Systemy Inteligentnego Przetwarzania

Rozpoznawanie osób na podstawie odcisków palców przy użyciu sieci Kohonena.

Szymon Bagiński
Dawid Aksamski

Prowadzący: Dr inż. Jacek Mazurkiewicz

Styczeń 2018

Spis treści

1 Wstęp

1.1 Cel projektu

Celem projektu było stworzenie klasyfikatora odcisków palca, opartego o sieć Kohonena[4]. Zadaniem sieci było rozpoznanie osoby, której odciski zostały użyte w procesie uczenia, na podstawie obrazu odcisku, który nie był w tym procesie wykorzystany. W temacie zadania nie zostały określone szczegółowe parametry sieci, takie jak na przykład jej pojemność, czy konkretne mechanizmy jakich należałoby użyć do stworzenia klasyfikatora, więc dostosowano je głównie do posiadanych możliwości.

1.2 Realizacja

Sieć Kohonena posiada ograniczoną pojemność, która jest równa ilości neuronów. Jej rozmiar jest więc bezpośrednio związany z ilością osób, które mają być rozpoznawane. Podczas pracy nad projektem skorzystano z darmowej bazy odcisków palca, dostępnej pod adresem:

`https://www.neurotechnology.com/download/CrossMatch_Sample_DB.zip`

Znajduje się tam 408 obrazów odcisków. Wśród nich jest po 8 różnych obrazów tego samego palca, tej samej osoby. Do trenowania klasyfikatora skorzystano z 7 z nich, natomiast jeden pozostawiono do oceny jego skuteczności. Pojemność sieci musiała więc wynosić co najmniej $\frac{7}{8}$ ilości odcisków, co jest równe 357. W praktyce wykorzystywano siatki neuronów o rozmiarach np. 35x35, co daje 1225 neuronów i powinno w zupełności wystarczyć.

Aby uzyskać dane, które można wprowadzić na wejście sieci Kohonena należy najpierw wydobyć cechy właściwe dla konkretnego odcisku oraz je odpowiednio przygotować. Proces ten został bardziej szczegółowo opisany w punkcie 2.2.

Niezbędne jest także wstępne przetworzenie danych wejściowych, tak aby pozbyć się niepotrzebnych informacji, wyeliminować niedoskonałości, jeśli jest to możliwe lub uwydatnić cechy przydatne z punktu widzenia klasyfikacji. Ten krok został opisany w punkcie 2.1.

Do weryfikacji działania sieci Kohonena, do namierzenia ewentualnych błędów i do wyciągnięcia wniosków, na końcu projektu stworzono prosty perceptron z jedną warstwą ukrytą, co jest opisane w sekcji 2.4. Do implementacji perceptronu, ale także i do sieci Kohonena, skorzystano z biblioteki TensorFlow, która posiada wiele wbudowanych algorytmów przydatnych w tego typu zadaniach.

2 Opis implementacji

Implementację można znaleźć w publicznym repozytorium programu git, pod adresem:

`https://github.com/sbag13/SIP_Fingerprint_recognition`

2.1 Przetwarzanie wstępne obrazu

W celu zmaksymalizowania efektywności algorytmów detekcji cech, obrazy zostały poddane obróbce graficznej.

2.1.1 Progowanie obrazu

Pierwszym etapem jest progowanie obrazu mające na celu przekształcenie obrazu do postaci binarnej. Obraz wejściowy jest w postaci odcieni szarości. Piksele powyżej zadanej wartości zostają zamienione na białe, a pozostałe na czarne. Wartość progowa jest wyliczana na podstawie całego obrazu wejściowego.

2.1.2 Odwrócenie kolorów

W celu zwiększenia efektywności procesu znajdowania szkieletu odcisku palca, kolory w obrazie zostają odwrócone. Odcisk palca z koloru czarnego zmieni się na biały, a tło z białego zmieni się na czarne.

2.1.3 Znajdowanie szkieletu

Ostatnim etapem obróbki obrazu odcisku palca jest znalezienie jego szkieletu. Poddaje się go algorytmowi zaimplementowanemu w bibliotece *OpenCV*. Wynikiem tej obróbki jest odcisk palca, w którym linie papilarne mają szerokość jednego piksela. Pozwala to na uwydatnienie cech kluczowych danego odcisku palca.

2.2 Wydobywanie cech

Po przetworzeniu obrazów odcisków palca należy je sprowadzić do poziomu wektora cech. Sieci Kohonena to tak naprawdę funkcje przyjmujące na wejściu zestaw liczb. Należy więc przedstawić obraz odcisku, bądź jego charakterystykę tak aby można go było podać na wejście takiej sieci. Służą do tego dedykowane algorytmy ekstrahowania cech. Poniżej zostały opisane algorytmy dostępne w bibliotece opencv[6], które zostały użyte w tym projekcie.

2.2.1 Algorytm Kaze

Algorytm Kaze to wieloskalowy detektor cech w obrazach 2D oraz algorytm opisujący w nieliniowych skalach. W przeciwieństwie do SIFT i SURF, które wykorzystują liniową dyfuzję (przenikanie cząstek), operuje na nieliniowych skalach (cechy na wielu poziomach). Wykorzystuje rozmycie nieliniowe, w przeciwieństwie do wspomnianych wcześniej algorytmów co zapewnia rozmywanie cech nieistotnych dla obrazu, a uwypuklanie cech kluczowych.

2.2.2 Algorytm Sift

Algorytm SIFT (skaloniezmiennicze przekształcenie cech) został opracowany w celu wykrywania powtarzających się sekwencji. Używany jest również do śledzenia, rozpoznawania obiektów jak również do łączenia obrazów w panoramy. Celem algorytmu jest wyznaczenia punktów, które są niezmiennicze ze względu na skalowanie i obroty, a częściowo niezmiennicze na zmiany oświetlenia i punktu widzenia (zmiany perspektywy).

Punkty skaloniezmiennicze są wydobywane w kilku krokach:

1. Wykrywanie punktów ekstremalnych (scale space extrema detection).
2. Dokładna lokalizacja punktów charakterystycznych (accurate keypoint location).
3. Przypisanie orientacji punktom charakterystycznym (keypoint orientation assignment).
4. Tworzenie deskryptorów punktów charakterystycznych (keypoint descriptors).

2.2.3 Algorytm Surf

Algorytm SURF (Speeded up robust features) to lokalny detektor i deskryptor cech obrazu. Może być używany do zadań takich jak rozpoznawanie obiektów, rejestracja obrazów, klasyfikacja lub rekonstrukcja 3D. Częściowo zainspirowany jest deskryptorem skalowalności cechującym SIFT (scale-invariant feature transform). Standardowa wersja SURF jest kilka razy szybsza niż SIFT, a jego autorzy uważają, że jest bardziej odporny na różne przekształcenia obrazu niż SIFT.

2.2.4 Algorytm Orb

Algorytm ORB (Oriented FAST and Rotated BRIEF) został stworzony przez OpenCV Labs głównie dlatego, że SIFT i SURF są opatentowanymi algorytmami. Jest prawie dwukrotnie szybszy od algorytmu SURF. Najpierw Korzysta z algorytmu FAST[3] aby znaleźć punkty kluczowe, a następnie stosuje miarę Harris Corner (sekcja 2.2.5) aby znaleźć najbardziej znaczące z nich. Jako, że algorytm FAST nie oblicza orientacji jest on w przypadku ORB zmodyfikowany. Wektorem kierunku zostaje wektor od punktu narożnego w kierunku ważonego środka ciężkości okrągłego regionu sąsiedztwa. Do generowania deskryptorów (binarnych) korzysta z algorytmu BRIEF[1], ale macierz próbek jest obrócona zgodnie z wektorem kierunku danego piksela znalezionym w poprzednim kroku.

2.2.5 Detektor Harris Corner

Detektor Harris Corner[2] służy do znajdowania na obrazie krawędzi oraz innych punktów szczególnych. Dla małych wycinków obrazu wylicza on wynik, następnie przesuwa okno o zadaną odległość i liczy wynik. Gdy wynik znacząco różni się od poprzedniego, oznacza to, że znajduje się w tym miejscu krawędź lub element charakterystyczny dla badanego obrazu.

2.3 Implementacja sieci Kohonena

Sieć Kohonena należy do rzadkiej domeny uczenia bez nadzoru w sieciach neuronowych. Jest to w istocie siatka neuronów, z których każdy oznacza jedną grupę wyuczoną podczas treningu. W sztucznych sieciach neuronowych zazwyczaj nie rozważa się lokalizacji neuronów. Jednak w SOM, każdy neuron ma konkretną lokalizację, a neurony leżące blisko siebie reprezentują gromady o podobnych właściwościach. Każdy neuron posiada wektor wagowy, które podczas procesu uczenia zostały dostrojone tak, aby dany neuron reagował najmocniej na podobne dane wejściowe. Proces uczenia polega na wybraniu neuronów, które odpowiadają za daną grupę rozwiązań. Odbywa się to zazwyczaj prze losowanie wag i określenie, która z odpowiedzi neuronów była najmocniejsza. Dalej iteracyjnie zmienia się wagi, tak aby zwiększyć jeszcze bardziej odpowiedź neuronu zwycięzcy (WTA - Winner Takes All) i ewentualnie jego sąsiadów (WTM - Winner Takes Most). W tym eksperymencie zastosowano strategię WTM.

Do tworzenia różnych sieci Kohonena stworzono specjalną klasę, która pozwala na sparametryzowanie tego procesu. Na topologię sieci wybrano prostokątną siatkę 2D. Poniżej zaprezentowane najważniejsze parametry z punktu widzenia projektowania sieci.

- Wymiar - Dwie liczby całkowite określające rozmiar siatki neuronów.
  Parametr obowiązkowy.
- Ilość iteracji - Całkowita liczba określająca ilość iteracji do wykonania.
  Parametr opcjonalny. Domyślna wartość: 100.
- Alfa - liczba rzeczywista określająca jak mocno mają być modyfikowane wagi aby wzmacniać odpowiednie neurony.
  Parametr opcjonalny. Domyślna wartość: 0.3.
- Sigma - liczba określająca wielkość sąsiedztwa neuronu.
  Parametr opcjonalny. Domyślną wartość stanowi połowa większego z wymiarów.

Zasadniczym elementem sieci Kohonena jest funkcja określająca, który z neuronów pasuje najlepiej do danych wejściowych (ang. bmu - best matching unit). W tym przypadku będzie to neuron, dla którego wartość poniższego równania będzie jak najmniejsza.

$$bmu = \sqrt{\sum (w - x)^2}$$

w - wektor wag pojedynczego neuronu.
x - wektor danych wejściowych.

Wartości alfa oraz sigma, wraz z postępem procesu uczenia, są modyfikowane w każdej iteracji, tak aby kolejne zmiany sieci były coraz mniejsze. Odbywa się to poprzez pomnożenie ich przez tak zwany współczynnik uczenia l, który oblicza się według wzoru:

$$l = 1 - \frac{i}{n}$$

i - numer bieżącej iteracji.
n - ilość wszystkich iteracji do wykonania.

Modyfikacja wag bierze pod uwagę wszystkie przykłady trenujące. Należy pamiętać, że wszystkie opisywane operację są operacjami na macierzach. Zmiana wartości wag dla konkretnego neuronu jest sumą delt obliczonych względem każdego przypadku treningowego. Delty są natomiast różnicą między wagami danego neuronu, a konkretnym zestawem danych wejściowych, pomnożoną przez współczynnik modyfikacji m, wyrażony równaniem:

$$m = \alpha \cdot e^{-(\frac{D}{\sigma})^2}$$

α - parametr alfa sieci.
σ - parametr sigma sieci.
D - odległość euklidesowa do wektora zwycięzcy.
Z powyższego równania można wywnioskować, że wartość współczynnika modyfikacji maleje dla coraz większych wartości D. To znaczy, że im neuron jest bardziej oddalony od neuronu zwycięzcy tym mniej zostanie nagrodzony zmianą wag. Widać także, że wartość ta zależy także od współczynnika σ, który wraz z postępem uczenia maleje. Wartość współczynnika m również będzie maleć dzięki temu. W praktyce oznacza to ograniczenie wielkości sąsiedztwa.

2.4 Implementacja perceptronu wielowarstwowego

Perceptron wielowarstwowy nie jest częścią tematu zadania. Powstał on jednak z myślą o weryfikacji działania sieci Kohonena oraz ułatwił określenie wadliwego komponentu całego klasyfikatora.

Perceptron składał się z trzech warstw. Ilość neuronów warstwy wejściowej była różna podczas badań. Zależało to od zadanej liczby punktów kluczowych, które miał wydobyć algorytm ekstrakcji cech, oraz od ilości liczb opisujących każdy z tych punktów. W praktyce, podczas badań, ilość wejść wahała się od 512 do 4096. Na warstwę ukrytą składało się 256 neuronów w przypadku gdy było 512 wejść, oraz 512 neuronów w przypadku gdy wejść było więcej. Wielkość warstwy wyjściowej była związana z ilością osób, których odciski były rozpoznawane i wyniosła ona 51.

Jako funkcji aktywacji, czyli funkcji, według której obliczana jest wartość wyjścia neuronów, użyto gotowej implementacji funkcji softmax[5] z pakietu TensorFlow. Wraz z postępem procesu nauczania, perceptron miał za zadanie minimalizować wartość entropii krzyżowej wyrażającej się wzorem:

$$-\frac{1}{n}\sum_{j=1}^{n} y_j^{(i)} \log(y_{j\_}^{(i)}) + (1 - y_j^{(i)}) \log(1 - y_{j\_}^{(i)})$$

gdzie $y_j^{(i)}$ jest wzorcowym wyjściem dla próbki j, $y_{j\_}^{(i)}$ jest prognozowanym wyjściem dla próbki j, a n jest ilością wszystkich próbek treningowych.

3 Podsumowanie

3.1 Wyniki

Pierwsze wyniki, zanim postanowiono stworzyć weryfikujący perceptron wielowarstwowy, pokazywały skuteczność sieci Kohonena nie większą niż 4%. Dla sieci o wymiarach 25x25 możemy więc założyć, że poprawna klasyfikacja była dziełem przypadku. Podczas analizy lokalizacji neuronów, do których przypisywane były cechy odcisków palca, zauważono że wiele z nich jest mapowanych do tego samego regionu tworząc klastry. Natomiast w innych rejonach siatki były widoczne duże regiony, w których neurony nie były odpowiedzialne za żaden z odcisków. Przypuszczono więc, że problem leży po stronie algorytmu ekstrahującego cechy z obrazów, który dla różnych odcisków zwracał podobne wektory cech. Takie działanie mogłoby powodować przypisywanie im neuronów bliskich sobie. Aby zweryfikować tę tezę podjęto decyzję o stworzeniu perceptronu, który został opisany w sekcji 2.4. Po bardzo krótkim treningu, udało się osiągnąć 100% skuteczność dla danych trenujących, dla algorytmów Kaze i Surf. Niemniej jednak skuteczność perceptronu na "świeżych" danych testowych oscylowała wokół kilku procent.

3.2 Wnioski

Otrzymane wyniki zdecydowanie nie są zadowalające. Udało się jednak stwierdzić, który komponent klasyfikatora odpowiada najprawdopodobniej za niepowodzenie. Jest to część projektu odpowiadająca za przedstawienie najważniejszych cech odcisku jako wektor liczb. Z uwagi na ograniczenia czasowe nie stwierdzono jednak konkretnie czego zabrakło. Możliwe także, że jakość wykorzystanych obrazów wymagała dużo bardziej zaawansowanych technik przetwarzania niż te, które zostały użyte. Wiadomo także, że obie części klasyfikatora odpowiadające za dalsze przetwarzanie wektorów cech (sieć Kohonena i perceptron) przy innych zastosowaniach spisują się zdecydowanie lepiej. Działanie sieci Kohonena było początkowo przetestowane na prostym przykładzie mapowania kolorów opisanych trzema liczbami (RGB), natomiast perceptron został użyty w projekcie klasyfikacji faz snu, gdzie dla nowych danych testowych uzyskał skuteczność ponad 74%. To tym bardziej przemawia za prawdziwością tezy, że problem leży po stronie ekstrakcji cech.

Odwołania

[1] *BRIEF (Binary Robust Independent Elementary Features)* [online], Data dostępu: 18.01.2019. https://docs.opencv.org/3.1.0/dc/d7d/tutorial_py_brief.html

[2] *Combined corner and edge detector* [online], Data dostępu: 20.01.2019. http://www.bmva.org/bmvc/1988/avc-88-023.pdf

[3] *FAST Algorithm for Corner Detection* [online], Data dostępu: 18.01.2019. https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_fast/py_fast.html

[4] *Sieci neuronowe - Klasyfikator Kohonena* [online], Data dostępu: 18.01.2019. http://galaxy.agh.edu.pl/~vlsi/AI/koho_t/

[5] *Softmax* [online], Data dostępu: 18.01.2019. https://www.tensorflow.org/api_docs/python/tf/nn/softmax

[6] *OpenCv* [online], Data dostępu: 18.01.2019. https://opencv.org/