

AKADEMIA GÓRNICZO-HUTNICZA



im. Stanisława Staszica w Krakowie

LOKALIZACJA TWARZY NA OBRAZIE ZA POMOCĄ
KONWOLUCYJNYCH SIECI NEURONOWYCH

Autorzy:

Bartłomiej BUŁAT
Tomasz CZARNIK
Krzysztof ŚMIŁEK

Opiekun projektu:

dr inż. Mirosław JABŁOŃSKI

Wydział Elektroniki, Automatyki, Informatyki i Elektrotechniki
Katedra Automatyki
Laboratorium Biocybernetyki

18 czerwca 2011

1 Streszczenie

Celem naszego projektu jest zbudowanie prototypu systemu pozwalającego na detekcję twarzy na obrazie w oparciu o sieć neuronową typu CNN (Convolutional Neural Network).

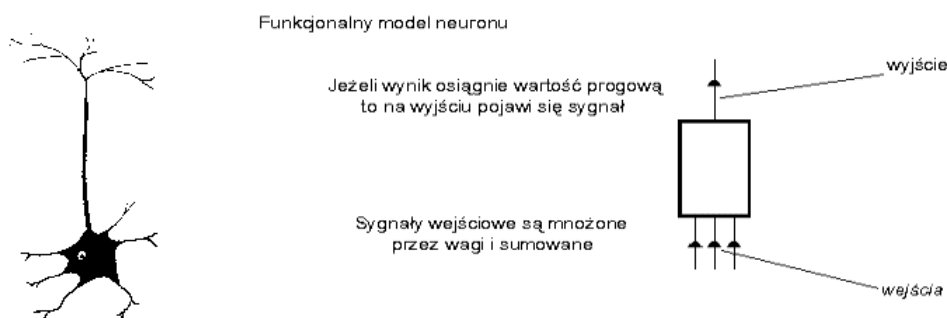
2 Wstęp

2.1 Co to jest sieć neuronowa?

Sieć neuronowa jest modelem matematycznym składającym się z sieci węzłów obliczeniowych zwanych neuronami i ich połączeń. Jest to pewna technika obliczeniowo-statystyczna, należąca do dziedziny sztucznej inteligencji. Jej zadaniem jest symulacja działania ludzkiego mózgu.

2.2 Budowa sieci neuronowej

Nasz mózg jest zbudowany z ogromnej ilości neuronów które komunikują się ze sobą za pomocą impulsów elektrycznych. Z informatycznego punktu widzenia neuron stanowi układ przetwarzający pewne dane, posiadający wiele wejść oraz jedno wyjście. Oto porównanie neuronu naturalnego ze sztucznym:



Działanie sztucznego neuronu można opisać w przybliżeniu następująco:
Sygnały x_i podane na wejścia są mnożone przez odpowiednie wagi w_i oraz sumowane. Wagi mogą przyjmować dowolne wartości rzeczywiste. Równanie neuronu wygląda zatem następująco:

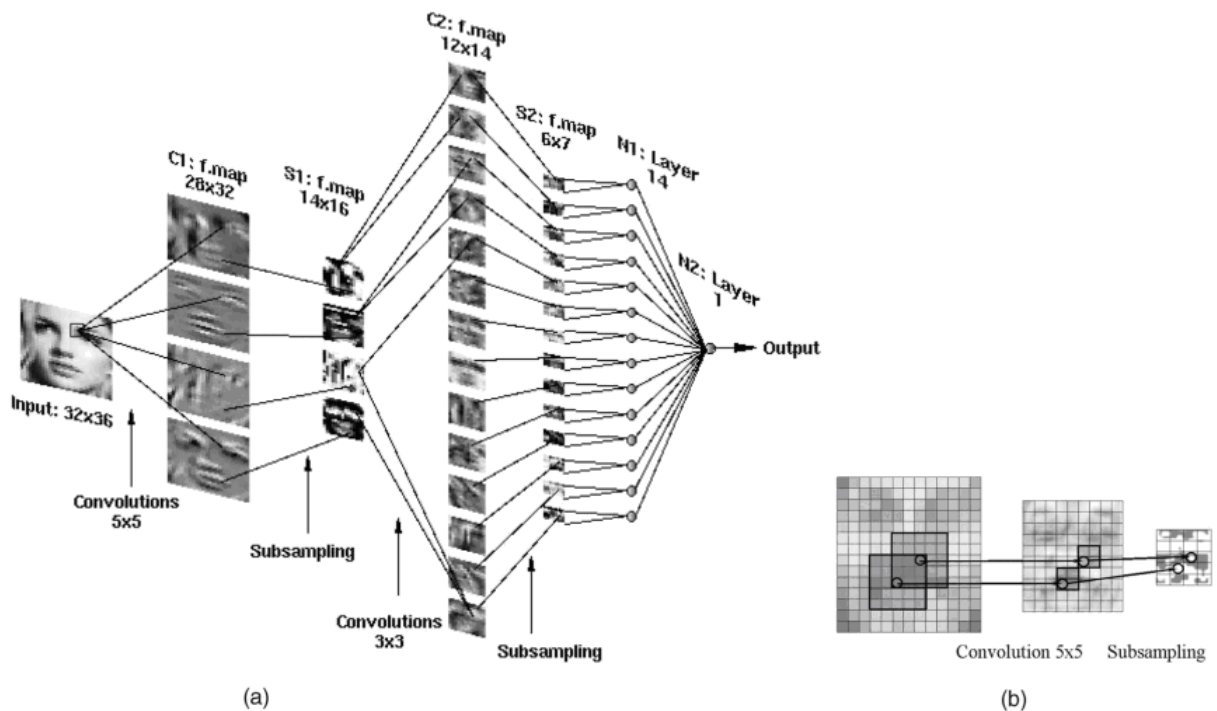
$$Y = x_1w_1 + x_2w_2 + x_3w_3 + \dots + x_nw_n + b$$

Suma ta staje się następnie argumentem funkcji aktywacji, która zwraca wartości z zakresu $(-1,1)$. Wynik tej funkcji pojawia się zarazem na wyjściu z neuronu.

Jak zapewne możemy zauważyć pojemność pojedynczego neuronu nie jest duża, a więc nie może on zapamiętać zbyt dużej ilości wzorców. Aby tego dokonać należy połączyć wiele pojedynczych neuronów w sieć. Dokonuje się tego tworząc kolejne warstwy. Czym więcej warstw tym sieć bardziej złożona. Ilość neuronów w poszczególnych warstwach może się różnić i zależy od konkretnego zastosowania sieci. Należy do tego dodać, że neurony w jednej warstwie nie są połączone ze sobą, natomiast neurony z dwóch sąsiednich warstw są połączone "każdy z każdym". Sygnał przechodzi do warstwy wejściowej poprzez ukrytą aż do wyjściowej, skąd jest odbierany i interpretowany.

3 Proponowane rozwiązanie

3.1 Konwolucyjne znajdowanie twarzy



Sieć ta składa się z sześciu warstw. Do warstw nie wlicza się obraz wejściowy o rozmiarach 32x36, który będzie klasyfikowany jako „twarz” lub „nie-twarz”.

Warstwy od **C1** do **S2** składają się z serii obrazów na których są przeprowadzane operacje konwolucji lub subsamplingu. Obrazy te są nazywane inaczej mapami cech, ponieważ każdy z nich wydobywa z obrazu wejściowego inną cechę. Warstwa **N1** zawiera szereg niezależnych od siebie neuronów podłączonych wyjściami do **N2**, która jest zarazem zakończeniem całej sieci neuronowej. Każdy element w danej warstwie otrzymuje na wejście informacje od małego zbioru elementów z warstwy poprzedniej. Koncepcję tę pokazano na rysunku (b).

Postaramy się teraz opisać dokładniej poszczególne elementy tworzące przedstawioną powyżej architekturę sieci neuronowej. Różne parametry opisujące całą sieć tj. ilość warstw, ilość elementów w warstwie, rodzaj połączeń między nimi a także wielkość poszczególnych elementów została wybrana doświadczalnie przez C. Garcia i M. Delakis. Przetestowali oni szeroki zakres wielu architektur sieci i przedstawiona powyżej okazała się najbardziej skuteczna zarówno pod względem wykrywalności twarzy jak i odporności na zakłócenia obrazu.

Warstwa **C1** jest złożona z czterech map cech. Rozmiar każdej z nich to 28x32 pikseli. Każdy piksel w każdej mapie jest połączony z 5x5 sąsiedztwem z obrazu wejściowego i jego wartość odpowiada sumie ważonej tego sąsiedztwa (5x5). Dodatkowo do sumy tej jest dodawana dodatkowa zmienna trenowana zwana biasem. Podsumowując, w warstwie tej znajduje się 106(4x26) trenowanych zmiennych.

Warstwa **S1** złożona jest z czterech map cech. Każda z map powstała poprzez subsampling mapy z poprzedniej warstwy z maską 2×2 , przez co jej rozmiar to połowa rozmiaru poprzedniego. Z każdych 4 sąsiednich pikseli maski **C1** jest brana średnia, która jest następnie mnożona przez trenowalną zmienną. Do otrzymanej wartości dodawany jest bias, po czym wynik staje się argumentem do funkcji aktywacji (tangensa hiperbolicznego). Funkcja ta używana jest po to, by uniknąć wykroczenia poza zakres zmiennych. Podsumowując, warstwa **S1** składa się z 4 map cech o rozmiarze 14×16 oraz $8(4 \times 2)$ trenowalnych parametrów.

Jak można zauważyć na rysunku (**a**), kolejne warstwy składają się z coraz większej ilości map cech, ale za to ich rozdzielczość się zmniejsza. Taki rodzaj systemu sieci został zaproponowany przez Hubela i Weisela i dobrze wyodrębnia on z obrazu cechy potrzebne do późniejszego przetwarzania przez neurony.

Warstwa **C2** to konwolucyjna warstwa z 14 mapami cech. Pierwsze 8 powstało w wyniku konwolucji z maską 3×3 map z poprzedniej warstwy (każda mapa z **S1** produkuje 2 wyjścia). Pozostałe 6 to złożenia dwóch z czterech map z **S1** ($\binom{4}{2} = 6$ kombinacji). Oto schemat przykładowego złożenia map o numerach 1 i 4:

$$Y = maska_{1_{3 \times 3}} * mapa_1 + maska_{2_{3 \times 3}} * mapa_2 + bias$$

Podsumowując, warstwa ta składa się z 14 map cech o rozmiarze 12×14 pikseli każda oraz z $196(8 \times (3 \times 3 + 1) + 6 \times (2 \times 3 \times 3 + 1))$ trenowalnych zmiennych.

Warstwa **S2** to warstwa subsamplingu analogiczna do warstwy **S1**. Zmienia się tylko ilość i rozmiar map cech. Dostajemy zatem: 14 map o rozmiarach 6×7 oraz $28(4 \times 2)$ trenowalnych zmiennych.

Ostatnie 2 warstwy składają się już z neuronów. Ich zadaniem nie jest ekstrakcja cech jak miało to miejsce w poprzednich warstwach, ale ich klasyfikacja. W **N1** każdy z pojedynczych neuronów jest w pełni połączony z wszystkimi pikselami tylko jednej odpowiadającej mu mapy cech z **S2**. Natomiast w warstwie **N2** istnieje tylko jeden neuron, który łączy się z wyjściami ze wszystkich neuronów z **N1**.

Wszystkie neurony z warstw **N1** i **N2** wykonują klasyczny iloczyn skalarny pomiędzy wektorem wejściowym a wektorem wag, do czego później dodawany jest jeszcze bias. Powstała ważona suma jest następnie przepuszczana przez funkcję tangensa hiperbolicznego, po to, by wyjście z neuronu stanowiła liczba z zakresu $(-1, 1)$.

Wyjście z ostatniej warstwy **N2** jest używane do klasyfikacji obrazu jako twarz ($result > 0$) lub nie-twarz ($result < 0$). Warstwy **N1** i **N2** mają zatem odpowiednio $602(14 \times 43)$ i 15 trenowalnych zmiennych.

3.2 Trenowanie sieci

Stworzyliśmy zestaw szkoleniowy wybierając ponad 800 obrazków zawierających twarze i nie-twarze. Zdjęcia pozyskaliśmy przeważnie z internetowych baz danych które udostępniały twarze w różnych perspektywach. Naszym zadaniem było ich wyselekcjonowanie i odpowiednie przycięcie, tak by do sieci neuronowej wprowadzić jasno i czytelnie które zdjęcia powinien zakwalifikować jako twarz, a które nie. Zdjęcia staraliśmy się także dobierać w ten sposób aby skutecznie wyłapać różne cechy na obrazach, tak aby nasz system stał się jak najbardziej niezawodny i aby wykrywał twarze z jak największą skutecznością.

Sam proces przycinania obrazków polegał na tym, aby obrazek przeskalować, bądź przyciąć do wielkości 32×36 px. Przy czym głównym celem było to, aby usta i oczy wybranych osób znajdowały się na każdym obrazie mniej więcej w tym samym miejscu. Odległość pomiędzy ustami

a oczami jest bowiem znacząca, ponieważ pozwala ona na zachowanie odpowiednich proporcji twarzy. Proporcje te to : odległość między oczami (około 16px), a między oczami i ustami (18px).

Zbieranie zbioru nie-twarzy okazało się o wiele trudniejsze niż sądziliśmy. Praktycznie każdy obraz nie zawierający twarzy mógłby posłużyć jako obraz testowy. Jednak w tym wypadku proces testowania jest mniej dokładny. Większą dokładność uzyskuje się podczas testowania obrazków które nie są twarzami, ale je znacząco przypominają. Jednak taki zbiór obrazów jest z oczywistych względów bardzo trudny do pozyskania, przez co posłużyliśmy się do testów obrazami losowo wyciętymi z większych zdjęć przedstawiających krajobrazy.

Sam proces trenowania przedstawia się natomiast w następujących krokach:

1. Stworzenie zbioru 400 obrazów przedstawiających twarz i 400 nie zawierających jej.
2. Ustawienie zmiennych: $\text{BootsIter} = 0$
3. Wybranie spośród zbioru utworzonego w (1) twarzy po 300 obrazów z twarzą i 300 bez.
4. Trenowanie sieci przez około 10 trenowalnych epok. W każdej epoce używana jest taka sama liczba pozytywnych i negatywnych przykładów. Trenowanie odbywa się natomiast metodą wstecznej propagacji błędów ze współczynnikiem uczenia (od 0.005)
5. Inkrementacja zmiennej $\text{BootsIter} = \text{BootsIter} + 1$;
6. Jeśli $\text{BootsIter} < 6$ to idź do (3)
7. Trenowanie sieci przez kolejne 10 epok, ale już dla całego zbioru obrazów.

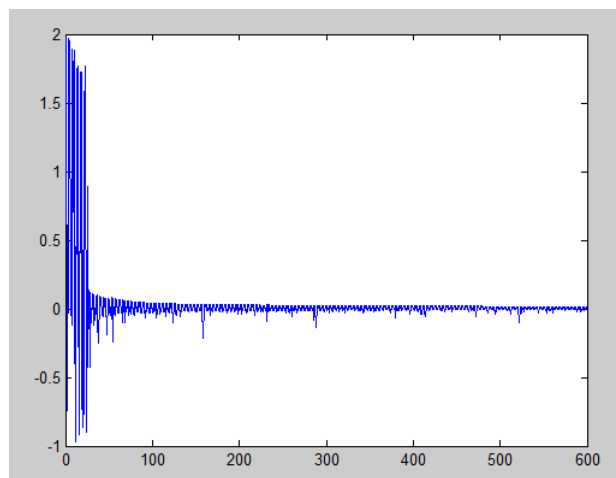
Przedstawiony wyżej algorytm został dokładniej opisany w artykule [1]. Do końca nie udało nam się go w pełni zrozumieć (dlaczego akurat taka a nie inna liczba iteracji itd), jednakże ufamy w pełni jego twórcom.

3.3 Testowanie

Proces testowania rozpoczęliśmy wykonując wcześniej wspomniany algorytm krok po kroku. Jednakże podczas nauki za pomocą metody wstecznej propagacji błędów zauważyliśmy, że nauka ta nie przynosi żadnych rezultatów.

Po konsultacji z prowadzącym doszliśmy wszyscy do wniosku, że wina najprawdopodobniej leży w tym, że twarz jest sama w sobie jest dosyć skomplikowanym obiektem, o mnóstwie cech. Postanowiliśmy zatem, że zamiast wykrywać twarze będziemy wykrywać na obrazach cyfry, które są prostsze do opisania, ponieważ posiadają mniejszą liczbę własności. Ponownie zatem stworzyliśmy bazę obrazów z cyframi i wykonaliśmy wspomniany algorytm dla nowych danych. Jak się okazało rezultaty były zaskakujące. Dla tej samej budowy sieci program zaczął rozróżniać cyfry.

Wykres uczenia się systemu dla przeprowadzonych testów prezentuje się następująco:

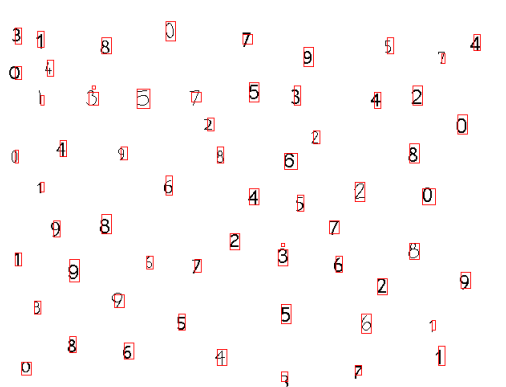


Jak możemy zauważyć, z każdą przeprowadzoną iteracją wartość błędu obliczeń się zmniejsza, co symbolizuje że sieć uczy się prawidłowo.

Tak wyuczoną sieć można już poddać głębszym testom. Postanowiliśmy tego dokonać tworząc obraz testowy:



Po przeprowadzeniu testu otrzymaliśmy zadowalający obraz wynikowy, który potwierdza, że stworzona przez nas sieć neuronowa działa i nadaje się do wykrywania prostych obiektów np. liczb. Oto otrzymany obraz wynikowy:



4 Rezultaty i wnioski

5 Podsumowanie

6 Literatura

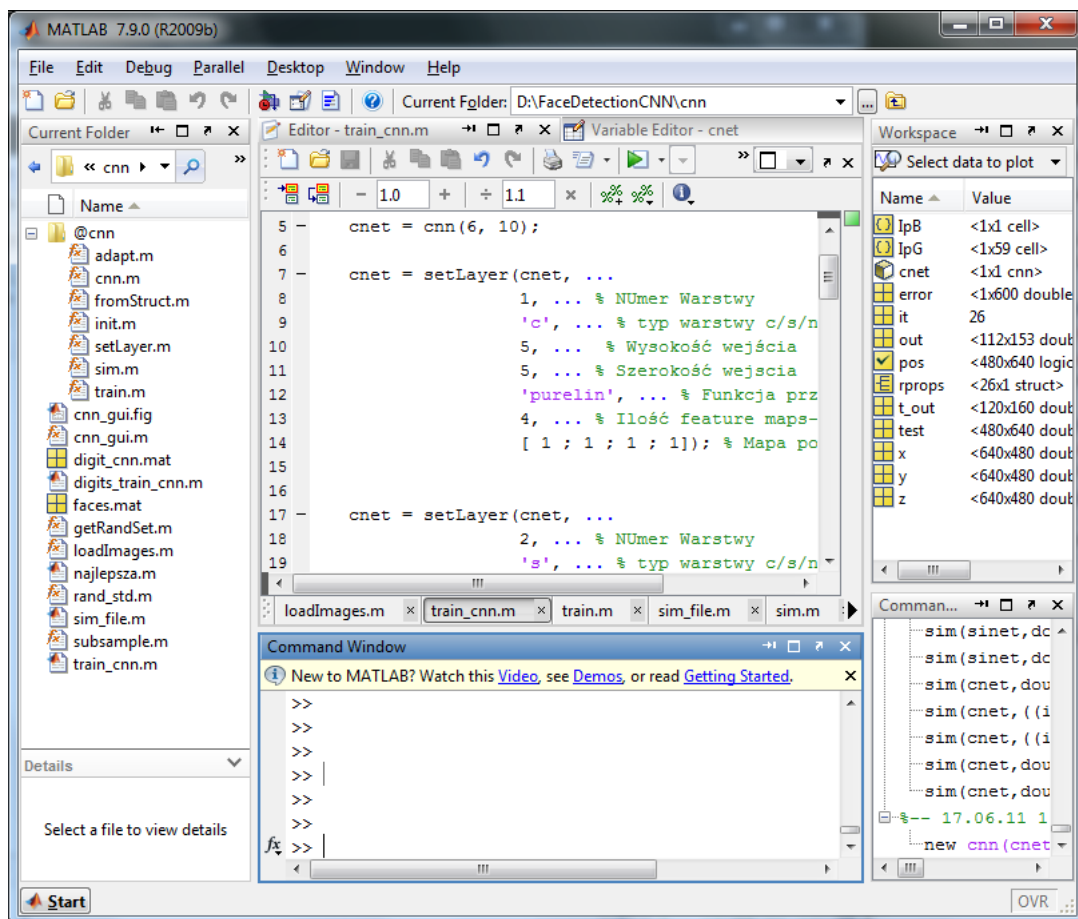
1. Convolutional Face Finder: A Neural Architecture for Fast and Robust Face Detection, Christophe Garcia and Manolis Delakis
2. Convolutional neural networks for image processing, Matthw Browne, Saeed Shiry Ghidary
3. An Embedded Robust Facial Feature Detector, Sebastien Roux, Christophe Garcia
4. Sieci Neuronowe, Ryszard Tadeusiewicz

7 Dodatek A: Opis narzędzi i metod postępowania

7.1 Implementacja algorytmu

Działanie algorytmu wdrażaliśmy w środowisku MathWorks Matlab R2009b. Zdecydowaliśmy się na nie ze względu na mnogość łatwo dostępnych funkcji wspomagających obliczenia, szybkość implementacji oraz to iż w ciągu studiów nabyliśmy sporo doświadczenia przy pracy w nim, szczególnie przy obróbce grafiki.

Matlab posiada wbudowane toolkity wspomagające pracę nad sieciami neuronowymi, ale niestety nie ma wśród nich obsługi konwolucyjnych sieci neuronowych. Dlatego też rozpoczęliśmy poszukiwania potrzebnych narzędzi w Internecie. Znaleźliśmy projekt Mikhail Sirotenki¹ w którym wykorzystane zostały konwolucyjne sieci neuronowe do rozpoznawania odręcznie napisanych cyfr. Klasa odpowiedzialna za sieci neuronowe została napisana przez Yann LeCun'a. Początkowo modyfikowaliśmy skrypt dla naszych potrzeb, jednak po jakimś czasie uznaliśmy, że lepiej będzie go przepisać na nowo opierając się na pierwotnym z zachowaniem struktury klasy. Zrzut ekranu okna programu Matlab:



Z prawej strony widać strukturę plików projektu. Folder @cnn zawiera pliki klasy cnn, zawierającą między innymi konstruktor, funkcję treningową (train.m), funkcję adaptacyjną (adapt.m)

¹projekt dostępny pod adresem <https://sites.google.com/site/mihailsirotenko/projects/convolutional-neural-network-class> [dostęp 10.06.2011]

oraz skrypt sprawdzający działania sieci neuronowej na podanym obiekcie (sim.m).

Podstawowymi zadaniami jakie wykonaliśmy było wprowadzenie odpowiednich parametrów konwolucyjnych sieci neuronowych, przeprowadzenie treningu dla wybranych obrazów treningowych oraz analiza wyników działania sieci dla obrazów testowych po czym ponowna konfiguracja parametrów. Aby rozpocząć naukę nowych sieci neuronowych należy uruchomić plik `train_cnn.m` i poczekać na zakończenie treningu (wyświetli się wtedy wykres błędów). Aby przeprowadzić symulację wykorzystujemy skrypt `sim_file.m`.

Podjęliśmy próbę użycia opracowanej przez firmę Nvidia architektury CUDA służącej do wspomagania obliczeń przy wykorzystaniu wydajnych układów graficznych. Niestety po zainstalowaniu pakietu deweloperskiego i wgraniu potrzebnych bibliotek otrzymaliśmy błędy przy wywołaniu funkcji, których z braku wiedzy o architekturze oraz dostępu do źródeł kodu nie udało nam się naprawić.

7.2 Pozyskiwanie obrazów do treningu i testów

Zdjęcia treningowe pochodzą z bezpłatnych baz zdjęć twarzy znalezionych w Internecie. Zdobyte zdjęcia zostały poddane ręcznej obróbce w edytorach grafiki gimp, MS Paint. Twarz należało odpowiednio przeskalować oraz wykadrować tak, aby nos znalazł się na środku obrazka a cała twarz zmieściła się na obrazku o rozdzielczości 36x32. Szczególnie przydatny był program IrfanView, którego użyliśmy do automatycznego, seryjnego przeskalowywania, zmiany nazw plików, konwersji do odcieni szarości i innych.

Obrazy nie przedstawiające twarzy (potrzebne w procesie nauki jako antywzorce) uzyskaliśmy poprzez pokrojenie obrazów krajobrazów oraz fragmentów zdjęć niezawierających twarzy. W tym celu napisaliśmy w Matlabie skrypt `slice.m`, którego zadaniem jest utworzenie wielu plików małych obrazków o zadanym rozmiarze z jednego dużego.

7.3 Organizacja pracy grupowej

W celu realizacji projektu wielokrotnie spotykaliśmy się by wspólnie pracować. Po wyznaczeniu zakresu prac zaistniała potrzeba aby każdy z nas mógł pracować indywidualnie nad swoją częścią. Aby umożliwić równoczesną pracę zdalną skorzystaliśmy z rozproszonego systemu kontroli wersji GIT. Założył konto w ogólnodostępnym, w dużej części bezpłatnym serwisie github (<https://github.com/>) oraz zainstalowaliśmy potrzebne oprogramowanie. Po utworzeniu repozytorium i przypisaniu członków można było bardzo szybko i łatwo nanosić swoje poprawki do projektu czy wysyłać pliki.

```
MINGW32:/d/FaceDetectionCNN
Tsm@HATSUNE /d/FaceDetectionCNN (master)
$ git pull
Enter passphrase for key '/c/Users/Tsm/.ssh/id_rsa':
Already up-to-date.







Tsm@HATSUNE /d/FaceDetectionCNN (master)
$ git add .

Tsm@HATSUNE /d/FaceDetectionCNN (master)
$ git commit -m "Dokumentacja: rozdzial 7 - Opis narzedzi i metod postepowania"
[master cbd22db] Dokumentacja: rozdzial 7 - Opis narzedzi i metod postepowania
1 files changed, 2 insertions(+), 0 deletions(-)

Tsm@HATSUNE /d/FaceDetectionCNN (master)
$ git push
Enter passphrase for key '/c/Users/Tsm/.ssh/id_rsa':
```

W razie wystąpienia problemu można było bez większego zachodu przywrócić poprzednią, działającą wersję. Każdą, nawet drobną zmianę (tzw. commit) dało się opisać komentarzem, co pozwalało reszcie zorientować się co się zmieniło.

2011-05-26

Matlabowe skrypty + MNIST do OCRa dla cyfr
 tsm (author) May 26, 2011
Poprawki w szablonie
 barthez (author) May 26, 2011
Merge branch 'master' of github.com:barthez/FaceDetectionCNN
 barthez (author) May 26, 2011
Dodanie szablonu sprawozdania - do dopracowania
 barthez (author) May 26, 2011
Merge branch 'master' of github.com:barthez/FaceDetectionCNN
 smilasek (author) May 26, 2011
Wrzucam dokumentacje jeszcze niekompletna, bo slabo z czasem :(
 smilasek (author) May 26, 2011

7.4 Dokumentacja

Niniejszą dokumentację stworzyliśmy w TeXie przy użyciu programu MikTeX 2.9. Zdecydowaliśmy się na to, ponieważ TeX umożliwia wstawianie nawet dosyć skomplikowanych wzorów matematycznych, pozwala łatwo i szybko formatować tekst oraz wspomaga automatyczną indeksację treści i eksport do PDF.

8 Dodatek A: Opis realizacji rozwiązania

9 Dodatek A: Opis informatycznych procedur

10 Dodatek A: Spis zawartości dołączonych nośników