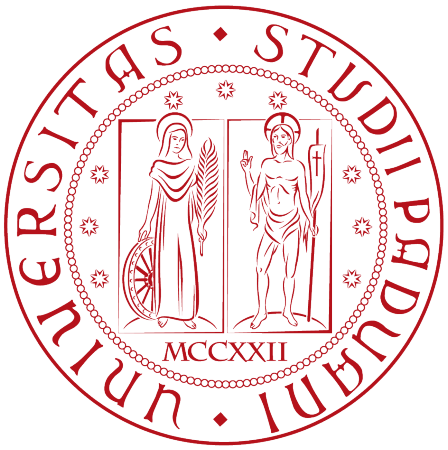
Università degli Studi di Padova




DIPARTIMENTO
MATEMATICA

Corso di Laurea Triennale in Informatica

# Progetto di Machine Learning

**Studente**

Bile E. Christian Prince Carlos
1096677

**Professore**

Aiolli Fabio

# Indice

## Obiettivo

Questo progetto ha lo scopo di risolvere un problema di classificazione dei membri della famiglia Simpson. Cercherò di trovare un classificatore con le migliori performance utilizzando 2 algoritmi di apprendimento: il Multiple Layer Perceptron e il Support Vector Machine. L'apprendimento avverrà impostando diversi parametri di apprendimento. Infine si andrà alla scoperta di un'altra tecnica di image classification, il Convolutional Neural Network, e si farà un confronto finale dei risultati ottenuti.

## Dataset

I Simpson è una popolare sitcom animata statunitense, creata nel 1987. La serie è una parodia satirica della società e dello stile di vita statunitense, impersonificati dalla famiglia Simpson, protagonista dell'opera, composta da Homer e Marge e dai loro tre figli Bart, Lisa e Maggie. Ambientato in una cittadina statunitense chiamata Springfield, il cartone tratta in chiave umoristica molti aspetti della condizione umana, tra cui la cultura, la società e la stessa televisione.
Il dataset utilizzato per questo progetto è un insieme di immagini tratti da diverse puntate andate in onda, ed è stato reso disponibile da un membro attivo della communità Kaggle, al seguente link:

`https://www.kaggle.com/alexattia/the-simpsons-characters-dataset`

Il dataset contiene per ora 20 934 immagini, in formato jpeg, di 47 personaggi della serie TV da classificare. Date le limitate capacità computazionali e di memoria RAM del mio calcolatore, mi limiterò alla sola classificazione dei membri della famiglia Simpson composta da 6 personaggi ( Abraham, Homer, Marge, Maggie, Lisa e Bart), per un totale di 7274 immagini di training e 300 immagini di test.

## Strumenti, script e Librerie utilizzate

Per questo progetto è stato utilizzato il linguaggio di programmazione Python 3.6, con inoltre le seguenti librerie:

- Scikit Learn: `http://scikit-learn.org/`
- Scikit Image: `http://scikit-image.org/`
- OpenCv: `https://opencv.org/`
- Numpy: `http://www.numpy.org/`
- Pandas: `https://pandas.pydata.org/`
- Matplotlib: `https://matplotlib.org/`
- tqdm: `https://github.com/tqdm/tqdm`

- Keras: `https://keras.io/`

Insieme a questa relazione sono stati consegnati degli script python che ho scritto. Quest'ultimi rappresentano l'applicazione che avvia gli algoritmi di apprendimento e ne visualizza i risultati. Questi file sono:

- **main.py:** Script principale da eseguire per avviare l'applicazione;
- **classification.py:** Modulo che contiene classi di oggetti che, dati un Training set, un Test set, un preprocessore e un algoritmo di apprendimento, apprende, visualizza e restituisce i risultati;
- **proprocessing.py:** Modulo che contiene classi di oggetti stream che, dato un dataset di tipo *list* multidimensionale, e impostati i vari passi di preprocessing, preprocessano i dati e restituiscono i risultati;
- **fileloaders.py:** Modulo che contiene classi di oggetti che caricano file e li restituiscono in forma matriciale o vettoriale;
- **ploting.py:** Modulo che contiene funzioni di ploting di dati;
- **console.py:** Modulo che contiene funzioni di printing su terminale di messaggi particolari come il passo di esecuzione, il tempo trascorso ecc..;

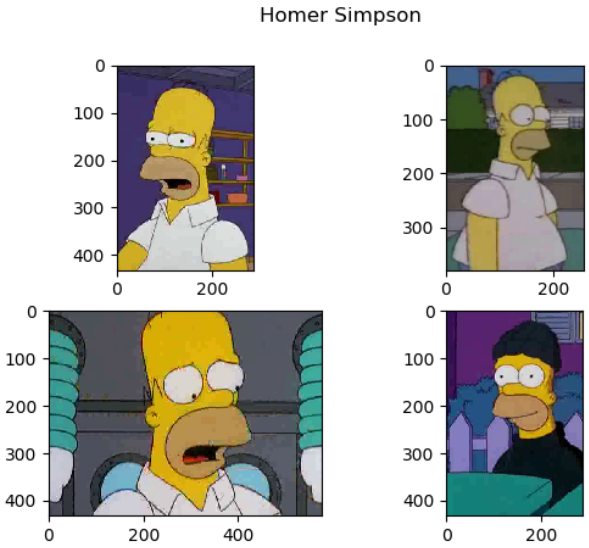
Gli algoritmi sono stati eseguiti sul mio calcolatore che ha le seguenti caratteristiche:

```
Sistema operativo: Windows 10
Processore: Intel Core i7-6700HQ, x64 based
Ram: 16 GB
```

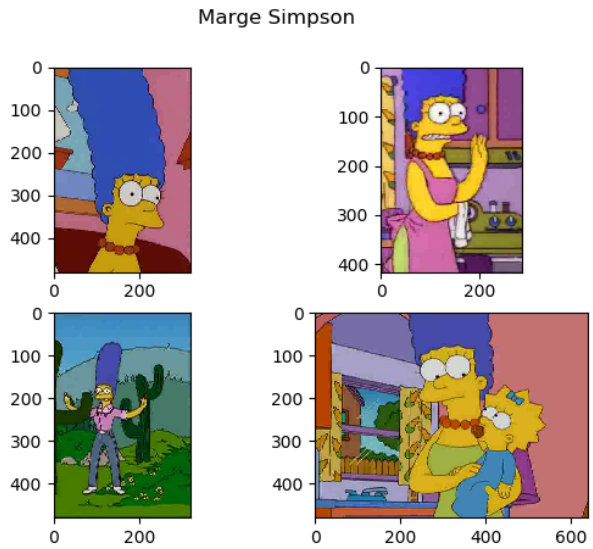
### Analisi dei dati

Il training set si compone dunque di 7274 istanze. Le classi sono:

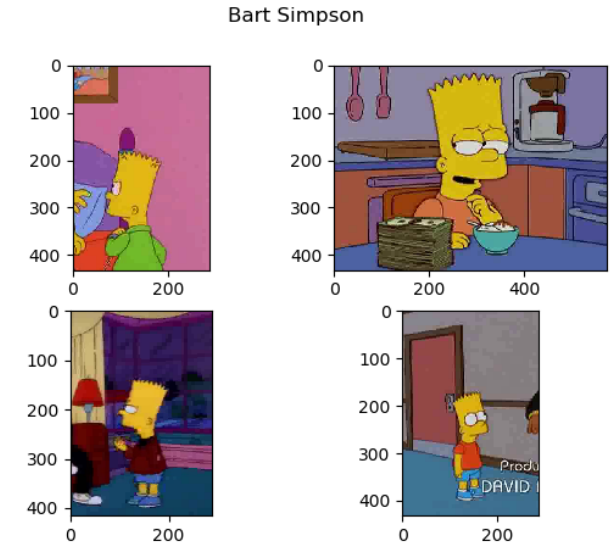
- *homer_simpson*: La classe delle immagini che contengono il personaggio Homer Simpson.



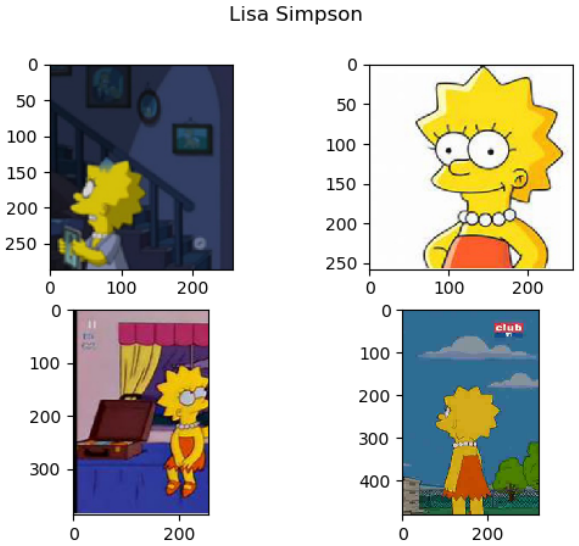

- *marge_simpson*: La classe delle immagini che contengono il personaggio Marge Simpson, moglie di Homer.



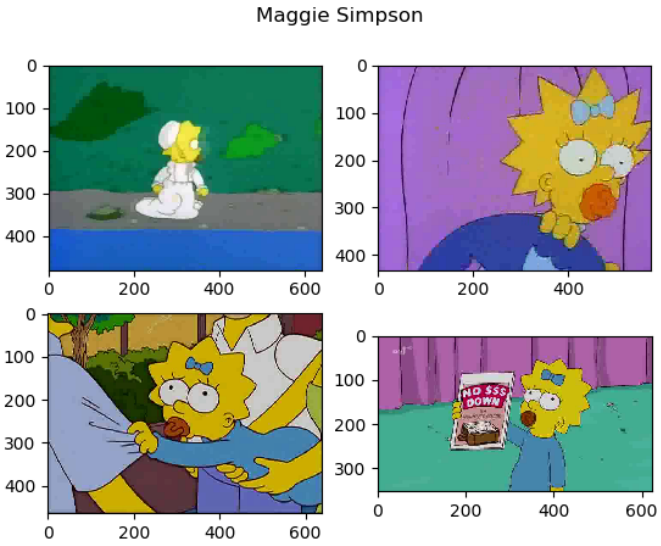

- *bart_simpson*: La classe delle immagini che contengono il personaggio Bart Simpson, figlio di Homer e Marge Simpson;



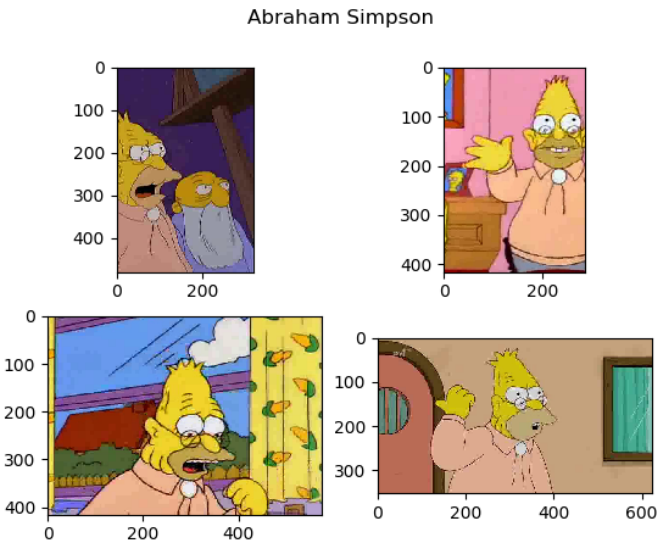

- *lisa_simpson*: La classe delle immagini che contengono Lisa Simpson, figlia di Homer e Marge Simpson;



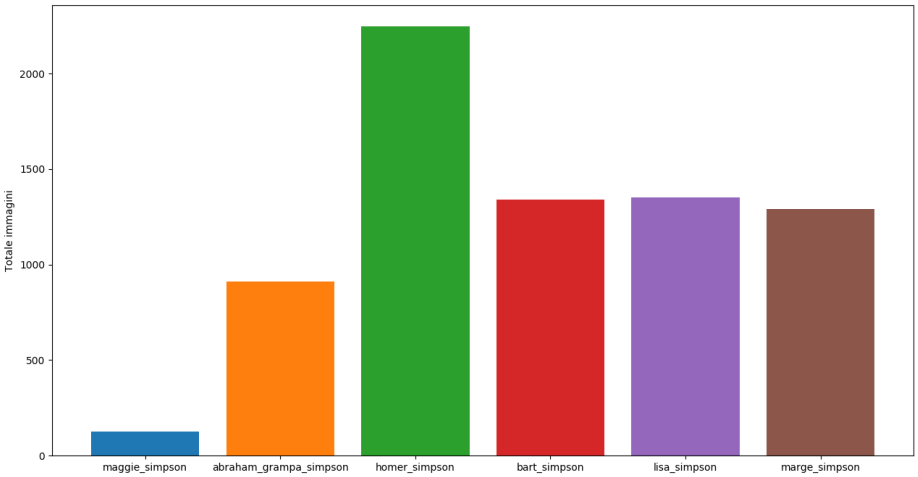

- *maggie_simpson*: La classe delle immagini che contengono Maggie Simpson, figlia di Homer e Marge Simpson.

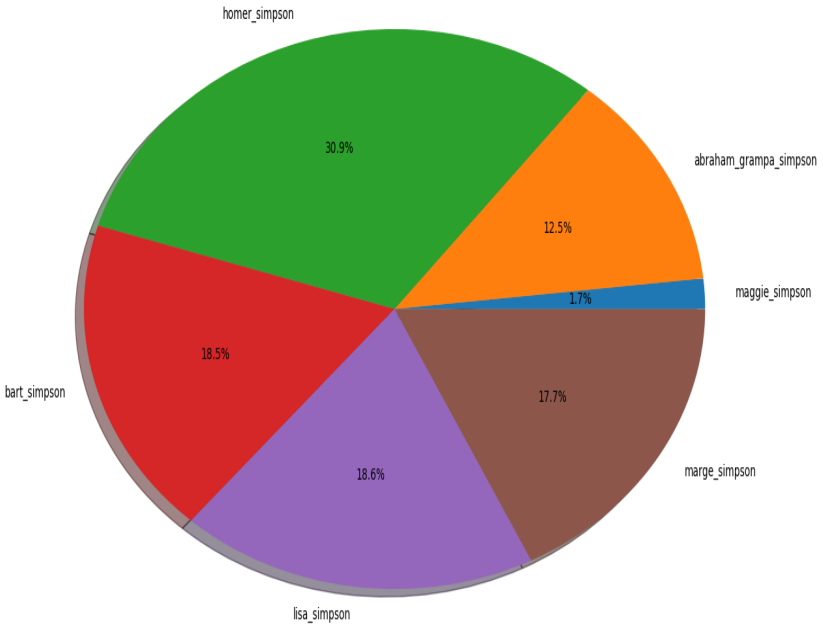



- *abraham_grampa_simpson*: La classe delle immagini che contengono il personaggio Abraham Simpson, padre di Homer Simpson.




Possiamo subito notare che Lisa Simpson assomiglia tantissimo a Maggie Simpson, questa somiglianza potrebbe portare ad un classificatore che li confonderà. Inoltre le classi non risultano bilanciate e lo si osserva nei seguenti diagrammi:


Totale immagini
2000
1500
1000
500
0
maggie_simpson
abraham_grampa_simpson
homer_simpson
bart_simpson
lisa_simpson
marge_simpson


homer_simpson
30.9%
abraham_grampa_simpson
12.5%
maggie_simpson
1.7%
bart_simpson
18.5%
17.7%
marge_simpson
18.6%
lisa_simpson

Le istanze *homer_simpson* sono le più numerose nel dataset e questo porterebbe all'apprendimento di un classificatore più performante nella classificazione delle immagini contenenti Homer Simpson. Il discorso inverso vale per la classe meno popolata *maggie_simpson*, l'algoritmo potrebbe restituire un classificatore poco performante su quella classe. Di conseguenza l'accuratezza come misura di valutazione non è applicabile, ma altre misure come la precision, la recall e l' $F_\beta$-score sono da preferire. Per mitigare gli effetti del problema dei dati non bilanciati, posso optare per almeno una delle seguenti soluzioni:

1. **Raccogliere più dati per la classe** *maggie_simpson***:** Richiederebbe troppo tempo;

2. **Ricampionare i dati:** aggiungendo copie di istanze della classe *maggie_simpson* (Oversampling), soluzione non fattibile per la memoria limitata del mio calcolatore. Inoltre per classi molto piccole di cardinalità, come è il caso della classe *maggie_simpson*, c'è il rischio di andare a fare test sugli stessi dati di apprendimento se si addotta delle tecniche di model selection come il Hold-out o il Cross Validation;

3. **Provare diverse tecniche di apprendimento:** In questo progetto userò il Multiple Layer Perception, il Support Vector Machine e il CNN, per poi confrontare i risultati ottenuti;

4. **Usare metodi di penalizzazione:** In modo da avere meno overfitting sulle classi più popolate ed avere un test error che assomiglia sempre di più a classificazioni verso le classi meno popolate, ad esempio la classe *maggie_simpson*. Tuttavia se i dati contengono tanto rumore questa tecnica rischia di non funzionare e di avere un test error elevato causato da un errore empirico elevato perché avremo overfitting su dati non caratteristici;

5. **Procedere senza la classe** *maggie_simpson*

E' difficile applicare il principio di Structural Risk Minimization con la classe *maggie_simpson* che un ha numero di osservazioni troppo poco (circa 130 immagini), percui opterò per la soluzione 5, che è quella di procedere senza quella classe.
Altri problemi che sono sorti in questa fase di analisi di dati, sono le dimensioni delle immagini che variano tantissimo, e i rumori presenti nelle immagini( altri personaggi o oggetti che non caratterizzano il personaggio da classificare, che causano confusione che mena a risultati non desiderabili). Le dimensioni delle immagini possono essere tutte fissate ad un certo valore in fase di preprocessing, stando attenti a non scegliere un valore che faccia perdere troppa informazione. Per quanto riguarda il rumore, una soluzione potrebbe essere di scegliere un algoritmo di apprendimento robusto, oppure di impostare l'output come probabiltà di presenza del personaggio piuttosto che restituire l'etichetta esatta.

## Feature Detection

Per poter classificare un insieme di immagini che contengono un personaggio, abbiamo bisogno di identificare delle caratteristiche comuni a queste immagini. Queste caratteristiche devono essere non confondenti e facili da rintracciare. Quindi una caratteristica candidata deve sempre sembrare diversa dalle altre carateristiche quando essa si presenta in qualsiasi punto dell'immagine. La ricerca di queste caratteristiche uniche e comuni a tutte le immagini di una classe, viene detta **Feature Detection**. Una volta identificate le caratteristiche, bisogna dare una loro definizione comprensibile al calcolatore in modo che le possa rintracciare in altre immagini. Tali definizioni vengono chiamate **Feature Descriptors**. Esistono tanti algoritmi di Feature Detection, come ad esempio gli algoritmi: SIFT(Scale-Invariant Feature Transform), HCD(Harris Corner Detection), SURF (Speeded-Up Robust Features) ecc..., che si differenziano per le caratteristiche che si vuole descrivere. Per questo progetto userò il Feature Descriptor HOG(Histogram of Oriented Gradients).

### HOG (Histogram of Oriented Gradients)

I personaggi della serie TV Simpson sono caratterizzati da forme ben precise del viso, capelli, corpo ecc... . Queste caratteristiche possono essere utilizzate per classificarli piuttosto che usare i valori RGB(Red Green Blue) che sono confondenti. Per tali motivi è necessario sostituire le immagini con un descrittore della forma di un oggetto.
HOG è un descrittore di caratteristiche usato per il riconoscimento di oggetti. La tecnica conta le occorrenze dell'orientamento del gradiente in porzioni localizzate di una immagine. L'immagine è divisa in piccole regioni collegate chiamate celle e per i pixel all'interno di ciascuna cella viene computato un istogramma di direzioni del gradiente. Il descrittore è la concatenazione di questi istogrammi. Per una maggiore precisione, gli istogrammi locali possono essere normalizzati mediante il contrasto calcolando una misura dell'intensità su una regione più ampia dell'immagine, chiamata blocco, e quindi utilizzando questo valore per normalizzare tutte le celle all'interno del blocco. Questa normalizzazione si traduce in una migliore invarianza ai cambiamenti di illuminazione e ombreggiamento. Maggiori informazioni sono disponibili ai seguenti link:

- `https://www.learnopencv.com/histogram-of-oriented-gradients/`
- `https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients`

Per poter applicare l'HOG alle immagini, bisogna inanzitutto ridimensionare le immagini in modo tale da non perdere troppa infomazione, tipicamente si fissa un ratio, ad esempio nel mio caso scelgo 1:1, e in base a queste ratio si valutano diverse dimensioni fisse delle immagini che non fanno perdere troppa informazioe. Poi si sceglie la size ideale delle celle per catturare le features, e infine si fa un'eventuale normalizzazione scegliendo la size dei blocchi che servono a ridurre la sensibiltà a variazioni di luce. Inoltre nel calcolo dell'HOG feature descriptor di un'immagine, è spesso applicata una correzione gamma di valore maggiore o uguale ad 1 per evidenziare al meglio le parti di maggiore focus

nell'immagine, cioè le parti non ombreggiate o scure. Ciò porta spesso ad un leggero aumento delle performance.
Per questo progetto, dopo diverse prove, la scelta dei valori che mediamente descrivono meglio le feature che ricerco sono di ridimensionare le immagini a 288x288 pixels, con cells di 8x8 pixels, un cell per ogni block. Il risultato è il seguente:


Figura 1: HOG di un'immagine contenente Homer Simpson

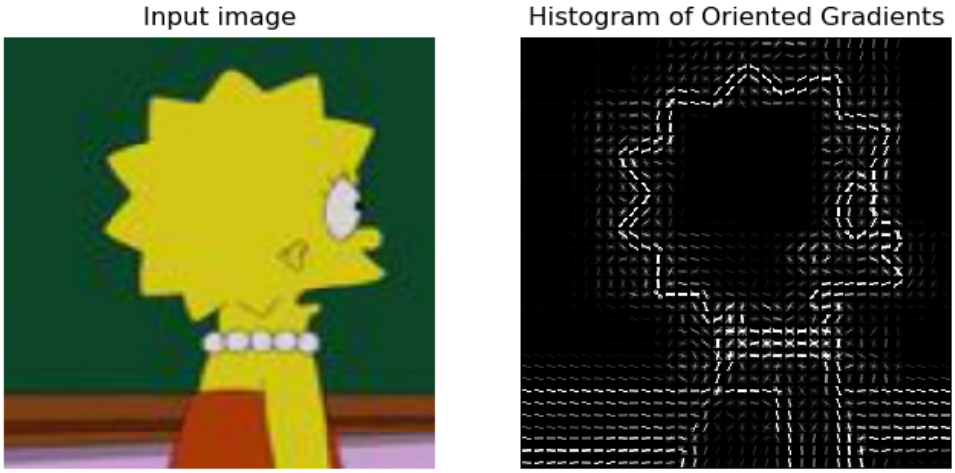



Figura 2: HOG di un'immagine contenente Lisa Simpson

### Procedura generale di apprendimento

Per gli algoritmi di apprendimento MLP e SVM, adotterò un procedimento commune che è il seguente:

- Carico tutti i dati di training e di test;
- Elimino i dati relativi alla classe *maggie_simpson*, in quanto ho deciso di fare apprendimento senza quella classe che non ha abbastanza dati per ora;
- Faccio un preprocessing dei dati di training e di test;
- Con il supporto della classe *StratifiedShuffleSplit* di *sklearn.model_selection*, avvio l'algoritmo di apprendimento sui dati di training con cross validation su 3 split stratificati e randomizzati, 25% dei dati di training vengono presi come dati di test ad ogni split. Anche se la randomizzazione non garantisce che i dati presenti nei fold saranno diversi, rischiando quindi di fare test sui dati di training, questa scelta di model selection è spiegata dal fatto che i dati vengono caricati in ordine di classe, perciò usare una K-Fold non garantisce stratificazione delle classi nei fold;
- Seleziono il modello che ha la migliore performance nel cross validation;
- Rialleno il modello selezionato su tutti i dati del training set senza crossvalidation;
- Faccio un test finale sui dati di test che non sono stati usati per fare apprendimento.

## Multiple Layer Perceptron classifier

Il Multiple Layer Perceptron (MLP), è una rete neurale artificiale feed forward che si compone di almeno 3 strati: il primo strato è lo strato input che si compone di tanti nodi quanti sono i valori del vettore input, l'ultimo strato è lo strato output che si compone di tanti nodi quanti sono i valori del vettore output, e infine gli strati intermedi vengono chiamati strati nascosti. L'apprendimento del classificatore avviene tramite l'apprendimento dei pesi dei valori che i neuroni ricevono in input, che minimizzano una funzione di costo, ad esempio l'errore quadratico medio. Si usano vari algoritmi per l'apprendimento, tra cui il Back-propagation, il più usato, che aggiornano i pesi su discesa del gradiente.
Gli strati nascosti aggiungono una non linearità alla funzione appresa definendo così delle superfici di decisioni più o meno complesse. Percui un numero elevato di nodi negli strati nascosti abbasserebbe il bias nei dati di apprendimento ma tuttavia aumenterebbe la varianza che causerà un aumento dell'errore ideale. Tale fenomeno è chiamato *Overfitting*, ed è spiegato dal generalization bound:

`http://www.math.unipd.it/~aiolli/corsi/1718/aa/Lez04_pac.pdf`

Il Multiple Layer Perceptron, come le altre tipologie di reti neurali multistrato, è un black box, cioè non è possibile sapere la forma della funzione appresa a scopo di interpretazione.

### Apprendimento del classificatore MLP

Il classificatore MLP che cercherò avrà quindi una certa topologia con un numero di nodi fissato per gli strati input e output, e quindi la ricerca si farà sul numero di strati nascosti e il numero di nodi per ogni strato nascosto. L'apprendimento dei pesi avverrà usando la classe MLPClassifier di Scikit-Learn, e utilizzando vari valori per gli iperparametri di regolarizzazione. Molti studi affermano che un solo strato nascosto è sufficiente per spiegare con buona approssimazione i dati, l'aggiunta di altri nascosti potrebbero migliorare l'accuratezza ma di poco. La ricerca del modello adatto avverrà con la tecnica di cross validation citata nella procedura generale di apprendimento, e nel seguente modo:

1. Parto inizialmente da un solo strato nascosto e 50 nodi in quello strato;
2. Finché c'è incremento di performance media sui vari split, aumento in multipli di 50 il numero di nodi nello strato nascosto;
3. Se c'è un declino di performance, aggiungo un termine di regolarizzazione sempre più crescente e ripeto il passo 2;
4. Se il punto 3 non dà segni di miglioramento, elimino il termine di regolarizzazione e faccio *pruning* sul numero di nodi dello strato nascosto fino a trovare il numero ottimale di nodi;
5. Una volta trovato il numero ottimale di nodi nello strato nascosto, provo ad aggiungere un altro strato nascosto e ripeto i punti 1, 2, 3 e 4 per quello strato. Se non ci sono migliramenti significativi, elimino questo strato, altrimenti lo tengo.

### Preprocessing

Prima di procedere con la ricerca del classificatore, preprocesso i dati di input:

- Ridimensiono, grazie alla libreria OpenCV, tutte le immagini ad una size fissa di 288x288;
- Applico una gamma correction di valore pari a 1;
- Sostituisco le immagini con i loro valori HOG grazie alla libreria Scikit Image;
- Converto ogni immagine rappresentata per da un array multidimensionale (288×288) dell'HOG, in un vettore ad una dimensione sola $x \in R^n$, dove $n = 288 \times 288$;
- Applico una Z-score standardization ai vettori ottenuti.

Non binarizzo le labels in quanto supportate dalla versione non troppo recente di MPClassifier della libreria di Scikit Learn.

### Modello finale ottenuto

Dopo aver provato diversi modelli e provato diversi valori degli iperparametri di apprendimento, il classificatore MLP che offre la migliore performance per i dati a disposizione, ha la seguente topologia:

- Strato di input composto da 82944 ($288 \times 288$) neuroni, features del vettore HOG di un'immagine;
- Un solo strato nascosto composto da 350 neuroni con funzione di attivazione ReLU: $f(x) = \max(0, x)$;
- Strato di output composto da 5 neuroni (le classi);

### Risultati nel training

Nel training sono stati ottenuti i seguenti score per i tre split stratificati e randomizzati:
**Split 1:**

| | precision | recall | f1−score | support |
|---|---|---|---|---|
| abraham_grampa_simpson | 0.61 | 0.49 | 0.54 | 228 |
| bart_simpson | 0.50 | 0.49 | 0.50 | 335 |
| homer_simpson | 0.56 | 0.72 | 0.63 | 562 |
| lisa_simpson | 0.61 | 0.49 | 0.54 | 339 |
| marge_simpson | 0.59 | 0.53 | 0.56 | 323 |
| avg / total | 0.57 | 0.57 | 0.56 | 1787 |

**Split 2:**

| | precision | recall | f1−score | support |
|---|---|---|---|---|
| abraham_grampa_simpson | 0.56 | 0.44 | 0.49 | 228 |
| bart_simpson | 0.53 | 0.52 | 0.53 | 335 |
| homer_simpson | 0.60 | 0.72 | 0.65 | 562 |
| lisa_simpson | 0.58 | 0.54 | 0.56 | 339 |
| marge_simpson | 0.61 | 0.54 | 0.57 | 323 |
| avg / total | 0.58 | 0.58 | 0.58 | 1787 |

**Split 3:**

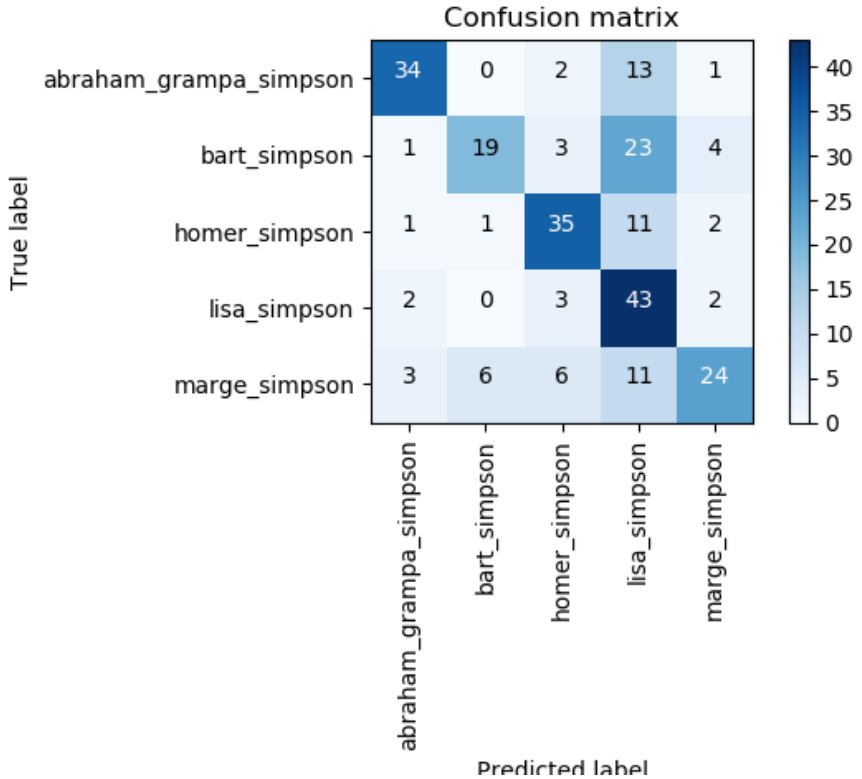
| | precision | recall | f1−score | support |
|---|---|---|---|---|
| abraham_grampa_simpson | 0.56 | 0.43 | 0.49 | 228 |
| bart_simpson | 0.52 | 0.53 | 0.53 | 335 |
| homer_simpson | 0.59 | 0.67 | 0.63 | 562 |
| lisa_simpson | 0.55 | 0.54 | 0.54 | 339 |
| marge_simpson | 0.63 | 0.59 | 0.61 | 323 |
| avg / total | 0.57 | 0.57 | 0.57 | 1787 |

Come previsto in fase di analisi dei dati, i risultati mostrano performance sulla classe *homer_simpson* mediamente migliori rispetto alle altre classi, in quanto classe predominante in termini di quantità di dati empirici.

### Risultati del test

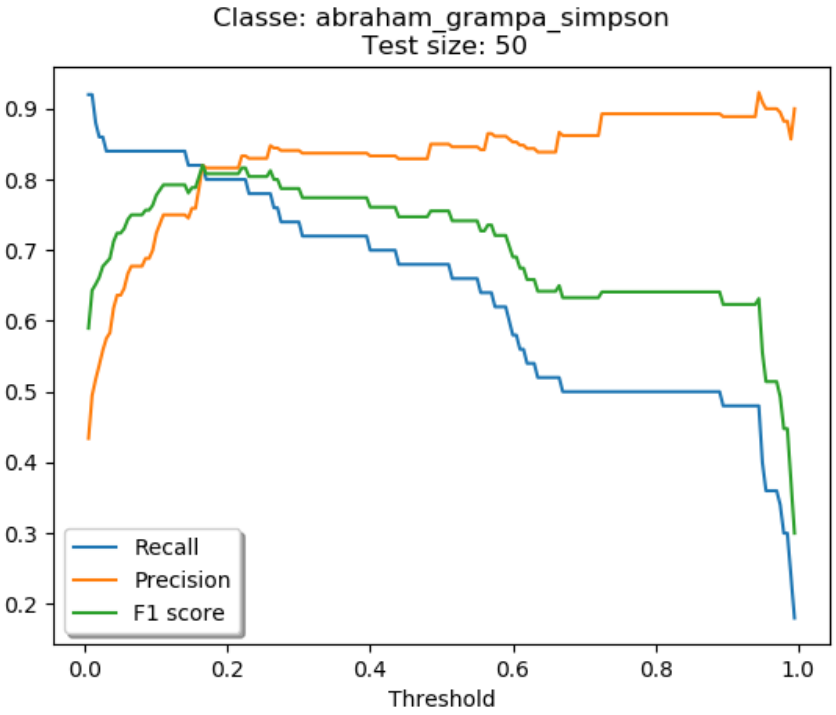
Il classificatore applicato ai dati di test non utilizzati per il training, ha dato i seguenti score:

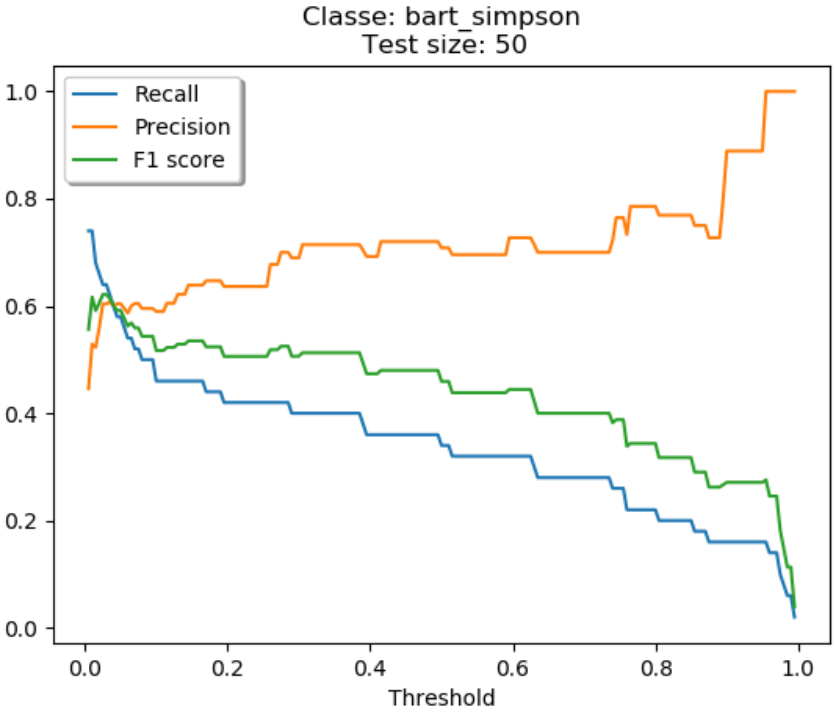
| | precision | recall | f1−score | support |
|---|---|---|---|---|
| abraham_grampa_simpson | 0.83 | 0.68 | 0.75 | 50 |
| bart_simpson | 0.73 | 0.38 | 0.50 | 50 |
| homer_simpson | 0.71 | 0.70 | 0.71 | 50 |
| lisa_simpson | 0.43 | 0.86 | 0.57 | 50 |
| marge_simpson | 0.73 | 0.48 | 0.58 | 50 |
| avg / total | 0.69 | 0.62 | 0.62 | 250 |

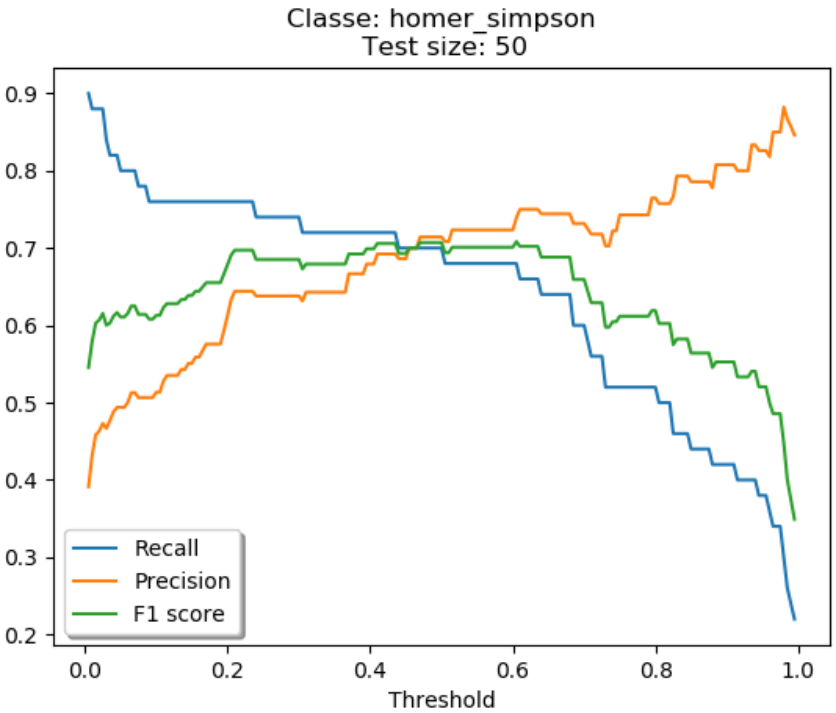


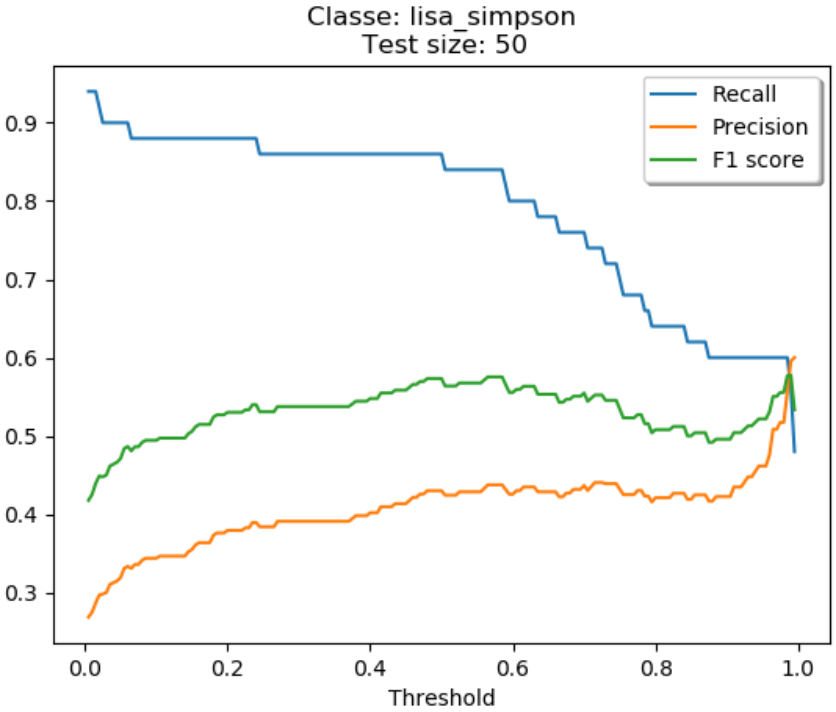

Questi risultati ottenuti sui dati di test sono leggermente maggiori di quelli di training in quanto il modello è stato riallenato su tutti i dati training senza cross validation. Il difetto di questo modello è che ha i falsi negativi che tendono a corrispondere a classificazioni come *lisa_simpson*, dovuto probabilmente a translazioni delle caratteristiche principali, oppure a dati non caratteristici, che sono stati appresi come caratteristiche di *lisa_simpson* in fase di training. Infatti questo comportamento si accentua quando si aumenta la complessità del modello. Purtroppo abbassare la complessità del modello non mitiga questo effetto ma peggiora le performance globali del classificatore a causa dell'errore empirico che aumenta. C'è bisogno di qualche algoritmo che abbia una buona capacità ad apprendere le caratteristiche principali senza confondere con il rumore e senza essere sensibile alle translazioni di quest'ultime, ad esempio il Convolutional Neural Network Tuttavia il classificatre MLP ottenuto potrebbe essere migliorato se abbiamo maggiori dati di training a disposizione. Il fatto che le medie della precision, la recall e il F1 score siano vicini a quelli ottenuti nel training, fa pensare, grazie al generalization bound, che l'errore di test che approssima l'errore ideale, è vicino all'errore empirico, e quindi si ha una VC-Confidence tende ad essere bassa. Ciò significa che, dato il numero di osservazioni attualmente molto basso, il modello ha una complessità bassa, sufficiente per spiegare i dati, ed ha quindi bisogno di dati in più per generalizzare di più la funzione target.
Adesso vediamo un possibile miglioramento delle performance del classificatore trovato,

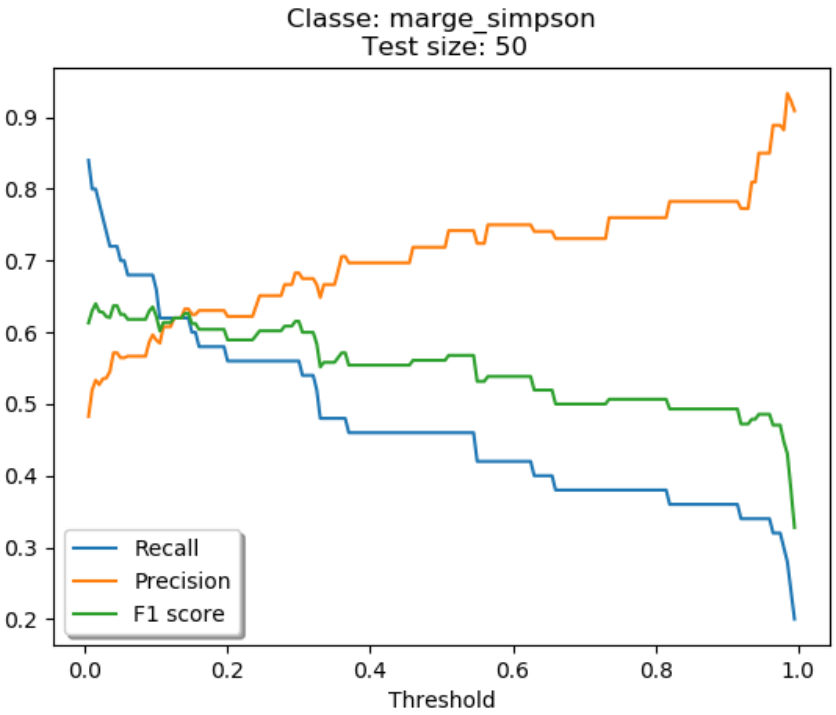
facendo una predizione di probabilità di appartenenza ad una classe, e fissando una soglia minima di probabilità per la classificazione positiva. Usando la tecnica One Vs Rest, è possibile rappresentare la funzione degli score del classificatore per ogni classe, al variare della soglia:




Classe: bart_simpson
Test size: 50
Recall
Precision
F1 score
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Threshold


Classe: homer_simpson
Test size: 50
Recall
Precision
F1 score
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.0
0.2
0.4
0.6
0.8
1.0
Threshold


Classe: lisa_simpson
Test size: 50
Recall
Precision
F1 score
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.0
0.2
0.4
0.6
0.8
1.0
Threshold


Classe: marge_simpson
Test size: 50
Recall
Precision
F1 score
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.0
0.2
0.4
0.6
0.8
1.0
Threshold

In questi grafici si osserva una recall che tende a decrescere all'aumentare della soglia, questo significa che il classificatore ottenuto non è in grado di classificare con buona approsimazione i personaggi, e di conseguenza aumentano i falsi negativi. Perciò fissare delle soglie di probabilità sopra i 0.5 non è ragionevole.

### Support Vector Machine (SVM)

Il Support Vector Machine è un classificatore binario caratterizzato dalla seguente funzione:

$$h_w(x) = \text{sign}(\sum_{i=0}^{n} w_i x_i) = \text{sign}(\mathrm{w} \cdot \mathrm{x})$$

L'apprendimento avviene tramite la ricerca di un iperpiano di massimo margine $\rho = \frac{2}{\|\mathrm{w}\|}$. Questo perché se i dati sono linearmente separabili l'errore empirico è nullo, e possiamo ridurre il bound del rischio ideale riducendo la VC-dimension dello spazio degli iperpiani. La VC-dimension ottimale è definita come segue:

$$VC_{opt} \leq \min\{\left\lceil \frac{R^2}{\rho^2} \right\rceil, m\} + 1$$

Dove $R$ è il raggio dell'ipersfera che contiene tutti i dati. L'apprendimento consiste quindi nel risolvere un problema quadratico vincolato, che in caso di dati linearmente separabili si presenta nella seguente forma:

$$\min_{\mathrm{w},b} \tfrac{1}{2}\|\mathrm{w}\|^2$$

$$\text{s.t.} \quad \forall i \in \{1, ...., n\} : y_i(\mathrm{w} \cdot x_i + b) \geq 1$$

Il quale viene risolto con il duale:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \tfrac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j (\mathrm{x}_i \mathrm{x}_j)$$

$$\text{s.t.} \quad \forall i \in \{1, ...., n\} : \alpha_i \geq 0 \quad \text{e} \quad \sum_{i=1}^{n} y_i \alpha_i = 0$$

Gli $\alpha_i$ sono i moltiplicatori di lagrange e sono non nulli quando i vettori $\mathrm{x}_i$ associati hanno una distanza dall'iperpiano pari al margine diviso due, quest'ultimi vengono chiamati **Support Vectors**.
La soluzione del primale sarà:

$$\mathrm{w} = \sum_{i=1}^{n} y_i \alpha_i \mathrm{x}_i$$

$$b = y_k - \mathrm{w} \cdot \mathrm{x}_k \quad \text{per tutti gli } \mathrm{x}_k \quad \text{t.c} \quad \alpha_k > 0$$

In caso di dati non linearmente separabili, i vettori input vengono proiettati, tramite una trasformazione non lineare, in uno spazio di dimensione maggiore detto **feature space** dove i dati saranno linearmente separabili con alta probabilità. L'iperpiano che

separa i dati nel feature space corrisponde ad una funzione non lineare nello spazio originario. Permettiamo al problema quadratico vincolato di violare alcuni vincoli inserendo ulteriori variabili $\xi_i$, dette variabili slack.
Primale:

$$\min_{\mathrm{w},b} \frac{1}{2}\|\mathrm{w}\|^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{s.t.} \quad \forall i \in \{1, ...., n\} : y_i(\mathrm{w} \cdot x_i + b) \geq 1 - \xi_i$$

Duale:

$$\max_{\alpha} \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j (\mathrm{x}_i \mathrm{x}_j)$$

$$\text{s.t.} \quad \forall i \in \{1, ...., n\} : 0 \leq \alpha_i \leq C \quad \text{e} \quad \sum_{i=1}^{n} y_i \alpha_i = 0$$

Il parametro $C$ è un parametro di regolarizzazione inserito per evitare overfitting. Maggiore è, minore sarà il margine e viceversa. I prodotti scalari tra i vettori input trasformati vengono computati con una funzione che rispetta le condizioni di Mercer, chiamata Kernel. Quest'ultima offre un enorme vantaggio computazionale in quanto calcola il prodotto scalare nel feauture space senza trasformare i vettori input.
Nel caso di più classi, si usano i metodi One vs Rest, oppure One vs One, che consistono rispettivamente a trovare un classificatore SVM per ogni classe, o per ogni coppia di classe.

### Apprendimento del classificatore SVM

Avendo pochi dati a disposizione e risorse computazionali limitate, procederò la mia ricerca con il metodo One vs Rest per la sua efficienza computazionale. Inoltre la ricerca del modello che offre maggiori performance sul test set, avverrà con la tecnica di cross validation presentata nella procedura generale di apprendimento:

1. Usando la classe SVC della libreria SciKit Learn, partirò con i valori di default del parametro $C$ e del Kernel;
2. Per ogni Kernel cercherò il parametro $C$ ottimale incrementandolo discretamente finché c'è miglioramento di performance;
3. Sceglierò la configurazione $C$ - Kernel che offre maggiore performance.

### Preprocessing

Prima di procedere con la ricerca del classificatore, preprocesso i dati:

- Ridimensiono, grazie alla libreria OpenCV, tutte le immagini ad una size fissa di 288x288;
- Applico una gamma correction di valore pari a 1;

- Sostituisco le immagini con i loro valori HOG grazie alla libreria Scikit Image;
- Converto ogni immagine rappresentata per ora da un array multidimensionale ($288 \times 288$), in un vettore ad una dimensione sola $x \in R^n$, dove $n = 288 \times 288$;
- Applico una Z-score standardization ai vettori ottenuti.

Non binarizzo le labels in quanto supportate dalla versione non troppo recente di SVC della libreria di Scikit Learn.

### Modello finale

Il modello finale ottimale per il dataset a disposizione è stato ottenuto con il metodo One vs Rest ed è caratterizzato da :

- I parametri di regolarizzazione $C_i = 5 \times \frac{n}{t_c \times n_i}$ associati al classificatore $h_i$ della classe $i$,
  - $n$ : totale istanze;
  - $t_c$ : totale classi a cui appartengono le istanze;
  - $n_i$ : totale istanze di classe $i$;

  Questa definizione dei parametri cerca di costringere il modello a non fare overfitting sulla classe che ha il maggior numero di istanze;
- Radial basis function kernel per i prodotti scalari nel feature space, in tutti classificatori $h_i$

### Risultati durante il training

Nel training sono stati ottenuti i seguenti score per i tre split stratificati e randomizzati:
**Split 1:**

| | precision | recall | f1−score | support |
|---|---|---|---|---|
| abraham_grampa_simpson | 0.74 | 0.42 | 0.54 | 228 |
| bart_simpson | 0.56 | 0.53 | 0.55 | 335 |
| homer_simpson | 0.57 | 0.80 | 0.66 | 562 |
| lisa_simpson | 0.64 | 0.51 | 0.57 | 339 |
| marge_simpson | 0.63 | 0.54 | 0.58 | 323 |
| avg / total | 0.61 | 0.60 | 0.59 | 1787 |

**Split 2:**

| | precision | recall | f1−score | support |
|---|---|---|---|---|
| abraham_grampa_simpson | 0.71 | 0.39 | 0.51 | 228 |

| | | | | |
|---|---|---|---|---|
| bart_simpson | 0.58 | 0.54 | 0.56 | 335 |
| homer_simpson | 0.57 | 0.79 | 0.66 | 562 |
| lisa_simpson | 0.59 | 0.54 | 0.57 | 339 |
| marge_simpson | 0.65 | 0.54 | 0.59 | 323 |
| avg / total | 0.61 | 0.60 | 0.59 | 1787 |

**Split 3:**

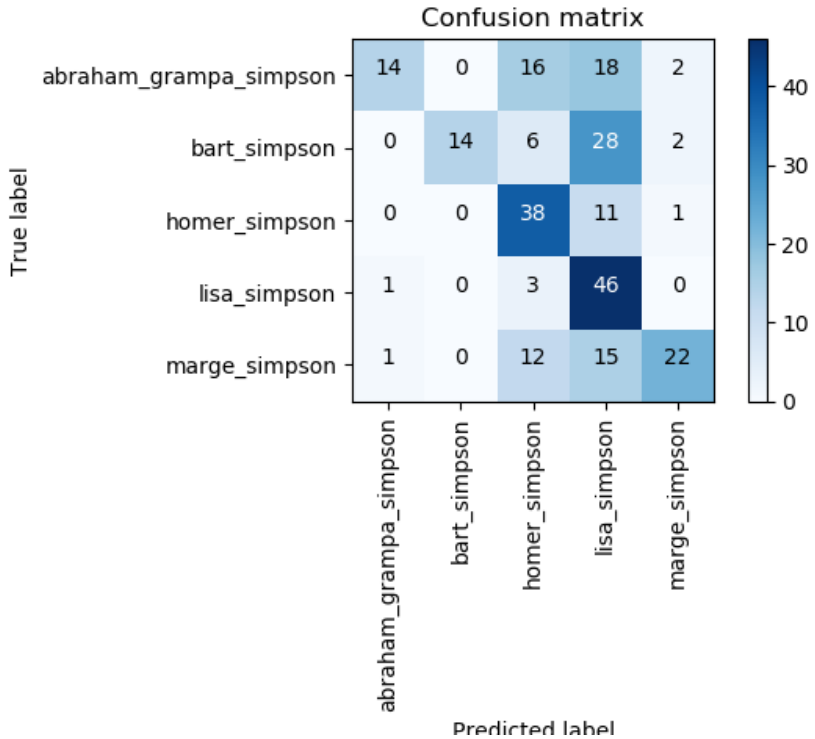
| | precision | recall | f1−score | support |
|---|---|---|---|---|
| abraham_grampa_simpson | 0.75 | 0.38 | 0.50 | 228 |
| bart_simpson | 0.55 | 0.52 | 0.54 | 335 |
| homer_simpson | 0.57 | 0.78 | 0.66 | 562 |
| lisa_simpson | 0.62 | 0.54 | 0.57 | 339 |
| marge_simpson | 0.64 | 0.58 | 0.61 | 323 |
| avg / total | 0.61 | 0.60 | 0.59 | 1787 |

Benché gli scores ottenuti nel training sono mediamente sufficienti rispetto ad altri modelli svm provati, c'è sempre un favoritismo del modello nei confronti della classe più popolata, la classe *home_simpson*, confondendo di più le classi meno popolate (metrica recall).

### Risultati del test

I seguenti risultati sono stati ottenuti dopo aver allenato di nuovo il modello su tutto il training set, e fatto il test sui dati non utilizzati per fare training.

| | precision | recall | f1−score | support |
|---|---|---|---|---|
| abraham_grampa_simpson | 0.88 | 0.28 | 0.42 | 50 |
| bart_simpson | 1.00 | 0.28 | 0.44 | 50 |
| homer_simpson | 0.51 | 0.76 | 0.61 | 50 |
| lisa_simpson | 0.39 | 0.92 | 0.55 | 50 |
| marge_simpson | 0.81 | 0.44 | 0.57 | 50 |
| avg / total | 0.72 | 0.54 | 0.52 | 250 |




Il modello SVM ottimale ottenuto, tende a confondere le classi meno popolate con quelle più popolate, ha delle performance che sarebbero più che discrete in classificazione binaria. Questa confusione si concentra in maggioranza verso la classe *lisa_simpson* come per il classificatore MLP, in quanto hanno distanza maggiore da l'iperpiano *lisa_simpson* vs Rest, rispetto agli altri iperpiani, cioè sono molto vicini ai veri positivi della classe *lisa_simpson*, quindi presentano caratteristiche predominanti in comune con quest'ultimi. Un errore dovuto probabilmente a translazioni di caratteristiche o rumore mal interpretato come nel classificatore MLP. Un altro problema del modello è il fatto di aver calcolato un numero di support vectors molto vicino ai dati di partenza, oltre ad aumentare il tempo computazionale di training e di predizione, questo significa che la maggior parte dei dati si concentrata sui margini dell'iperpiano. Questo numero elevato di suport vectors può essere segno di overfitting che tuttavia non si riflette nei risultati ottenuti, infatti gli score ottenuti non sono molto lontani da quelli ottenuti nel training, e grazie al generalization bound, si può dire che l'errore nel test è maggiormente dovto alla VC-Confidence.

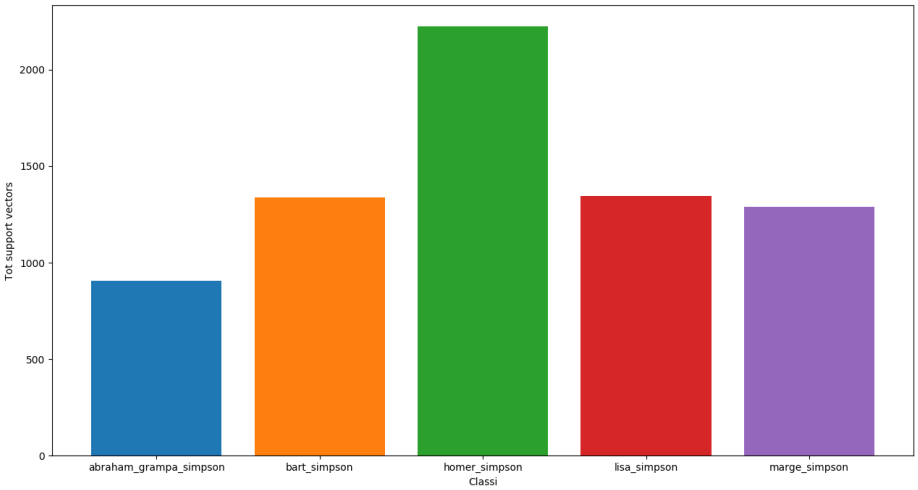



Figura 3: Distribuzione dei support vectors rispetto alle classi di appartenenza

Quindi questo numero di support vectors risulta giusto rispetto ai dati a disposizione. Il modello non può essere migliorato aumentando il parametro $C$ per avere superfici di decisione più precise, in quanto i moltiplicatori di lagrange calcolati sono in valore assoluto minore di 5, lo si può osservare dal seguente grafico ottenuto grazie alla proprietà *dual_coef_* dell'oggetto SVC di Scikit:

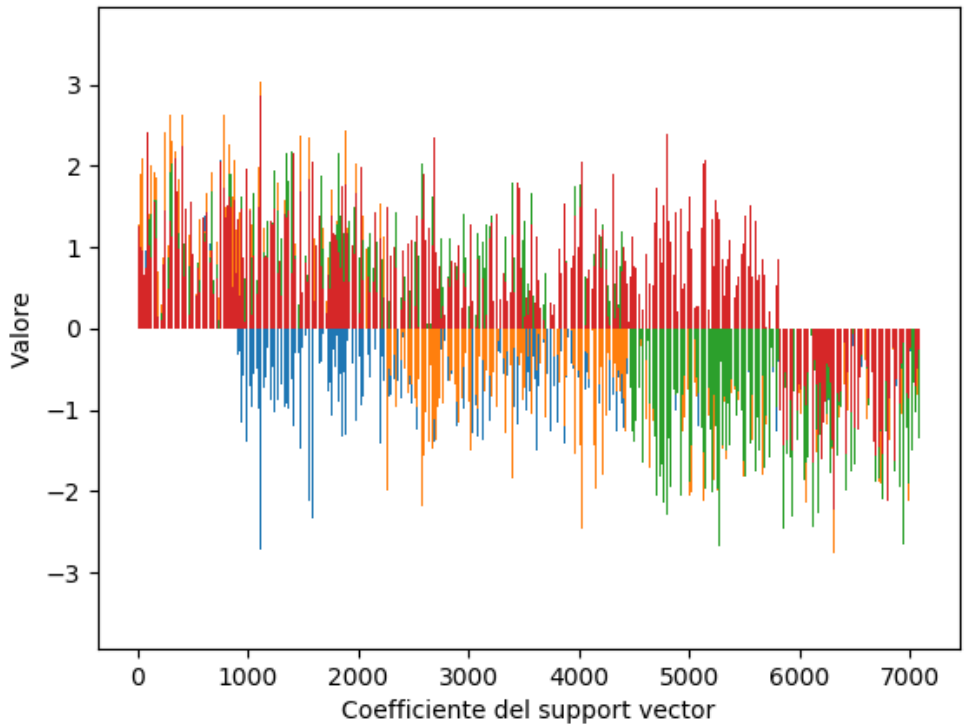



Figura 4: Valori dei coefficienti dei support vectors ($y_i\alpha_i$ dove $y_i \in \{-1, 1\}$)

```
>>> modello = fitter.getModel()
>>> print('Massimo valore in modulo dei coefficienti dei SV=',
          np.amax(np.abs(modello.dual_coef_)))
Massimo valore in modulo dei coefficienti dei
SV = 3.6053279505660223
```

Inoltre non è possibile abbassare il parametro $C$ per avere meno overfitting in quanto l'errore empirico aumenta drasticamente oltre alla VC-Confidence che rimarrà più o meno alta come prima, causando così un test error che aumenta come è stato osservato con gli altri modelli svm candidati in fase di training. L'unico modo per migliorare il modello rimane quello di avere maggiori dati a disposizione.

### Convolutional Neural Network

Noto come CNN o ConvNet, è una tipologia di deep feedforward neural network che fa uso di layer convoluzionali in 2D che filtrano caratteristiche di immagini date in input. Eliminano la necessità di estrazione manuale delle feature, in modo che l'utente non

debba identificare le feature utilizzate per la classificazione delle immagini. L'estrazione automatica delle feature consente un'elevata precisione del modello nella la classificazione di oggetti, e risolve il problema dei gradienti che svaniscono o esplodono nell'addestramento di reti neurali tradizionali multistrato con molti strati usando il backpropagation, in quanto riduce enormemente il numero di parametri da apprendere.
Un CNN è composto da uno strato di input e uno strato di output, oltre a più strati nascosti. Quest'ultimi sono costituiti da strati convoluzionali, strati di pooling, strati fully connected e strati di perdita. Grazie a questa loro topologia, i CNN sono invarianti a translazioni delle caratteristiche apprese.

### Strati di convoluzione

Il core delle CNN è l'operazione di convoluzione, che può operare in 1D (es. speech processing), 2D (es. image processing) o 3D (video processing). La rappresentazione tipica di un'immagine, è una matrice $x \times y \times z$ dove $x$ è la larghezza dell'immagine, $y$ è l'altezza e $z$ è il numero di canali, che in immagini colorate vale 3 e per immagini in bianco nero vale 1. L'operazione di convoluzione coinvolge la matrice che rappresenta l'immagine e una matrice kernel che funge da filtro di una certa caratteristica della matrice immagine. La matrice kernel ha dimensione strettamente della matrice immagine. L'output dell'operazione è la matrice immagine filtrata. Esempi di possibili matrici kernel sono:

$$K = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Figura 5: Matrice Kernel identità

$$K = \begin{bmatrix} -1 & -1 & -1 \\ -1 & -8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Figura 6: Matrice kernel per il rilevamento dei bordi

$$K = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Figura 7: Matrice kernel per la nitidezza

$$K = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Figura 8: Gaussian blur $3 \times 3$

Una volta scelta la matrice Kernel, partendo dal primo pixel della matrice input, la matrice viene posta sopra ad ogni sottomatrice, di dimensione pari alla sua, della matrice input, e si esegue l'operazione di convoluzione che consiste nel moltiplicare ogni elemento della matrice Kernel con l'elemento della matrice che si sovrappone ad esso, e infine si sommano tutti i prodotti ottenuti e il risultato viene messo nella posizione della matrice filtrata, che corrisponde al centro del Kernel nella matrice di immagini. Questa operazione prosegue spostando orrizzontalmente e vertcalmente, la matrice Kernel di un numero $x$ di pixel detto *stride*, che solitamente vale 1. Ad esempio, data una matrice input corrispondente ad un'immagine in bianco nero, e la matrice Kernel per la nitidezza, l'operazione di convoluzione è la seguente:

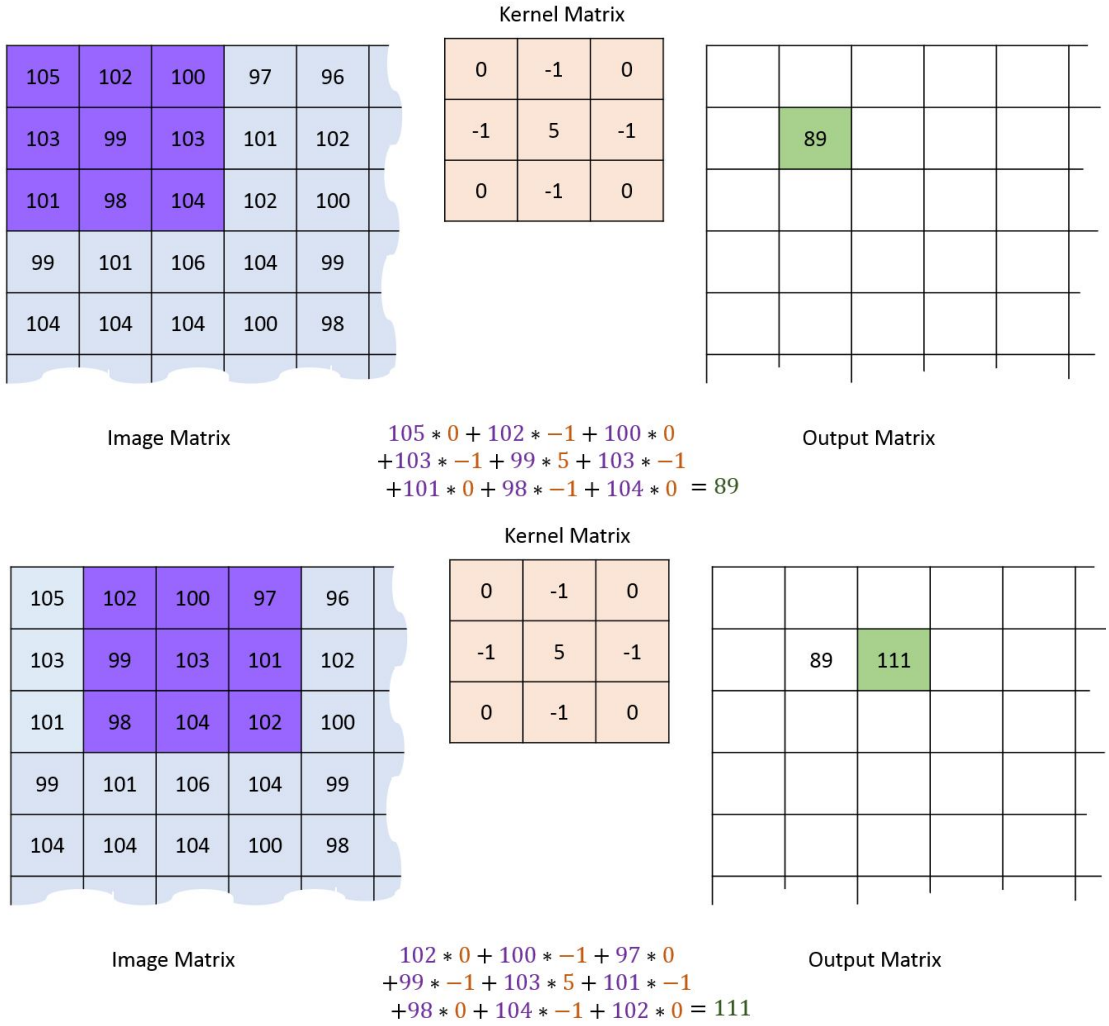



Figura 9: Operazione di convoluzione

Per i pixel sul bordo della matrice immagine, alcuni elementi del kernel potrebbero non avere alcun elemento corrispondente dalla matrice immagine. In questo caso, si può eliminare l'operazione di convoluzione per queste posizioni, la dimensione della matrice output sarà minore rispetta a quella della matrice input, oppure possiamo applicare il padding alla matrice input in base alla dimensione del kernel, che consiste nell'aggiungere dei zeri sui bordi, il risultato sarà una matrice output di dimensione maggiore o uguale a quella di ouput, ma con un probabile calo di contrasto dell'immagine. La dimensione della matrice output può essere calcolata ne seguente modo:
Data una matrice input di dimensione $N \times N$, un filtro di dimensione $F \times F$, uno stride $S$ e un Zero Padding di dimensione $P$, la dimensione della matrice output sarà $M \times M$ dove:

$$M = 1 + \frac{N-F+2P}{S}$$

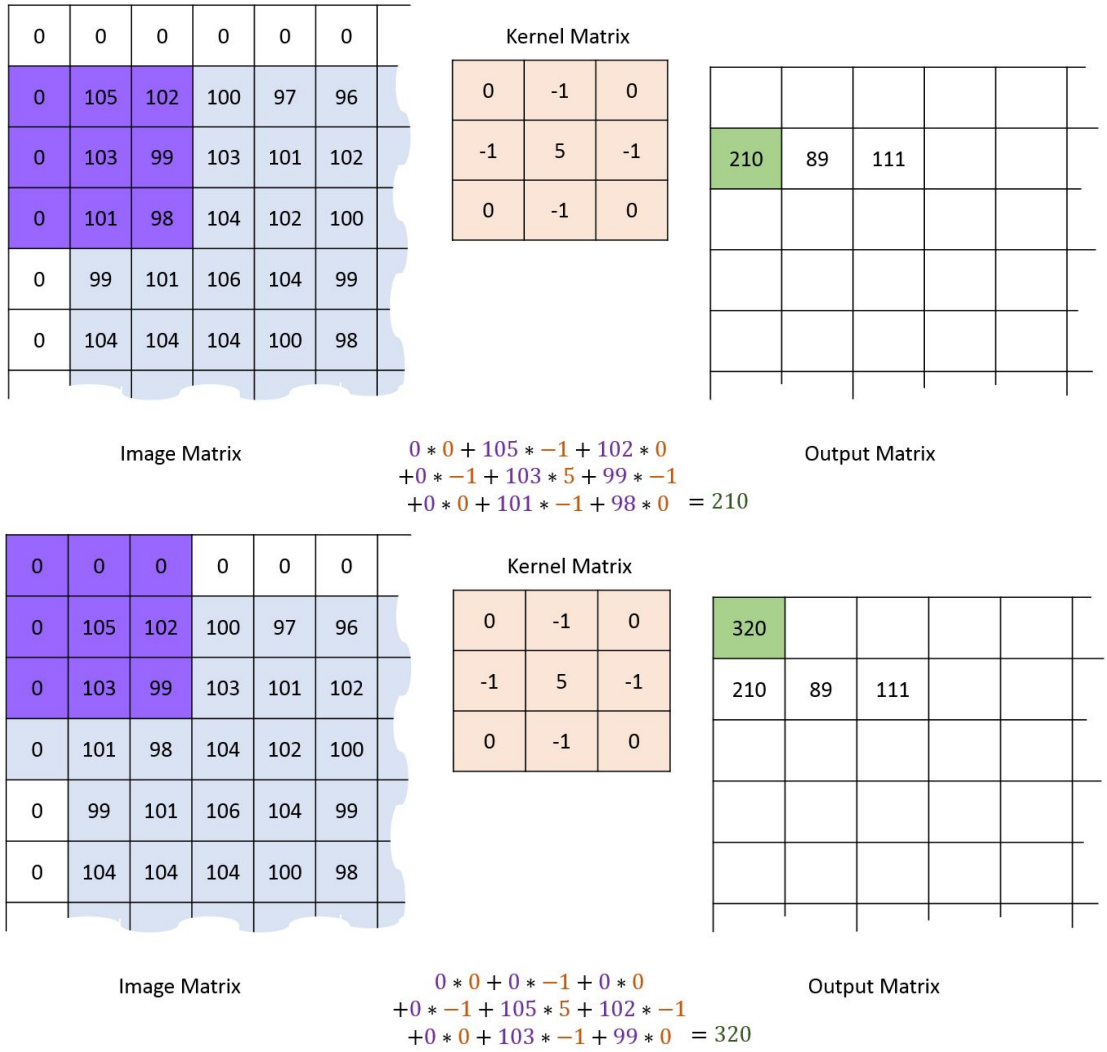



Figura 10: Operazione di convoluzione sul bordo aggiungendo padding

La matrice output di convoluzione potrebbe violare l'intervallo di input di [0-255] dei canali. La soluzione è portare tali valori alla normalità riproporzionandoli. Ciò può essere fatto facendo un Min Max Scaling e moltiplicare il risultato per 255. Questi elementi della matrice output corrispondono ciascuno ad un singolo neurone i quali li passano allo strato successivo con una funzione di attivazione, e condividono gli stessi pesi, in numero uguale alla dimensione della matrice kernel, e i termini bias in totale 1.
Spesso vengono applicati più filtri della stessa dimensione, in uno strato convoluzionale.

In questo caso il numero di pesi condivisi tra neuroni è $F \times F \times y$ e i termini bias sono in totale $y$, dove $F \times F$ è la dimensione delle matrici kernel e $y$ è il numero di filtri applicati.

### Strati di pooling

Lo strato di pooling si trova subito dopo lo strato di convoluzione, e serve a ridurre la dimensione dell'input e a ridurre il numero di parametri, e quindi a ridurre l'overfitting. Viene applicata una funzione non lineare alle matrici output dello strato precedente per ottenere matrici di dimensioni minori. Esistono diverse funzioni non lineari per implementare il pooling tra le quali la più utilizzata è il *Max pooling*, che consiste nell'applicare un filtro che estrae il valore massimo presenti nelle sottomatrici di dimensione pari al filtro applicato, della matrice immagine. Un Max pooling di dimensione $2 \times 2$ con stride uguale a 2, applicato ad una matrice $N \times N$ dimezza la dimensione di quest'ultima.

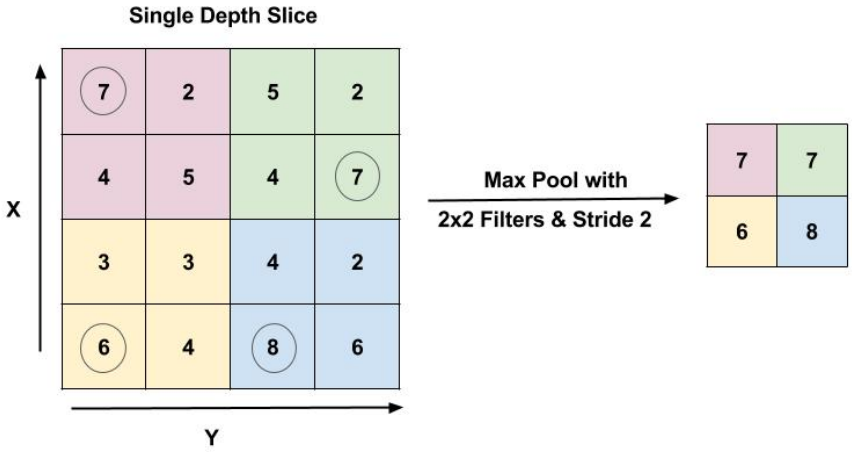



Figura 11: Max pooling

### Strato fully connected

Dopo diversi strati convoluzionali e di pooling, il ragionamento ad alto livello nella rete neurale viene fatto tramite lo strato fully connected. I neuroni in uno strato fully connected hanno connessioni a tutte le attivazioni nello strato precedente, come nelle normali reti neurali. Le loro attivazioni possono quindi essere calcolate con una moltiplicazione di matrice seguita da un offset di bias.

### Svantaggi del CNN

Il CNN tende a fare overfitting sui dati di training in quanto modello complesso che fa features learning su dati di training. Per mitigare questo effetto, si usano diverse tecniche che possono essere:

- **Data augmentation:** I dati di training vengono replicati con variazioni di caratteristiche. Ad esempio nel caso di Image augmentation, le immagini vengono replicate con rotazioni, flipping delle copie. L'apprendimento avviene selezionando un sottoinsiemi disgiunti del training set augmented, ad ogni epoch. In questo modo siamo certi che il modello non rivedrà la stessa immagine nei vari epoch di apprendimento;
- L'uso degli strati di perdita.

### Strato di perdita

Lo strato di perdita è uno strato di regolarizzazione che viene inserito tra due strati di quelli citati sopra. Possono essere utilizzate varie funzioni di perdita appropriate per diversi compiti, la più usata è il Dropout. Informazioni sulla tecnica Dropout può essere trovata a questo link:

`http://jmlr.org/papers/volume15/srivastava14a.old/srivastava14a.pdf`

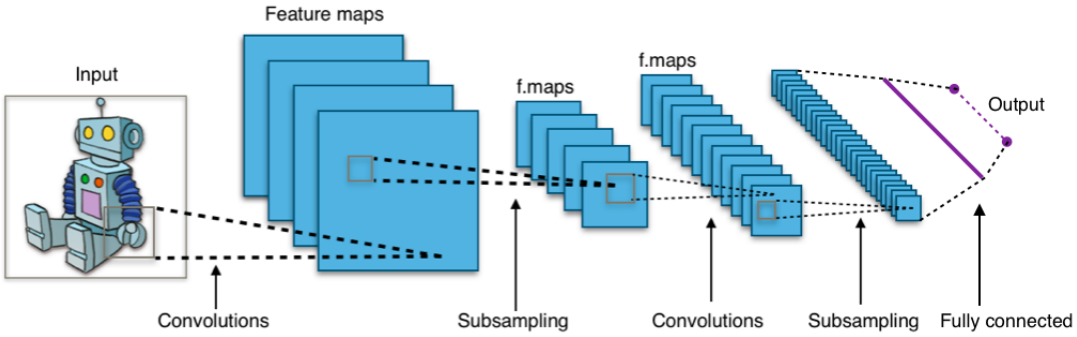



Figura 12: Architettura tipica di un CNN

Il numero di strati di convoluzione è proporzionale all'accuratezza del classificatore, maggiore sono gli strati e più features verranno apprese: angoli, bordi, occhi, piede, viso ecc...

### Limiti del CNN

Il CNN non è in grado di identificare molti personaggi nella stessa immagine ed eventualmente ritornare la loro posizione. Per rimediare a questo difetto, si ricorre spesso alle

versioni deep learning del CNN: R-CNN, Fast R-CNN, Faster R-CNN, di cui non parlerò in questo progetto.

### Classificazione con la libreria Keras

Esistono numerose librerie per che permettono di implementare un classificatore CNN: TensorFlow, Caffe, Theano ecc... Ma la più interessante, e tra l'altro la più usata dalla comunità Kaggle, è Keras. Questa libreria si pone l'obiettivo di astrarre le librerie TensorFlow, CNTK, e Theano, riducendo il numero di righe di codice da scrivere e il delay di apprendimento a scopi di esperimenti scientifici. Maggiori informazioni sulla libreria sono disponibili a questo link:

`https://keras.io/`

Uno dei punti forti di Keras è la possibilità di creare da capo modelli di deep learning più o meno complessi, partendo dalla scelta di uno dei modelli, il Sequential Model e la classe Model usata insieme alle API procedurali di Keras. Questi modelli predefiniti dicono essenzialmente come sono posti gli strati, sequenziale per il Sequential Model, e non sequenziale per la classe Model. Grazie a queste caratteristiche della libreria Keras, possiamo costruire un CNN da capo scegliendo i vari strati e metodi di regolarizzazione. Keras riduce notevolmente i tempi di apprendimento per CNN poco complessi (pochi strati e pochi filtri), per questo motivo le tecniche di model selection sono difficili da applicare se si vuole ottenere buoni risultati a breve. Spesso è utile partire da un modello di riferimento, appreso in passato da un'altra persona su dati simili. Per ridurre ulteriormente il tempo di esecuzione, si consiglia di eseguire l'algoritmo su GPUs, ridimensionare tutte le immagini e convertirle in scala grigia se i colori non sono fattori di classificazione.
Per questo progetto ho avuto difficoltà a fare apprendimento di un CNN per classificare i Simpson. Difficoltà dovute alle capicità limitate del mio calcolatore. Per questo motivo presenterò un possibile CNN appreso da un membro della community Kaggle, che è tra l'altro colui che ha pubblicato il dataset. La sua soluzione è disponibile al seguente link:

`https://www.kaggle.com/alexattia/visualizing-predicted-characters`

Il modello proposto è il seguente:

```
pic_size = 64 # Ridimensionamento delle immagini
num_classes = 10 # totale classi

model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same', input_shape=(3,pic_size,
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
```

```
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(256, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(256, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(1024))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(18, activation='softmax'))
opt = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)

model.compile(loss='categorical_crossentropy',
              optimizer=opt, metrics=['accuracy'])

# Image augmentation
datagen = ImageDataGenerator(
featurewise_center=False, # set input mean to 0 over the dataset
samplewise_center=False, # set each sample mean to 0
featurewise_std_normalization=False, # divide inputs by std
samplewise_std_normalization=False, # divide each input by its std
rotation_range=0, # randomly rotate images in the range
width_shift_range=0.1, # randomly shift images horizontally
height_shift_range=0.1, # randomly shift images vertically
horizontal_flip=True, # randomly flip images
vertical_flip=False) # randomly flip images

model.fit_generator(datagen.flow(train_data, train_labels_one_hot, ba
steps_per_epoch=int(np.ceil(train_data.shape[0] / float(batch_size)))
epochs=200,
validation_data=(test_data, test_labels_one_hot),
```

workers=4)

Questo modello sequenziale è stato definito e allenato per la classificazione di 18 personaggi della serie TV. Esso si compone di: 6 strati di convoluzione (uno di input) aggiunti con la funzione Conv2D, 3 strati di pooling aggiunti con la funzione MaxPooling2D, 4 strati di perdita di tipo Dropout, uno strato fully connected aggiunto con la funzione Dense, e uno strati di output aggiunto ancora con la funzione Dense.

- **Conv2D:** Questa funzione aggiunge uno strato di convoluzione ad un modello, per filtrare immagini(features learning). Il primo parametro corrisponde al numero di filtri da applicare e il secondo parametro corrisponde alla loro dimensione. Il parametro *padding='same'* chiede di applicare un padding in modo tale che le dimensioni della matrice filtrata e la matrice input corrispondano.Non è possibile sceglielre il tipo di matrici Kernel per il filtro, esse vengono create dallo strato stesso. Infine il parametro *input_shape* viene applicato allo strato di input (il primo strato aggiunto), e specifica le dimensioni $channels \times width \times height$ della matrice input. Attenzione che tale dimensione viene accettata soltanto se nel file JSON di configurazione di Keras, il parametro *image_data_format* è settato a *channels_first*, altrimenti l'unica dimensione accettata è $width \times height \times channels$;

- **Activation:** Definisce la funzione di attivazione dei neuroni di uno strato. Questa funzione può essere sostituita inserendo il nome della funzione desiderata nel parametro *activation* della funzione che aggiunge lo strato, ad esempio la funzione Conv2D

- **MaxPooling2D:** Strato di pooling con operazione di max pooling su immagini. Il parametro *pool_size* specifica la dimensione della matrice kernel da usare per l'operazione. Il valore (2,2) di questo parametro ha come effetto il dimezzamento delle dimensioni delle matrici che riceve in input;

- **Dropout:** funzione di perdita che setta a zero una frazione delle unità di input ad ogni aggiornamento dei pesi nell'apprendimento. Il primo parametro specifca la frazione;

- **Flatten:** può essere visto come la funzione che aggiunge uno strato di preprocessing prima di applicare strati che non accettano input di dimensioni maggiori di 2. Tipicamente viene aggiunto prima dello strato di fully connected e di output, e converte l'input ricevuto in vettore;

- **Dense:** può essere visto come la funzione che aggiunge uno strato di dense-connected, variazione dello strato di fully connected, e consiste ad applicare la funzione di attivazione specificata alla somma tra il termine bias e il prodotto scalare tra l'input ricevuto e la matrice Kernel creata dallo strato. Accetta input di dimensione al più 2 e produce ouput con dimensione uguale. Il primo parametro specifica il numero di unità di output dello strato. Questa funzione può essere

utilizzata per definire sia uno strato di dense-connected che uno strato di output o di input;

Le funzioni citate sopra hanno altri parametri che possono essere consultati dalla documentazione, per ridefinirli a seconda delle proprie esigenze.
Prima di procedere con l'allenamento, il modello viene compilato scegliendo la funzione di costo per l'apprendimento, le metriche di valutazione e i parametri di ottimizzazione. Vengono inoltre preprocessati i dati ridimensionando le immagini e applicandogli un Min Max scaling, le etichette vengono binarizzate usando il metodo One Hot Encoding. L'allenamento avviene con Image augmentation del training set e validazione su un test set, il tutto in 200 epochs e 32 batch. I risultati ottenuti sono i seguenti:

| | precision | recall | f1−score | support |
|---|---|---|---|---|
| abraham_grampa_simpson | 0.96 | 0.94 | 0.95 | 48 |
| apu_nahasapeemapetilon | 0.98 | 0.96 | 0.97 | 50 |
| bart_simpson | 0.91 | 0.96 | 0.93 | 50 |
| charles_montgomery_burns | 0.91 | 0.90 | 0.91 | 48 |
| chief_wiggum | 1.00 | 1.00 | 1.00 | 50 |
| comic_book_guy | 1.00 | 0.96 | 0.98 | 49 |
| edna_krabappel | 0.98 | 0.92 | 0.95 | 50 |
| homer_simpson | 0.94 | 0.92 | 0.93 | 50 |
| kent_brockman | 1.00 | 0.96 | 0.98 | 50 |
| krusty_the_clown | 0.98 | 1.00 | 0.99 | 50 |
| lisa_simpson | 0.96 | 0.88 | 0.92 | 50 |
| marge_simpson | 0.98 | 1.00 | 0.99 | 50 |
| milhouse_van_houten | 0.94 | 0.98 | 0.96 | 49 |
| moe_szyslak | 0.96 | 0.96 | 0.96 | 50 |
| ned_flanders | 0.87 | 0.96 | 0.91 | 49 |
| nelson_muntz | 0.96 | 0.88 | 0.92 | 50 |
| principal_skinner | 0.89 | 1.00 | 0.94 | 50 |
| sideshow_bob | 0.98 | 1.00 | 0.99 | 47 |
| avg / total | 0.96 | 0.95 | 0.95 | 890 |

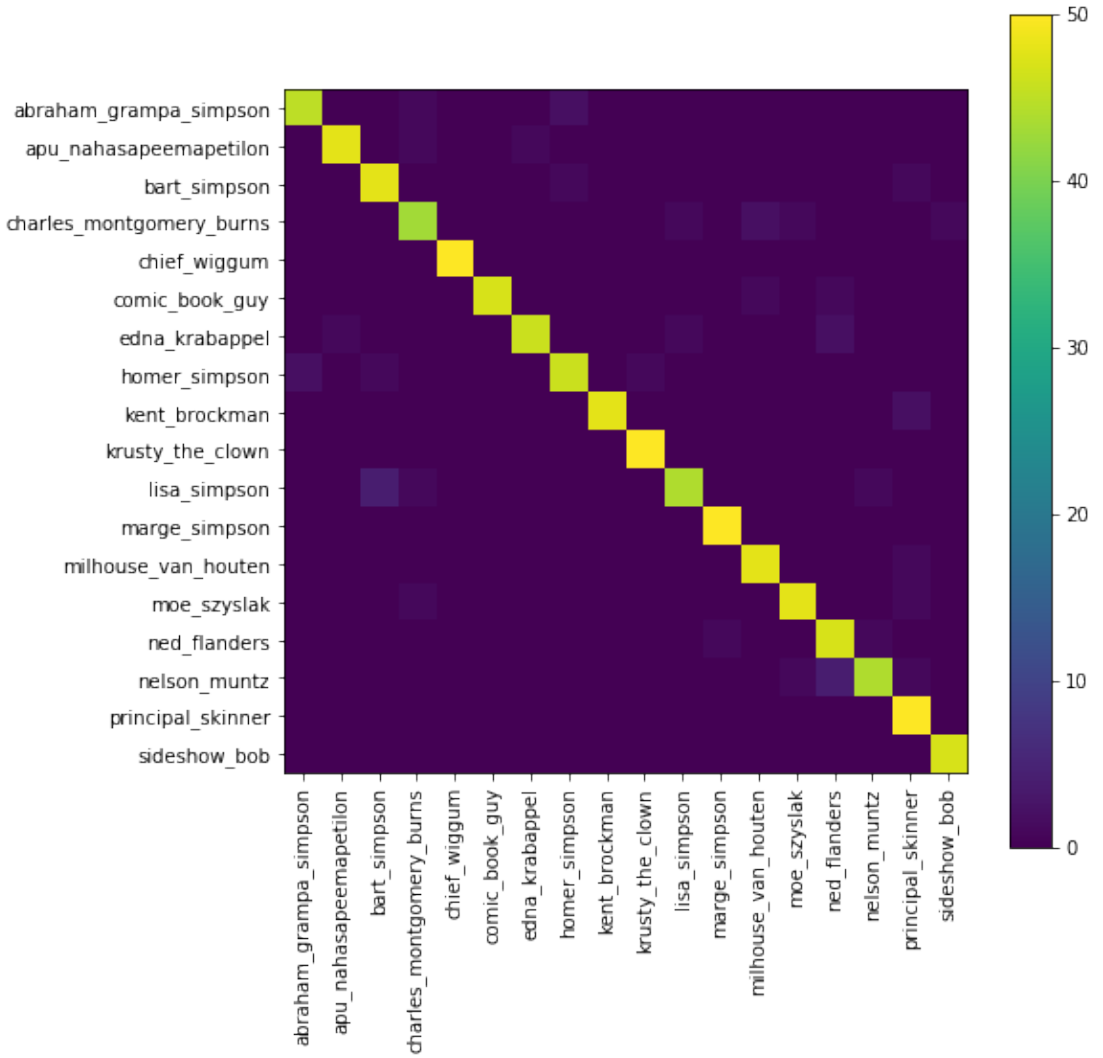



Figura 13: Matrice di confusione

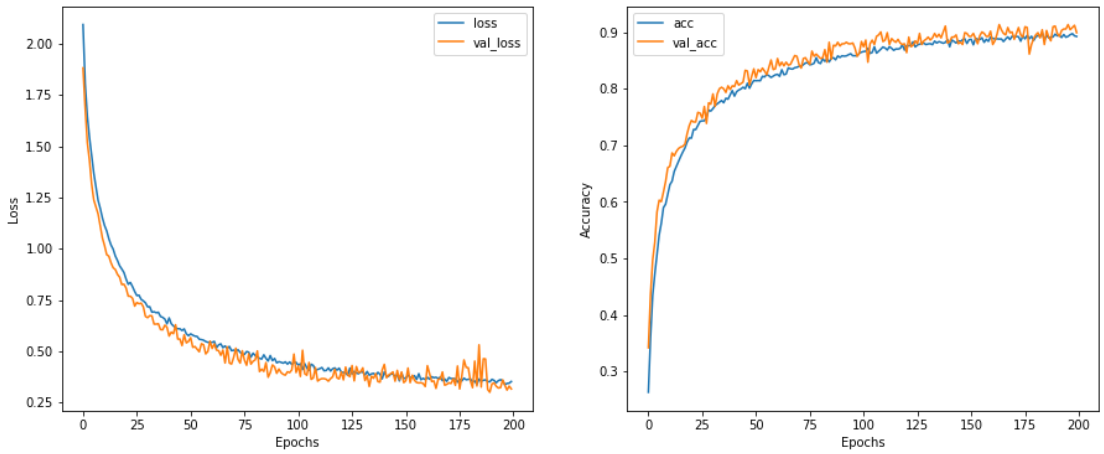



Figura 14: Andamento della funzione di costo e dell'accuratezza durante il training

## Conclusione

Il CNN ha le performance migliori grazie alle capacità di features learning e invarianza a translazioni di queste features. E quindi rappresenta la mia prima scelta di classificatore per questo dataset, e potrebbe essere migliorato in futuro con maggiori dati di apprentimento. Anche il modello MLP che ho trovato avrà sicuramente buone performance con un training set più ampio, anche se queste performance non supereranno quelle del CNN per i punti di forza posseduti da quest'ultimo che un MLP non ha, quindi rappresenta la mia seconda scelta. Il modello SVM potrebbe anche lui migliorare in presenza di maggiori dati, ma i tempi di apprendimento e di test aumenterebbero proporzionalmente, e quindi rappresenta la mia ultima scelta.
Il CNN fallisce nella classificazione multi-etichetta di un'immagine, che non è tuttavia presente nel problema oggetto di questo progetto, ma qualora si presentasse, gli algoritmi utilizzati in questo progetto non costituirebbero possibili scelte. Le scelte vanno invece valutate su altri algoritmi come il BP-MLL per una tradizionale rete neurale multistrato, e algoritmi di deep learning come il R-CNN.