Akademia Górniczo - Hutnicza im. Stanisława Staszica w Krakowie

Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej

Kierunek: Automatyka i robotyka

Wieloprocesorowe Systemy Wizyjne

Rozpoznawanie obiektów na obrazach w reprezentacji zdarzeniowej

styczeń 2018

Wykonali:
Marcin Kowalczyk
Jadwiga Piechota

Prowadzący:
dr inż. Mirosław Jabłoński

Przedmiot:	**Wieloprocesorowe Systemy Wizyjne.**			
			PR17wsw503	
Temat projektu:				
	Rozpoznawanie obiektów na obrazach w reprezentacji zdarzeniowej			
Wykonali:	Marcin	Kowalczyk	kowalczyk@agh.edu.pl	
	Jadwiga	Piechota	jadwiga.piechota@onet.pl	
	magisterski	rok V	Automatyka i Robotyka	
Rok akademicki	2017/18	zimowy		
Prowadzący	dr inż.	Mirosław	Jabłoński	

wersja 0.0

Kraków, styczeń, 2018.

Streszczenie

Celem projektu było zaprojektowanie i implementacja sieci neuronowej do rozpoznawania obiektów na obrazach w reprezentacji zdarzeniowej. Skorzystano z bazy danych *MNIST-DVS*, która zawiera obrazy cyfr zapisane w tej reprezentacji. W celu zmniejszenia ilości danych potrzebnych do przetworzenia oraz poprawy jakości obrazów wykonano proste przetwarzanie wstępne. Dane zostały następnie przekonwertowane do postaci, która może posłużyć do uczenia sieci neuronowej. Na koniec sprawdzono skuteczność w zależności od wielkości sieci.

Spis treści

1 Wstęp

1.1 Cel projektu

Celem projektu jest implementacja rozwiązania pozwalającego na rozpoznawanie obiektów na obrazach w reprezentacji zdarzeniowej. Idea reprezentacji zdarzeniowej polega na zaznaczaniu pikseli, dla których nastąpiła pewna zmiana w obrazie wejściowym kamery. Współrzędne wraz z typem zmiany(wzrost lub spadek wartości piksela) tego piksela następnie są wysyłane do zewnętrznego urządzenia, które przetwarza te dane. Metoda ta pozwala na zmniejszenie ilości danych, które potrzebne są do przesłanie z kamery i przetworzenia w innym urządzeniu. Projekt zakłada, że zapisane dane z reprezentacji zdarzeniowej są następnie przetworzone przez sieć neuronową zaimplementowaną w programie MATLAB.

1.2 Proponowane rozwiązanie

Zaproponowane i sprawdzone rozwiązanie wymaga wstępnego przetworzenia zarejestrowanych danych. Polega ono na agregowaniu współrzędnych pikseli z 10us. Dane te następnie formatowane jako obraz o rozmiarze 40×40px oraz jako wektor, którego kolejne elementy są kolejnymi elementami stworzonego obrazu. Stworzone w ten sposób wektory są danymi wejściowymi sieci neuronowej. Dane wyjściowe są zapisywane w postaci wektora one-hot. Jest to wektor, który ma na jednym miejscu jedynkę, a na pozostałych zera.

1.3 Alternatywne rozwiązania

2 Analiza złożoności i estymacja zapotrzebowania na zasoby

Zadanie projektowe zostało podzielone na dwie części. Pierwsza z nich dotyczyła odpowiedniego przygotowania danych, czyli wybrania odpowiednich zdarzeń z bazy *MNIST-DVS* i dostosowania ich do aktualnych potrzeb. Dane te zostały następnie użyte podczas właściwego działania algorytmu, czyli uczenia i testowania sieci neuronowej.

2.1 Przygotowanie danych

Cała baza [**baza**] zawierała bardzo dużo informacji. Nie było konieczności użycia tak dużej liczby wartości, dlatego zdecydowano się na wybranie około 10%. Liczba ta okazała się wystarczająca do zobrazowania wyników istotnych z punktu widzenia założonych celów. W celu znalezienia zbioru uczącego, napisano skrypt w programie *MATLAB 2017b*. Ta część zadania nie wymagała specjalnych zasobów obliczeniowych. Podstawowe wymagania systemowo - sprzętowe, potrzebne do poprawnego działania programu *MATLAB 2017b*, znalezione w [**wymagania_Matlab**], są następujące:

- system operacyjny: Windows 7 SP1 / Windows 8.1 / Windows 10
- procesor: Intel lub AMD x86-64 (rekomendowane wsparcie dla instrukcji AVX2)
- dysk: 4 - 6 Gb dla instalacji, 2 Gb wolnego miejsca dla prawidłowego działania programu
- RAM: co najmniej 2 Gb
- grafika: nie jest wymagana żadna konkretna karta graficzna, jednak polecana to taka, która ma wsparcie dla OpenGL 3.3 z 1 Gb pamięci GPU.

Zasoby pamięciowe, jakie dodatkowo musi posiadać komputer PC, na którym działa program, muszą pomieścić oryginalną bazę, która zajmuje około 9 Gb miejsca. Jednak już po przetworzeniu jej przez *MATLAB* zajmuje dużo mniej miejsca - to około 200 Mb. Aplikacja musi poradzić sobie z dużą ilością danych - na bazę składa się około 100 000 plików, przedstawiających zdarzenia przypadające na pewne przesunięcie danej liczby. Pliki te są zapisane w formacie .aedat, stworzonym przez twórców [**MNIST_DVS**]. Autorzy stworzyli specjalny skrypt, który pozwala na zamianę formatu .aedat na format, który może być odczytany i dalej przetwarzany przez *MATLAB* (format .mat). Wymaga to wykonaniu dodatkowego szeregu instrukcji przy każdorazowym przetwarzaniu pliku z bazy, a wyniki tu generowane są przedstawione w formacie double. To wszystko powoduje dość długi czas trwania obliczeń, dlatego im lepsze parametry ma komputer PC, na którym wykonywane są obliczenia, tym wyniki uzyskiwane są sprawniej. Po wykonaniu obliczeń, uzyskuje się dwie macierze, wykorzystywane później do uczenia sieci. W obu wartościami są liczby boolean. Ponieważ nie ma tutaj żadnych zależności z pozostałymi modułami (baza została przygotowana wcześniej i nie koliduje z procesem uczącym), nie ma znaczenia na jakim komputerze będą wykonywane obliczenia. Możliwości sprzętowe powodują jednak, że efektywność pracy rośnie.

2.2 Uczenie sieci neuronowej

Właściwa część algorytmu, czyli uczenie i testowanie sieci neuronowej została również wykonana w programie *MATLAB 2017b*. Wymagania sprzętowo - programowe są tu

więc identyczne co te, wymienione w sekcji 2.1. Jednak tutaj zastosowano dwa różne podejścia do tematu.

Pierwsze podejście zakładało użycie gotowego GUI programu *MATLAB - Neural Network Toolbox*, stworzonego do uczenia i testowania sieci neuronowej. Jedynym wymaganiem był tu dostęp do tego narzędzia z poziomu programu *MATLAB*.

Drugie podejście polegało na stworzeniu sieci neuronowej za pomocą funkcji programu *MATLAB*. Dodatkowo, w celu przyspieszenia obliczeń, wykorzystano tutaj GPU (z ang. *graphics processing unit*). Aby taki algorytm mógł działać, niezbędna jest tutaj karta graficzna NVIDIA wyposażona w jednostkę obliczeniową, obsługująca architekturę obliczeniową CUDA (z ang. *Compute Unified Device Architecture*). Takie podejście jest uzasadnione, ponieważ uczenie sieci neuronowych to proces równoległy. Zatem zwiększenie zrównoleglenia obliczeń poprzez użycie dodatkowego procesora znacznie usprawni przebieg działania i spowoduje, że testowanie, polegające na wielokrotnym uczeniu sieci z różnymi parametrami będzie szybsze, a cały proces bardziej efektywny. Wydajność w tym przypadku wzrasta w znacznym stopniu.

W obu podejściach jako wektory wejściowe podane są macierze:

- danych - 94 490 $\times$ 1600
- wyjść - 94 490 $\times$ 10

W obu przypadkach w pierwszym podejściu były to wartości logiczne, w drugim zostały zapisane jako typ *double*.

3 Koncepcja proponowanego rozwiązania

Pierwsza część pracy polegała na stworzeniu bazy danych, na których następnie można było uczyć sieć neuronową. Jak zostało wspomniane w sekcji 2, około 10% danych dostarczonych przez autorów bazy *MNIST-DVS* w źródle [**MNIST_DVS**] zostało wykorzystanych w niniejszej pracy. Ponieważ rozważano dwie koncepcje, dane zostały przygotowane do realizacji obu w podobny sposób, jednak na końcu zostały zapisane w innej formie, co zostanie omówione w późniejszej części rozdziału.

Dane zostały wybrane w sposób losowy, odrzucono tylko początkowe wartości, które mogłyby być nieco zaszumione i wprowadzać niepotrzebne rozbieżności. Mając już tak przygotowany zestaw danych, zauważono pewną powtarzalność w przedstawianiu zdarzeń. Otóż zdarzenia przedstawiające zmianę położenia liczby znajdują się w podobnym miejscu (jak już zostało wspomniane, nie cały zestaw danych, czyli zbiór zdarzeń został wzięty pod uwagę). Dzięki temu można było znacząco zmniejszyć wymiary danych wejściowych do rozmiaru 40×40. Taki zabieg przyspieszył i tak już zaawansowane obliczenia. W dalszej kolejności stworzono tablicę wartości boolean o rozmiarze 40×40 i przepisano tam wartość 1 w miejscach, gdzie wystąpiły zdarzenia. W ten sposób powstało 94 490 takich tablic (po około 10 000 na jedną cyfrę). Braki wynikają z błędów twórców bazy *MNIST-DVS*, ale zostały wyeliminowane w procesie działania zaprezentowanego algorytmu. Ponieważ uczenie sieci neuronowej zastosowanej w omawianej metodzie następowało z nauczycielem, potrzebne było stworzenie macierzy wyjść. Macierz do tego utworzona miała tyle wierszy, ile tablic wejściowych, zaś kolumn tyle co cyfr, czyli 10. Wartość 1 w danej kolumnie informowała tu z jaką cyfrą mamy w tym przypadku do czynienia - numer kolumny wskazywał cyfrę pomniejszoną o jeden (pierwsza kolumna dla 0).

Uczenie sieci odbyło się równolegle na dwa sposoby. Pierwszy wymagał przedstawienia danych w postaci wierszowej. Stworzono więc macierz mającą tyle wierszy, ile danych wejściowych i tyle kolumn, ile wartości przypadało na każdą daną, czyli 1600. Drugi sposób pozwalał na stworzenie tablicy celek. Dane wejściowe były przedstawione w taki sposób, że każda celka była tablicą 40×40, czyli jedną daną. Etykiety stworzono bazując na macierzy wyjść. Każda celka przestawiała tu wektor o rozmiarze 10, w którym na odpowiednim miejscu, oznaczającym daną cyfrę, znajdowała się wartość 1. Synchronizację otrzymano poprzez numer celki w danych wejściowych odpowiadał numerowi celki w tablicy wyjściowej zawierającej etykiety.

Pierwsze podejście uczenia sieci zakładało użycie GUI programu *MATLAB - Neural Network Toolbox*, a dokładnie jego części odpowiedzialnej za rozpoznawanie wzorców i klasyfikację danych - *nprtool*. Po wprowadzeniu tu wartości wejściowych i etykiet, ustala się liczbę neuronów ukrytych i można już przejść do nauki sieci neuronowej. *MATLAB* sam ustala tutaj liczbę danych potrzebnych do uczenia, walidacji i testowania. Program używa tutaj sieci neuronowej z użyciem algorytmu wstecznej propagacji błędów (z ang. *backpropagation*). Algorytm uczenia kończy działanie po udanej sześciokrotnej walidacji. Wyniki testowania można zaobserwować na stworzonych przez GUI statystykach.

Drugie podejście zostało stworzone po to, aby móc zastosować głęboką sieć neuronową. Uczenie tej sieci odbywa się używając macierzy wejściowej i etykiet wyjściowych, czyli jest to uczenie z nauczycielem. Dodatkowo wprowadzone parametry pozwalają na zastosowanie równoległości obliczeń na wielu rdzeniach oraz użycie procesora graficznego GPU. Zastosowanie jest uzasadnione ze względu na dużą liczbę danych wejściowych, co zostało omówione w sekcji 2. To wszystko pozwala na przyspieszenie tego etapu uczenia. Uzasadnione jest tu użycie autoenkodera w celu dodania warstw ukrytych. Użyto

tutaj GPU dodając odpowiedni parametr w funkcji *trainAutoencoder*, ustalona została też maksymalna liczba epok. W celu stworzenia pełnej sieci, dodano warstwę wyjściową. Uwzględniając fakt, że jest to problem klasyfikacyjny, stworzono funkcję aktywacji typu *softmax*. Tak nauczoną sieć poddano testowaniu. Wyniki uzyskane nieco różnią się od wyników uzyskanych w podejściu wcześniejszym, co zostanie omówione w sekcji 4.

4 Symulacja i testowanie

4.1 Modelowanie i symulacja

4.2 Testowanie a weryfikacja

5 Rezultaty i wnioski

6 Podsumowanie

7 DODATEK A: Szczegółowy opis zadania

7.1 Specyfikacja projektu

Tematem projektu było rozpoznawanie obiektów na obrazach w reprezentacji zdarzeniowej. Postawione zadanie wymagało znalezienia odpowiednich danych, które zostały przedstawione poprzez następujące po sobie zdarzenia. Podejście to nieco różni się od standardowego, trudno byłby wygenerować zbiór uczący w ramach realizacji tego zadania, bo to już jest temat na inny projekt. W literaturze poleconej przez prowadzącego znaleziono metodę, której wynikiem była baza *MNIST-DVS*. Baza ta została odnaleziona w [**baza**] i wykorzystana do uczenia sieci neuronowej. Celem, jaki został postawiony, było nauczenie sieci neuronowej, bazując na zdarzeniowej reprezentacji danych, aby była w stanie rozpoznać 10 cyfr: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Wybór rodzaju sieci oraz platform sprzętowo - programowych prowadzący zostawił autorom projektu.

7.2 Szczegółowy opis realizowanych algorytmów przetwarzania danych

Sekcja ta będzie rozszerzeniem informacji zawartych już wcześniej w paragrafie 3. Zastosowane algorytmy zostaną tu omówione bardziej od strony matematycznej.

Przygotowanie danych Przetwarzanie bazy *MNIST-DVS* zostało napisane w programie *MATLAB*. Było to konieczne, ponieważ autorzy artykułu [**MNIST_DVS**] zastosowali rozszerzenie .aedat. Dostarczyli dodatkowo skrypt *dat2mat*, który pozwalał na zamianę tych danych na pliki z rozszerzeniem .mat, które można już w łatwy sposób podglądnąć i przetworzyć. W tym przypadku zastosowano przetwarzanie sekwencyjne, tworząc skrypt, który zamieniał rozszerzenie i dostosowywał bazę do realizacji postawionych w projekcie celów.

Uczenie sieci Uczenie sieci przebiegało z użyciem wbudowanych funkcji programu *MATLAB*, co znacznie ułatwiło pracę. Dane były przetwarzanie współbieżnie, wykorzystując rdzenie procesora CPU oraz procesor graficzny GPU. Pierwszy sposób, uwzględniający GUI do uczenia sieci neuronowych, używał algorytmu wstecznej propagacji błędów i będzie omówiony w sekcji 7.2. Drugi sposób polegał na wykorzystaniu wbudowanych funkcji do uczenia maszynowego i używał głębokich sieci neuronowych do klasyfikacji danych.

Algorytm wstecznej propagacji błędów To podstawowy algorytm uczenia nienadzorowanego wielowarstwowych sieci neuronowych. Zaletą tej sieci jest to, że wagi można tu wytrenować, znajdując ich optymalny zestaw [**back**]. Metoda umożliwia modyfikację wag w sieci o architekturze warstwowej, we wszystkich jej warstwach. Po ustaleniu topologii, początkowe wagi są inicjowane tu losowo. Przyjmują bardzo małe wartości. Następnie dla danego wektora uczącego oblicza się odpowiedź sieci, warstwa po warstwie, stosując algorytm spadku gradientowego [**back2**]. Każdy neuron wyjściowy oblicza swój błąd, który następnie jest propagowany do wcześniejszych warstw. Następnie każdy neuron modyfikuje wagi na podstawie wartości błędu. Dodatkowo jest tu wprowadzona powtarzalność, czyli gdy wszystkie dane wejściowe zostaną już użyte, zmieniana jest ich kolejność i ponownie są wprowadzana do sieci. Proces trwa do momentu zatrzymania się średniego błędu kwadratowego. Jest to proces dość kosztowny obliczeniowo, zwłaszcza kiedy sieć jest rozbudowana.

Głębokie sieci neuronowe Uczenie głębokie sieci neuronowej następuje krok po kroku. Pozwala na stopniowe wyznaczenie wag dla poszczególnych warstw sieci w celu innej reprezentacji cech wspólnych. To poszczególne warstwy reprezentują tu cechy wspólne wzorców uczących i na tej podstawie tworzą reprezentacje bardziej skomplikowanych cech w kolejnych warstwach sieci głębokich. Jest to ulepszona metoda niż wielowarstwowe sieci neuronowe uczone algorytmem propagacji wstecznej, w której w warstwach oddalonych od wyjścia sieci sieć ma tendencję do dokonywania coraz mniejszych zmian [**deep**]. Tutaj sieć jest rozbudowywana powoli o kolejne warstwy dopiero wtedy, gdy w poprzednich warstwach pojawiły się takie cechy. Stosuje się tu sieci typu *autoencoder*, które używając aproksymacji identycznościowej wykorzystują warstwę ukrytą składającą się z mniejszej ilości neuronów niż ilość wejść lub wyjść z sieci. W przypadku głębokich sieci neuronowych trzeba uważać na to, aby sieć nie dopasowała się zbyt mocno do danych uczących, ponieważ na danych testowych może później niepoprawnie działać.

W przypadku realizacji przestawionego projektu wykorzystano głębokie sieci neuronowe do uczenia nadzorowanego, ponieważ rozważano problem klasyfikacji. Do realizacji głębokiego uczenia użyto tutaj autoenkodera, jako funkcja aktywacji posłużyła w warstwie wyjściowej funkcja typu *softmax*. Dokładny opis procedur znajduje się w sekcji 8.

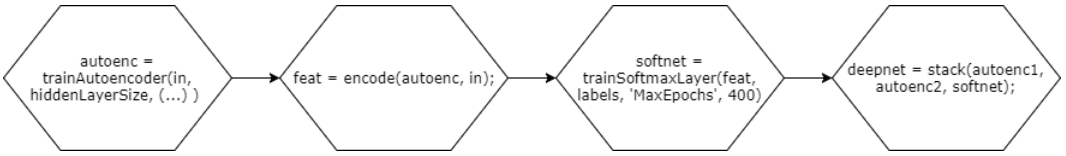
8 DODATEK B: Dokumentacja techniczna

Projekt został napisany w programie *MATLAB 2017b*. Korzystano z gotowego GUI programu *MATLAB - Neural Network Toolbox* oraz innych funkcji z tego *toolbox*. Baza danych potrzebna do realizacji postawionego zadania została pobrana z źródła.

8.1 Dokumentacja oprogramowania

Wszystkie funkcje użyte na potrzeby zaimplementowania algorytmu zostały zaczerpnięte z gotowych rozwiązań. Opis i dokumentacja techniczna GUI *Neural Network Toolbox* znajduje się w źródle [**gui**], zaś do funkcji użytych w procesie tworzenia głębokich sieci neuronowych w źródle [**funkcje**].

Kolejność użytych funkcji przedstawiono na schemacie 8.1.



Rys. 8.1: Schemat blokowy użytych funkcji do głębokiego uczenia sieci neuronowej.

Funkcja *trainAutoencoder* tworzy ukryte warstwy. Jako parametry przyjmuje dane wejściowe, ilość ukrytych warstw oraz parametry określające dynamikę i właściwości wag. Dodatkowo można tu zdecydować, czy obliczenia mają być wykonywane również na procesorze GPU. Funkcja *encode* służy do ekstrakcji cech z ukrytych warstw, a jako parametry przyjmuje dane wejściowe oraz wynik działania autoenkodera. Funkcja *trainSoftmaxLayer* tworzy funkcę aktywacji typu softmax, bazując na cechach zwróconych po wywiłaniu funkcji *encode* oraz etykietach danych wejściowych. Określa się tutaj także maksymalną liczbę epok. Ostatnia funkcja *stack* układa wszystkie warstwy, łącząc stworzone autoenkodery oraz warstwę wyjściową. Funkcja ta tworzy głęboką sieć neuronową, która może być już użyta do testowania.

8.2 Procedura symulacji, testowania i weryfikacji

Do realizacji projektu nie używano żadnej zewnętrznej platformy sprzętowej. Aby uruchomić aplikację wystarczy komputer PC spełniający wymagania sprzętowo - programowe opisane w sekcji 2.1 z zainstalowanym programem *MATLAB 2017b*. Potrzebny jest także *Neural Network Toolbox*, który będzie wbudowany w przypadku pełnej instalacji programu. W celu przygotowania danych do uczenia można pobrać bazę ze źródła [**baza**] i uruchomić skrypt *Data_selection* znajdujący się na nośniku CD. Można również wczytać przygotowane już dane w formacie .mat: data_matrix.mat oraz images_double.mat znajdujące się również na nośniku CD. W celu weryfikacji algorytmu należy skorzystać z *Neural Network Toolbox*, wybierając opcję *Pattern recognition and classification*. Jako wejście należy wczytać macierze input_matrix oraz output_matrix. Druga opcja zakłada skorzystanie z głębokiego uczenia z użyciem większej liczby warst ukrytych. W tym celu trzeba uruchomić skrypt *GPU_Mnist* (płyta CD). Należy zaznaczyć, że proces trwa dosyć długo, a liczenie przebiega tu w sposób równoległy i z użyciem procesora graficznego GPU.

9 DODATEK C: Spis zawartości dołączonego nośnika (płyta CD ROM)

Struktura folderów nośnika wygląda następująco:

- ◇ DOC - raport z projektu w formacie PDF
- ◇ SRC - zawiera skrypty źródłowe, Matlab2017b
  - dat2mat - stworzony przez autorów bazy [**MNIST_DVS**], pozwala na zmianę rozszerzenie plików .aedat na format .mat
  - Data_selection - skrypt stworzony do przygotowania danych do uczenia sieci neuronowej
  - GPU_Mnist - algorytmy uczenia i testowania głębokiej sieci neuronowej użyciem równoległych obliczeń i procesora GPU
- ◇ TEST - znajdują się tu zarówno pliki do uczenia i testowania sieci neuronowej, jak i zapis z *workspace* po wytrenowaniu sieci (trained.mat).

10 DODATEK D: Historia zmian

Tabela 1: Historia zmian

Autor	Data	Opis zmian	Wersja
Kowalczyk & Piechota	2018-01-07	Stworzenie pierwszej wersji projektu	0.0

PR17wsw503			**Karta oceny projektu**
Data	Ocena	Podpis	Uwagi