Marek Bardoński - nr ind. 336912

20 maja 2016

Systemy uczące się

Zadanie 3 - Raport

W zadaniu nr 3 udało mi się uzyskać wynik 0.9838 uśrednionej miary F1 według wskazań wstępnych na portalu Knowledgepits. Główną techniką, którą zastosowałem, było oznaczenie następujących 9 cech każdego z obrazków: rozmiarów obrazka (mx, my), ilości "kropelek" (ang. blob), ilości kresek oraz 5 cech związanych z natężeniem pikseli – w całym obrazku i każdym z 4 rogów o wielkości mx/2, my/2. Dodatkową techniką była ręczna weryfikacja grup, która okazała się być wyjątkowo efektywna z uwagi na zastosowane kryteria wyboru cech.

Wstęp

Rozpoznawanie znaków jest znanym problemem, który od dawna jest badany przez naukowców. Zostało opracowanych wiele metod automatycznej realizacji tego zadania, najbardziej skuteczną obecnie jest metoda rozpoznawania za pomocą głębokich sieci neuronowych[1]. Skuteczność tej metody wynika z konieczności rozpoznawania skomplikowanych, nieliniowych wzorów na obrazie, niezależnie od ich rozmiaru, relatywnego położenia oraz orientacji, do czego wielowarstwowe sieci neuronowe nadają się wyjątkowo dobrze, szczególnie jeśli zaczynają się od warstw konwolucyjnych.

Niestety, skuteczność tej metody jest związana z wcześniejszym wytrenowaniem sieci na setkach tysięcy przykładów oznaczonych ręcznie przez człowieka, a w zadaniu korzystanie z zewnętrznych zbiorów danych nie jest dozwolone. Dlatego należało zwrócić się do starszych metody, które pozwalały na klasteryzację nienadzorowaną.

Wykrywanie kropelek

Do wykrywania kropelek wykorzystałem pakiet ndimage z pakietu scipy[2]. Kropelkę[3] definiuje jako pewien ograniczony region obrazka o podobnym natężeniu pikseli, w naszym przypadku będą to w szczególności spójne fragmenty literek oraz spójne fragmenty przestrzeni wewnątrz literek (tzw. „oczka”).

Pierwszym krokiem algorytmu jest zastosowanie filtru minimów[4] o jądrze rozmiaru 2x2 (tzn. w każdym z okienek 2x2 wartość każdego z 4 pikseli jest zastępowana przez najmniejszą z nich. Ma to na celu usunięcie drobnych artefaktów zaburzających strukturę obrazków (np. niedomknięty okrąg w literze o). Rozmiar jądra został dobrany eksperymentalnie.

Kropelki stanowiły główny wyróżnik między literami „i” oraz „l”, „o” i „ó”. W moim rozwiązaniu powyższe litery są rozróżniane z niemal 100% dokładnością.

Wykrywanie kresek

Do wykrywania kresek wykorzystałem algorytm Hough transform[5] z pakietu opencv[6]. Jest to algorytm, który próbuje zaczepić wzorcową kreskę w każdym z pikseli i porównuje ile % pikseli obrazka i kreski się pokrywa. Następnie zlicza liczbę kresek, zliczając liczbę pikseli w których % pokrywających się pikseli przekracza wybrany próg czułości.

Algorytm również korzysta z obrazka poddanego filtrowi minimów o rozmiarze jądra 2x2. Następnie kreski wykryte bardzo blisko siebie o tym samym kierunku są scalane w jedną.

Liczba kresek dobrze odróżnia litery „m" od „n", „u" od „w" czy „k" od „l"

Obliczanie natężenia pikseli

Natężenie pikseli całego obrazku oraz każdego z 4 segmentów obliczyłem stosując trywialny algorytm iterujący po obrazku i rozpatrującym każdy piksel osobno. Natężenie pikseli przede wszystkim odróżnia duże litery od małych (z bardzo dobrą dokładnością) oraz szczupłe („i", „j", „l") od puszystych („w", „m"). Natomiast natężenie poszczególnych sektorów odróżnia litery które są niesymetryczne („k", „P", „T").

Algorytmy alternatywne

Ogólna dokładność była wystarczająca do zastosowania kroku nr 2 opisanego w następnym rozdziale, ale warto przedstawić algorytmy, które mogą ją jeszcze bardziej poprawić:

1. **SURF (Speeded up robust features)[7] –** Opatentowany algorytm który rozpoznaje "kropelki" w macierzy pochodnych Hessego. W przeciwieństwie do zastosowanego przeze mnie algorytmu, cechy uzyskane za pomocą SURF będą bardziej ogólne i niezależne od skali. Dzięki dodatkowym technikom (m. in. wykorzystaniu falek Harra[8]) będą one również mniej podatne na rozróżnianie tych samych cech, obróconych. Z tego algorytmu nie skorzystałem głównie z uwagi na jego komercyjny charakter.
2. **SIFT (Scale-invariant feature transform)[9] –** Kolejny opatentowany algorytm, pierwowzór SURF. Polega na rozpoznawaniu „kropelek" w macierzach pochodnych Hessego obliczonych na krokowo rozmytym filtrem Gaussa obrazie (tzw. przestrzeni skali [10])
3. **Sieci konwolucyjne + głęboka sieć neuronowa[11] –** Jest to obecnie najlepsza technika do rozpoznawania obrazów. Najlepszym sposobem byłoby wyuczenie sieci na zbiorach treningowych literek, a następnie rozpoznanie każdej z nich. Niestety, takie podejście wymaga dużej ilości danych treningowych, z których nie można korzystać w zadaniu. Można poniekąd to rozwiązać poprzez techniki rozszerzenia danych [12], jednak nakład pracy na zaimplementowanie opisanych technik nie jest uzasadniony możliwym polepszeniem wyniku.
4. **KPCA (Kernel principal component analysis) [13] –** Redukcja przestrzeni obrazków (która ma rozmiar około 30x30 = 900) na przestrzeń 100 cech. Dokładność metody byłaby jednak niska w naszym przypadku, z uwagi na duże odchylenia w obrazkach.
5. **Zliczanie kresek pod każdym kątem nachylenia osobno** – Dzięki wprowadzeniu dodatkowej cechy która zliczałaby ile kresek występuje pod danym kątem, byłaby możliwość łatwego oddzielenia par liter typu K i H które są oporne na pozostałe techniki użyte w pracy.

6. **Porównywanie z wzorcem (HM – Hough Matching) [14]** - Wykonanie algorytmu HM badającego proporcję wzorców u n oraz a e byłoby sposobem na oddzielenie wyżej wymieniony par literek, które również są oporne na pozostałe techniki użyte w pracy.

Półautomatyczna weryfikacja

Do półautomatycznej weryfikacji wykorzystałem algorytm Agglomerative Clustering z pakietu sklearn.cluster. Działa on analogicznie do pakietu za pomocą sprawdzane są rozwiązania na platformie Knowledgepits.

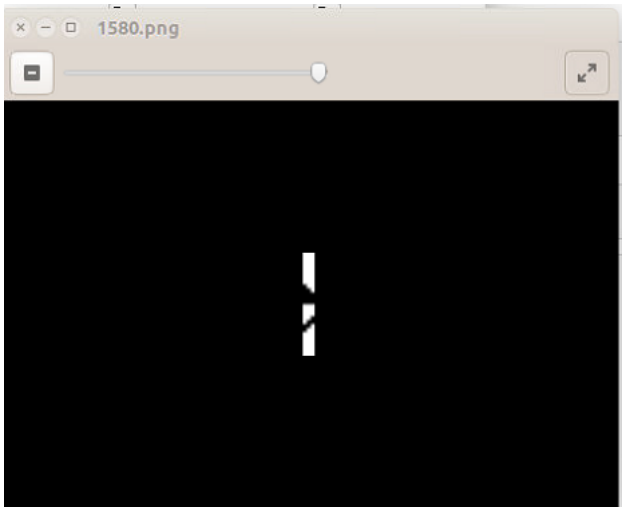
Weryfikacja półautomatyczna polegała na wygenerowaniu 59 folderów z przewidywaniami dotyczącymi klasyfikacji, a następnie ręcznym oddzieleniu poszczególnych klas literek. Jest to szczególnie istotne w przypadku literek 'a' / 'e', 'K' / 'H' oraz 'u' / 'n' które ciężko jest automatycznie rozpoznać.

Dzięki przyjętej strategii (użytych cech), w każdym z folderów do oddzielenia było nie więcej niż 3-4 różne, podobne kategorie znaków, dzięki czemu oddzielanie było bardzo szybkie (zaznaczanie prostokątami) i zajęło około 15 minut.

Po oddzieleniu literek, skrypt przeskanował katalogi i poprawił cechy tak aby były odzwierciedlały poprawną strukturę literek.

Podsumowanie

Osiągnięty przeze mnie wynik 0.983% uśrednionego F1 jest satysfakcjonującym wynikiem. Można go poprawić prawdopodobnie jedynie metodami drobiazgowej, ręcznej weryfikacji bądź używając głębokiej sieci neuronowej wytrenowanej na zewnętrznych danych. Próba dalszego 'podkręcania' wyniku nie niesie zbytniej wartości naukowej, szczególnie, że w zbiorze danych znajduje się kilka-kilkanaście przypadków których nawet człowiek nie jest w stanie zakwalifikować do żadnej z kategorii:



Bibliografia

[1] End-to-End Text Recognition with Convolutional Neural Networks - Tao Wang, David J. Wu, Adam Coates, Andrew Y. Ng, 2012

[2] http://docs.scipy.org/doc/scipy/reference/ndimage.html

[3] https://en.wikipedia.org/wiki/Blob_detection

[4] http://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.ndimage.filters.minimum_filter.html

[5] https://pl.wikipedia.org/wiki/Transformacja_Hougha

[6] http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html

[7] https://en.wikipedia.org/wiki/Speeded_up_robust_features

[8] https://pl.wikipedia.org/wiki/Falki_Haara

[9] https://en.wikipedia.org/wiki/Scale-invariant_feature_transform

[10] https://en.wikipedia.org/wiki/Scale_space

[11] https://en.wikipedia.org/wiki/Convolutional_neural_network

[12] https://www.techopedia.com/definition/28033/data-augmentation

[13] https://en.wikipedia.org/wiki/Kernel_principal_component_analysis

[14] Unsupervised Object Discovery and Localization in the Wild: Part-based Matching with Bottom-up Region Proposals - Minsu Cho, Suha Kwak, Cordelia Schmid, Jean Ponce, 2015