POLITECHNIKA WARSZAWSKA

WYDZIAŁ MATEMATYKI  
 I NAUK INFORMACYJNYCH

Detekcja typu terenu na podstawie zdjęć satelitarnych i danych z OpenStreetMap

Projekt na potrzeby przedmiotu „Aplikacje i usługi GIS”

Tomasz Luśtyk

Paweł Paradowski

Spis treści

[1. Wstęp teoretyczny 3](#_Toc388918029)

[2. Opis zadania 3](#_Toc388918030)

[3. OpenCV - krótki opis biblioteki 4](#_Toc388918031)

[3.1. OpenCV – wykorzystane funkcjonalności do projektu 4](#_Toc388918032)

[3.2. Co udało się zrobić 5](#_Toc388918033)

[4. Podejście awaryjne – Matlab 5](#_Toc388918034)

[5. Co udało się zrobić? 5](#_Toc388918035)

[5.1. Zmiany w podejściu z użyciem klasyfikatora Haara 5](#_Toc388918036)

[5.2. Canny 6](#_Toc388918037)

[5.3. Wykorzystanie Matlaba 6](#_Toc388918038)

[6. DeepLearning? 7](#_Toc388918039)

# Wstęp teoretyczny

Pokrycie terenu jest ściśle związane z jego użytkowaniem przez człowieka i jest bardzo istotnym elementem zmieniającego się środowiska na Ziemi. Odzwierciedla ono biofizyczne cechy środowiska, natomiast użytkowanie terenu mówi nam o jego funkcji w przestrzeni. W badaniach teledetekcyjnych, obserwując pokrycie terenu, możemy wnioskować o jego użytkowaniu.

Szczególnie istotnym źródłem informacji o środowisku geograficznym są dane satelitarne. Przy robieniu badań dotyczących zmian pokrycia terenu bardzo ważnymi cechami zdjęć satelitarnych są:

* synoptyczność,
* duża powtarzalność obrazowania takich samych terenów,
* automatyzacja procesu ich interpretacji,
* dostępność zdjęć.

[](http://wiki.openstreetmap.org/w/images/4/4f/MarekDuzyObszarNaRastrze4.jpg)

Obraz nr 1. Konstruowanie linii prostopadłych do osi konstrukcjyjnej

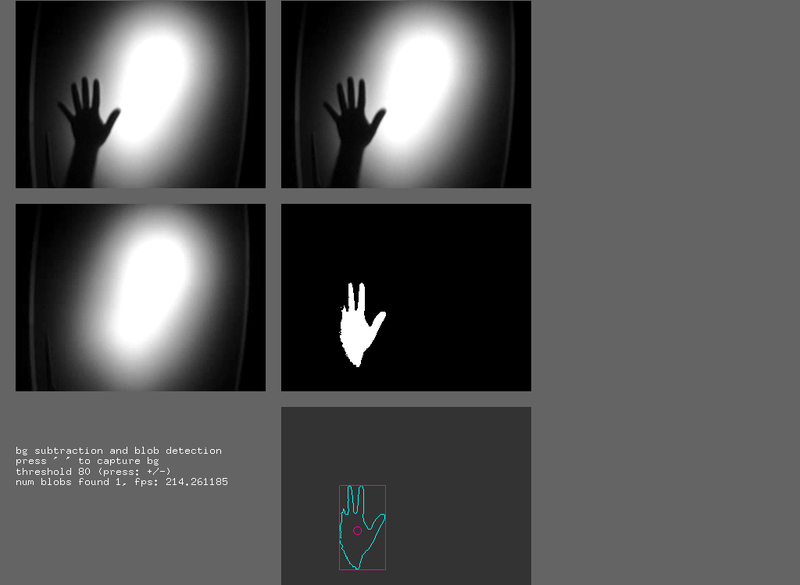
# Opis zadania

Celem naszego projektu jest rozpoznawanie różnych obszarów terenu na podstawie zdjęć satelitarnych i danych udostępnianych przez takie serwisy jak m.in. OpenStreetMap czy GoogleMaps. Chociaż w oryginalnym temacie projektu było wspomniane o celu minimalnym (rozróżnianie budynków od reszty terenu), będziemy starali się rozpoznawać w miarę możliwości inne rodzaje terenu.

Z dostępnych w literaturze oraz sieci materiałów, wybraliśmy dwa podejścia, które chcielibyśmy wykorzystać przy analizie zdjęć satelitarnych do rozpoznawania danego przez nas wzorca. Skupiliśmy się na pierwszym z nich (opisanym poniżej) mając przy tym nadzieję, że to właśnie on pomoże nam w rozwiązaniu naszego problemu projektowego.

# OpenCV - krótki opis biblioteki

Naszym pierwszym podejściem, które chcemy wykorzystać jest uznana, darmowa biblioteka – OpenCV. Jest to biblioteka funkcji wykorzystywanych podczas obróbki obrazu, oparta na otwartym kodzie i zapoczątkowana przez firmę Intel. Jest ona wieloplatformowa (m.in. możemy z niej korzystać pod Linuxem, Windowsem oraz w Mac OS X), stworzona w języku C. Istnieją także nakładki umożliwiające korzystanie z niej również w językach C++, C# oraz Python. Głównym zamysłem autorów było oparcie się na przetwarzaniu obrazu w czasie rzeczywistym.

[](http://upload.wikimedia.org/wikipedia/commons/8/87/OfxOpenCV.png)

Obraz nr 2. Działanie OpenCV w oprogramowaniu openFrameworks

## OpenCV – wykorzystane funkcjonalności do projektu

Główną funkcjonalnością, którą będziemy wykorzystywać z OpenCV będzie rozpoznawanie wzorca w obrazie. W samej bibliotece możemy korzystać z tzw. Klasyfikatora kaskadowego Haara, który zawiera w sobie sieć neuronową. Można w niej m.in. nauczyć klasyfikator na podstawie kolekcji przykładów (pozytywnych i negatywnych) rozpoznawać element (z obrazka pozytywnego) na dowolnym obrazku. Dwa główne kroki rozpoznawania wzorca to:

* Trenowanie
* Detekcja na obrazie

Dokumentacja, która dotyczy tej biblioteki opisuje m.in. jak wytresować klasyfikator (jak go przygotować, aby dobrze rozpoznawał zadane wzorce). Do realizacji tych kroków, OpenCV udostępnia dwie aplikacje – *opencv\_haarttraining* oraz *opencv\_traincascade*.

Dodatkowowo, są też udostępnione pomocne narzędzia:

* *opencv\_createsamples* – stosowana do przygotowania zbioru danych pozytywnych i negatywnych przykładów. Zbiór danych jest w formacie wspieranym przez obie (wymienione wyżej) aplikacje. Na wyjściu dostajemy plik z rozszerzeniem \*.vec, co jest binarnym odpowiednikiem zawierającym obrazek.
* *opencv\_performance* – może być użyte do oszacowania jakości klasyfikatora, ale tylko dla *opencv\_haartraining*.

## Co udało się zrobić

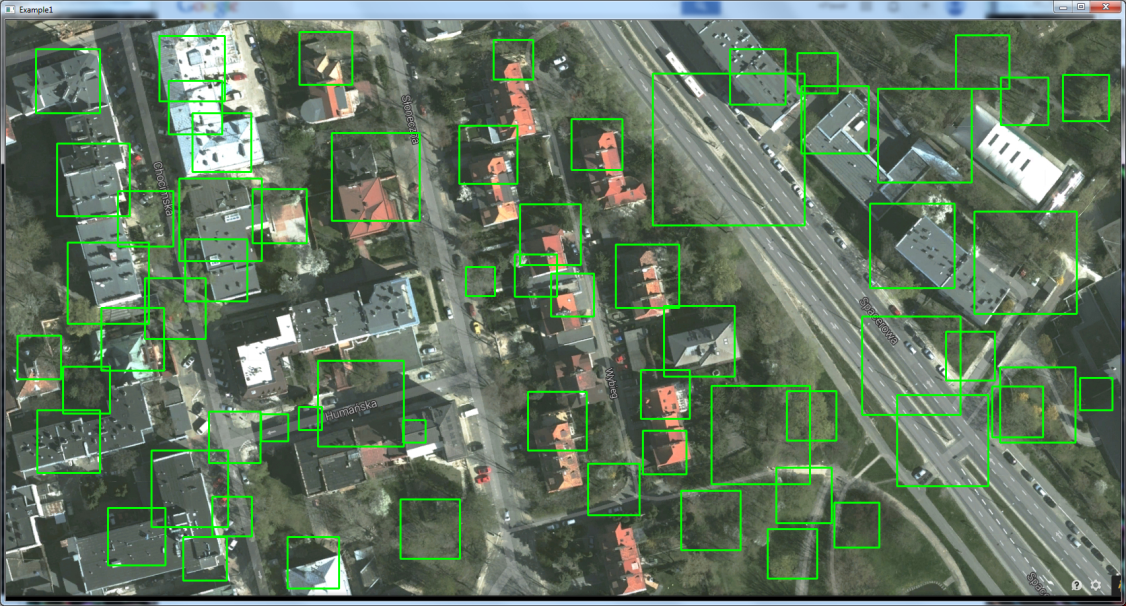
Na ten moment, udało nam się przygotować sample z wykorzystaniem opencv\_createsamples, zgodnie z tutorialem udostępnionym przy dokumentacji OpenCV. Niestety, nie udało nam się póki co wytrenować klasyfikatora. Napotykamy na swej drodze barierę w postaci zbytniego zużycia pamięci i zawieszania się aplikacji (z niewiadomego powodu) przy uruchomieniu testów.

# Podejście awaryjne – Matlab

W razie niepowodzenia z OpenCV, chcielibyśmy spróbować zdziałać coś za pomocą Matlaba, w którym również mielibyśmy możliwość rozpoznawania wzorców na obrazach podanych przez nas. Ponieważ temat pierwszy, dotyczący OpenCV jest dla nas priorytetowy i wciąż pokładamy w nim nadzieję, nie rozpisujemy się w tym rozdziale na temat tej technologii. Dopiero w sytuacji całkowitej rezygnacji z kontynuacji pracy.

# Co udało się zrobić?

## Zmiany w podejściu z użyciem klasyfikatora Haara

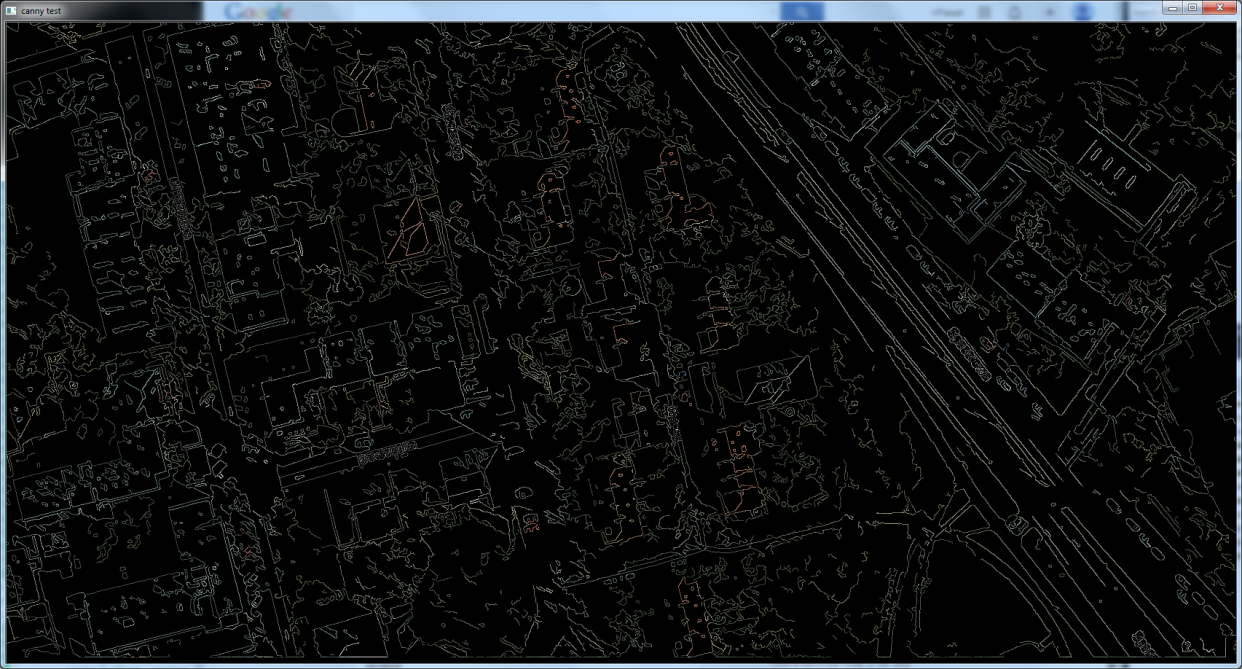
Jedną z podstawowych zmian, które zastosowaliśmy w podejściu do rozwiązania powyżej opisanego problemu była zmiana klasyfikatora z klasyfikatora Haara na LBP. Głównym atutem tej zmiany było to, że nauka następowała dużo szybciej (LBP uczył się w kilka minut, Haar w około 1,5 dnia). Jakość nauki za pomocą LBP jest słabsza, ale porównywalna do nauki Haara. Niestety jednak zmiana klasyfikatora nie pomogła nam przy rozpoznawaniu dachów z mapy satelitarnej. Na przykładowych danych wejściowych, zawierających pozytywne i negatywne przypadki, otrzymaliśmy taki wynik:

Obraz nr 3. Działanie nauki przy zastosowaniu klasyfikatora LBP

Niestety, wydaje nam się, że budynki są na tyle różnorodne i trudno odróżnialne od terenu, że tego typu podejście nie jest niezbyt dobrze się sprawdza (bądź brakuje nam wiedzy w tym temacie).

## Canny

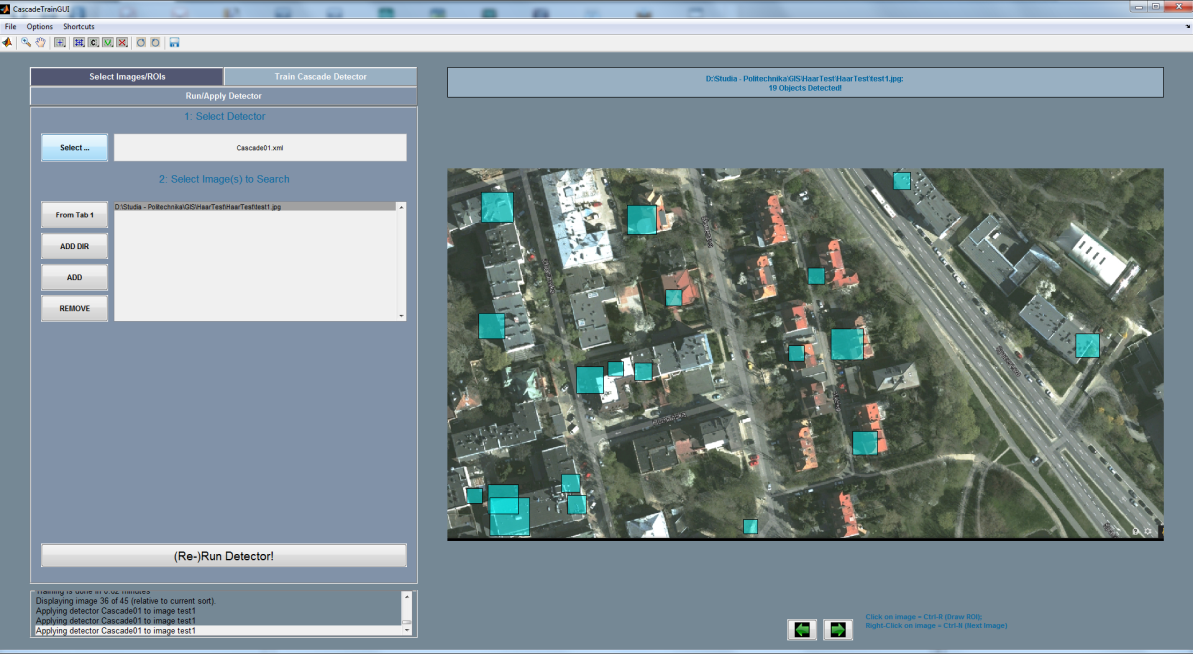
Nie poddaliśmy się i postanowiliśmy zmienić podejście – wykrywanie krawędzi w OpenCV za pomocą algorytmu Canny. Algorytm daje całkiem niezłe rezultaty, ale nie mamy pomysłu jak można wykorzystać ten obraz do znajdowania budynków.



Obraz nr 4. Działanie wykrywanie krawędzi algorytmem Canny

## Wykorzystanie Matlaba

Próbowaliśmy również uczyć kaskadę w Matlabie 2014 (Cascade Train GUI), wyszła bardziej ostrożna niż w OpenCV, co pokazuje poniższe zdjęcie. Niestety, tak jak w poprzednich przypadkach rezultat jest niezadowalający.



Obraz nr 5. Działanie nauczonej kaskady w Matlabie

Próbowaliśmy również drugiego wariantu z Matlabem, a mianowicie użycie klasyfikatora Matlaba (Haar features). Wyniki lepsze niż na obrazie nr 5, ale i tak słabe.



Obraz nr 6. Działanie nauczonej kaskady w Matlabie (Haar features)

# Wykorzystanie Deep Learningu - propozycja

Biorąc pod uwagę sukcesy metod z rodziny *deep learning* w rozpoznawaniu obrazów (m. in. sukcesy grupy Swiss AI Lab), takie podejście jest potencjalnie najskuteczniejsze. Ograniczymy się do *deep learningu* w oparciu o sieci neuronowe; sieć neuronowa jest głęboka, gdy posiada więcej niż jedną warstwę ukrytą.

Widzimy dwa możliwe sposoby zaatakowania problemu: pierwszy uczy sieć w celu klasyfikowania pojedynczego piksela, druga uczy sieć w celu klasyfikowania prostokątnego obrazu jako zawierającego budynek lub nie. Zauważmy, że w obu przypadkach wyjściem sieci jest pojedynczy bit, czyli odpowiedź: tak lub nie.

Przed prezentacją tych dwóch podejść, omówimy w sposób ogólny sieci Convolutional Neural Network w implementacji zaproponowanej przez Yann’a LeCun’a – często nazywane od jego nazwiska LeNet-ami.

## Convolutional Neural Network – typ LeNet

Sieci CNN stanowią szczególny rodzaj głębokich sieci neuronowych inspirowanych korą wzrokową zaobserwowaną u kota. Podobnie jak dla większości NN, uczenie sieci odbywa się za pomocą algorytmu propagacji wstecznej, a sieć ma architekturę typu feed-forward. Celem CNN jest rozpoznawanie wzorców w obrazie mając na wejściu piksele tego obrazu przy minimalnym bądź braku przetwarzaniu na wstępie. Ich zaletą jest elastyczność co do różnorodności przykładów danej klasy (np. różne ręcznie napisane cyfry 2) oraz dobre rozpoznawanie wzorca nawet, gdy uległ translacji, zniekształceniu, bądź obrotowi. Jest to znaczny postęp w porównaniu do możliwości klasycznego MLP z jedną warstwą ukrytą w tym zakresie.

By ogólnie opisać ideę: warstwy ukryte występują na przemian w dwóch rodzajach: convolution (filtrujące) oraz sub-sampling (max-pool, kompresja stratna analizowanych danych w celu uogólnienia i osiągnięcia większej generalizacji – wybieramy wartość maksymalną z obszaru). Typowo będziemy mieli:

1. Warstwę wejściową (piksele obrazu)
2. Warstwę convolution #1 z wyuczonym bądź arbitralnym filtrem. Rozmiar filtra: ok. 10% rozmiaru obrazu
3. Warstwę max-pool #1 próbkującą prostokąty 2x2, 3x3, 4x4 lub o innym rozmiarze – zależnie od rozmiaru obrazu na wejściu. Dla identyfikacji piksela sugerowalibyśmy zastosować 2x2, dla klasyfikacji budynku z prostokątnego obrazu o rozmiarze 100x100 – 200x200 pikseli – raczej 3x3 albo 4x4.
4. Warstwę convolution #2
5. Warstwę max-pool #2
6. Warstwę ukrytą odpowiadającą za klasyfikację
7. Warstwę wyjściową – 1 neuron.

Zaznaczmy, czym różni się takie podejście od przepuszczenia obrazu przez kilka filtrów, a następnie podaniu go na wejściu do MLP z jedną warstwą ukrytą: po pierwsze, filtry i warstwy max-pool podlegają uczeniu. Po drugie, warstwy max-pool działają w sposób nieliniowy w przeciwieństwie do typowych metod downsamplingu. Warstwy filtrów można też uczyć bez nauczyciela – choć nie jest to konieczne. Co ważne, wagi są współdzielone – ucząc się filtru, nie musimy wykonywać pełnej propagacji wstecznej, a jedynie obliczyć zmianę każdego „koloru” krawędzi (ten sam kolor oznacza krawędzie o arbitralnie tej samej wadze).

**Fragmenty kodu w Theano:**

layer0 = LeNetConvPoolLayer(rng, input=layer0\_input,

image\_shape=(batch\_size, 1, 28, 28),

filter\_shape=(nkerns[0], 1, 5, 5), poolsize=(2, 2))

W ten sposób tworzymy warstwę typu convolution i następującą po niej warstwę max-pool dla wejścia o wymiarze 28x28 pikseli. Filtr jest tutaj typu macierzowego o wymiarze 5x5, co daje nam obraz o wymiarze 24x24 pikseli na wyjściu, następnie mamy max-pool na (nienachodzących na siebie) prostokątach 2x2 pikseli, czyli wyjściem warstwy max-pool będzie obraz o wymiarze 12x12 pikseli.

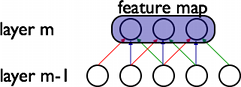
layer1 = LeNetConvPoolLayer(rng, input=layer0.output,

image\_shape=(batch\_size, nkerns[0], 12, 12),

filter\_shape=(nkerns[1], nkerns[0], 5, 5), poolsize=(2, 2))

Analogicznie tworzymy warstwę przyjmującą na wejściu wyjście warstwy utworzonej wcześniej (layer0). Wejściem jest obraz 12x12 pikseli, jądro filtru to macierz 5x5, max-pool na prostokątach 2x2. Otrzymujemy wyjście 4x4, które następnie podlega klasyfikacji przy pomocy warstwy ukrytej i warstwy wyjściowej. Klasa LeNetConvPoolLayer i wywołania jej konstruktora pochodzą z tutorialu umieszczonego na deeplearning.net, plik *convolutional\_mlp.py*.

**Obrazowe działanie filtru:**



**Wizualizacja schematu sieci typu LeNet:**

## File:Mylenet.png

## W oparciu o sieć klasyfikującą obrazek

**Wejście:** prostokątny obrazek, gdzie prostokąt jest opisany na budynku zakładając orientację prostokąta wzdłuż osi X i Y (istnieje dokładnie jeden taki prostokąt).

**Architektura sieci:** ConvNN – LeNet-5. Przykładowy rozmiar filtrów: 5x5, max-pool 3x3, 5x5, max-pool 3x3.

**Wyjście:** odpowiedź tak lub nie (analogicznie jak dla klasyfikacji piksela).

**Przykład treningowy:** Para – prostokątny obrazek oraz wartość 1 – obraz zawiera budynek, 0 – obraz nie zawiera budynku.

**Metoda pozyskiwania przykładów treningowych:**

1. Wybieramy dostawcę danych udostępniającego dane WFS oraz WMS.
2. Wyznaczamy dwie pary współrzędnych geograficznych (np. długości i szerokości geograficznej dla WGS 84) wyznaczające prostokąt otaczający duże miasto, np. Warszawę.
3. Arbitralnie wybieramy rozmiar prostokąta, dla którego będziemy iteracyjnie żądać danych opisujących wektorowo budynki (z serwisu WFS). Powinien być to rozmiar odpowiadający ok 500x500m ziemi.
4. Iteracyjnie przesuwamy prostokąt w prawo, a gdy już nie możemy, to resetujemy do lewej i przesuwamy w dół z zakładką 50m w obu osiach.
5. Dla bieżącego prostokąta:
   1. Pobierz grafikę WMS (dla odpowiadającego prostokątowi *bounding boxu*)
   2. Pobierz dane o budynkach w tym obszarze z serwisu WFS
   3. Dla każdego budynku, który całkowicie mieści się w obecnie rozważanym prostokącie mapy:
      1. Oblicz geograficzne współrzędne prostokąta opisanego na tym budynku
      2. Przelicz te współrzędne geograficzne na piksele obrazka otrzymanego z WMS (znamy bounding box, więc możemy dokonać takiego przeliczenia)
      3. Fragment obrazka będący prostokątem opisującym bieżący budynek zapisz do zbioru przykładów treningowych jako przykład pozytywny.

**Wykorzystanie:** Dla obrazu o wymiarach nxn, na którym chcemy wykryć budynki, wykonujemy następujący algorytm:

1. Stwórz pomocniczą strukturę danych – tablicę liczb całkowitych o wymiarze takim samym, jak obraz satelitarny, który badamy.
2. Dla rozmiaru okna [RxR] od 20x20 do 300x300, zwiększając je każdorazowo o współczynnik s równy np. 1.2 (tutaj to da rozmiary okna 20x20, 24x24 itd.):
   1. Przesuwaj okno co 1/20 R i dla każdego pozycjonowania okna:
      1. Podaj sieci na wejście obraz ograniczony bieżącym oknem przeszukiwania
      2. Jeżeli sieć zwróci NIE, przesuń okno w punkcie (a)
      3. Jeżeli sieć zwróciła TAK, inkrementuj wartości w pomocniczej strukturze danych odpowiadające pikselom pokrytym bieżącym oknem.
   2. Grupuj punkty w budynki (algorytm daje nam na wyjściu coś w rodzaju heat-map bądź height-map, gdzie niektóre punkty przynależały do bardzo wielu wykryć budynku, inne do mniejszej liczby i na tej podstawie musimy oszacować granice właściwego budynku).
   3. Resetuj strukturę danych
   4. Zapamiętaj, gdzie wykryto budynki, by przy kolejnych iteracjach już ich nie nadpisywać (można przyjąć kryterium np. budynki nie mogą nachodzić na siebie, bądź: mogą na siebie nachodzić w ograniczonym stopniu itd.)

**Przewidywana skuteczność:** Przewidujemy, że sieć LeNet nauczyłaby się dobrze klasyfikować, czy obraz przedstawia budynek, czy nie, skuteczność powyżej 80%, a nawet powyżej 90% powinna być osiągalna, choć pewnie nie przy pierwszym podejściu (wnioskujemy z rezultatów tego typu sieci w konkursach).

Mimo to, zaznaczmy, że problem określenia przynależności piksela jest trudny – dużo trudniejszy, niż problem klasyfikacji obrazu jako budynek bądź nie-budynek (taki problem przypomina standardowe testy CIFAR bądź MNIST i dla takiego problemu spodziewamy się skuteczności jak wyżej). Niestety, w tym przypadku wyuczenie sieci tak, że jest w stanie dobrze rozpoznawać, czy prezentowany obraz przedstawia budynek jest niewystarczające.

Wykorzystanie takiej sieci przysparza problemów: zakładając podejście z przesuwającym się oknem, narażamy się na błędne wyznaczenie faktycznych granic budynku. Musimy też balansować szybkość działania z dokładnością przeszukiwania.

## W oparciu o sieć klasyfikującą piksel

**Wejście:** piksel, który chcemy sklasyfikować oraz jego otoczenie o promieniu R, gdzie R jest hiperparametrem (ustalone z góry dla sieci neuronowej). Dla np. R = 20 na wejściu będzie kwadrat o wymiarze 40x40, gdzie piksel podlegający klasyfikacji będzie miał współrzędne (20, 20).

**Architektura sieci:**LeNet-5. Mały rozmiar filtrów – najlepiej 2x2. Max-pool 2x2. Dobrym pomysłem byłoby wzorowanie się na sieci z tutorialu na deeplearning.net, rozpoznającej cyfry z zestawu MNIST ze względu na podobny rozmiar danych wejściowych.

**Wyjście**: odpowiedź tak lub nie (neuron będzie logistyczny, więc wyjście <0.5 może oznaczać odpowiedź NIE, natomiast wyjście >= 0.5 – odpowiedź TAK).

**Przykład treningowy:** para - kwadratowy obrazek o boku 2R zawierający interesujący nas piksel we współrzędnych (R, R) oraz wartość liczbowa – 1, jeżeli piksel (R, R) należy do budynku, 0 w przeciwnym przypadku.

**Metoda pozyskiwania przykładów treningowych:** Analogicznie, jak dla prostokątnego obrazka z tą różnicą, że wycinamy prostokąt opisany na budynku z buforem o rozmiarze R pikseli. Dodatkowo zapisujemy do powiązanego pliku:

1. współrzędne geograficzne opisujące prostokąt, który wycięliśmy
2. współrzędne punktów wielokąta odpowiadającemu budynkowi reprezentowanemu przez dane wektorowe.

Tak opisane dane mogą być względnie łatwo przetworzone na przykłady treningowe dla sieci przez aplikację pośredniczącą. Taka aplikacja działałaby w taki sposób:

1. Weź obrazek i jego opis
2. Na podstawie współrzędnych geograficznych opisujących prostokąt oraz współrzędnych budynku stwórz obraz binarny odpowiadający rozmiarem prostokątowi z budynkiem
3. Za pomocą algorytmu skan-linii lub innego wyznacz w obrazie binarnym piksele należące do budynku, ustawiając wartość na 1.
4. Iteruj po pikselach obrazu binarnego i dla każdego piksela (p, p):
   1. Zapisz obszar (p-R, p-R), (p+R, p+R) obrazu z budynkiem i powiąż go z wartością równą wartości pikselu (p, p) obrazu binarnego (jest to 0 – brak budynku lub 1 – piksel należący do budynku)

W ten sposób jesteśmy w stanie podać na wejściu sieci przykłady pozytywne i negatywne, na których może się uczyć. Jeżeli uznamy to za stosowne, możemy poprzestać na zapisaniu danych treningowych w dowolnym formacie, a następnie podamy te dane programowi uczącemu sieć już bez udziału aplikacji pośredniczącej – aplikacja posłuży jako narzędzie do spreparowania danych.

**Wykorzystanie:** Dla obrazu nxn, na którym chcemy wyznaczyć lokalizację budynków, wykonujemy następujący algorytm:

1. Stwórz pomocniczą strukturę danych odpowiadającą obrazowi wejściowemu
2. Dla każdego piksela obrazu wejściowego (z marginesem R od krawędzi):
   1. Podaj sieci na wejściu prostokąt 2Rx2R
   2. Sieć zwraca, czy piksel (R, R) tego prostokąta należy do budynku
3. Grupuj wartości niezerowe z pomocniczej struktury danych, wyznaczając lokalizację budynków.

**Przewidywana skuteczność:** Niska. Nie udało nam się dotrzeć do literatury opisującej klasyfikację przynależności pikseli do obiektów, bądź analogicznie: opisującej wyznaczanie granic obiektu za pomocą sieci neuronowych, głębokich bądź nie. Takie zadanie wydaje się bardzo trudne.

Intuicyjnie wydaje nam się, że otoczenie piksela przynależącego do budynku może być tak różne, że szum będzie zbyt duży, aby sieć zdołała stworzyć generalizację, co charakteryzuje otoczenie piksela należącego do budynku.

Niektóre piksele będą leżeć w samym środku budynku, inne bliżej krawędzi, niektóre na krawędzi cienia, niedaleko drzew, zaparkowanych samochodów, ulicy, trawnika – są to bardzo różne otoczenia. W przypadku identyfikacji, czy obraz przedstawia budynek pewne cechy będą nieodzowne – występowanie kształtu zawierającego liczne krawędzie raczej bliżej krawędzi analizowanego obszaru (bo budynek wypełnia obraz wejściowy), budynek stanowi kontrastowy i dość jednolity element itd.

Klasyfikowanie przynależności pikseli bezpośrednio przez sieć neuronową wydaje nam się bardzo trudnym zadaniem, podatnym na błędy i według naszej obecnej wiedzy – pionierskim. To zadanie wydaje nam się jakościowo inne z punktu widzenia sieci neuronowej, niż odpowiedź sieci na pytanie: „czy ten obraz jest psem, krową, czy koniem?”, „czy ten obraz jest budynkiem, czy nie?” „czy ten obraz jest cyfrą 0, 1, …, 8, czy 9?”. Tego typu problemy były i są rozwiązywane przez sieci neuronowe, literatura i przykłady na ten temat są dostępne i dlatego przychylamy się do rozwiązania operującego na sieci klasyfikującej obraz otrzymany na wejściu jako budynek lub nie-budynek. W tamtym przypadku mamy podstawy twierdzić, że sieć nauczyłaby się dobrze wykonywać swoje zadanie; przypadku klasyfikacji piksela nie jesteśmy co do tego przekonani.

# Bibliografia – Deep Learning:

1. <http://yann.lecun.com/exdb/lenet/>
2. <http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf>
3. <http://deeplearning.net/tutorial/lenet.html>
4. <http://white.stanford.edu/teach/index.php/An_Introduction_to_Convolutional_Neural_Networks>