 0.0978 - mae: 0.0978
Epoch 20/20
600/600 [==============================] - 3s 4ms/step - loss: 0.0977 - mae: 0.0977

<keras.callbacks.History at 0x7fbc4ed4d490>
```

```
example_images = np.array([mnist_images_test[0], mnist_images_test[5], mnist_images_test[7]]

utils.plot_images(example_images, space='gray', columns=3)

reconstructed_images = autoencoder.predict(example_images)
utils.plot_images(reconstructed_images, space='gray', columns=3)
```



```
1/1 [==============================] - 0s 15ms/step
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0
```

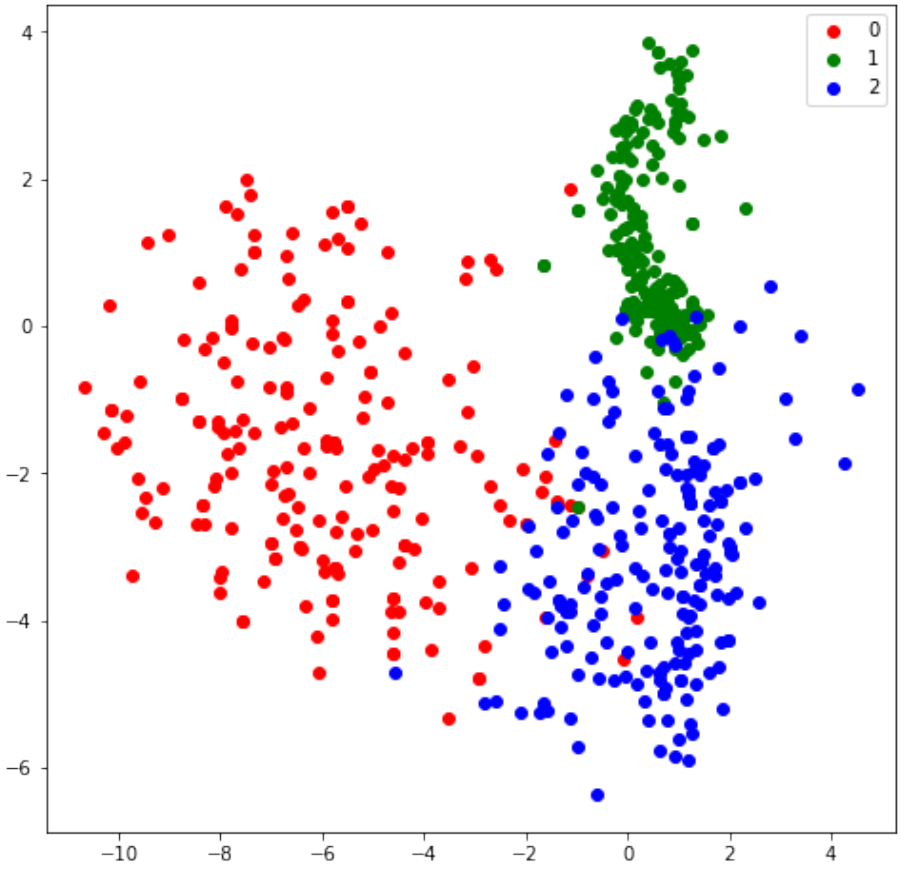


Con 2 dimensioni è facile vedere che gli encoding sono abbastanza *disentangled* (immagini concettualmente simili hanno encoding simili)

```
def random_images_with_label(images, labels, target_label, count):
    matching_indices = np.nonzero(np.equal(labels, target_label))[0]
    chosen_indices = np.random.choice(matching_indices, count)

    return images[chosen_indices]

import matplotlib.pyplot as plt


plt.figure(figsize=(8, 8))

labels = [0, 1, 2]
colors = ['red', 'green', 'blue']

for label, color in zip(labels, colors):
    chosen_images = random_images_with_label(mnist_images_test, mnist_labels_test, label, 20
    encodings = encoder_2.predict(chosen_images)
    plt.scatter(encodings[:, 0], encodings[:, 1], color=color)
```

```
plt.legend([str(n) for n in labels])
plt.show()
```

```
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
```



Generazione

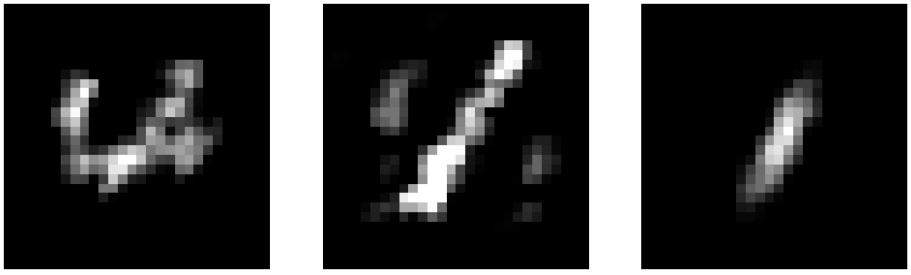
Se è possibile usare l'encoder per convertire un'immagine in un encoding, è anche possibile usare il decoder per convertire un encoding in un'immagine. Questo include encoding che non esistono:

```
# Genera 3 encoding da due valori ciascuno
random_encodings = np.random.rand(3,2) * 6

generated_images = decoder_2.predict(random_encodings)
```

```
utils.plot_images(generated_images, columns=3, space='gray')
```

```
1/1 [==============================] - 0s 14ms/step
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0
```



Questo concetto è alla base dei Variational AutoEncoders (VAEs), che combinano gli autoencoder con principi statistici.

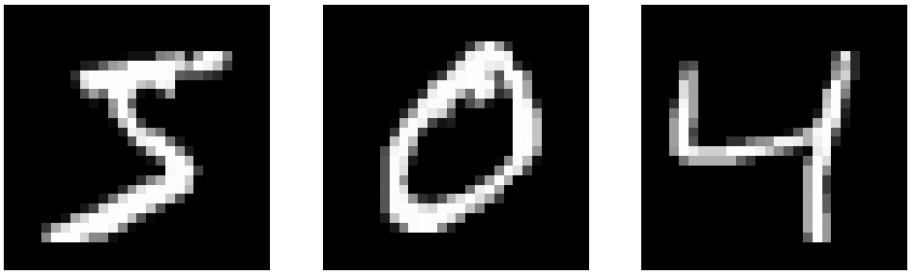
Anomaly Detection

L'Anomaly Detection consiste nell'identificare anomalie senza sapere prima come sono fatte queste anomalie.

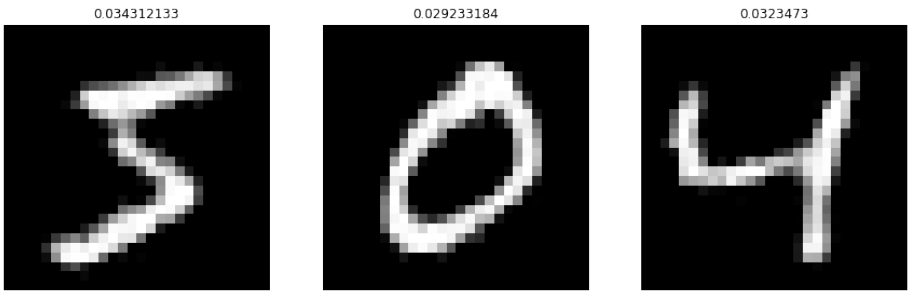
L'approccio più comune all'anomaly detection è di modellare quello che si sa essere normale e definire qualunque cosa non-normale come un'anomalia. Gli autoencoder sono quindi lo strumento ideale: un'immagine "normale" verra ricostruita bene, un'immagine "anomala" verrà ricostruita male.

Immagini normali

```
normal_images = mnist_images_train[:3]
normal_reconstructed = autoencoder(normal_images).numpy()

normal_maes = np.abs(normal_images - normal_reconstructed).mean((1, 2))

utils.plot_images(normal_images, columns=3, space='gray')
utils.plot_images(normal_reconstructed, columns=3, space='gray', titles=[str(mae) for mae in
```



```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0
```



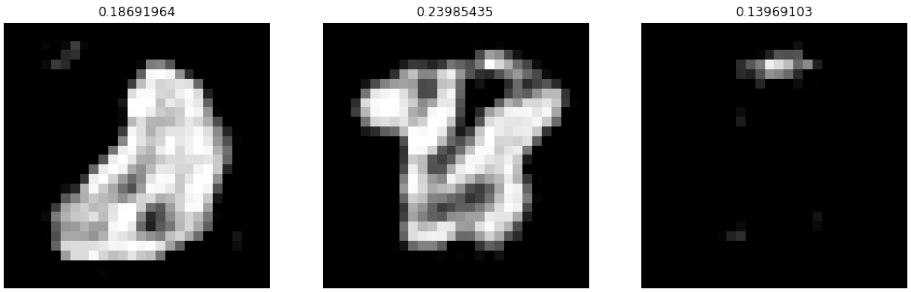
Immagini Anomale

```
from keras.datasets import fashion_mnist
(fashion_mnist_train, _), (_, _) = fashion_mnist.load_data()

fashion_mnist_train = fashion_mnist_train.astype(np.float32) / 255
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labe
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-imag
26421880/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-label
5148/5148 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-image
4422102/4422102 [==============================] - 0s 0us/step
```

```
anomalous_images = fashion_mnist_train[:3]

anomalous_reconstructed = autoencoder(anomalous_images).numpy()
anomalous_maes = np.abs(anomalous_images - anomalous_reconstructed).mean((1, 2))

utils.plot_images(anomalous_images, columns=3, space='gray')
utils.plot_images(anomalous_reconstructed, columns=3, space='gray', titles=[str(mae) for mae
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0
```



Denoising Autoencoders

Il problema degli autoencoder è che a volte si vuole imparare una rappresentazione "utile" (dati simili sono vicini) senza necessariamente voler comprimere i dati (perché si perdono informazioni).

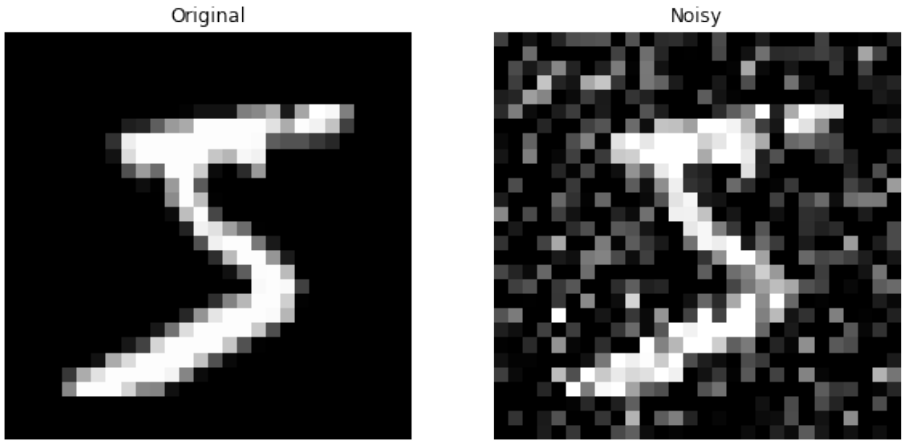
I denoising autoencoders usano un approccio diverso: dato un input corrotto da noise, la rete deve restituire l'input originale. Il fatto che debba togliere il noise obbliga la rete a capire cosa conta come noise e cosa conta come immagine vera.

Un altro vantaggio dei denoising autoencoders è che possono essere usati per togliere il noise da immagini vere (ma il noise deve essere molto simile a quello che la rete ha imparato a riconoscere).

Creare il Noise

```
mnist_noise_factor = 0.3

def add_noise(image):
    # Notate che qui stiamo usando tensorflow, non NumPy (molti metodi sono simili)
    image = image + tf.random.normal(tf.shape(image)) * mnist_noise_factor
    # Clip a [0, 1]
    return tf.clip_by_value(image, 0, 1)
```

```
mnist_dataset_train_noisy = mnist_dataset_train.map(lambda input, output: (add_noise(input),
mnist_dataset_test_noisy = mnist_dataset_test.map(lambda input, output: (add_noise(input), o

# Visualizziamo che noise stiamo applicando
example_image = mnist_images_train[0]
noisy_image = add_noise(example_image).numpy()

utils.plot_images([example_image, noisy_image], titles=['Original', 'Noisy'], space='gray')
```



Creare l'Autoencoder

```
# Personalmente ho osservato che ridurre un po' la dimensione delle immagini
# usando convoluzioni/pooling aiuta comunque anche nei denoising autoencoder
# La mia spiegazione è che ciò permette ai layer successivi di ragionare più
# ad alto livello

denoising_encoder = keras.Sequential([
    layers.Reshape((28, 28, 1)),
    layers.Conv2D(8, (3, 3), padding='same'), # 28x28x8
    layers.ReLU(),
    layers.MaxPool2D((2, 2)),
    layers.Conv2D(8, (3, 3), padding='same'), # 14x14x8
    layers.ReLU(),
    layers.MaxPool2D((2, 2)), # 7x7x8
    layers.Flatten(), # 7 * 7 * 8 = 392
])

# Notate che ora l'encoding ha 392 valori!

denoising_encoder.build((None, 28, 28))
```

```
denoising_decoder = keras.Sequential([
    layers.Reshape((7, 7, 8)), # 7x7x8 | È il contrario di Flatten()
    layers.Conv2DTranspose(8, (3, 3), strides=2, padding='same'), # 14x8x8
    layers.ReLU(),
    layers.Conv2DTranspose(1, (3, 3), strides=2, padding='same'), # 28x28x1
    layers.Reshape((28, 28)) # 28x28
])

denoising_decoder.build((None, 392))

denoising_autoencoder = keras.Sequential([
    denoising_encoder,
    denoising_decoder
])

denoising_autoencoder.build((None, 28, 28))
```

Addestrare l'Autoencoder

```
denoising_autoencoder.compile(
    optimizer=keras.optimizers.Adam(learning_rate=1e-3),
    loss=keras.losses.MeanAbsoluteError(),
    metrics=['mae']
)

denoising_autoencoder.fit(
    mnist_dataset_train,
    epochs=30
)
```

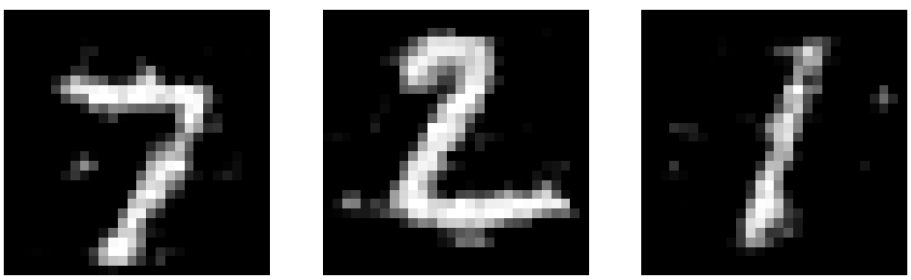
```
Epoch 1/30
600/600 [==============================] - 3s 4ms/step - loss: 0.0648 - mae: 0.0648
Epoch 2/30
600/600 [==============================] - 2s 4ms/step - loss: 0.0407 - mae: 0.0407
Epoch 3/30
600/600 [==============================] - 2s 4ms/step - loss: 0.0371 - mae: 0.0371
Epoch 4/30
600/600 [==============================] - 2s 4ms/step - loss: 0.0348 - mae: 0.0348
Epoch 5/30
600/600 [==============================] - 2s 4ms/step - loss: 0.0333 - mae: 0.0333
Epoch 6/30
600/600 [==============================] - 2s 4ms/step - loss: 0.0322 - mae: 0.0322
Epoch 7/30
600/600 [==============================] - 2s 4ms/step - loss: 0.0313 - mae: 0.0313
Epoch 8/30
```

```
600/600 [==============================] - 2s 4ms/step - loss: 0.0305 - mae: 0.0305
Epoch 9/30
600/600 [==============================] - 2s 4ms/step - loss: 0.0299 - mae: 0.0299
Epoch 10/30
600/600 [==============================] - 2s 4ms/step - loss: 0.0293 - mae: 0.0293
Epoch 11/30
600/600 [==============================] - 2s 4ms/step - loss: 0.0288 - mae: 0.0288
Epoch 12/30
600/600 [==============================] - 2s 4ms/step - loss: 0.0284 - mae: 0.0284
Epoch 13/30
600/600 [==============================] - 2s 4ms/step - loss: 0.0281 - mae: 0.0281
Epoch 14/30
600/600 [==============================] - 2s 4ms/step - loss: 0.0279 - mae: 0.0279
Epoch 15/30
600/600 [==============================] - 2s 4ms/step - loss: 0.0277 - mae: 0.0277
Epoch 16/30
600/600 [==============================] - 2s 4ms/step - loss: 0.0275 - mae: 0.0275
Epoch 17/30
600/600 [==============================] - 2s 4ms/step - loss: 0.0274 - mae: 0.0274
Epoch 18/30
600/600 [==============================] - 2s 4ms/step - loss: 0.0273 - mae: 0.0273
Epoch 19/30
600/600 [==============================] - 2s 4ms/step - loss: 0.0271 - mae: 0.0271
Epoch 20/30
600/600 [==============================] - 2s 4ms/step - loss: 0.0269 - mae: 0.0269
Epoch 21/30
600/600 [==============================] - 2s 4ms/step - loss: 0.0268 - mae: 0.0268
Epoch 22/30
600/600 [==============================] - 2s 4ms/step - loss: 0.0265 - mae: 0.0265
Epoch 23/30
600/600 [==============================] - 2s 4ms/step - loss: 0.0264 - mae: 0.0264
Epoch 24/30
600/600 [==============================] - 2s 4ms/step - loss: 0.0262 - mae: 0.0262
Epoch 25/30
600/600 [==============================] - 2s 4ms/step - loss: 0.0261 - mae: 0.0261
Epoch 26/30
600/600 [==============================] - 2s 4ms/step - loss: 0.0260 - mae: 0.0260
Epoch 27/30
600/600 [==============================] - 2s 4ms/step - loss: 0.0258 - mae: 0.0258
Epoch 28/30
600/600 [==============================] - 2s 4ms/step - loss: 0.0257 - mae: 0.0257
Epoch 29/30
600/600 [==============================] - 2s 4ms/step - loss: 0.0256 - mae: 0.0256
Epoch 30/30
600/600 [==============================] - 2s 4ms/step - loss: 0.0254 - mae: 0.0254
```

```
<keras.callbacks.History at 0x7fbc4eabbdd0>
```

```
# Prendi una batch e di quella batch tre elementi
example_noisy_images = [image.numpy() for image, _ in list(mnist_dataset_test_noisy.take(1))
utils.plot_images(example_noisy_images, space='gray', columns=3)

denoised_images = denoising_autoencoder.predict(example_noisy_images)
utils.plot_images(denoised_images, space='gray', columns=3)
```

```
1/1 [==============================] - 0s 103ms/step
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0
```

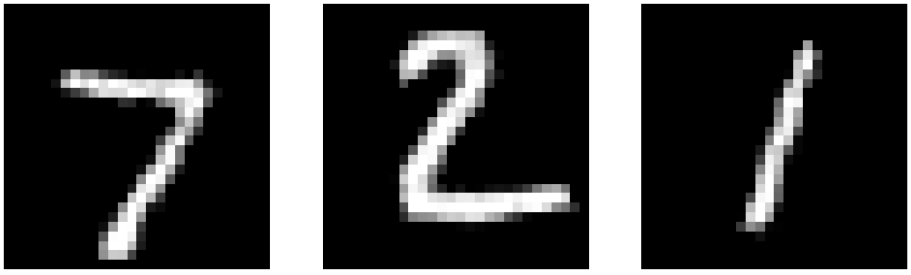


Cosa succede se mostriamo un'immagine normale a un denoising autoencoder?

```
example_noiseless_images = [image.numpy() for _, image in list(mnist_dataset_test_noisy.take
utils.plot_images(example_noiseless_images, space='gray', columns=3)

denoised_images_noiseless = denoising_autoencoder.predict(example_noiseless_images)
utils.plot_images(denoised_images_noiseless, space='gray', columns=3)
```

```
1/1 [==============================] - 0s 20ms/step
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0
```

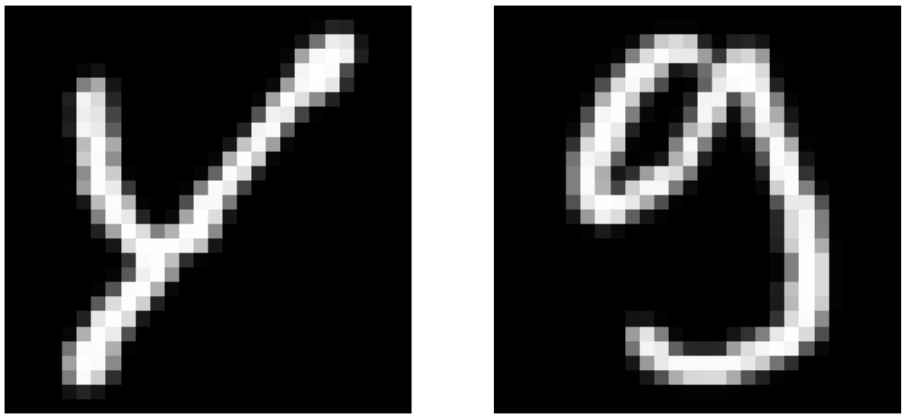


Esercizio: Denoising Autoencoders

Creare un denoising autoencoder per il dataset EMNIST-Letters e calcolare il MAE sul dataset di test.

```
import tensorflow_datasets as tfds

letters_dataset = tfds.load('emnist/letters')

emnist_train_dataset, emnist_test_dataset = letters_dataset['train'], letters_dataset['test'

# Questo fa sì che il dataset abbia la tipica struttura
def basic_preparation(element):
    image = tf.reshape(element['image'], (28, 28))
    image = tf.transpose(image)
    image = tf.cast(image, tf.float32) / 255 # <-----
    return image


emnist_train_dataset = emnist_train_dataset.map(basic_preparation)
emnist_test_dataset = emnist_test_dataset.map(basic_preparation)
```

```
# Il dataset è così grande che non è possibile trattarlo come un tensore:
# È possibile prendere piccoli sottoinsiemi usando .take() e convertirli in array NumPy
example_letters = list(emnist_train_dataset.take(2))
example_images = [image.numpy() for image in example_letters]
ml_intro.utils.plot_images(example_images, space='gray')
```



```
tf.random.set_seed(47)
# Addestramento
...
exercise_autoencoder.compile(
    metrics=['mae'],
    ...
)
...

.fit(train_dataset.batch(100))
...

tf.random.set_seed(99)
# Testing
loss, exercise_mae = exercise_autoencoder.evaluate(emnist_test_dataset_noisy.batch(100))
```

Esercizio Extra: Super-Risoluzione

La super-risoluzione è una tecnica per imparare ad ingrandire un'immagine "inventandosi" i pixel mancanti.

È importante notare che la rete sta facendo essenzialmente un'ipotesi ragionevole sui pixel mancanti; questo significa che non sempre è corretta (soprattutto se viene usata su dati diversi dal dataset di addestramento).

L'esercizio consiste nello scrivere un autoencoder che faccia "super-risoluzione" di immagini EMNIST-Letters da 7x7 a 28x28

```
# Numpy non ha una funzione resize; bisogna usare cv.resize
# oppure skimage.transform.resize

# Tensorflow/Keras hanno una funzione resize, è tf.image.resize
```

```
!git clone https://github.com/samuelemarro/ml_intro
```

```
Cloning into 'ml_intro'...
remote: Enumerating objects: 318, done.ote: Counting objects: 100% (61/61), done.ote: Compre
```

```
import numpy as np
import tensorflow as tf

np.random.seed(1)
```

LSTMs

L'idea di base delle reti LSTM (Long-Short Term Memory) è di avere una struttura per una rete neurale che cattura i meccanismi comuni della memoria:

- Capire quali informazioni sono importanti immediatamente e quali saranno utili nel futuro
- Dimenticare informazioni inutili
- Combinare informazioni vecchie e nuove

Vantaggi delle LSTMs:

- Sono progettate per capire quando devono ricordare e quando dimenticare
- A differenza delle RNNs normali, possono essere addestrate velocemente anche quando lavorano su sequenze lunghe (non hanno gradient vanishing)

Svantaggi delle LSTMs:

- Il loro output è di default tra 0 e 1 (può richiedere un rescale)
- Sono difficili da parallelizzare (dato che leggono gli elementi della sequenza uno dopo l'altro)

Struttura delle LSTMs

Refresh: una rete LSTM prende un input e due stati (hidden e carry). Emette un hidden (che coincide con l'output) e un carry.

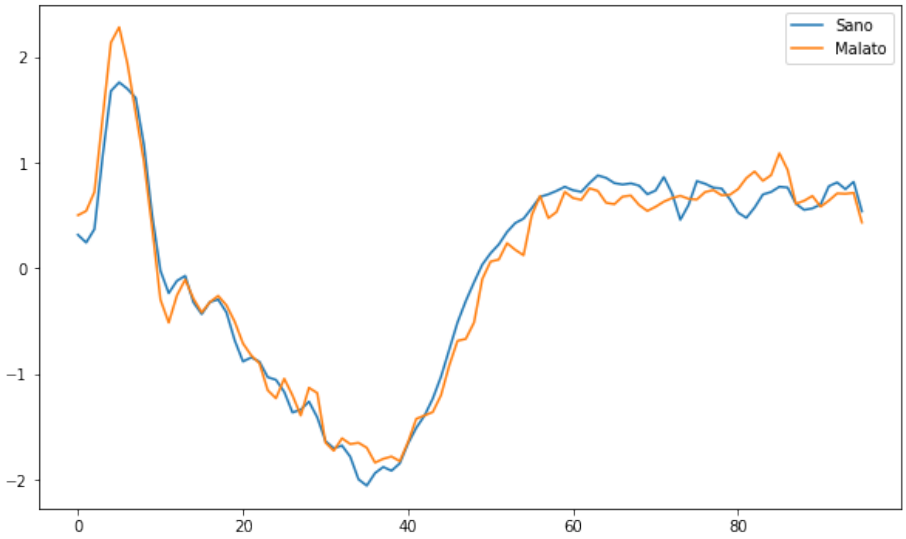
IMPORTANTE: Quando si parla di LSTM, parlare di "hidden" e di "output" è la stessa cosa!

Classificazione con LSTMs

Partiamo da un dataset semplice: dato l'elettrocardiogramma di un individuo, dobbiamo capire se è sano (classe 1) o ha un problema di cuore (classe 0).

```
import ml_intro.data as data
import matplotlib.pyplot as plt

ecg_values, ecg_labels = data.ecg_dataset()
```

```
plt.figure(figsize=(10, 6))
plt.plot(ecg_values[2])
plt.plot(ecg_values[0])
plt.legend(['Sano', 'Malato'])
plt.show()
```




```
# Facciamo il solito split train-test
from sklearn.model_selection import train_test_split

ecg_values_train, ecg_values_test, ecg_labels_train, ecg_labels_test = train_test_split(ecg_

print(ecg_values_train.shape, ecg_labels_train.shape)
```

```
(80, 96) (80,)
```

Costruiamo una rete che contiene un layer LSTM:

```
import keras

# 20 è il numero di neuroni nell'hidden layer
# La prima dimensione è la batch size (che è None perché dipende)
# La seconda dimensione è quanto è lunga la sequenza (che è None perché le sequenze possono
# La terza dimensione è il numero di feature in input (in questo caso 1, perché la misura d
ecg_lstm = keras.layers.LSTM(20, batch_input_shape=(None, None, 1))

# Di default, un'LSTM in Keras restituisce il valore dell'hidden layer quando ha finito
# di leggere la sequenza (quindi è un tensore di dimensione [batch_size, 20])

# Inoltre, un'LSTM restituisce valori in [0, 1]
```

```
ecg_model = keras.Sequential([
    ecg_lstm,
    keras.layers.Dense(1) # L'output è uno, ed è 1 quando il cuore è sano, 0 quando è malat
])
```

```
ecg_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 20)                1760

 dense (Dense)               (None, 1)                 21

=================================================================
Total params: 1,781
Trainable params: 1,781
Non-trainable params: 0
_________________________________________________________________
```

```
ecg_model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)

ecg_model.fit(
    ecg_values_train,
    ecg_labels_train,
    epochs=100
)
```

```
Epoch 1/100
3/3 [==============================] - 5s 71ms/step - loss: 4.7147 - accuracy: 0.3250
Epoch 2/100
3/3 [==============================] - 0s 66ms/step - loss: 4.1362 - accuracy: 0.3000
Epoch 3/100
3/3 [==============================] - 0s 65ms/step - loss: 3.8386 - accuracy: 0.2875
Epoch 4/100
3/3 [==============================] - 0s 59ms/step - loss: 3.2126 - accuracy: 0.2750
Epoch 5/100
3/3 [==============================] - 0s 92ms/step - loss: 2.6517 - accuracy: 0.2750
Epoch 6/100
3/3 [==============================] - 0s 67ms/step - loss: 2.3857 - accuracy: 0.2875
Epoch 7/100
3/3 [==============================] - 0s 101ms/step - loss: 2.1204 - accuracy: 0.2500
```

```
Epoch 100/100
3/3 [==============================] - 0s 32ms/step - loss: 0.3464 - accuracy: 0.8500

<keras.callbacks.History at 0x7f1b72f54280>
```

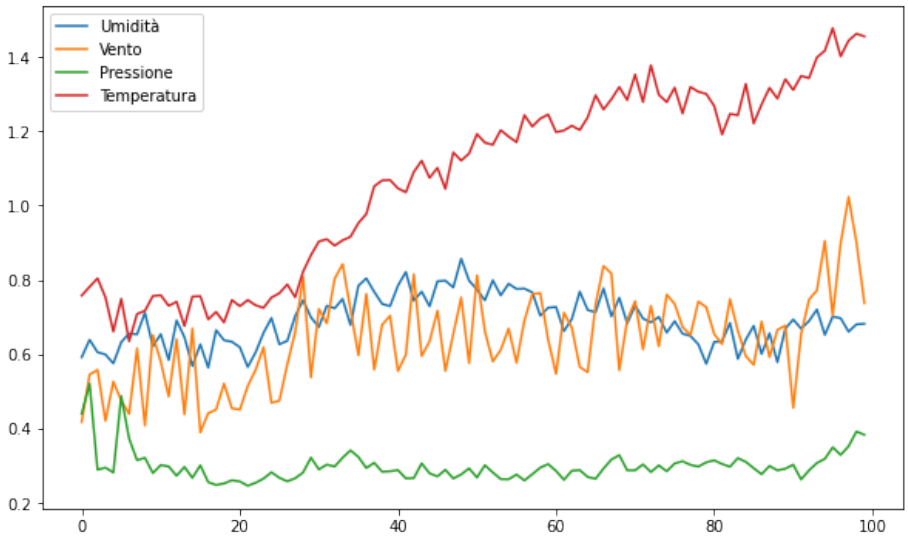
```
_, accuracy = ecg_model.evaluate(ecg_values_test, ecg_labels_test)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

```
1/1 [==============================] - 1s 517ms/step - loss: 0.3792 - accuracy: 0.8500
Accuracy: 85.00%
```

Seq2Seq Semplificato

Sequence-to-Sequence è l'insieme di tecniche che prendono in input una sequenza di lunghezza arbitraria e restituiscono sequenze di lunghezza arbitraria. Partiamo con un caso semplice: prevedere la temperatura giornaliera data l'umidità, la velocità del vento e la pressione. C'è un legame tra questi fenomeni, ma spesso la temperatura dipende dalla pressione, umidità e velocità del vento *passati*. I dati per semplicità sono già stati riscalati.

```
import ml_intro.data

# Forma: [50, 100, 4]
# 50 è il numero di stazioni di rilevamento (si suppone che siano indipendenti tra di loro)
# 100 è il numero di giorni
# 4 è il numero di dati (umidità, velocità del vento, pressione, temperatura)
meteo_data = ml_intro.data.meteo_dataset()

plt.figure(figsize=(10, 6))
plt.plot(meteo_data[0, :, :])
plt.legend(['Umidità', 'Vento', 'Pressione', 'Temperatura'])
plt.show()
```




```
# 50 stazioni di rilevamento, 100 giorni
meteo_X = meteo_data[:, :, :3] # Dati pressione, vento e umidità
meteo_Y = np.expand_dims(meteo_data[:, :, 3], axis=-1) # Trovare la temperatura

meteo_X_train, meteo_X_test, meteo_Y_train, meteo_Y_test = train_test_split(meteo_X, meteo_Y
```

In teoria, se usassimo direttamente un'LSTM dovremmo avere una cella di dimensione 1 (il numero di dati da prevedere), il che potrebbe essere limitante:

```
meteo_model_1 = keras.Sequential([
    keras.layers.LSTM(1, batch_input_shape=(None, None, 4), return_sequences=True) # Senza
])
```

Possiamo però mettere più LSTM uno dopo l'altro:

```
meteo_model_2 = keras.Sequential([
    keras.layers.LSTM(20, batch_input_shape=(None, None, 4), return_sequences=True),
    keras.layers.LSTM(5, batch_input_shape=(None, None, 20), return_sequences=True),
    keras.layers.LSTM(1, batch_input_shape=(None, None, 5), return_sequences=True),
])
```

C'è ancora un problema: l'output di un'LSTM è in [0, 1], ma la temperatura non assume necessariamente quel range (e non sappiamo a priori il range della temperatura). Un'opzione sarebbe di usare una `SimpleRNN` (che, come dice il nome, è una RNN molto semplice), però è un po' un'overkill, visto che dobbiamo semplicemente riscalare dei dati. La strategia più semplice è di usare un layer Dense normale:

```
meteo_model_3 = keras.Sequential([
    keras.layers.LSTM(20, batch_input_shape=(None, None, 3), return_sequences=True),
    keras.layers.LSTM(5, batch_input_shape=(None, None, 20), return_sequences=True),
```

```
    keras.layers.LSTM(1, batch_input_shape=(None, None, 5), return_sequences=True),
    keras.layers.TimeDistributed(keras.layers.Dense(1)) # TimeDistributed fa sì che lo stess
])

meteo_model_3.compile(loss='mse', optimizer='adam')

meteo_model_3.fit(
    meteo_X_train,
    meteo_Y_train,
    epochs=600
)
```

```
Epoch 1/600
2/2 [==============================] - 5s 97ms/step - loss: 0.7473
Epoch 2/600
2/2 [==============================] - 0s 103ms/step - loss: 0.7255
Epoch 3/600
2/2 [==============================] - 0s 98ms/step - loss: 0.7059
Epoch 4/600
2/2 [==============================] - 0s 99ms/step - loss: 0.6879
Epoch 5/600
2/2 [==============================] - 0s 106ms/step - loss: 0.6721
Epoch 6/600
2/2 [==============================] - 0s 101ms/step - loss: 0.6577
Epoch 7/600
2/2 [==============================] - 0s 111ms/step - loss: 0.6447
Epoch 8/600
2/2 [==============================] - 0s 105ms/step - loss: 0.6327
Epoch 9/600
2/2 [==============================] - 0s 107ms/step - loss: 0.6218
Epoch 10/600
2/2 [==============================] - 0s 102ms/step - loss: 0.6115
Epoch 11/600
2/2 [==============================] - 0s 99ms/step - loss: 0.6019
Epoch 12/600
2/2 [==============================] - 0s 100ms/step - loss: 0.5931
Epoch 13/600
2/2 [==============================] - 0s 99ms/step - loss: 0.5846
Epoch 14/600
2/2 [==============================] - 0s 97ms/step - loss: 0.5767
Epoch 15/600
2/2 [==============================] - 0s 101ms/step - loss: 0.5691
Epoch 16/600
2/2 [==============================] - 0s 99ms/step - loss: 0.5618
Epoch 17/600
2/2 [==============================] - 0s 103ms/step - loss: 0.5547
Epoch 18/600
```

```
2/2 [==============================] - 0s 108ms/step - loss: 0.0096
Epoch 594/600
2/2 [==============================] - 0s 123ms/step - loss: 0.0102
Epoch 595/600
2/2 [==============================] - 0s 112ms/step - loss: 0.0099
Epoch 596/600
2/2 [==============================] - 0s 114ms/step - loss: 0.0095
Epoch 597/600
2/2 [==============================] - 0s 133ms/step - loss: 0.0100
Epoch 598/600
2/2 [==============================] - 0s 120ms/step - loss: 0.0096
Epoch 599/600
2/2 [==============================] - 0s 118ms/step - loss: 0.0095
Epoch 600/600
2/2 [==============================] - 0s 113ms/step - loss: 0.0091
```

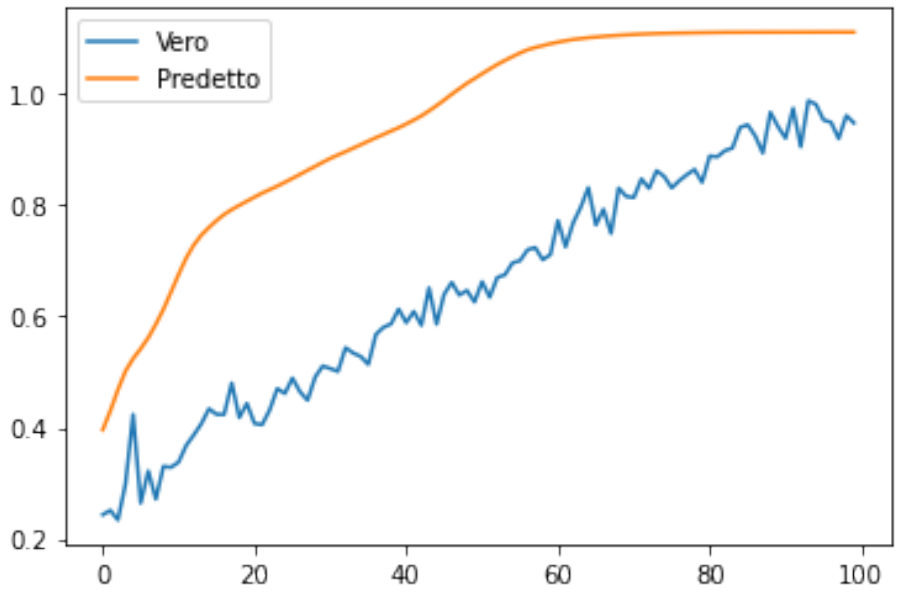
```
<keras.callbacks.History at 0x7f1b5e0559d0>
```

```
loss = meteo_model_3.evaluate(meteo_X_test, meteo_Y_test)
print('Loss:', loss)
```

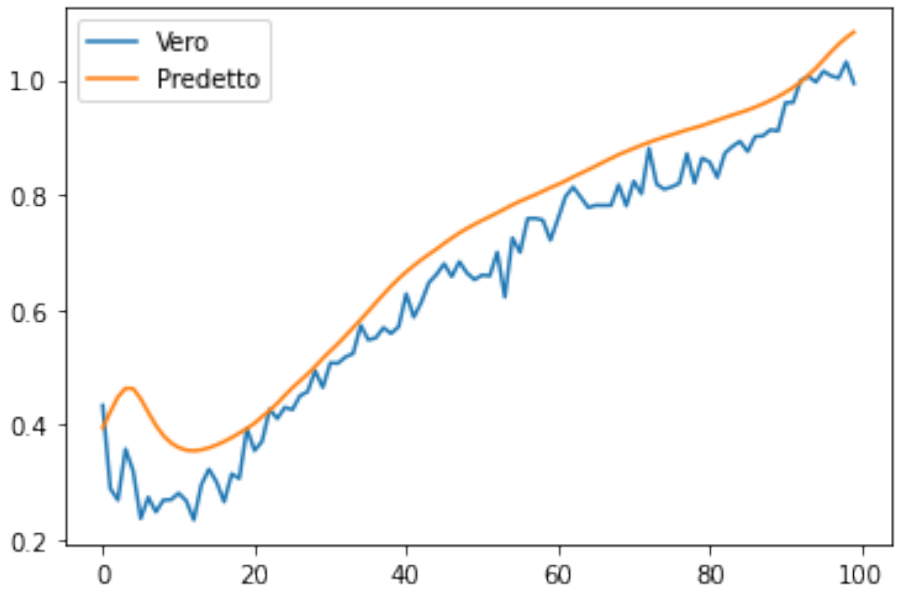
```
1/1 [==============================] - 2s 2s/step - loss: 0.0233
Loss: 0.023311957716941833
```

```
for i in range(3):
    plt.plot(meteo_Y_test[i])
    plt.plot(meteo_model_3.predict(meteo_X_test)[i])
    plt.legend(['Vero', 'Predetto'])
    plt.show()
```

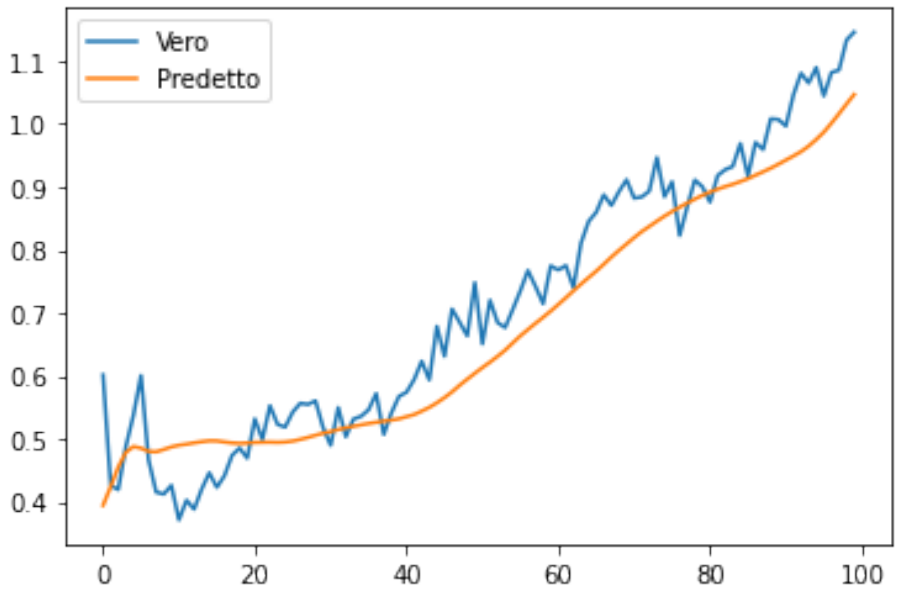
```
1/1 [==============================] - 1s 1s/step
```




```
1/1 [==============================] - 0s 41ms/step
```




```
1/1 [==============================] - 0s 44ms/step
```

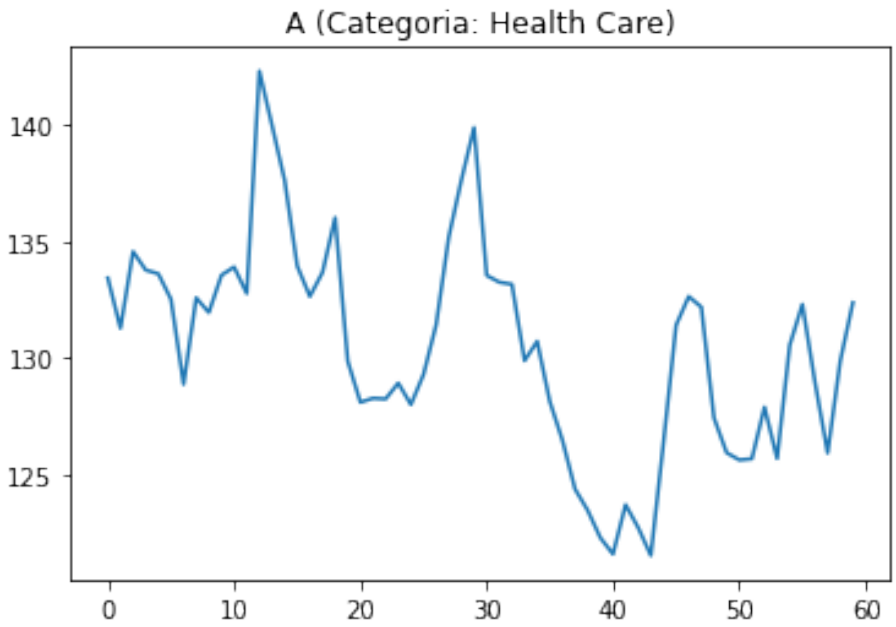


Esercizio: Studio di Correlazioni nei Mercati

Il mercato finanziario è interentemente imprevedibile, però si possono fare studi sulla correlazione tra i titoli (es. "se sale il settore informatico e scende quello industriale, allora sale il settore agricolo").

Questi sono i prezzi di chiusura degli ultimi 60 giorni dei primi 250 titoli (in ordine alfabetico) di S&P 500, con anche la categoria di riferimento.

```
# sp250_stock_names e sp250_category_names non servono ai fini dell'esercizio,
# è più per dare un senso ai dati
sp250_prices, sp250_categories, sp250_stock_names, sp250_category_names = ml_intro.data.sp25

print(sp250_stock_names)

['A', 'AAL', 'AAP', 'AAPL', 'ABBV', 'ABC', 'ABMD', 'ABT', 'ACN', 'ADI', 'ADM', 'ADP', 'ADSK'

print(sp250_category_names)

['Communication Services', 'Consumer Discretionary', 'Consumer Staples', 'Energy', 'Financia

plt.plot(sp250_prices[0, :])
plt.title(f'{sp250_stock_names[0]} (Categoria: {sp250_category_names[sp250_categories[0]]})'
plt.show()
```



L'esercizio consiste nel prevedere l'andamento del 10% dei titoli nel periodo 50° giorno - 60° giorno sapendo:

- L'andamento di questo 10% dei titoli nel periodo 1° giorno - 49° giorno
- L'andamento del 90% dei titoli restanti su tutti i giorni

Essenzialmente ci si sta chiedendo: se nell'ultimo periodo (50° - 60°) i titoli che compongono il 90% si comportassero in questo modo, come si comporterebbero i titoli del 10% rimasto?

Miglioriamo l'LSTM

L'LSTM usata sopra ha due problemi:

- Funziona solo se la sequenza di input ha la stessa lunghezza della sequenza di output
- Non può fare previsioni al di fuori del range temporale su cui è addestrata (non può "estendere la linea")

Per questo, usiamo un approccio diverso: addestriamo la rete a prevedere, dati T elementi della sequenza, qual è il $T+1$-esimo.

A livello pratico è come se stessimo addestrando un suggeritore da tastiera:

Cambiamo quindi l'obiettivo: avendo tutti e quattro i dati per i primi 95 giorni, dobbiamo riuscire a prevedere l'andamento degli ultimi 5 giorni senza sapere

nulla su quest'ultimi.

```
meteo_data_known = meteo_data[:, :95, :]
meteo_data_unknown = meteo_data[:, 95:, :]

# Dato l'input fino a quel punto...
meteo_data_X = np.zeros_like(meteo_data_known)
# L'elemento i-esimo avrà l'input (i-1)-esimo (il primo elemento ha valore 0 e basta)
meteo_data_X[:, 1:95, :] = meteo_data_X[:, :94, :]

# ...prevedere l'input successivo
meteo_data_Y = meteo_data_known.copy()
```

Usiamo l'API funzionale.

Nota: gli stiamo facendo ritornare tutte le previsioni per l'elemento successivo a ogni singolo step. Questo è perché così possiamo addestrare in parallelo su tutta la timeseries.

```
lstm_inputs = keras.Input(shape=(None, 4))
next_step_lstm = keras.layers.LSTM(15, batch_input_shape=(None, None, 4), return_state=True,
lstm_outputs, state_h, state_c = next_step_lstm(lstm_inputs)

dense = keras.layers.TimeDistributed(keras.layers.Dense(4))
model_outputs = dense(lstm_outputs)

next_step_model = keras.Model([lstm_inputs], [model_outputs])

next_step_model.compile(optimizer='adam', loss='mse')

next_step_model.fit(
    meteo_data_X,
    meteo_data_Y,
    epochs=300
)
```

```
Epoch 1/300
2/2 [==============================] - 2s 49ms/step - loss: 0.3628
Epoch 2/300
2/2 [==============================] - 0s 37ms/step - loss: 0.3569
Epoch 3/300
2/2 [==============================] - 0s 39ms/step - loss: 0.3507
Epoch 4/300
2/2 [==============================] - 0s 37ms/step - loss: 0.3441
Epoch 5/300
2/2 [==============================] - 0s 33ms/step - loss: 0.3371
Epoch 6/300
2/2 [==============================] - 0s 36ms/step - loss: 0.3295
Epoch 7/300
```

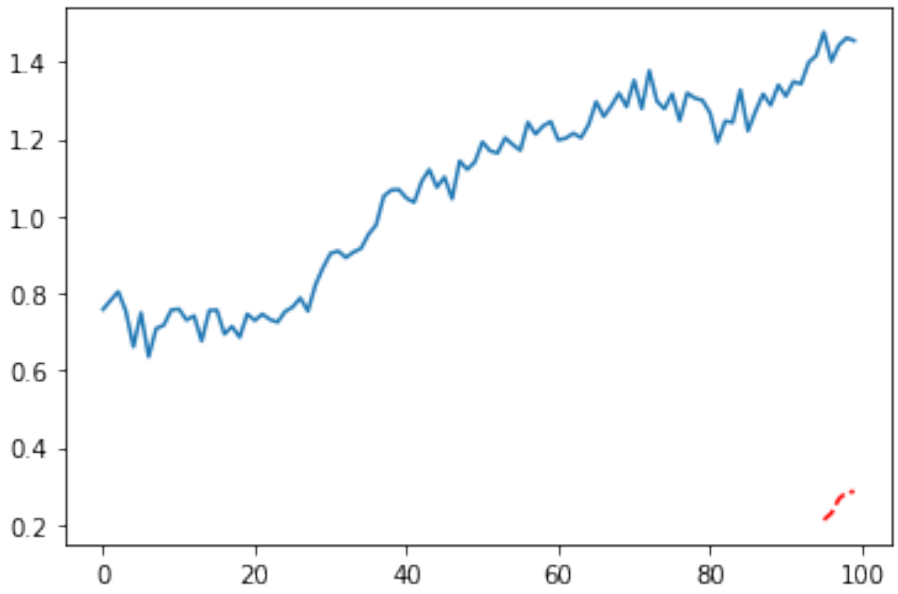
```
    return next_step_model.predict(inputs)[:, -1, :]

def predict_next_k(inputs, window_size):
    timeseries_length = inputs.shape[1]
    new_inputs = np.zeros((inputs.shape[0], timeseries_length + window_size, inputs.shape[2]
    new_inputs[:, :timeseries_length, :] = inputs

    for i in range(window_size):
        print(f'Running on new_inputs[{i}:{timeseries_length + i}] to predict {timeseries_le
        subsequence = new_inputs[:, :timeseries_length + i, :]
        new_inputs[:, timeseries_length + i, :] = predict_next(subsequence)

    # Restituisci gli ultimi window_size elementi
    return new_inputs[:, timeseries_length:, :]

predicted_5_days = predict_next_k(meteo_data_known, 5)
print('Final shape:', predicted_5_days.shape)
```

```
Running on new_inputs[0:95] to predict 95
2/2 [==============================] - 0s 12ms/step
Running on new_inputs[1:96] to predict 96
2/2 [==============================] - 1s 11ms/step
Running on new_inputs[2:97] to predict 97
2/2 [==============================] - 0s 11ms/step
Running on new_inputs[3:98] to predict 98
2/2 [==============================] - 0s 11ms/step
Running on new_inputs[4:99] to predict 99
2/2 [==============================] - 0s 15ms/step
Final shape: (50, 5, 4)
```

```
print('MSE:', np.mean((predicted_5_days - meteo_data_unknown) ** 2))
```

```
MSE: 0.2501247375440814
```

```
# Plottiamo la temperatura della prima stazione di rilevamento
plt.plot(meteo_data[0, :, 3])
plt.plot(np.arange(95, 100), predicted_5_days[0, :, 3], color='red', linestyle='dashed')
plt.show()
```



Esercizio Extra: Previsione Vendite

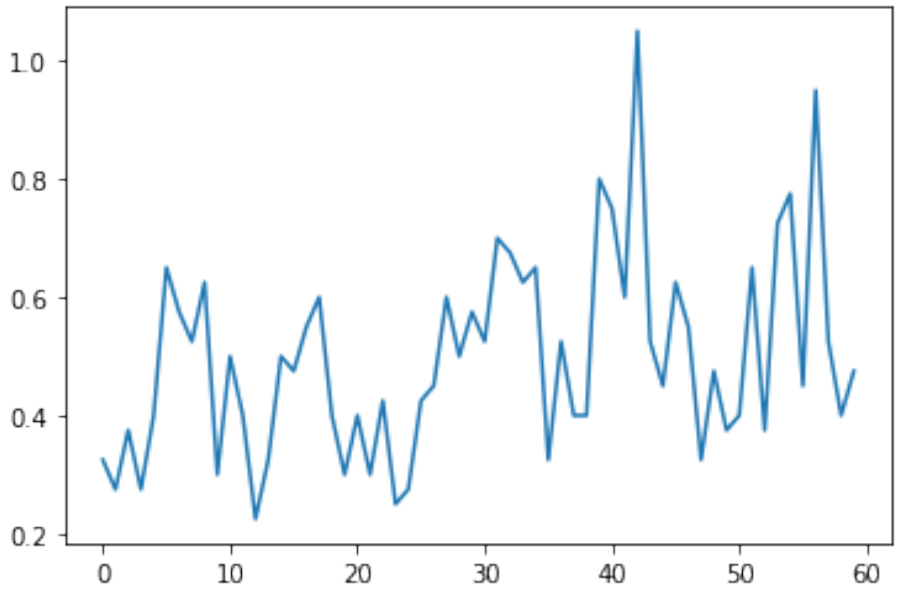
Dato il seguente dataset, che contiene le vendite mensili (in migliaia) per una serie di prodotti:

```
import pandas as pd
df = pd.read_csv('ml_intro/data/sales.csv')
print(df)
```

```
            date  store  item  sales
0     2013-01-01      1     0     13
1     2013-02-01      1     0     11
2     2013-03-01      1     0     15
3     2013-04-01      1     0     11
4     2013-05-01      1     0     16
...          ...    ...   ...    ...
2995  2017-08-01      1    49     68
2996  2017-09-01      1    49     64
2997  2017-10-01      1    49     66
2998  2017-11-01      1    49     59
2999  2017-12-01      1    49     51

[3000 rows x 4 columns]
```

```
sales = np.zeros([50, 60, 1])
for i in range(50):
    sales[i, :, 0] = df[df['item'] == i]['sales']
```

```
# Anche con la correzione di scala fatta da un Dense, le LSTM impiegano molto a
# imparare scale lontane da [0, 1]. Per velocizzare la spiegazione divido per 40
# Questo è puramente per scopi didattici, ci sono approcci migliori
sales /= 40

plt.plot(sales[0])
plt.show()
```



Prevedete le vendite dei prossimi 6 mesi conoscendo le vendite dei 54 mesi passati

```
# Come loss si usa mse
```

Lavorare con il Testo

Le LSTMs, come ogni rete neurale, prendono in input vettori, non parole. Per questa ragione quando vogliamo lavorare con il testo si usano due strumenti:

- Un tokenizer, che divide il testo in ”token”, ovvero unità (solitamente parole)
- Un embedding, che converte i token in vettori che la rete può usare

C'è inoltre da gestire un altro problema: frasi diverse hanno lunghezze diverse.

```
import json

with open('./ml_intro/data/positive-reviews.json') as f:
    positive_reviews = json.load(f)
```

```
with open('./ml_intro/data/negative-reviews.json') as f:
    negative_reviews = json.load(f)
```

```
print(positive_reviews[:2])
print(negative_reviews[:2])
```

```
['\nSphere by Michael Crichton is an excellant novel. This was certainly the hardest to put
["\nTHis book was horrible.  If it was possible to rate it lower than one star i would have
```

Preparazione

Per gestire la lunghezza diversa delle frasi, aggiungiamo esplicitamente due parole (es. <START> ed <END>) al testo per far capire alla rete che sono l'inizio e la fine della frase. Questo normalmente aiuta anche la rete a ragionare meglio sul testo. Tornerà anche utile dopo per seq2seq.

```
positive_reviews = ['<START> ' + s + ' <END>' for s in positive_reviews]
negative_reviews = ['<START> ' + s + ' <END>' for s in negative_reviews]
print(positive_reviews[0])
```

```
<START>
Sphere by Michael Crichton is an excellant novel. This was certainly the hardest to put down

The story revolves around a man named Norman Johnson. Johnson is a phycologist. He travels w

This novel does not have the research that some of the other Crichton novels have, but it st

I would strongly recommend this book
 <END>
```

Prepariamo poi i dati

```
all_reviews = positive_reviews + negative_reviews
# Genera un array con valori [1, 1, 1, 1... 1, 0, 0, 0, ..., 0]
review_labels = np.array([1] * len(positive_reviews) + [0] * len(negative_reviews))
```

```
review_text_train, review_text_test, review_labels_train, review_labels_test = train_test_sp
```

Tokenizer

```
tokenizer = keras.layers.TextVectorization(max_tokens=20000, output_sequence_length=200)
text_ds = tf.data.Dataset.from_tensor_slices(review_text_train).batch(128)
tokenizer.adapt(text_ds)
```

Stampiamo le 10 parole più comuni:

```
print(tokenizer.get_vocabulary()[:10])
```

```
['', '[UNK]', 'the', 'of', 'and', 'to', 'a', 'is', 'i', 'in']
```

```
example_output = tokenizer([['the pen is on the table']])
print(example_output.numpy()[0, :20])
```

```
[   2 6642    7   19    2  946    0    0    0    0    0    0    0    0
    0    0    0    0    0    0]
```

Embedding

Potremmo addestrare un embedding nostro, ma è molto più semplice usare un embedding pre-addestrato.

Nota: questo codice sotto serve principalmente per far funzionare questo embedding specifico. Non prendetelo come oro colato.

```
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip -q glove.6B.zip
```

```
--2022-12-12 21:35:28--  http://nlp.stanford.edu/data/glove.6B.zip
Resolving nlp.stanford.edu (nlp.stanford.edu)... 171.64.67.140
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://nlp.stanford.edu/data/glove.6B.zip [following]
--2022-12-12 21:35:28--  https://nlp.stanford.edu/data/glove.6B.zip
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
--2022-12-12 21:35:29--  https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)... 171.64.64.22
Connecting to downloads.cs.stanford.edu (downloads.cs.stanford.edu)|171.64.64.22|:443... co
HTTP request sent, awaiting response... 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: ‘glove.6B.zip’

glove.6B.zip        100%[===================>] 822.24M  5.08MB/s    in 2m 39s

2022-12-12 21:38:08 (5.17 MB/s) - ‘glove.6B.zip’ saved [862182613/862182613]
```

```
import os

embeddings_index = {}
with open('glove.6B.100d.txt') as f:
    for line in f:
        word, coefs = line.split(maxsplit=1)
        coefs = np.fromstring(coefs, "f", sep=" ")
        embeddings_index[word] = coefs
```

```
# Prepara la matrice dell'embedding (che quindi associa ogni parola del dizionario del token
# a un embedding)

embedding_dim = 100 # GloVe 6B 100 usa un embedding che dà un tensore di 100 elementi per p

vocabulary = tokenizer.get_vocabulary()
num_tokens = len(vocabulary) + 2 # I 2 extra sono '' (vuoto) e [UNK] (sconosciuto)

word_index = dict(zip(vocabulary, range(len(vocabulary))))

embedding_matrix = np.zeros((num_tokens, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector

embedding_layer = keras.layers.Embedding(
    num_tokens,
    embedding_dim,
    embeddings_initializer=keras.initializers.Constant(embedding_matrix),
    trainable=False,
)
```

Se avessimo voluto addestrare noi l'embedding, avremmo semplicemente dovuto scrivere:

```
embedding_layer = keras.layers.Embedding(
    num_tokens,
    embedding_dim
)
```

Stampiamo come viene salvata la parola "pen":

```
print(embedding_layer(tokenizer([['pen']]))[0][0])
```

```
tf.Tensor(
[-0.34552    0.29789   -0.37904   -0.31938    0.62888    0.61416
  0.49733   -0.41696   -0.3151    -0.20626   -0.97058    0.38394
 -0.73213   -0.51389   -0.13533    0.81919    0.13479   -0.54161
  0.36336   -0.44535    0.30318    0.89861    0.18018    0.017982
  0.97207   -0.3767    -0.34253   -0.62245    0.53802   -0.45462
  0.37347    0.66855    0.045919   0.31317    0.57272    0.13682
  0.077149   0.1073    -0.38126   -0.33555   -0.35833   -0.28619
 -0.3241    -1.1862    -0.59932   -0.19396   -0.36689   -0.40929
 -0.55322    0.049855  -0.39643    0.36457    0.09228    0.83146
 -0.38847   -1.126      0.29753    0.62646    0.47811   -0.22676
  0.71105    0.63182   -0.36587    0.54473    0.16763    0.16875
  0.16516    0.33061   -0.12752   -0.16011   -0.26389    0.45155
  0.3081    -0.62813   -0.57092    0.39991    0.69702    0.2313
```

```
 -0.61316    0.58607    0.19509   -0.36331   -0.13374   -0.72725
 -1.0138    -0.51908    0.6747     0.15295    0.7359     0.23369
  0.15363   -0.0087955  1.3251     0.93143    0.1579     0.051251
 -0.24631   -0.38782    0.20755   -0.49034  ], shape=(100,), dtype=float32)
```

Confrontiamo l'embedding della parola "cat" con "paper" e "feline"

```
cat_embedding = embedding_layer(tokenizer([['cat']]))[0][0]
paper_embedding = embedding_layer(tokenizer([['paper']]))[0][0]
feline_embedding = embedding_layer(tokenizer([['feline']]))[0][0]
print('Distance between cat and paper:', tf.reduce_sum((cat_embedding - paper_embedding) **
print('Distance between cat and feline:', tf.reduce_sum((cat_embedding - feline_embedding) *
```

```
Distance between cat and paper: tf.Tensor(41.78264, shape=(), dtype=float32)
Distance between cat and feline: tf.Tensor(25.394722, shape=(), dtype=float32)
```

Classificazione Testo

Ora che abbiamo come input dei vettori, possiamo addestrare una LSTM come prima.

Nota: potremmo mettere direttamente il tokenizer e l'embedding nel modello, così:

```
review_model = keras.Sequential([
    tokenizer,
    embedding_layer,
    review_lstm,
    ...
])
```

però visto che bisognerebbe passare al modello tutto il testo (e visto che il tokenizer non è molto efficiente su GPU), è più efficiente calcolare una volta i token delle parole delle recensioni e lavorare direttamente su quelli.

```
review_text_train_tokenized = tokenizer(review_text_train)
print(review_text_train_tokenized.shape)
```

```
(1600, 200)
```

```
review_lstm = keras.layers.LSTM(50, batch_input_shape=(None, None, 100))
```

```
review_model = keras.Sequential([
    embedding_layer,
    review_lstm, # Output: [batch_size, 50]
    keras.layers.Dense(10),
    keras.layers.ReLU(),
    keras.layers.Dense(1)
])
```

```
review_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

review_model.fit(review_text_train_tokenized, review_labels_train, epochs=50)
```

```
Epoch 1/50
50/50 [==============================] - 5s 70ms/step - loss: 7.7609 - accuracy: 0.4969
Epoch 2/50
50/50 [==============================] - 3s 70ms/step - loss: 7.7607 - accuracy: 0.4969
Epoch 3/50
50/50 [==============================] - 3s 69ms/step - loss: 7.7607 - accuracy: 0.4969
Epoch 4/50
50/50 [==============================] - 4s 71ms/step - loss: 7.7607 - accuracy: 0.4969
Epoch 5/50
50/50 [==============================] - 4s 71ms/step - loss: 7.7607 - accuracy: 0.4969
Epoch 6/50
50/50 [==============================] - 4s 70ms/step - loss: 7.7607 - accuracy: 0.4969
Epoch 7/50
50/50 [==============================] - 4s 70ms/step - loss: 7.7607 - accuracy: 0.4969
Epoch 8/50
50/50 [==============================] - 4s 71ms/step - loss: 7.7607 - accuracy: 0.4969
Epoch 9/50
50/50 [==============================] - 3s 70ms/step - loss: 7.7607 - accuracy: 0.4969
Epoch 10/50
50/50 [==============================] - 3s 70ms/step - loss: 7.7607 - accuracy: 0.4969
Epoch 11/50
50/50 [==============================] - 3s 69ms/step - loss: 7.7607 - accuracy: 0.4969
Epoch 12/50
50/50 [==============================] - 4s 70ms/step - loss: 7.7607 - accuracy: 0.4969
Epoch 13/50
50/50 [==============================] - 4s 70ms/step - loss: 7.7607 - accuracy: 0.4969
Epoch 14/50
50/50 [==============================] - 4s 70ms/step - loss: 7.7607 - accuracy: 0.4969
Epoch 15/50
50/50 [==============================] - 4s 70ms/step - loss: 7.7607 - accuracy: 0.4969
Epoch 16/50
50/50 [==============================] - 4s 71ms/step - loss: 7.7607 - accuracy: 0.4969
Epoch 17/50
50/50 [==============================] - 3s 70ms/step - loss: 7.7607 - accuracy: 0.4969
Epoch 18/50
50/50 [==============================] - 4s 70ms/step - loss: 7.7607 - accuracy: 0.4969
Epoch 19/50
50/50 [==============================] - 3s 69ms/step - loss: 7.7607 - accuracy: 0.4969
Epoch 20/50
50/50 [==============================] - 3s 70ms/step - loss: 7.7607 - accuracy: 0.4969
Epoch 21/50
50/50 [==============================] - 4s 70ms/step - loss: 7.7607 - accuracy: 0.4969
```

```
Epoch 45/50
50/50 [==============================] - 4s 70ms/step - loss: 7.7607 - accuracy: 0.4969
Epoch 46/50
50/50 [==============================] - 3s 70ms/step - loss: 7.7607 - accuracy: 0.4969
Epoch 47/50
50/50 [==============================] - 4s 70ms/step - loss: 7.7607 - accuracy: 0.4969
Epoch 48/50
50/50 [==============================] - 4s 70ms/step - loss: 7.7607 - accuracy: 0.4969
Epoch 49/50
50/50 [==============================] - 4s 70ms/step - loss: 7.7607 - accuracy: 0.4969
Epoch 50/50
50/50 [==============================] - 3s 69ms/step - loss: 7.7607 - accuracy: 0.4969
```

```
<keras.callbacks.History at 0x7f1b72ac1a00>
```

```
review_text_test_tokenized = tokenizer(review_text_test)
_, accuracy = review_model.evaluate(review_text_test_tokenized, review_labels_test)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

```
13/13 [==============================] - 1s 29ms/step - loss: 7.5197 - accuracy: 0.5125
Accuracy: 51.25%
```

Seq2Seq con Testo

Così come si può fare seq2seq con dati numerici, usando un tokenizer possiamo fare seq2seq con il testo. Un esempio classico è riassumere il testo.

Rispetto al caso dei dati numerici, ci sono due aspetti da gestire:

- Frasi diverse avranno lunghezze diverse
- La rete non ha modo di ragionare sui sinonimi o su parole di significato collegato (es. ”gatto” e ”felino”)

```
headlines, complete_texts = ml_intro.data.headlines_dataset()


print(headlines[0])
print(complete_texts[0][:1000])
```

```
Daman & Diu revokes mandatory Rakshabandhan in offices order
The Administration of Union Territory Daman and Diu has revoked its order that made it compu
```

```
headlines = ['<START> ' + headline + ' <END>' for headline in headlines]
complete_texts = ['<START> ' + complete_text + ' <END>' for complete_text in complete_texts]

print(headlines[0])
print(complete_texts[0][:1000])
```

```
<START> Daman & Diu revokes mandatory Rakshabandhan in offices order <END>
<START> The Administration of Union Territory Daman and Diu has revoked its order that made
```

Si tratta semplicemente di tokenizzare e addestrare un modello seq2seq come quelli di cui sopra

```
tokenizer = keras.layers.TextVectorization(max_tokens=1000, output_sequence_length=500)
text_ds = tf.data.Dataset.from_tensor_slices(headlines + complete_texts).batch(128)
tokenizer.adapt(text_ds)
```

```
headlines_tokenized = tokenizer(np.array(headlines))
complete_texts_tokenized = tokenizer(np.array(complete_texts))

complete_texts_tokenized_prev = np.zeros_like(complete_texts_tokenized)
print(complete_texts_tokenized.shape)
complete_texts_tokenized_prev[:, 1:] = complete_texts_tokenized[:, :499]
```

```
(4514, 500)
```

```
# Prepara la matrice dell'embedding (che quindi associa ogni parola del dizionario del token
# a un embedding)

embedding_dim = 100 # GloVe 6B 100 usa un embedding che dà un tensore di 100 elementi per p

vocabulary = tokenizer.get_vocabulary()
num_tokens = len(vocabulary) + 2 # I 2 extra sono '' (vuoto) e [UNK] (sconosciuto)

word_index = dict(zip(vocabulary, range(len(vocabulary))))

embedding_matrix = np.zeros((num_tokens, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector

embedding_layer = keras.layers.Embedding(
    num_tokens,
    embedding_dim,
    embeddings_initializer=keras.initializers.Constant(embedding_matrix),
    trainable=False,
)
```

C'è però un aspetto extra da gestire: noi vogliamo che il riassunto venga generato *dopo* aver letto l'articolo. Un approccio possibile è di trattare tutto come un unico testo, e chiedere di prevedere il completamento:

```
<START-FULL> Full text: In Chicago, stores have begun to prepare for the holidays [...] <END
```

questo richiede un po' di lavoro (es. evitare che il testo intero sia troppo lungo, magari tagliandolo), però si può fare (e infatti viene usato spesso nei transformer). Con le LSTM si tende però a usare un approccio diverso:

- Esegui una LSTM (chiamata encoder) sul primo testo

- Trasferisci lo stato della prima LSTM a una seconda LSTM, chiamata decoder
- Usa la seconda LSTM per prevedere la *i*-esima parola data la *(i-1)*-esima

Nota: usiamo l'API funzionale (invece di usare Sequential, diciamo esplicitamente noi come vengono gestiti i dati dei layer). Per un po' di esempi di API funzionale, vedete i notebook di Asperti

```
encoder_inputs = keras.Input(shape=(500))

# return_state fa ritornare anche lo stato dell'LSTM
encoder = keras.layers.LSTM(50, return_state=True, batch_input_shape=(None, 500, 100),)
encoder_outputs, state_h, state_c = encoder(embedding_layer(encoder_inputs))

# Ignoriamo encoder_outputs, guardiamo solo lo stato
encoder_states = [state_h, state_c]

decoder_inputs = keras.Input(shape=(500))


decoder_lstm = keras.layers.LSTM(50, batch_input_shape=(None, 500, 100), return_sequences=Tr
decoder_outputs, _, _ = decoder_lstm(embedding_layer(decoder_inputs), initial_state=encoder_
decoder_dense = keras.layers.Dense(len(vocabulary) + 2, activation="softmax")
decoder_outputs = decoder_dense(decoder_outputs)

# Define the model that will turn
# `encoder_input_data` & `decoder_input_data` into `decoder_target_data`
summarization_model = keras.Model([encoder_inputs, decoder_inputs], decoder_outputs)
```

Valutare la qualità di un riassunto è un lavoro difficilmente automatizzabile. Il modo più semplice (e più automatizzabile) e valutare se ha usato esattamente le stesse parole della soluzione "ufficiale". Non è un granché come metrica, ma è molto semplice.

```
summarization_model.compile(
    optimizer='adam',
    loss=keras.losses.SparseCategoricalCrossentropy() # Uguale a crossentropy, ma non richie
)

summarization_model.fit([headlines_tokenized, complete_texts_tokenized_prev], complete_texts
```

```
Epoch 1/10
142/142 [==============================] - 109s 763ms/step - loss: 2.5362
Epoch 2/10
142/142 [==============================] - 109s 771ms/step - loss: 2.5220
Epoch 3/10
142/142 [==============================] - 107s 751ms/step - loss: 2.5104
Epoch 4/10
142/142 [==============================] - 111s 782ms/step - loss: 2.4967
```

```
Epoch 5/10
142/142 [==============================] - 106s 744ms/step - loss: 2.4853
Epoch 6/10
142/142 [==============================] - 106s 745ms/step - loss: 2.4748
Epoch 7/10
142/142 [==============================] - 108s 763ms/step - loss: 2.4649
Epoch 8/10
142/142 [==============================] - 106s 744ms/step - loss: 2.4554
Epoch 9/10
142/142 [==============================] - 108s 761ms/step - loss: 2.4461
Epoch 10/10
142/142 [==============================] - 106s 743ms/step - loss: 2.4376
```

```
<keras.callbacks.History at 0x7f1b7c822100>
```

```python
def predict_next_token(inputs, inputs_2):
    return summarization_model.predict([inputs, inputs_2])

def predict_next_k_tokens(inputs, inputs_2, window_size):
    timeseries_length = inputs.shape[1]
    new_inputs = np.zeros((inputs.shape[0], timeseries_length + window_size))
    new_inputs[:, :timeseries_length] = inputs
    new_inputs_2 = np.zeros((inputs_2.shape[0], timeseries_length + window_size))
    new_inputs_2[:, :timeseries_length] = inputs_2

    for i in range(window_size):
        print(f'Running on new_inputs[{i}:{timeseries_length + i}] to predict {timeseries_le
        subsequence = new_inputs[:, i:timeseries_length + i]
        subsequence_2 = new_inputs_2[:, i:timeseries_length + i]
        new_inputs[:, timeseries_length + i], new_inputs_2[:, timeseries_length + i] = predi

    # Restituisci gli ultimi window_size elementi
    return new_inputs[:, timeseries_length:], new_inputs_2[:, timeseries_length:]
```

Nota: per riconvertire i token in testo, si può usare `tokenizer.sequences_to_texts`

Esercizio Extra: Autoencoder LSTM

Ci sono molte somiglianze tra l'ultima rete fatta e gli autoencoder. Infatti, è possibile utilizzare le LSTM per fare autoencoder su testo o altre sequenze.

Addestrare un autoencoder su uno dei due dataset (review o news) e confrontare gli encoding di frasi simili.