Università degli Studi di Verona

DIPARTIMENTO DI INFORMATICA
Corso di Laurea in Ingegneria e scienze informatiche

Progetto di Teorie e Tecniche del Riconoscimento

Food Recognition

Candidato:
Alessia Bodini
Matricola VR451051

Anno Accademico 2019–2020

Indice

1 Motivazioni e fondamento logico

Il seguente progetto si pone lo scopo di identificare una serie di cibi facendo uso di modelli visti durante il corso di studio (KNN, SVM e NN). Tale tipo di riconoscimento può risultare molto utile per quanto riguarda la classificazione di piatti in tutto il mondo, ad esempio per viaggiotori o stranieri che vogliono avere maggiori informazioni sul piatto o per coloro che sono interessati a conoscere i valori nutrizionali del cibo proposto, il tutto con una sola foto.

2 Stato dell'arte

L'applicazione maggiormente conosciuta per quanto riguardo il riconoscimento di cibi è al momento *Calorie Mama* [2]. Tale applicazione è disponibile per Apple e Android e permette non solo di riconoscere i cibi ma anche di mostrarne i valori nutrizionali e di far gestire all'utente le calorie assunte giornalmente e relativi programmi di fitness. La funzione di *istant food recognition* viene alimentata da *Food AI API* [6] basata sulle ultime innovazioni in campo di deep learning e in grado di riconoscere ad oggi 756 cibi diversi (gran parte cibi tipici di Singapore). Ogni piatto viene poi legato a specifici valori nutrizionali che l'utente utilizza per controllare le proprie diete direttamente dall'app.

3 Obbiettivi

Il mio progetto non si pone di superare i risultati già raggiunti dall'applicazione nè da *Food AI API*, ma di eseguire un'analisi sulle migliori tecniche di classificazione conosciute e decretare la più efficiente tra queste. In particolare il mio lavoro si è concentrato sull'analisi di tre principali metodi per la classificazione: KNN (*K-Nearest Neighbors*), SVM (*Support Vector Machine*) e reti neurali.

4 Metodologia

Il lavoro si è suddiviso nella ricerca di un dataset e relativa estrazione dei dati e delle features e nell'implementazione di alcuni dei modelli di riconoscimento visti durante il corso. Si spiegano di seguito nei dettagli tali processi.

4.1 Ricerca del dataset

Il dataset scelto denominato *Food-101* [3] è disponibile sul sito kaggle.com e presenta un totale di 10100 fotografie di piatti e cibi diversi. In particolare, il dataset è suddiviso in 101 categorie di cibi, ognuna composta da 1000 foto. La classe di appartenza è deducibile dalla cartella in cui essa è contenuta.

Per tutti e tre i metodi di riconoscimento usati si è fatto uso di un campione di sole 10 classi, prendendo 100 immagini ciascuna per la fase di training e 10 per quella di testing, ridimensionate in un formato 64x64. La scelta di questo insieme ridotto di immagini ha permesso di svolgere le operazioni in tempi relativamente brevi. Solo nel caso delle reti neurali si sono presi in considerazione per ogni categoria anche un training set di 750 immagini e un testing set di 250, prendendo le foto in input con due formati diversi 64x64 e 128x128. Negli altri due casi invece si è visto fin da subito che l'utilizzo di un campione più grande portava a risultati ben peggiori e si sono, per questo motivo, evitati ulteriori test.

4.2 Estrazione delle features

Per l'estrazione delle features si è fatto uso di una rete neurale disponibile tra i modelli di Torchvision e già richiamata tramite il file *resnet.py* rilasciato per questo progetto. Tale modello è il ResNet-50, definito a partire dalla ricerca *Deep Residual Learning for Image Recognition* [1].

ResNet-50 è stato usato per i primi due metodi di riconoscimento usati (KNN e SVM), mentre alla rete neurale definita successivamente sono state date direttamente in pasto le immagini del dataset (nel formato specificato sopra).

4.3 Metodi di riconoscimento usati

I metodi di riconoscimento implementati sono i seguenti.

KNN Il metodo dei *K-Nearest Neighbors* è stato costruito utilizzando diversi tipi di metriche e un diverso numero di vicini (K) considerati per l'attribuzione a una certa categoria. In particolare si è fatto uso delle seguenti metriche per il calcolo delle distanze tra le features:

- distanza euclidea: $\|u - v\|$;
- distanza di Minkowski: $\|u - v\|_p$;
- distanza del coseno: $1 - \frac{u \cdot v}{\|u\|_2 \cdot \|v\|_2}$;
- correlazione: $1 - \frac{(u-\bar{u}) \cdot (v-\bar{v})}{\|(u-\bar{u})\|_2 \cdot \|(v-\bar{v})\|_2}$;

Per ognuna delle precedente se ne è calcolata l'efficienza per K pari a 1, 3 e 7.

Dal punto di vista dell'implementazione, si è fatto uso delle seguenti funzioni e classi:

- *sklearn.preprocessing.StandardScaler* [5] per standardizzare le features in ingresso prima di darle in pasto al sistema;
- *scipy.spatial.distance.cdist* [7] per il calcolo della distanza tra una coppia di features;
- *scipy.stats.mode* [7] per trovare tra i K vicini la classe più frequente.

SVM Per l'implementazione della *Support Vector Machine* si sono presi in considerazione anche in questo caso di kernel diversi:

- lineare;
- polinomiale, con grado pari a 3, 5 e 7;
- RBF (*Radial Basis Function*), usando diversi valori per *gamma* ($\frac{1}{n\_features \cdot set.var()}$ se posta uguale a *auto* o $\frac{1}{n\_features \cdot set.var()}$ se posta su *scale*) e di C (0.1 e 1).

Per tutti i casi si è testato il modello su 10, 100 e 1000 iterazioni totali.

Dal punto di vista dell'implementazione, si è fatto uso delle seguenti funzioni e classi:

- *sklearn.preprocessing.StandardScaler* [5] (come in precedenza);
- *sklearn.svm.SVC* [5]: classe che definisce un modello per la classificazione con SVM;
- *sklearn.svm.SVC.fit*: funzione che adatta il modello definito in base ai dati di training;
- *sklearn.svm.SVC.predict_proba*: probabilità di appartenenza di un campione al modello sopra definito.

NN La rete nurale è stata creata ad hoc per il dataset e comprende 6 diversi strati:

1. convoluzione 2D con kernel di dimensione 3x3, passando da 3 canali in input (RGB) a 6 finali;
2. *max-pooling* di dimensione 2x2;
3. seconda convoluzione 2D con uguale kernel (3x3), passando da 6 canali in input a 16 in output;
4. trasformazione lineare che prende come features in input l'insieme dei valori che riguardano l'immagine come finora è stata modificata, cioè con 16×14 (o 30) $\times 14$ (o 30) ($numerodicanali \times altezza \times larghezza$), che confluiscono in 2048 features in output;
5. seconda trasformazione lineare che riduce le features in 1024;
6. terza e ultima trasformazione lineare che da 1024 features passa a sole 10 che rappresentano il numero di classi finali di appartenza. La feature che presenta il valore più alto sarà identificata come la classe di appartenza.

Tale configurazione è stata ispirata da quella presente in nel tutorial di PyTorch: Training a classifier. Il tasso di apprendimento è stato impostato tra 0.001 e 0.01 e il numero di batch a 4. La fase di addestramento è stata fatta proseguire per 2, 5 e 10 epoche.

Per la costruzione della rete neurale è stato utilizzato il framework PyTorch [4] e in particolare:

- *torch.utils.data.DataLoader*: produce dataset sotto forma di tensore su cui iterare;
- *torch.true_divide* per normalizzare i valori in input dei pixel (da 0255 a 01);
- *torch.nn.Module, torch.nn.Conv2d, torch.nn.MaxPool2d, torch.nn.Linear, torch.nn.functional.relu* per definire la topologia della rete neurale;
- *torch.nn.CrossEntropyLoss* per definire la funzione di perdita e la relativa funzione di *backwards*;
- *torch.optim.SGD* per stabilire l'ottimizzatore e le sue funzioni *step* e *zero_grad*.

5 Esperimenti e risultati raggiunti

Sfortunatamente, i test eseguiti sui tre diversi metodi di riconoscimento non hanno condotto a ottimi risultati. Il miglior valore di accuratezza, raggiunto tramite il metodo dei *K-Nearest Neighbors*, è stato del 20%, con valori di precisione e recall rispettivamente pari a 0.33 e 0.34).

In tutti e tre i casi le performance sono state calcolate in base ad accuratezza e secondo i valori di *precision* e *recall*, estratti dalla matrice di confusione calcolata a partire dalle predizioni.

Per ogni metodo, in particolare, sono stati conseguiti i seguenti risultati.

5.1 KNN

Metric \ K	1	3	7
Euclidean	18%	17%	9%
Minkowski	18%	17%	9%
Cosine	**20%**	19%	12%
Correlation	19%	16%	14%

Tabella 1: Risultati ottenuti per il modello KNN.

Si ha quindi che la maggior accuratezza nei risultati (20%) si ha utilizzando come metrica la distanza del coseno e come 1 come numero di vicini considerati.

5.2 SVM

Nel caso della *Support Vector Machine* si distinguono i rusltati ottenuti con kernel lineare e polinomiale (praticamente uguali per grado pari a 3, 5 o 7) e quelli raggiunti tramite RBF (*Radial Basis Function*), suddivisi in base ai valori scelti per γ e per C.

Kernel \ Iterations	10	100	1000
Linear	10%	14%	15%
Polynomial	10%	7%	5%

Tabella 2: Risultati ottenuti con la SVM con kernel lineare e polinomiale.

γ, C \ Iterations	10	100	1000
$\gamma = scale, C = 1$	15%	**18%**	16%
$\gamma = scale, C = 0.1$	8%	17%	16%
$\gamma = auto, C = 1$	8%	16%	14%
$\gamma = auto, C = 0.1$	9%	13%	16%

Tabella 3: Risultati ottenuti con la SVM con kernel RBF.

Come si può notare dalle tabelle il risultato migliore, 18% di accuratezza, viene ottenuto usando un kernel RBF, con $\gamma = scale$ e C = 1 (valori di default). In questo caso si ha un $precision = 0.33$ e una $recall = 0.60$.

5.3 NN

Per addestrare la rete neurale sono state fatte molte prove differenti, in base al numero di immagini considerate (100/10 o 750/250), in base alla dimensione di quest'ultime (64x64 o 128x128), in base al numero di epoche (2, 5 e 10) e infine rispetto al *learning rate*. Per semplificare maggiormente le tabelle contenenti di valori di accuratezza risultati da questi esperimenti vengono riportati di seguito i valori maggiori ottenuti con uno tra i due *learning rate* presi in considerazione, ovvero 0.01 e 0.001.

Size \ Epochs	2	5	10
64x64	11%	11%	10%
128x128	7%	**16%**	16%

Tabella 4: Risultati ottenuti con il NN con un train set di 100 immagini e un test set di 10.

Size \ Epochs	2	5	10
64x64	12,60%	12,20%	10,60%
128x128	12,16%	11,16%	10,64%

Tabella 5: Risultati ottenuti con il NN con un train set di 750 immagini e un test set di 250.

Utilizzando questo tipo di classificazione quindi ci si aspetta al massimo un 16% di accuratezza, con valori di precisione e recall rispettivamente pari a 0.22 e 0.40.

6 Conclusioni

Le sperimentazioni fatte utilizzando questi tre diversi metodi di classificazione fanno capire, nonostante i bassi risultati raggiunti, come talvolta sistemi più semplici quali i *K-Nearest Neighbors* possono battere le più complesse reti neurali.

Riferimenti bibliografici

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[2] Azumio Inc. Calorie mama, 2017.

[3] K Scott Mader. Food-101, 2018.

[4] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[6] Prof. Steven HOI R&D team. Foodai.

[7] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.