UNIVERSITÀ
degli STUDI
di CATANIA

Local Binary Pattern: Riconoscimento della presenza del COVID-19 in TAC polmonari

Fabio Merola W82000188. Professore: Sebastiano Battiato.

Dipartimento: D.M.I. Informatica LM-18 A.A. 2019/2020

Data documento: 25 Settembre 2020.

Indice

1 Introduzione

Lo scopo di questo elaborato è quello di riuscire ad individuare la presenza o l'assenza del COVID-19 con l'ausilio di un dataset di immagini raffiguranti TAC(Tomografia Assiale Computerizzata), tramite l'utilizzo dell'algoritmo di estrazione features LBP(Local Binary Pattern). Verranno anche proposti diversi metodi utilizzati per l'elaborazione dei vettori di features uscenti da quest'ultimo. Infine, verranno esaminati i risultati allo scopo di stimare quali sono le percentuali di successo dell'algoritmo abbinato ai vari metodi di elaborazione.

Ciò che ci si aspetta dall'elaborato è che la granulosità della texture, propria delle TAC di pazienti affetti dal COVID-19, risulti evidente ad un algoritmo come LBP che si occupa del riconoscimento di texture[1].

Il progetto è quindi diviso in quattro fasi principali:

- Analisi del dataset a disposizione;
- Definizione dei metodi di elaborazione;
- Sviluppo script;
- Interpretazione dei risultati.

2 Analisi del dataset a disposizione

Il dataset utilizzato proviene da una ricerca chiamata "COVID-CT-Dataset: a CT scan dataset about COVID-19"[2]. Questo risulta essere diviso in due parti, TAC con COVID-19 e TAC senza. Le parti contengono rispettivamente 317 e 397 immagini non omogenee in dimensioni e formato, alcune anche con palesi segni di alterazione.

I difetti elencati risultano ammissibili nella ricerca citata, dato l'uso di algoritmi di machine learning, ma risultano poco omissibili in questo elaborato basato su LPB. Questo, inoltre, non permetterà l'utilizzo della variante Multi-Block[1] del Local Binary Pattern per incompatibilità tra le immagini.

Considerata la numerosità dei campioni per le due tipologie di immagini, si è optato per svolgere ogni iterazione dell'esperimento utilizzando 200 immagini prese randomicamente dalle due parti del dataset come referenze a cui, successivamente, comparare 100 immagini(fra le restanti) per tipologia.

[1] La variante Multi-Block prevede la divisione dell'immagine in svariati blocchi, di ognuno se ne computa l'LBP e si concatenano i vettori(o istogrammi) risultanti.

3 Definizione dei metodi di elaborazione

L'algoritmo in questione non prevede un metodo per unire i diversi vettori(o istogrammi) risultanti al fine ultimo di creare una referenza consistente per ogni tipologia di immagine. Per supplire a questa mancanza sono stati ideati ed utilizzati tre metodi di elaborazione dei 200 istogrammi per referenza.

3.1 Primo metodo

Il primo e più banale implica l'uso della media puntuale di tutti gli istogrammi contenuti nel set creato per ogni referenza. Alla fine un solo istogramma rappresenterà la referenza(o classe).

Pro	**Contro**
Complessità lineare.	Arrotondamento eccessivo dei valori dell'istogramma.

3.2 Secondo metodo

Questo prevede la costruzione di più istogrammi per la stessa classe, ponendo in ogni sotto-classe solo istogrammi il cui valore medio delle differenze puntuali risulti sotto una determinata soglia.

Nel dettaglio:

- **Passo base**: Non esiste nessuna sotto-classe per la referenza corrente, l'istogramma ricevuto comporrà la prima sotto-classe.

- **Passo successivo**: Per ogni sotto-classe si calcola la differenza puntuale tra il suo rappresentante[2] e l'istogramma ricevuto. Del vettore uscente da ogni differenza si calcola il valor medio e questi si ordinano in maniera crescente. Se il primo valore risulta essere sotto una determinata soglia(parametrizzabile), l'istogramma viene aggiunto alla sotto-classe a cui quel valore fa riferimento; altrimenti verrà creata una nuova sotto-classe per l'istogramma in questione.

Passati al vaglio tutti i campioni si otterranno diverse sotto-classi, ognuna con il suo rappresentante.

Pro	**Contro**
Precisione parametrizzabile. Annulla il difetto principale del dataset ovvero la disomogeneità delle immagini.	Arrotondamento eccessivo nel caso peggiore. Complessità non lineare, quadratica nel caso peggiore.

[2]Il rappresentante è calcolato facendo la media degli istogrammi appartenenti alla sotto-classe.

3.3 Terzo metodo

L'ultimo metodo è estremamente simile al secondo tranne che per la natura del valore utilizzato per riconoscere la similitudine tra due istogrammi; questo infatti non è più il valore medio delle differenze puntuali, bensì il valore computato dalla funzione di divergenza di Kullback-Leibler[3].

Nel dettaglio:

- **Passo base**: Non esiste nessuna sotto-classe per la referenza corrente, l'istogramma ricevuto comporrà la prima sotto-classe.
- **Passo successivo**: Per ogni sotto-classe si calcola la divergenza di Kullback-Leibler tra il suo rappresentante[3] e l'istogramma ricevuto. I valori risultanti si ordinano in maniera crescente e, se il primo valore risulta essere sotto una determinata soglia(parametrizzabile), l'istogramma viene aggiunto alla sotto-classe a cui quel valore fa riferimento; altrimenti verrà creata una nuova sotto-classe per l'istogramma in questione.

Passati al vaglio tutti i campioni si otterranno diverse sotto-classi, ognuna con il suo rappresentante.

Pro	**Contro**
Precisione parametrizzabile.	Arrotondamento eccessivo nel caso peggiore.
Annulla il difetto principale del dataset ovvero la disomogeneità delle immagini.	Complessità non lineare, quadratica nel caso peggiore.

[3]Il rappresentante è calcolato come nel secondo metodo.

4 Sviluppo script

Tutti gli script sono sviluppati in Python 3.6.9 e sono disponibili, insieme alle analisi principali su GitHub[4]. Le librerie utilizzate sono alle seguenti versioni:

Libreria	Versione
Skimage	0.17.2
Numpy	1.15.2

4.1 Fase iniziale

Inizialmente vengono caricate tutte le immagini necessarie alla creazione delle referenze, di ognuna di queste viene subito computata la funzione di LBP:

```
...
from skimage.feature import local_binary_pattern
...
LBP = {}
LBP["radius"] = 1 #2,3
LBP["n_points"] = 8 * LBP["radius"]
LBP["method"] = "uniform"
...
temp_data["training_set"] = load_random_training_set(
    TRAINING_IMGS_NUMBER,
    [NON_COVID_PATH, COVID_PATH]
    )
lbps_covid = [local_binary_pattern(
    image,
    LBP["n_points"],
    LBP["radius"],
    LBP["method"]
    ) for image in TRAINING_COVID_IMGS]
lbps_non_covid = [local_binary_pattern(
    image,
    LBP["n_points"],
    LBP["radius"],
    LBP["method"]
    ) for image in TRAINING_NON_COVID_IMGS]
lbps_collections = {
    "COV": lbps_covid,
    "NON_COV": lbps_non_covid
}
...
```

La funzione `local_binary_pattern` prende in input:

- Immagine;
- Numero di pixel vicini a quello interessato da prendere in considerazione[4];
- Raggio dell'intorno entro il quale prendere i vicini;
- Metodo LBP da utilizzare[5].

Mentre la funzione `load_random_training_set` si occupa di caricare le immagini in memoria.

Quindi vengono svolte le elaborazioni con i metodi citati poc'anzi, in questo caso parallellizzati per abbreviare i tempi di esecuzione come segue:

```
...
temp_data["method_2_avg_difference_limit"] = 0.003
temp_data["method_3_avg_divergences_limit"] = 0.01

first_method_thread = Thread(
    target=method_1,
    args=(lbps_collections,)
    )
second_method_thread = Thread(
    target=method_2,
    args=(
        lbps_collections,
        temp_data["method_2_avg_difference_limit"]
        )
    )
third_method_thread = Thread(
    target=method_3,
    args=(
        lbps_collections,
        temp_data["method_3_avg_divergences_limit"]
        )
    )

for x in [
    first_method_thread,
    second_method_thread,
    third_method_thread
]:
    x.start()
...
```

[4]Solitamente si usa considerare questo numero come il prodotto tra otto ed il raggio dell'intorno.

[5]Impostando il metodo a `"uniform"` si computa la variante LBP invariante alla rotazione ed alla luminanza

Come si può notare, tutti e tre prendono come parametro `lbps_collections` computato poco prima, mentre gli ultimi due richiedono anche il valore parametrizzabile discusso prima, quello che funge da soglia per la creazione di una nuova sotto classe. Un esempio di questi è possibile trovarlo nelle prime due righe.
Ora, in breve, le righe di codice più rappresentative di ogni metodo:

- **Primo metodo**

```
...
for key, lbps_collection in lbps_collections.items():
    ...
    for lbp in lbps_collection:
        # n_bins = numero di colonne dell'istogramma
        n_bins = int(lbp.max() + 1)
        hist, _ = np.histogram(
            lbp,
            density=True,
            bins=n_bins,
            range=(0, n_bins)
        )
        if refs[key] is None:
            refs[key] = hist
        else:
            refs[key] += hist
    refs[key] /= len(lbps_collection)
...
```

Come anticipato, di ogni vettore uscente dalla funzione LBP se ne calcola l'istogramma tramite la funzione `numpy.histogram`, questo si va a sommare ad una variabile temporanea che sarà, infine, divisa per la quantità di vettori ricevuti;

- **Secondo metodo**[6]

```
...
goup_counter = 0
for lbp in lbps_collection:
    n_bins = int(lbp.max() + 1)
    hist, _ = np.histogram(...)
    if len(temp_refs[key].keys()) == 0: # Passo base
        temp_refs[key][...] = {"h": hist, "c":1}
        goup_counter += 1
    else: # Passo successivo
        avg_differences = []
        for key_ in temp_refs[key].keys():
            temp = temp_refs[key][key_]["h"]
            temp /= temp_refs[key][key_]["c"]
            temp = np.abs(hist - temp)
            avg_diff = np.sum(temp)/len(temp)
            avg_differences.append(
                {"key":key_,"val":avg_diff}
            )
        avg_differences = sorted(
            avg_differences, key=lambda k: k["val"]
        )
        # Confronto con la soglia
        if avg_differences[0]["val"] <= avg_diff_limit:
            temp_refs[key][...]["h"]+= hist
            temp_refs[key][...]["c"]+= 1
        else:
            temp_refs[key][...] = {"h": hist, "c":1}
            goup_counter += 1
# Media di ogni sotto-classe
for key_ in temp_refs[key].keys():
    temp = temp_refs[key][key_]["h"]
    temp /= temp_refs[key][key_]["c"]
    temp_refs[key][key_] = temp
...
```

[6]I commenti nel codice indicano le varie fasi del processo, come esposto precedentemente.

- **Terzo metodo**[7]

```
...
goup_counter = 0
for lbp in lbps_collection:
    ...
    if len(temp_refs[key].keys()) == 0: # Passo base
        temp_refs[key][...] = {"h": hist, "c":1}
        goup_counter += 1
    else: # Passo successivo
        avg_divergences = []
        for key_ in temp_refs[key].keys():
            temp = temp_refs[key][key_]["h"]
            temp /= temp_refs[key][key_]["c"]
            temp = np.abs(hist - temp)
            avg_div = kullback_leibler_divergence(
                temp, hist
            )
            avg_divergences.append(
                {"key":key_,"val":avg_div}
            )
        avg_divergences = sorted(
            avg_divergences, key=lambda k: k["val"]
        )
        # Confronto con la soglia
        ...
# Media di ogni sotto-classe
...
```

Infine, vengono caricate le 100 immagini(per tipologia) per testare le referenze, anche loro vengono quindi manipolate tramite `local_binary_pattern`[8] prima e `numpy.histogram` dopo. Quindi si confrontano con le varie referenze, sempre attraverso la funzione di divergenza di Kullback-Leibler, e se ne salvano i risultati in un file json.

Di seguito l'implementazione della funzione di divergenza di Kullback-Leibler:

```
def kullback_leibler_divergence(p, q):
    p = np.asarray(p)
    q = np.asarray(q)
    filt = np.logical_and(p != 0, q != 0)
    return np.sum(p[filt] * np.log2(p[filt] / q[filt]))
```

[7]Le righe di codice in comune con il secondo metodo sono state omesse.

[8]I parametri quali raggio, numero di punti e metodo rimangono invariati rispetto a quelli utilizzati per le referenze.

5 Interpretazione dei risultati

5.1 Premessa

Col fine di produrre risultati analizzabili, sono stati svolti dodici test con i seguenti input:

Test	Raggio LBP	Soglia 2º metodo	Soglia 3º metodo
0	3	0.05	0.25
1	3	0.01	0.05
2	3	0.005	0.025
3	2	0.05	0.25
4	2	0.01	0.05
5	2	0.005	0.025
6	1	0.05	0.25
7	1	0.01	0.05
8	1	0.005	0.025
9	3	0.003	0.01
10	2	0.003	0.01
11	1	0.003	0.01

Ogni test prevede dieci iterazioni dell'intera sequenza di funzioni descritte prima, e per ognuna di queste iterazioni vengono utilizzate:

- 400 immagini scelte randomicamente da quelle a disposizione per creare le referenze(200 COVID-19, 200 NON COVID-19);
- 200 immagini scelte randomicamente da quelle restanti dopo la scelta delle precedenti per testare le referenze(100 COVID-19, 100 NON COVID-19);

I risultati sono da intendere come percentuale di successo nella corretta classificazioni delle immagini. Sono inoltre valutati in base alle variabili in gioco, ovvero:

- Raggio LBP(1,2,3);
- Livelli di precisione determinati dalle soglie(quattro differenti livelli);
- Metodo di elaborazione dei vettori.

5.2 Analisi risultati

Il risultato migliore è stato ottenuto nel nono test dal secondo metodo, con una media sulle 10 iterazioni del 81.5% di successo; mentre il peggior risultato è stato ottenuto nel quarto test sempre dallo stesso metodo, con una percentuale di successo del 54,45%.

Il test che, indipendentemente dal metodo utilizzato, ha fornito i risultati migliori, facendo quindi una media dei valori di tutti i metodi di elaborazione utilizzati, è l'undicesimo con una percentuale di successo media pari al 73,7%.

Il metodo che, sul totale dei test e delle iterazioni, è risultato essere il migliore è il secondo, con una percentuale media di successo del 70,45%. Allo stesso modo, il raggio dell'intorno dei pixel vicini risultato ottimale è pari a uno, con una percentuale di successo del 68,18%.

La precisione scaturita dalle soglie imposte al secondo ed al terzo metodo ha definito il miglioramento più grande, al punto che tra il livello di precisione più basso ed il più alto c'è un miglioramento di circa il 20%.

Tutti i risultati ed il codice utilizzato sono reperibili su GitHub[4].

5.3 Sviluppi futuri

Tra gli sviluppi futuri si può includere l'utilizzo di(in ordine di importanza):

1. Dataset qualitativamente superiore, con immagini di TAC ad alta risoluzione, sicuramente i risultati saranno migliori;
2. Altri metodi per l'elaborazione dei vettori;
3. Varianti LBP come la Multi-Block o la Over-Complete(OCLBP), dataset permettendo.

Riferimenti

[1] LBP technical study.
[https://github.com/Dophy6/LBP-COVID-19-Recognition/LBP_technical_study].
Ultimo accesso: 25 Settembre 2020.

[2] COVID-CT-Dataset: a CT scan dataset about COVID-19.
[https://github.com/UCSD-AI4H/COVID-CT].
Ultimo accesso: 25 Settembre 2020.

[3] Divergenza di Kullback-Leibler.
[https://it.wikipedia.org/wiki/Divergenza$_d i_K ullback - Leibler$].
Ultimo accesso: 25 Settembre 2020.

[4] Repository GitHub dell'elaborato: LBP COVID-19 Recognition.
[https://github.com/Dophy6/LBP-COVID-19-Recognition].
Ultimo accesso: 25 Settembre 2020.