Uczymy komputer rozpoznawać obrazy

Sztuczna inteligencja (ang. artificial intelligence), w skrócie SI, to dział informatyki starający się za pomocą narzędzi techniki naśladować zachowania człowieka, takie jak np. myślenie, podejmowanie decyzji, generalizacja informacji, rozpoznawanie mowy, obrazów. Jedną z metod SI są sztuczne sieci neuronowe. Są to modele matematyczne wymyślone w latach 50. XX wieku. Od tamtego czasu powstało wiele odmian sieci neuronowych.

Dowiesz się:

- Jak zastosować sztuczną inteligencję do rozpoznawania obrazów.

Powinieneś wiedzieć:

- Podstawowa znajomość języka C++.

Popularną architekturą wśród sztucznych sieci neuronowych jest samoorganizująca się sieć Kohonena, nazywana SOM (*ang. Self-Oranizing Map*). Jest to sieć, która uczy się bez nadzoru. Sieci SOM mogą byś stosowane w zadaniach klasyfikacji. W niniejszym artykule zapoznamy się ze sposobami klasyfikacji obrazów. Opisaną poniżej metodę możemy zastosować w wielu problemach:

- wyszukiwanie obrazu w bazie za pomocą jego zawartości (a nie tekstowych tagów) - CBIR (*ang. Content Based Image Retrieval*) ;
- systemy rozpoznawania twarzy;
- grupowanie obrazów o podobnym wyglądzie.

Na początku artykułu przedstawię sposób przygotowania obrazu umożliwiający zrozumienie go przez sieć neuronową, a następnie omówię architekturę sieci SOM oraz proces jej uczenia. Przedstawionym elementom towarzyszą przykłady implementacji w języku C++.

Przygotowanie obrazów

Bardzo ważnym elementem systemu rozpoznawania obrazów jest sposób przygotowania kodowania obrazów, tzw. *pre-processing*. Polega on na ekstrakcji cech obrazu umożliwiającej opisanie go liczbami – stworzeniu deskryptora obrazu. Generalnie w obrazie można wyróżnić trzy cechy: kolor, teksturę i kształt. Każda z nich może zostać opisana za pomocą zbioru liczb. By uprościć procedurę kodowania obrazu, będziemy używać tylko dwóch cech obrazu: koloru i tekstury. Struktura przedstawiająca obrazy w naszym systemie jest pokazana na Listingu 1. Za pomocą funkcji `Picture::readFromFile()` wczytujemy obraz i zmieniamy jego rozmiar na 160x160 pikseli. Następnie obraz dzielimy siatką 4x4, na 16 równych kwadratów, każdy o boku 40 pikseli. W każdym kwadracie zliczamy średnią liczbę dla każdej wartości z kolorów RGB. W ten sposób każdy kwadrat opisany jest trzema zmiennymi, a całkowita liczba zmiennych opisująca kolor obrazka wynosi 16*3 = 48 (16 kwadratów razy 3 kolory dla każdego). Każda zmienna będzie zawierać się w przedziale (0; 255). W ten sposób zakodujemy kolory, jakie pojawiają się w obrazie. Aby wyznaczyć teksturę obrazu, konwertujemy go do obrazu w skali szarości, ponieważ tekstura nie zależy od kolorów. Konwersji dokonujemy na każdym pikselu za pomocą równania: $Y = 0.3 * R + 0.59 * G + 0.11 * B$. W tak otrzymanym obrazie wyznaczamy macierz tekstury za pomocą prostego algorytmu, który wyznacza różnicę między największą a najmniejszą wartością w sąsiedztwie piksela. Sąsiedztwo piksela określmy jako macierz 3x3 piksele, gdzie środkowy piksel to piksel rozpatrywany, a 8 pikseli na bokach macierzy to „sąsiedzi". Jeżeli będziemy rozważać sąsiedztwo na krańcach obrazu, sąsiedztwo będzie tylko częściowe, a na krawędziach nie będziemy mogli zrobić całej macierzy 3x3. Pomimo to wyznaczymy wartość tekstury z pikseli, które jednak będą w sąsiedztwie. Przypadek najbardziej skrajny będzie miał miejsce, gdy będziemy wyznaczać teksturę dla piksela na rogu obrazu; wtedy macierz sąsiedztwa zawierać będzie tylko 4 piksele. Z otrzyma-

nych wartości tekstury wyznaczamy histogram, który będzie składał się z 32 przedziałów, w zakresie wartości od 0 do 255. Wartość zliczeń w każdym przedziale posłuży do zakodowania tekstury obrazu. Ponieważ pikseli w całym obrazie jest 160*160=25600, wszystkie liczby kodujące teksturę będą zawierać się w przedziale (0; 25600).

Podsumowując, do przedstawienia cech obrazu (koloru i tekstury) potrzebujemy 48 + 32 = 70 liczb. Po wyznaczeniu liczb kodujących obraz powinniśmy je znormalizować do 1, co oznacza, że wszystkie liczby powinny się zawierać w przedziale (0; 1). Aby to zrobić, pierwsze 48 liczb musimy podzielić przez 255, a liczby kodujące teksturę podzielić przez 25600. Normalizacja ułatwi uczenie sieci, ponieważ każda liczba w wektorze kodującym zawierać się będzie w przedziale (0; 1). Każdy obraz, który będzie przedstawiany sieci, musi zostać przekodowany na wektor 70 liczb, które zawsze będą ułożone w tej samej kolejności. Na Listingu 1 przedstawiona jest funkcja `Picture::makeDesc()`, która tworzy deskryptor obrazu. Warto podkreślić, że kolejność w jaką ustawimy liczby kodujące obraz nie ma znaczenia, natomiast bardzo ważne jest by dla każdego obrazu była ona zawsze taka sama.

Struktura sieci

Wiemy już jak zakodować obraz do postaci rozpoznawalnej przez sieć neuronową, prześledźmy teraz budowę i działanie sieci. Sieć SOM najczęściej zbudowana jest z dwuwymiarowej tablicy neuronów. W naszym systemie zastosujemy właśnie taką architekturę sieci. Na Listingu 2 przedstawiona jest klasa reprezentująca neuron. Neuron w sieci będziemy oznaczać *Uij*, co znaczy, że neuron leży w *i*-tym wierszu w *j*-tej kolumnie. Każdy neuron składa się z wektora wag, który łączy neuron z wejściem sieci, schemat takiej sieci pokazany jest na Rysunku 1. Ponieważ deskryptor obrazu jest przedstawiony jako *N*=70 liczb, które będą podawane na wejście, każdy neuron powinien mieć 70 wag. Liczba neuronów w tablicy zależy od tego, jak dużą liczbę obrazów i o jakiej zmienności (jak bardzo różnych od siebie) chcemy rozpoznać za pomocą sieci. Można przyjąć, że około 1 neuron jest w stanie nauczyć się 2-3 podobnych do siebie obrazów. Zakładając, że mamy w bazie około 300 obrazów, powinniśmy przyjąć rozmiar sieci 10x10, a zwiększając liczbę obrazów w bazie odpowiednio powinniśmy zwiększyć rozmiar sieci. Przy doborze odpowiedniej liczby neuronów w sieci zawsze warto jest poeksperymentować i sprawdzić kilka różnych wielkości sieci, i wybrać taką, która według naszego odczucia daje najlepsze wyniki. Przeanalizujmy działanie sieci, przy założeniu, że jest ona już nauczona, co znaczy, że ma odpowiednio dobrane wartości wag. Na wejściu sieci przedstawiony zostaje wektor *X* kodujący obraz. Następnie dla każdego neuronu wyznaczana jest odległość *Dij* w metryce euklidesowej między jego wagami *W* a wektorem wejściowym *X* na podstawie wzoru:

$$D_{ij}(W,X)=\sqrt{\sum_{q=1}^{N}(W_q-X_q)^2} \quad (1)$$

O neuronie, dla którego odległość *Dij* będzie najmniejsza, mówi się, że zostaje on pobudzony wektorem wejściowym lub też że jest najbliższy obrazowi wejściowemu, w skrócie nazywa się go BMU (*ang. Best Maching Unit*). Zastanówmy się teraz, jak będzie wyglądało działanie sieci neuronowej? Na początku nauczymy sieć, jest to równoznaczne z doborem odpowiednich wartości wag dla każdego neuronu. Za pomocą wag każdy neuron będzie „pamiętał" kilka obrazów. Następnie sieci przedstawimy obraz, którego sieć nie widziała wcześniej, co spowoduje pobudzenie jednego neuronu - BMU. Ponieważ neuron ten „pamięta" kilka obrazów, to właśnie one według sieci będą najbardziej podobne do obrazu przedstawionego sieci. W ten sposób możemy wyszukiwać obrazy w bazie najbardziej podobne do obrazu pokazywanego sieci. Jednak aby cały system działał prawidłowo, sieć musi zostać wcześniej nauczona. Listing 3 pokazuje implementację sieci oraz funkcji uczącej i testującej.

Algorytm uczenia sieci

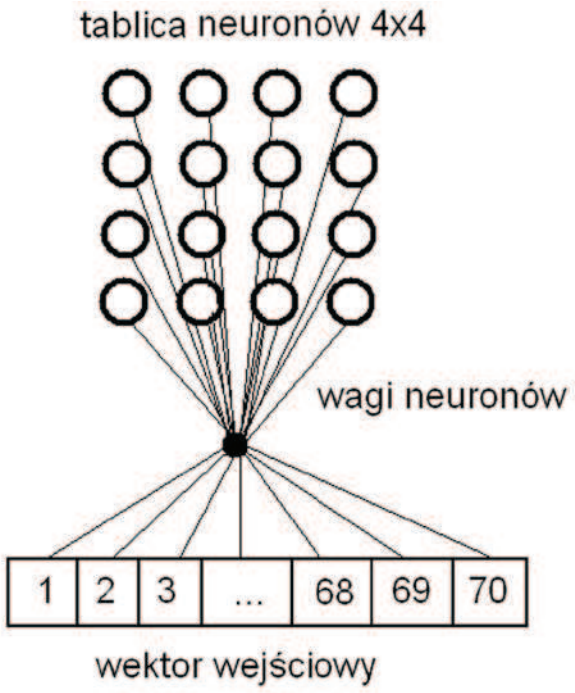
Uczenie sieci SOM jest uczeniem bez nauczyciela. Sieć sama przystosowuje się do danych uczących. Przed nauką wszystkie wagi w sieci są losowo inicjalizowane wartościami losowanymi z przedziału *(0; 1)*. Teraz zaczynana jest nauka sieci. Liczbą *t* będziemy oznaczać numer iteracji, natomiast *e* będzie oznaczać epokę uczenia. Jedna iteracja to pokazanie jednego obrazu, epoka uczenia to pokazanie sieci wszystkich obrazów ze zbioru uczącego. Ze zbioru obrazów uczących wybierany jest losowo jeden obraz, który nie był jeszcze pokazany w danej epoce, oznaczmy go jako *X(t)*. Następnie przedstawiany jest on sieci. Sieć odpowiada na sygnał wejściowy, wskazując jeden neuron zwycięski BMU, który daje największe pobudzenie. Oznaczmy współrzędne neuronu zwycięskiego jako *(bi, bj)*. Kolejnym krokiem jest aktualizacja wag sieci neuronu zwycięskiego oraz neuronów sąsiadujących według wzoru:

Może wydawać się on trochę zagmatwany, ale spokojnie,

$$W(t+1)=W(t)+L(e)\cdot S(i,j)\cdot(X(t)-W(t)) \quad (2)$$

przeanalizujemy go dokładnie i zobaczycie, że jest bardzo prosty. Najpierw wyjaśnię, co znaczą liczby *L(e)* i *S(i,j)*? *L(e)* jest to współczynnik uczenia sieci, który jest jednakowy dla wszystkich neuronów i zależy tylko od numeru epoki. *L(e)* maleje wraz ze wzrostem liczby epok. Zmniejszanie współczynnika uczenia ma na celu utrwalenie nauczonej informacji przez sieć. *L(e)* wyraża się wzorem:

$$L(e)=L_0\exp\left(-\frac{e}{\lambda}\right) \quad (3)$$



Rysunek 1. *Schemat struktury sieci neuronowej*

współczynnik λ przyjmiemy równy *10*, a *L0 = 0,2. S(i,j)* to funkcja sąsiedztwa, określa jak bardzo neurony oddalone są od neuronu zwycięskiego BMU. Wartość maksymalną *S(i,j)* przyjmuje dla neuronu zwycięskiego – co znaczy że to on najwięcej się uczy, jego wagi są najbardziej zmieniane. Natomiast im dalej od neuronu zwycięskiego tym wartości funkcji sąsiedztwa maleją. Funkcja sąsiedztwa opisana jest wzorem:

$$S(i,j)=\exp\left(\eta\sqrt{(B_i-i)^2+(B_j-j)^2}\right) \quad (4)$$

gdzie η przyjmiemy jako 0,1. Wartości λ i η zostały przyjęte za pomocą metody prób i błędów. Takie wartości zapewniają dobrą naukę sieci. Znając współczynniki *L(e)* i *S(i,j)*, wyznaczamy dla każdego neuronu wartości, o jakie należy zaktualizować wagi. Dodajemy je do aktualnych wartości wag sieci. W taki sposób otrzymamy wartości wag sieci, jakie będzie ona miała w następnej iteracji – t+1. W przedstawiony sposób pokazujemy sieci losowo wszystkie obrazki. Teraz warto zastanowić się, ile czasu należy uczyć sieć? Można uczyć sieć określoną z góry liczbą epok około 10-20 lub można obserwować, kiedy stan wiedzy sieci jest stabilny. Można stwierdzić, że sieć jest już nauczona, gdy dla tych samych obrazów sieć będzie zawsze pobudzała te same neurony. Jako warunek zatrzymania uczenia można przyjąć zdarzenie, kiedy sieć 3 razy pod rząd pobudzi te same neurony dla tych samych obrazów. Jeżeli zajdzie takie zdarzenie, to przerywamy uczenie. Aby móc skorzystać z tego warunku, musimy w czasie nauki zapamiętywać, dla jakich obrazów jakie neurony zostały najbardziej pobudzone. Dzięki takiej informacji po nauce sieci wiemy, jaki neuron jest pobudzany przez jaki obraz.

Cały algorytm nauki sieci można zapisać w punktach:

1. Inicjalizujemy wagi sieci wartościami losowanymi z przedziału *(0; 1)*.
2. Powtarzamy punkty 3-7, aż stan wiedzy sieci będzie stabilny.
3. Losowo wybieramy obraz, który nie był jeszcze pokazany w danej epoce, kodujemy go jako X(t).
4. Pokazujemy sieci wektor X(t) i wyznaczamy neuron najbardziej pobudzony BMU, zapamiętujemy, który neuron został pobudzony.
5. Aktualizujemy wagi wszystkich neuronów według wzoru (2).
6. Jeżeli pokazaliśmy już wszystkie obrazy, to zwiększamy licznik epok i aktualizujemy wartość współczynnika uczenia *L*.
7. Sprawdzamy, czy stan wiedzy sieci jest stabilny, jeżeli tak, to koniec nauki.
8. Po nauce prezentujemy sieci wszystkie obrazy i zapamiętujemy, który neuron jest najbardziej pobudzany dla którego obrazu.

Uczenie sieci możemy przeprowadzić jedynie na początku, a wagi sieci i informacje o tym, jakie neurony odpowiadają jakim obrazom, możemy zapisać na dysku. Listing 2 i 3 pokazuje implementację funkcji umożliwiających zapis i odczyt informacji o sieci.

Test sieci

Następnym krokiem jest sprawdzenie działania sieci. Wybieramy jakiś obraz, który nie był pokazywany sieci w czasie nauki, jednocześnie wiedząc, że jest podobny do niektórych obrazów, jakie brały udział w uczeniu. Pokazujemy go sieci. Sieć pobudza najmocniej jeden neuron, który odpowiada obrazom biorącym udział w uczeniu. Wskazujemy te obrazy jako najbardziej zbliżone wyglądem do obrazu wejściowego. W ten sposób wyznaczymy obrazy najbardziej podobne.

Podsumowanie

Opisane w artykule działanie sieci samoorganizujących się może posłużyć jako trzon systemu, który będzie rozpoznawał obrazy, wykorzystując sztuczną inteligencję. Dokonując w systemie kilku zmian, można go dostosowywać do trudności problemu, który chcemy rozwiązać. Wystarczy dokonać kilku modyfikacji w zależności od poziomu trudności. Można zmienić liczbę kodującą cechy obrazu lub zmienić liczbę neuronów w sieci czy też zastosować inne współczynniki w nauce sieci. Pamiętajmy, że stosując techniki sztucznej inteligencji, zawsze warto poeksperymentować, sprawdzić kilka różnych konfiguracji.

PIOTR PŁOŃSKI

Student Wydziału Elektroniki i Technik Informacyjnych Politechniki Warszawskiej. Autor interesuje się sztuczną inteligencją i bioinformatyką. Ulubionym językiem programowania jest C++. kontakt: pplonski86@gmail.com