Leucemia linfoblastica acuta

Confronto tra classificatori di immagini oncologiche

- Quirile Alessandro N97/402
- Matrisciano Ivano N97/392

https://github.com/alessandroquirile/Leukemia/

Panoramica

Contenuti:

- Data exploration e image pre-processing
- Feature extraction
- Feature selection
- Analisi sperimentale

LA LEUCEMIA LINFOBLASTICA ACUTA

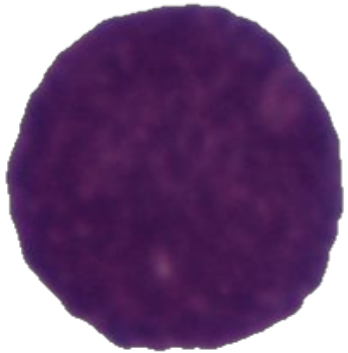
La leucemia linfoblastica acuta (ALL) è il tipo di cancro più comune nei bambini, rappresentando circa il 25% dei tumori in età tra 0 e 14 anni [1].

Il processo di identificazione di cellule malate è affidato a personale altamente specializzato, è complesso e **time-consuming**. In questo lavoro la diagnosi è automatizzata partendo da immagini di cellule visualizzate al microscopio.

UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

Analisi del dataset

QUAL È LA CELLULA MALATA?

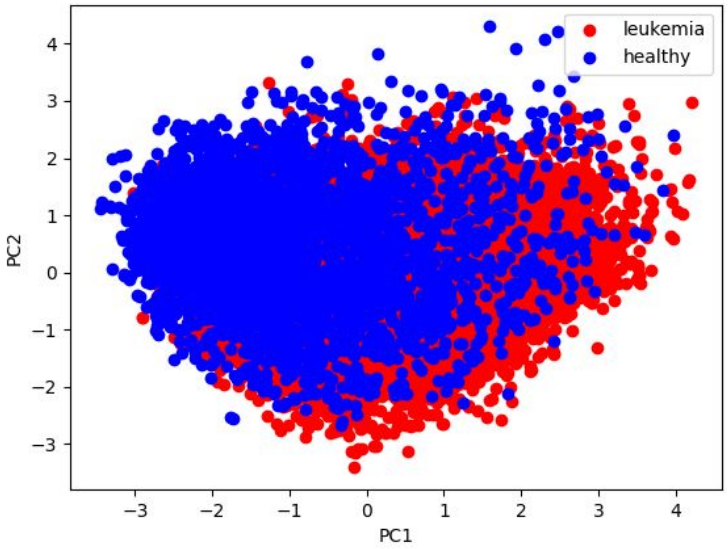


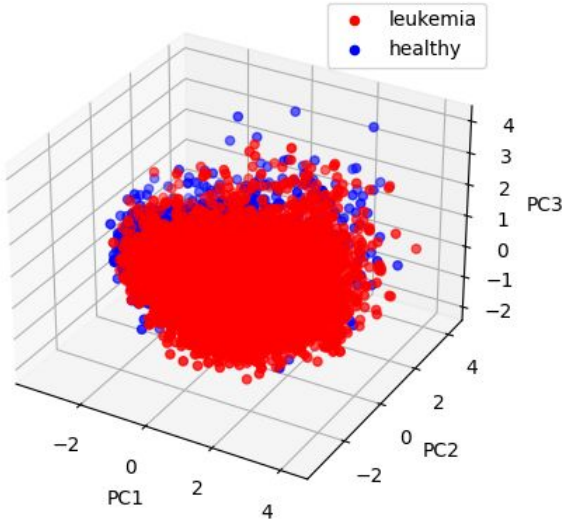
QUAL È LA CELLULA MALATA?

MALATA

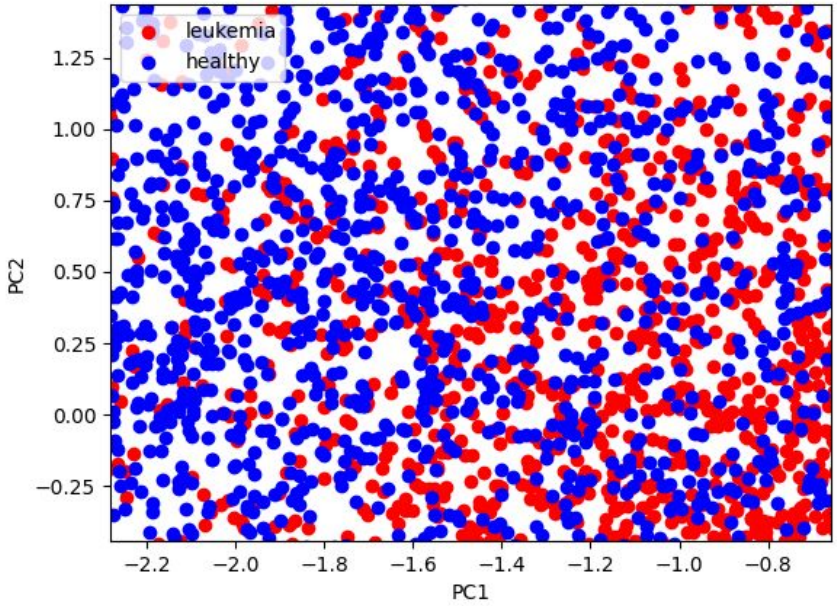
SANA

CELLULE MALATE E SANE SONO SIMILI





CELLULE MALATE E SANE SONO SIMILI



IL DATASET NON È BILANCIATO

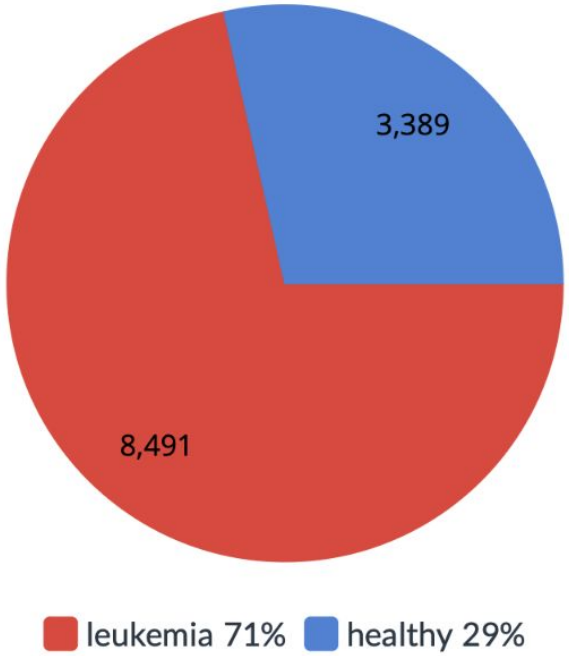


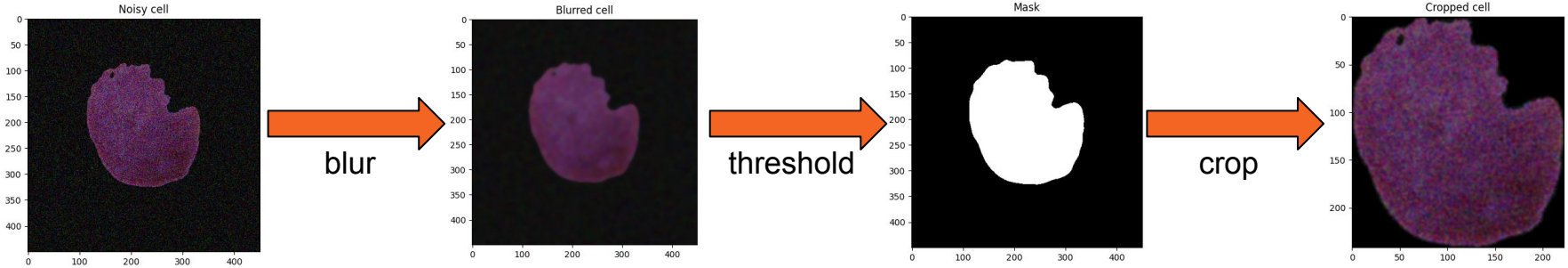
Image pre-processing

Immagini di partenza

L’addestramento dei modelli è stato effettuato su immagini senza rumore significativo.

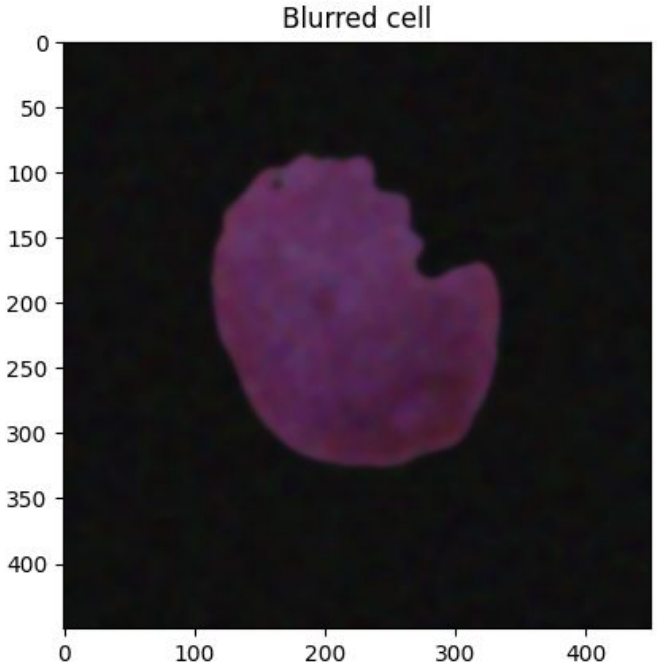
Nonostante ciò è stata implementata una tecnica di pulizia delle immagini per ridurre eventuale rumore gaussiano e segmentare la cellula: così facendo il sistema risulta robusto anche qualora venga data in input un’immagine rumorosa.

Segmentare la cellula



(450, 450, 3) (450, 450, 3) (450, 450, 1) (242, 223, 3)

Step 1: Blurring

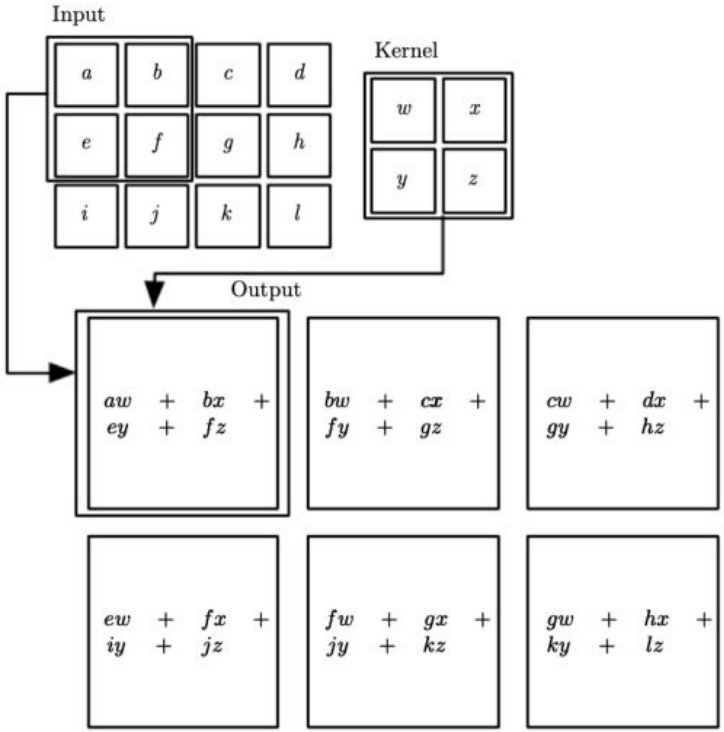


L'immagine rumorosa in input è stata filtrata mediante una combinazione di filtri gaussiani e a mediana.

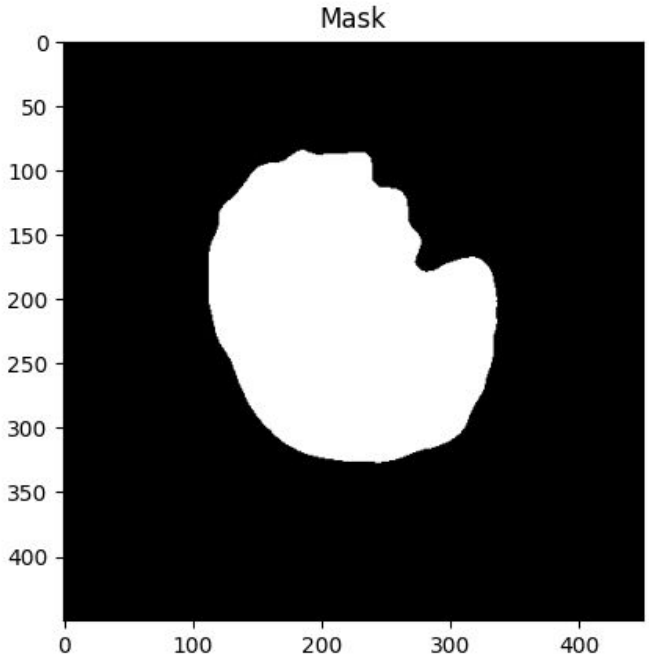
1. Filtro **gaussiano**, con dimensione del kernel di (13, 13) ha lo scopo di ridurre il rumore sia sulla cellula che sullo sfondo.

2. Filtro a **mediana** avente kernel con dimensione (11, 11) con lo scopo di filtrare ulteriormente eventuali pixel rimasti illuminati nella zona dello sfondo.

La convoluzione



Step 2: Thresholding (1)



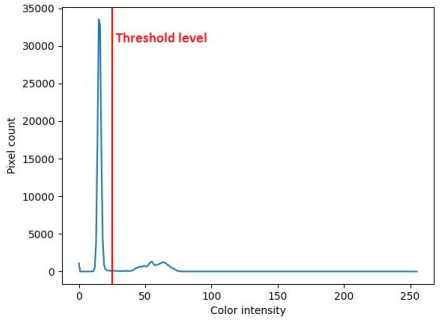
L'immagine blurred è stata trasformata in toni di grigio ed è stato applicato un **thresholding binario** che permette di separare la cellula dallo sfondo scuro.

$$\text{dst(x,y)} = \begin{cases} 255 & \text{se src(x,y)} > 25 \\ 0 & \text{altrimenti} \end{cases}$$

Tutte le immagini sono codificate in *uint8*.

I pixel che nell'immagine blurred hanno un'intensità maggiore di 25 vengono trasformati nel colore bianco (255); tutti gli altri nel colore nero (0).

Step 2: Thresholding (2)

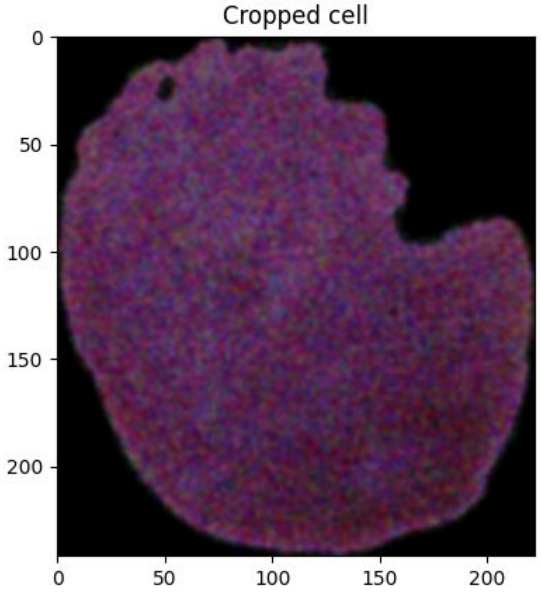


Abbiamo osservato empiricamente che la **sogliatura binaria è bastevole** e funziona bene per il nostro problema.

Alternativamente avremmo potuto utilizzare metodi più sofisticati come la **sogliatura adattiva**, ad esempio per gestire casi di varianza intra-classe: in generale un'immagine può avere aree illuminate diversamente; questo, comunque, è poco probabile per immagini oncologiche catturate da macchine professionali *ad hoc*.

Ancora, in alternativa, avremmo potuto utilizzare la **sogliatura di Otsu**: analizza automaticamente l'istogramma di colore dell'immagine e determina una valle di scissione tra le due creste (se l'immagine è bimodale).

Step 3: Cropping



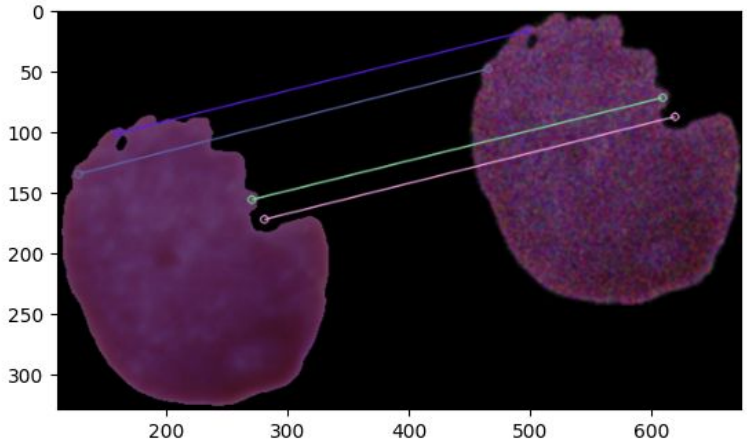
Una volta individuata la maschera, l'immagine *cropped* della cellula è ottenuta ritagliando dall'immagine originale la zona data dal **bounding box** della maschera, ovvero la sua regione significativa.

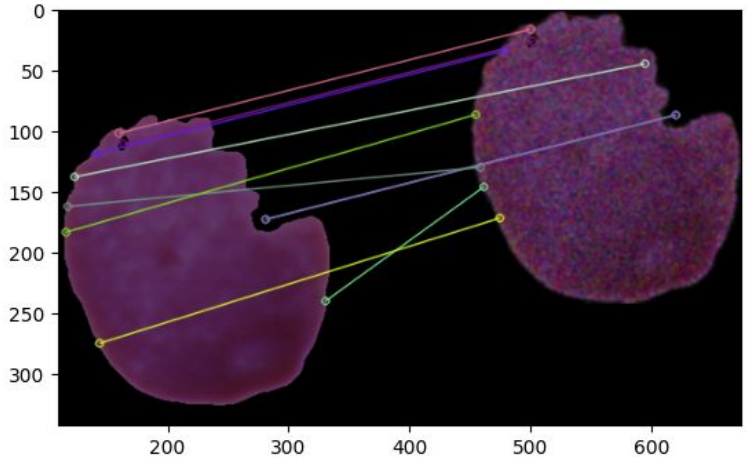
Una volta ritagliata l'immagine viene applicato un **blur gaussiano** con kernel di dimensione (5,5) per trovare un compromesso tra riduzione del rumore e conservazione dei dettagli.

Questa fase non solo consente di limitare la computazione alla regione significativa dell'immagine (la cellula), ma anche di ridurne la dimensione e rendere i calcoli meno dispendiosi.

Verifica: SIFT

Le caratteristiche dell'immagine originale risultano conservate a seguito del cropping.





Feature extraction

Panoramica

Modelli di feature extraction:

- VGG19
- ResNet50
- ResNet101

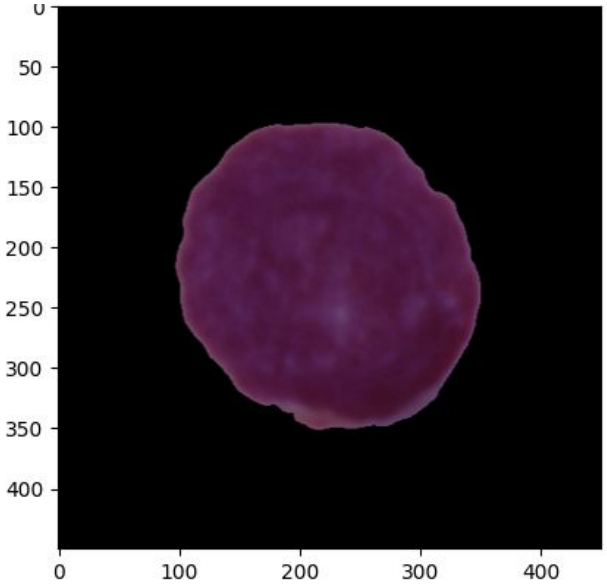
La necessità della feature extraction

L’obiettivo della **feature extraction** è quello di **rilevare** ed **estrarre** delle caratteristiche **locali e significative** in un’immagine.

Possono essere utilizzate reti neurali convoluzionali per estrarre tali feature.

Abbiamo sfruttato il **transfer learning** utilizzando modelli pre-addestrati su dataset di immagini come *imagenet* (15 milioni di immagini, 10.000 classi).

Esempio di feature extraction



feature extraction

0	1	2	3	...	2045	2046	2047
0.079193	0.0	0.11129	0.231761	...	0.038104	0.060423	0.317076

[1 rows x 2048 columns]

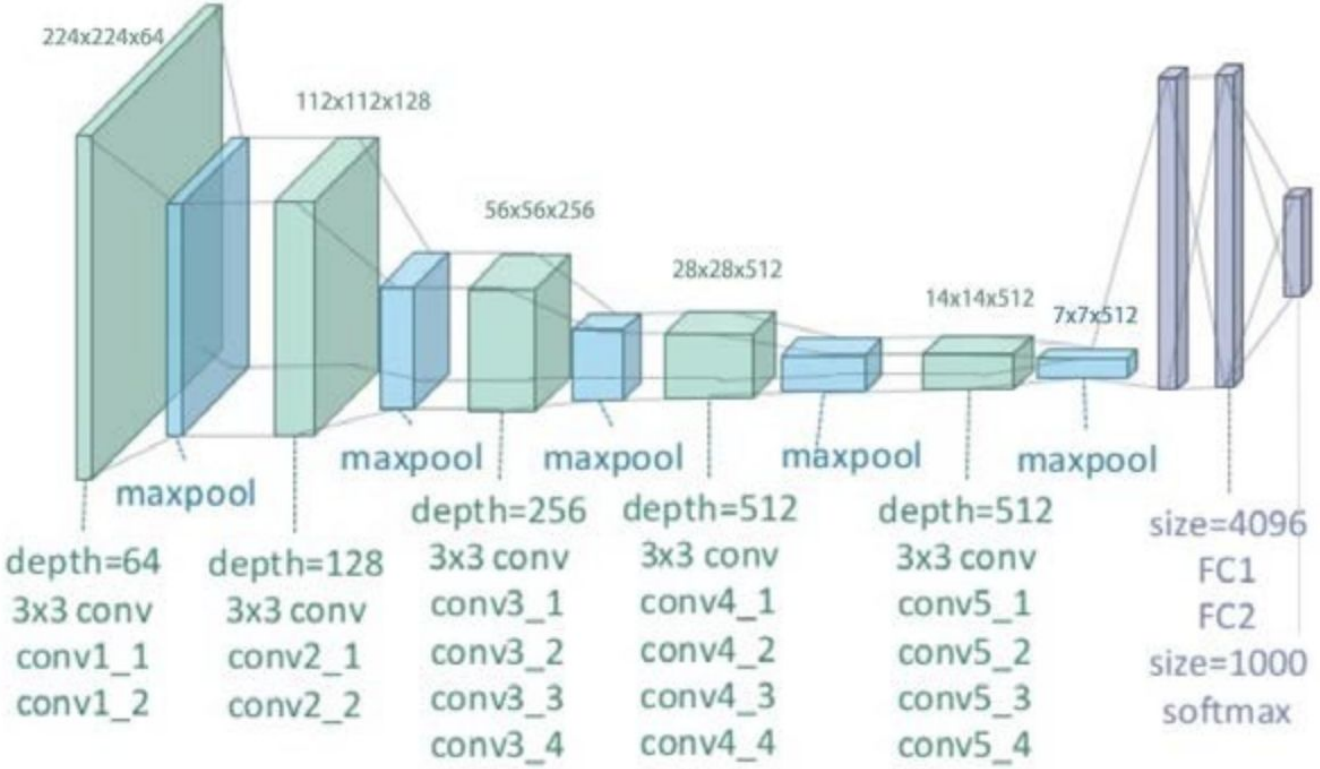
VGG19

Si tratta di una CNN con 19 layer di cui:

- 16 layer di convoluzione
- 3 layer fully connected
- 5 layer di (max) pooling
- 1 layer di output (softmax) — escluso nel transf. learning

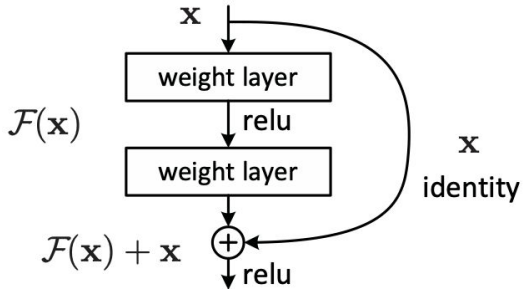
Abbiamo sfruttato il transfer learning di questo modello pre-addestrato sul dataset *imagenet*:
include_top = False
weights = “imagenet”

VGG19: architettura [3]



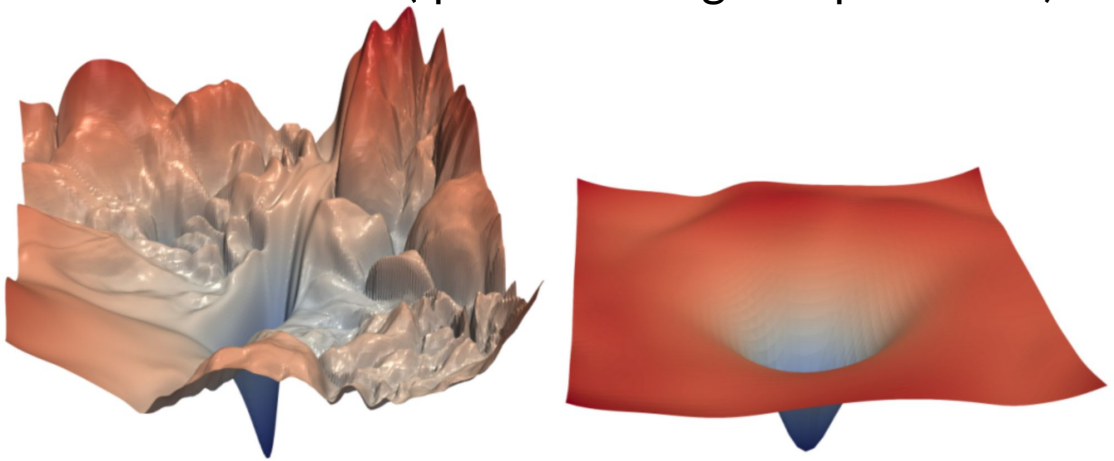
Residual Neural Networks (1)

Si tratta di reti neurali convoluzionali (estremamente) profonde che sono in grado di risolvere alcuni problemi tipici del deep learning, tra cui il vanishing e l'exploding gradient e la difficoltà nell'addestramento.



Residual Neural Networks (2)

Gli autori di ResNet hanno osservato che la superficie relativa alla funzione di errore (*loss surface*) diventa **più liscia** quando si **saltano le connessioni** (quindi convergenza più veloce).



(a) without skip connections

(b) with skip connections

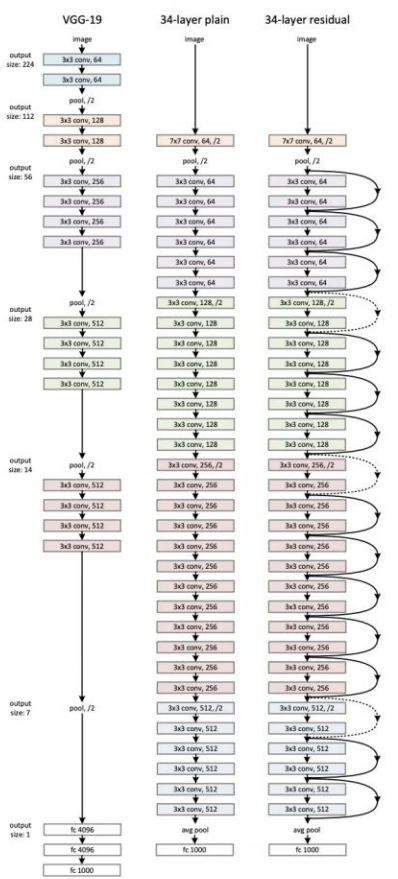


Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. **Table 1** shows more details and other variants.

ResNet50 e ResNet101

Sono residual neural network con 50 o 101 strati.

Di fianco è riportato un confronto[2] tra l'architettura VGG-19, una rete convolutiva a 34 strati e una rete residuale a 34 strati

Abbiamo sfruttato il transfer learning di questo modello pre-addestrato sul dataset *imagenet*:

include_top = False
weights = "imagenet"

Feature selection

Panoramica

Metodi di feature selection:

- Analisi della varianza (ANOVA)
- Recursive Feature Elimination (RFE)
- Random Forest Selection
- Principal Component Analysis (PCA)

La necessità della feature selection

L’obiettivo della feature selection è quello di selezionare un sottoinsieme **significativo** delle feature di partenza.

Di conseguenza i predittori diventano più efficienti in termini di spazio e/o tempo.

Inoltre limitare la classe delle ipotesi aiuta a **prevenire il rischio di overfitting**.

Bisogna trovare un **tradeoff** tra il numero di feature selezionate e le prestazioni del sistema.

Esempio di feature selection

0	1	2	3	...	2045	2046	2047
0.079193	0.0	0.11129	0.231761	...	0.038104	0.060423	0.317076

[1 rows x 2048 columns]

feature selection

116	441	948	1312	1781
0.562989	0.499496	0.49074	0.301969	0.485022

[1 rows x 5 columns]

ANOVA

Analizza la varianza delle feature rispetto alla media per trovare le *k* più **significative**.

Il calcolo della “significatività” per un problema di classificazione viene fatto attraverso la funzione ***f_classif***.

L’algoritmo si domanda “quanto una feature è capace di **discriminare** una classe piuttosto che l’altra”.

Recursive Feature Elimination (RFE)

Sfrutta uno **stimatore** che assegna a ciascuna **feature un'importanza**.

L'algoritmo si richiama **ricorsivamente** su un insieme di feature ed **elimina** da esso, ad ogni iterazione, **la feature con minore importanza**.

L'algoritmo **termina** quando la cardinalità dell'insieme di feature corrente è **uguale** al numero di feature desiderate.

Attenzione **all'overhead** dovuto alla ricorsione!

Random Forest Selection

Tecnica **simile** alla RFE.

Non è un algoritmo ricorsivo.

Seleziona una feature soltanto se la sua importanza **supera** una certa **soglia** intesa come **iperparametro**.

Tecnica basata su **ensemble learning**.

Principal Component Analysis (PCA)

Tecnica di **feature reduction**: non è detto che le feature selezionate appartengano all'insieme di feature di partenza.

Le feature sono ottenute tramite **combinazione lineare** delle feature di partenza ritenute più significative.

Algoritmo poco robusto rispetto alla **varianza intra-classe**: una stessa cellula acquisita con luce diversa avrà diversa rappresentazione. Soluzione: Linear Discriminant Analysis (LDA)

Classificatori

Panoramica

Confronto tra classificatori

- *k*-Neighbors
- Naive Bayes'
- SVM

k-Neighbors Classifier

Algoritmo di tipo **instance-based**.

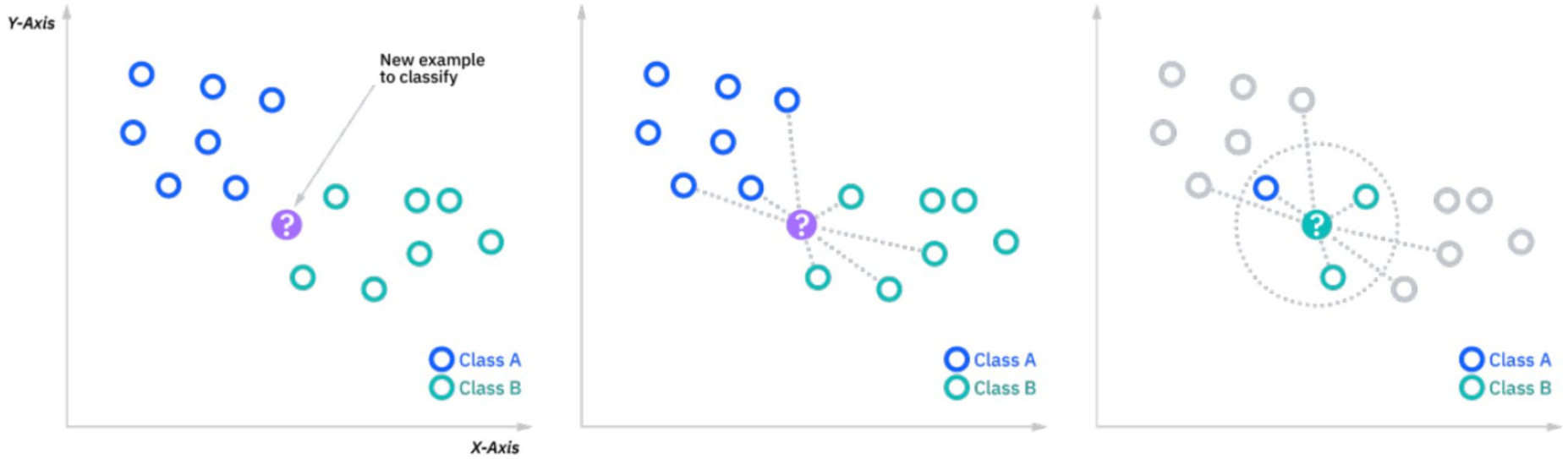
Abbiamo usato la **distanza euclidea** per calcolare la distanza tra due punti dati.

$$\sqrt{\sum_{k=1}^{n} (p_k - q_k)^2}.$$

La **grandezza del vicinato** è un iperparametro. Se troppo grande, diminuisce il rumore ma produce regioni meno distinte.

La classe dell'elemento da classificare è la classe che occorre **più volte nel suo vicinato**.

k-Neighbors Classifier [4]



Bayes' Classifier (in generale)

Un generico predittore bayesiano: $h_{Bayes}(\mathbf{x}) = \text{argmax}_k \; p(C_k \mid \mathbf{x})$

Per stimare correttamente tale **probabilità** occorre un numero esponenziale di parametri perché considera ogni combinazione delle feature e la associa alla probabilità di appartenenza ad una determinata classe.

Se occorre un numero esponenziale di feature, occorre un numero esponenziale di esempi (altrimenti rischio l'overfit).

L'assunzione **naive** fa in modo che il numero di feature (e quindi di esempi) che occorrono sia lineare.

Naive Bayes' Classifier

Un classificatore bayesiano **naive** assume che le **feature siano indipendenti tra loro**. Quindi…

$$p(\mathbf{x} \mid C_k) = \prod_{i=1}^{n} p(x_i \mid C_k)$$

$$\begin{aligned}
h_{NB}(\mathbf{x}) &:= \operatorname{argmax}_k \; p(C_k \mid \mathbf{x}) && \text{predittore bayesiano generico} \\
&= \operatorname{argmax}_k \; \frac{p(C_k)\; p(\mathbf{x} \mid C_k)}{p(\mathbf{x})} && \text{teorema di Bayes} \\
&= \operatorname{argmax}_k \; p(C_k)\; p(\mathbf{x} \mid C_k) && \text{il denominatore non dipende dalle classi} \\
&= \operatorname{argmax}_k \; p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k) && \text{assunzione naive}
\end{aligned}$$

La verosimiglianza di una feature rispetto alla classe C_k può essere campionata dalla distribuzione gaussiana (GaussianNB).

SVM Classifier

Il compito del classificatore a SVM è quello di individuare un **iperpiano** $\mathbf{wx} - b = 0$ che **separi** le istanze di cellule malate da quelle sane.

L'**etichetta predetta** dal classificatore sarà $\hat{y} = sign(\mathbf{wx} - b)$

Occorre quindi **determinare** la coppia $(\mathbf{w}^*, b^*) \in \mathbb{R}^d \times \mathbb{R}$ ottimale.

Per farlo risolve un **problema di ottimizzazione**: tra gli infiniti iperpiani che separano il dataset (nel caso realizzabile), si sceglie quello che ha il **margine** più ampio tra le istanze malate da quelle sane: si è visto generalizzare meglio.

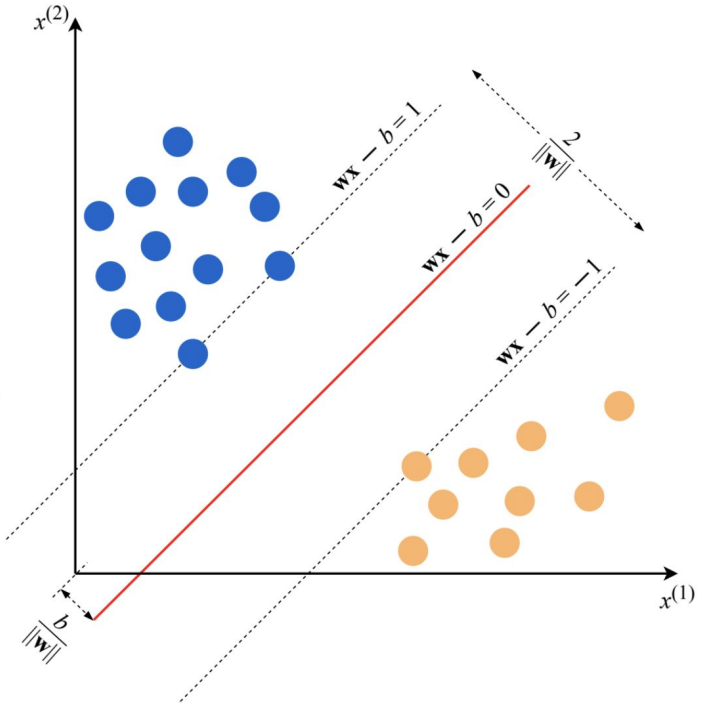
Hard-margin SVM

Per **massimizzare il margine** occorre minimizzare la norma $||\mathbf{w}||$

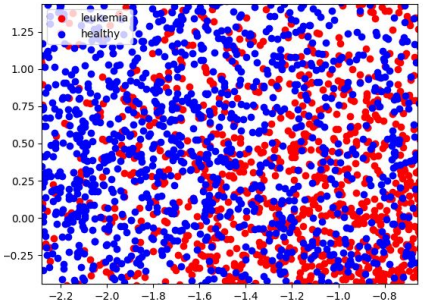
Quindi

$$\min ||\mathbf{w}||$$
$$\text{s.a. } y_i(\mathbf{wx_i} - b) \geq 1 \quad \forall 1 \leq i \leq n$$

Questo è un classificatore **lineare** e potrebbe non essere adatto a molti problemi.



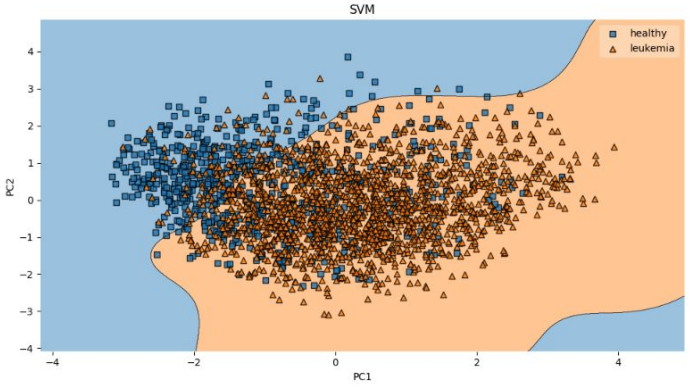
Soft-Margin SVM

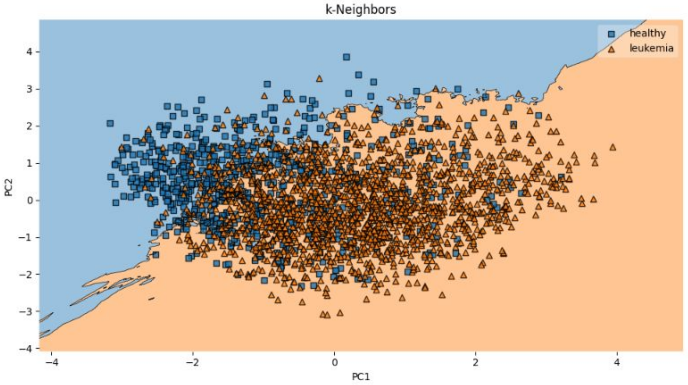


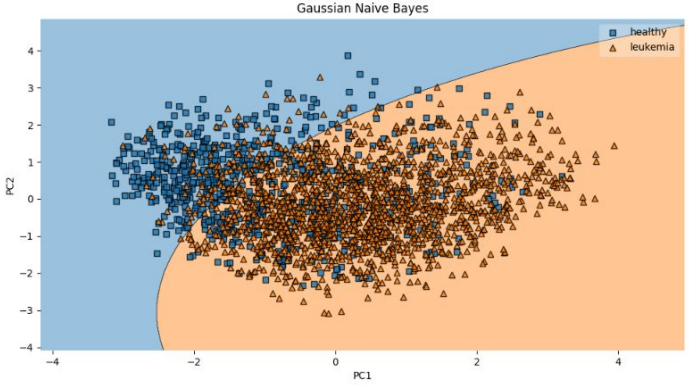
Il dataset non è linearmente separabile in due/tre dimensioni. Potrebbe aver senso fare un tuning degli iperparametri tra cui:

- **Fattore di regolarizzazione *C***: regola il tradeoff tra classificare correttamente i dati di training (ERM, C alto, margine più piccolo) e classificare correttamente i dati futuri (generalizzazione, C basso, margine più ampio). Il cosiddetto *bias-variance tradeoff*.
- Il **kernel**: regolo il kernel (*poly*, *rbf*…) per gestire casi di *inherent nonlinearity* (applicando il *kernel trick*)
- Il **coefficiente *gamma*** del kernel

Visualizzazione decision boundary







Setup sperimentale

Panoramica

Grid Search e Stratified *k*-fold Cross Validation per

- *k*-Neighbors Classifier
- Naive Bayes' Classifier
- SVM Classifier

Considerazioni sperimentali

Si ricorda che il dataset ***non* è bilanciato**: l'analisi della sola accuracy *non* può bastare.

Abbiamo scelto di studiare congiuntamente **precision** e **recall** attraverso la **media armonica F1** restituendo, a seguito della ricerca esaustiva a griglia **Grid Search**, il classificatore che la massimizza (tenendo conto anche della deviazione standard). Una recall alta, a parità di TP, ci garantisce un numero esiguo di FN.

Sarebbe stato possibile applicare anche la **Random Search** al posto di quella a griglia: si è osservato empiricamente che funziona altrettanto bene.

k-Neighbors Classifier

Ogni classificatore basato su ***k*-Neighbors** è stato addestrato eseguendo una **Grid Search** sull'iperparametro *n_neighbors* ∈ [5;30), eseguendo una **Stratified 5-fold Cross Validation**. È stato scelto il modello con miglior valore di **F1**, facendone il **refit**.

```
def knn(x, y, cv=5):
    param_grid = {"n_neighbors": list(range(5, 30))}
    grid = GridSearchCV(
                KNeighborsClassifier(), param_grid,  cv=cv,
                scoring=("accuracy", "precision", "recall", "f1"),
                refit="f1")
    grid.fit(x, y)
    return grid.best_estimator
```

Naive Bayes' Classifier

Ogni classificatore **Naive Bayes** è stato addestrato facendo un **Grid Search** sull'iperparametro *var_smoothing* $\in \{10^{-9}, 10^{-8}, 10^{-7}\}$. Ogni istanza è stata validata con una **Stratified 5-fold Cross Validation**, scegliendo il modello con miglior valore di **F1**, facendone il **refit**.

```
def naive_bayes(x, y, cv=5):
    param_grid = {"var_smoothing": [10 ** -9, 10 ** -8, 10 ** -7]}
    grid = GridSearchCV(
                GaussianNB(), cv=cv, param_grid=param_grid,
                scoring=("accuracy", "precision", "recall", "f1"),
                refit="f1")
    grid.fit(x, y)
    return grid.best_estimator_
```

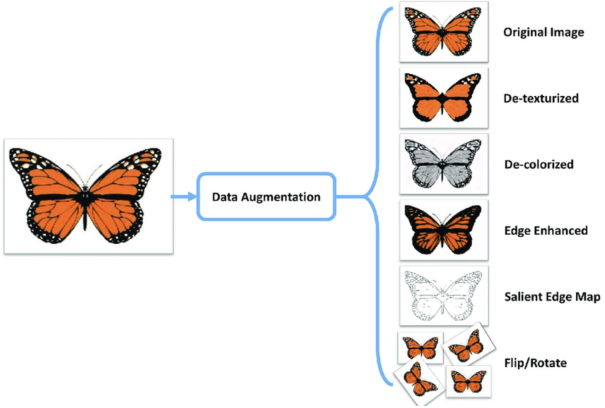
SVM Classifier

Ogni classificatore **SVM** è stato addestrato facendo una **Grid Search** sugli iperparametri $C \in \{10^{-2}, 10^{-1}, 1, 10\}$ e *kernel* $\in$ {‘*linear*’, ‘*rbf*’, ‘*poly*’}. Ogni istanza è stata validata con una **Stratified 5-fold Cross Validation** scegliendo il modello con miglior valore di **F1**, facendone il **refit**.

```
def svc(x, y, cv=5):
    param_grid = { 'C': [0.01, 0.1, 1, 10],
                   'kernel': ['linear', 'rbf', 'poly'],
                   'gamma': ['auto']}
    grid = GridSearchCV(
                  svm.SVC(), param_grid,  cv=cv,
                  scoring=("accuracy", "precision", "recall", "f1"),
                  refit="f1")
    grid.fit(x, y)
    return grid.best_estimator_
```

Considerazioni sperimentali (2)

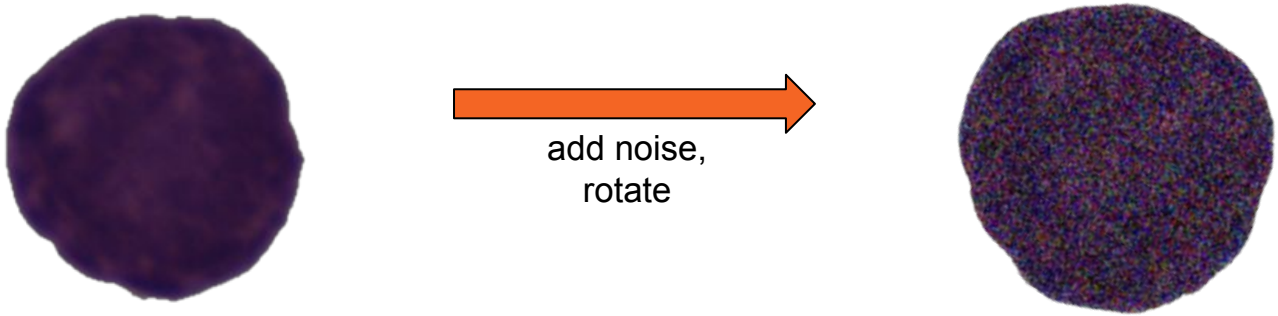
Sarebbe stato possibile, per cercare di bilanciare il dataset, applicare la **data augmentation** gonfiando il training set con dati fittizi (ad esempio aggiungendo del rumore alle immagini);



è fondamentale, tuttavia, che nel test set ci siano *solo* dati reali! Può essere vista come una forma di regolarizzazione.

Siccome ci sono più cellule malate che sane, si potrebbe pensare di fare data augmentation su quelle sane.

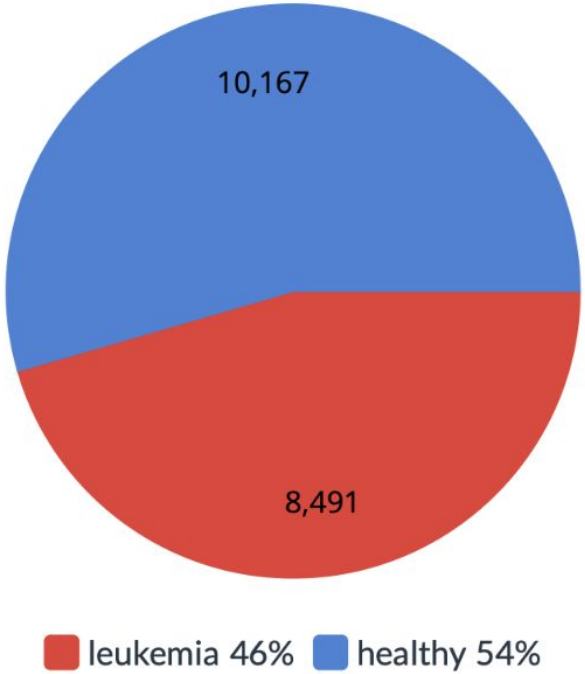
Considerazioni sperimentali (3)



Considerazioni sperimentali (4)

Il dataset può diventare pressoché bilanciato se decidiamo di applicare **data augumentation** alle immagini di **cellule sane** usate per l'**addestramento**.

Può essere utile, in quest'ottica, generare campioni con aggiunta di **rumore gaussiano** e **rumore sale e pepe**.

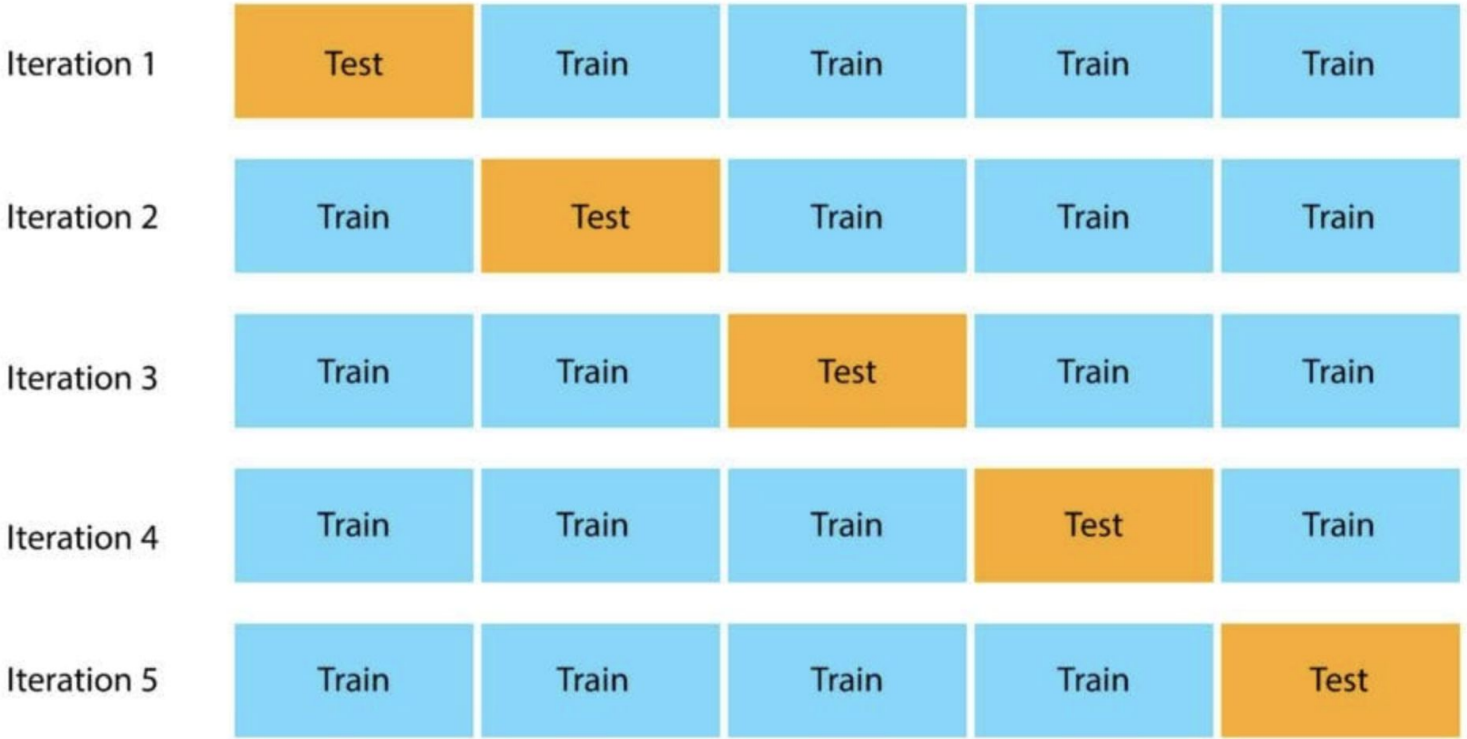


Considerazioni sperimentali (4)

La ***k*-fold Cross Validation** è stata usata per avere, fissati gli iperparametri di un classificatore, un'unica misura di prestazioni statisticamente più significativa di quella che avrei con la tecnica **Hold-Out**. Ottengo, di conseguenza, *k* modelli distinti in termini di parametri (non iperparametri). La misura ultima che esprime la valutazione del modello sottoposto al *k*-fold CV la si può ottenere tramite una media sperimentale ed una deviazione standard rispetto ad essa.

Abbiamo scelto di utilizzare, in particolare, la versione **Stratified** per garantire che, ad ogni iterazione, ciascun fold sia rappresentativo per il problema.

k-fold Cross Validation



Analisi sperimentale

Panoramica

- Confronto feature extractor
- Confronto classificatori
- Presentazione del miglior classificatore

Test di ipotesi (1)

I **modelli di machine learning** vengono **confrontati** in base alle loro **prestazioni** (tipicamente ottenute applicando un *k*-fold CV).

Nel nostro caso, per bilanciare tra precision e recall, abbiamo detto che se un classificatore A ha una prestazione F1 più alta di B, allora A è meglio di B. Ma chi garantisce che **la differenza tra le prestazioni** di A e B sia **statisticamente significativa** e non dettata dalla fortuna?

La soluzione è utilizzare un **test di ipotesi** per verificare se la **differenza** tra le due prestazioni è **reale oppure no**.

Test di ipotesi (2)

Si definisce un'**ipotesi nulla** (null hypothesis) = **A e B hanno stesse prestazioni** (le prestazioni di A e B sono campionate dalla stessa distribuzione di probabilità).

Definisco una **soglia di accettabilità α** (tipicamente 0.05).

Calcolo la ***p***-value: **se è minore o uguale di α**, allora la differenza tra le prestazioni di A e B è reale (**rigetto l'ipotesi nulla**); **altrimenti** A e B sono equivalenti (**accetto l'ipotesi nulla**).

Ovviamente è auspicabile **rigettare l'ipotesi nulla**.

Confronto metodi di estrazione (1)

Supponiamo di voler sapere, al **variare del metodo di feature extraction**, qual è il miglior classificatore in termini di F1. Tramite un'analisi sperimentale concludiamo che…

Per **VGG19** otteniamo:

- 350 feature selezionate da RFE
- Classificatore: *SVC(C=1, kernel='linear')*

Per **ResNet50** otteniamo:

- 100 feature selezionate da PCA
- Classificatore: *SVC(C=10, kernel='rbf')*

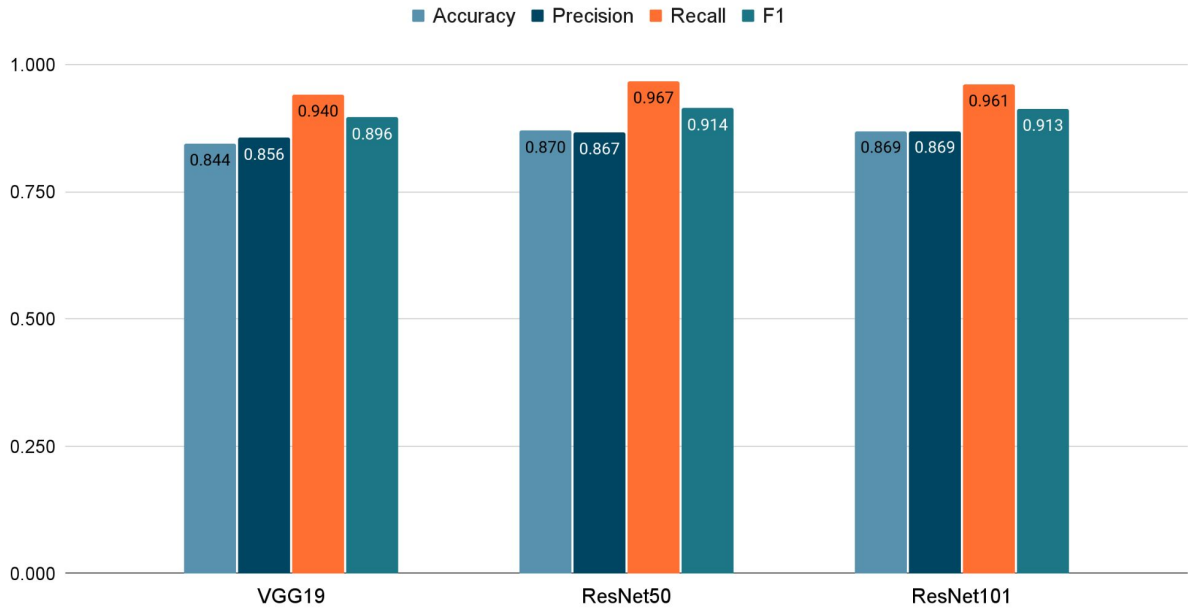
Per **ResNet101** otteniamo:

- 400 feature selezionate da PCA
- Classificatore: *SVC(C=0.1, kernel='linear')*

Confronto metodi di estrazione (2)

Prestazione dei migliori classificatori, in termini di F1, ottenuti al variare del metodo feature extraction:

	VGG19	ResNet50	ResNet101
Accuracy	0.844 ± 0.009	0.870 ± 0.003	0.869 ± 0.004
Precision	0.856 ± 0.007	0.867 ± 0.003	0.869 ± 0.004
Recall	0.940 ± 0.004	0.967 ± 0.001	0.961 ± 0.003
F1	0.896 ± 0.006	0.914 ± 0.002	0.913 ± 0.003



Confronto classificatori (1)

Supponiamo di voler sapere, **in assoluto**, quali sono i **tre** migliori classificatori (uno per ogni tipologia cioè SVM, kNN e NB) in termini di F1:

- *SVM(C=10, kernel='rbf', gamma='auto')*: dopo aver estratto le caratteristiche con ResNet50 ed averne selezionate 100 con PCA

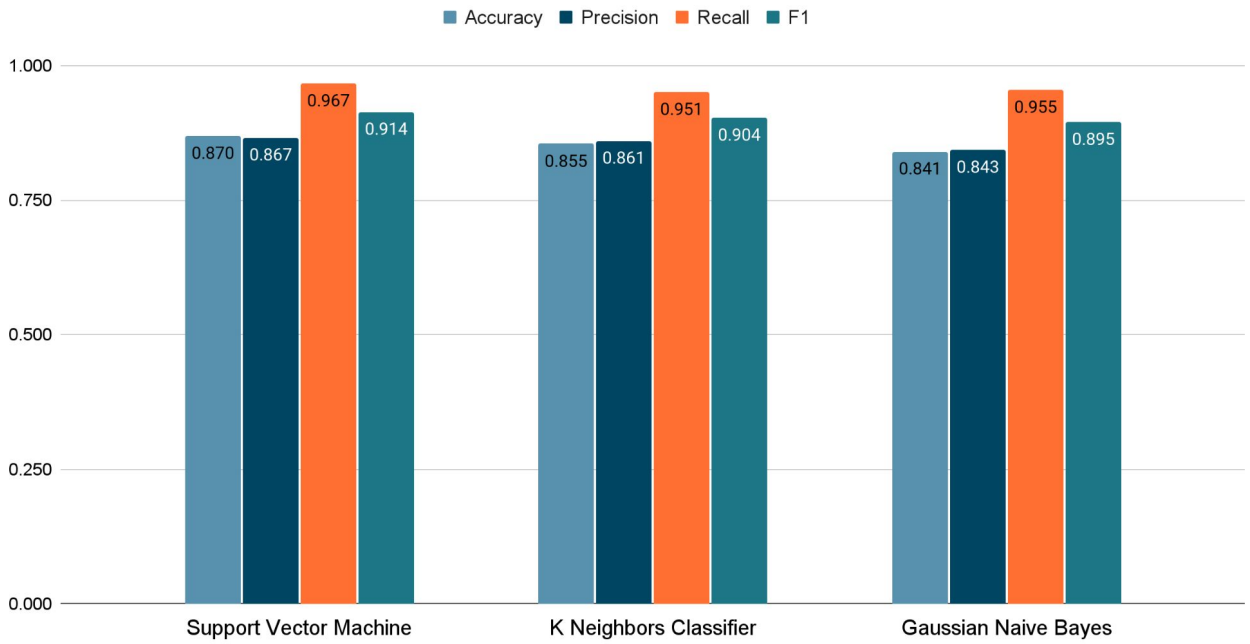
- *k-Neighbors(n_neighbors=13)*: dopo aver estratto le caratteristiche con ResNet50 ed averne selezionate un set appropriato tra le 400 più significative, sfruttando 200 decision tree (ensemble learning - formano una Random Forest)

- *GaussianNB(var_smoothing=*10^{-9}*)*: dopo avere estratto le caratteristiche con ResNet50 ed averne selezionate le 250 più significative con PCA

Confronto classificatori (2)

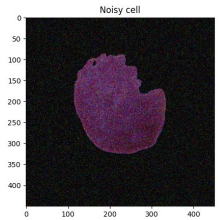
Prestazione dei migliori classificatori in termini di F1:

	SVM	k-Neighbors	GNB
Accuracy	0.870 ± 0.003	0.855 ± 0.004	0.841 ± 0.014
Precision	0.867 ± 0.003	0.861 ± 0.006	0.843 ± 0.012
Recall	0.967 ± 0.001	0.951 ± 0.005	0.955 ± 0.005
F1	0.914 ± 0.002	0.904 ± 0.003	0.895 ± 0.009



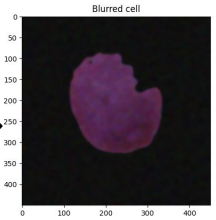
Il miglior classificatore: SVM

Il **miglior classificatore**, in termini di F1, è ***SVM****(C=10, kernel='rbf', gamma='auto')*, dopo aver estratto le feature con **ResNet50** ed averle ridotte alle 100 più significative con **PCA**.



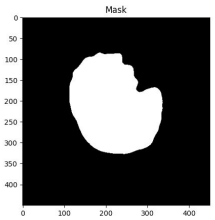
Input, RGB cell
(450, 450, 3)

blur



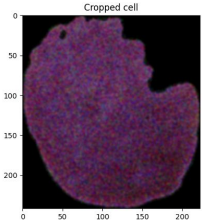
Blurred RGB cell
(450, 450, 3)

threshold



Gray-scale Mask
(450, 450, 1)

crop



Cropped RGB cell
(242, 223, 3)

Feature Extraction
(ResNet50)

$\mathbf{f} = [f_1, f_2, \ldots, f_{2048}]$

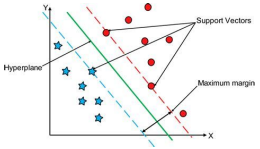
Features
(1, 2048)

Feature
Reduction
(PCA)

$\mathbf{f_s} = [\bar{f}_1, \bar{f}_2, \ldots, \bar{f}_{100}]$

Selected Features
(1, 100)

input



Support Vector
Classifier

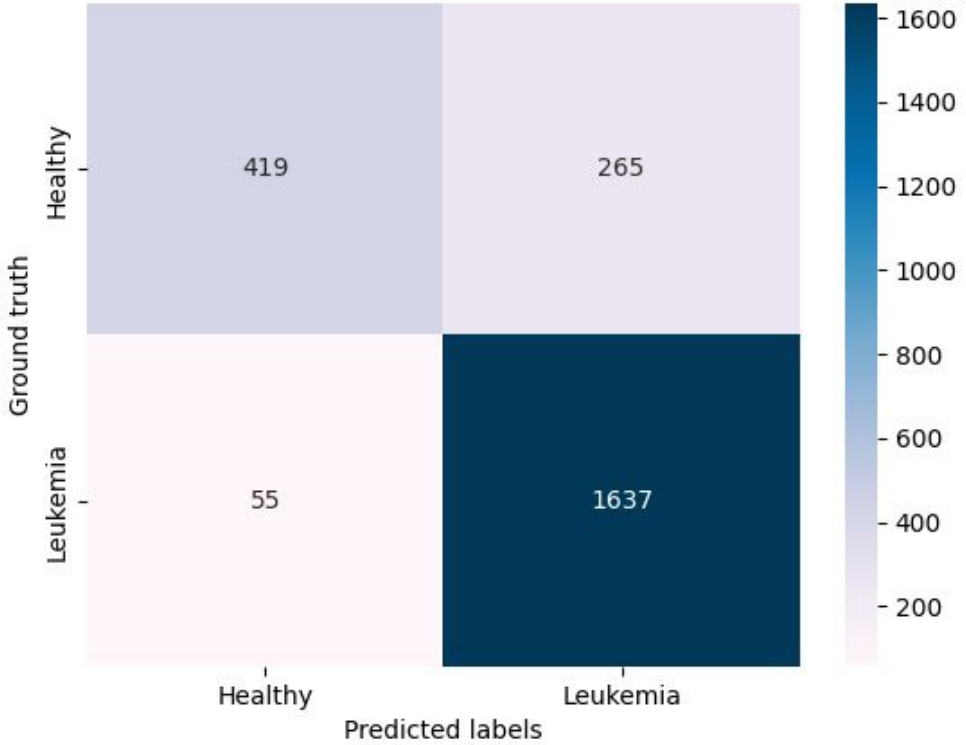
output

$y \in \{1, 0\}$

La matrice di confusione

Metrica	Media ± std
Accuracy	0.870 ± 0.003
Precision	0.867 ± 0.003
Recall	0.967 ± 0.001
F1	0.914 ± 0.002

test_size = 0.2



Bibliografia

[1]: *Leucemia Linfoblastica Acuta*, AIRC, 13/03/2023
https://www.airc.it/cancro/informazioni-tumori/guida-ai-tumori/leucemia-linfoblastica-acuta

[2]: *Deep Residual Learning for Image Recognition*, Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, arXiv:1512.03385 [cs.CV]

[3]: *Breast cancer screening using convolutional neural network and follow-up digital mammography*, Yufeng Zheng, Clifford Yang, Alex Merkulov, Computational Imaging III, DOI:10.1117/12.2304564

[4]: *What is the k-nearest neighbors algorithm?*, IBM, 17/03/2023
https://www.ibm.com/topics/knn