Informatyka stosowana
2017/18

Sztuczne Sieci Neuronowe

Rozpoznawanie podstawowych kształtów przy użyciu konwolucyjnych sieci neuronowych u użyciem technologii TensorFlow

Autorzy:

Julian Czerwonka
Kamila Kalecińska
Tomasz Kąkol
Mateusz Karpik

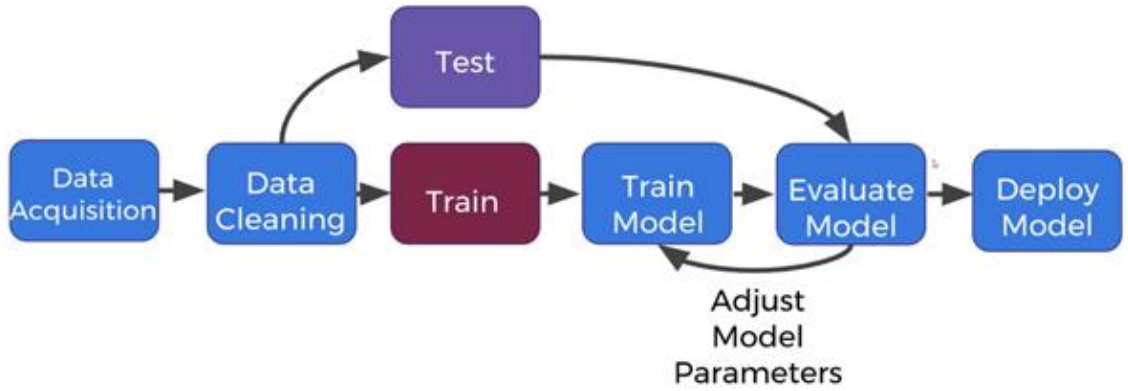
Cel i specyfikacja problemu

Celem projektu było utworzenie sieci neuronowej zdolnej do rozpoznawania podstawowych figur geometrycznych. Technologie wykorzystywane do realizacji tego przedsięwzięcia to Python i biblioteka sztucznej inteligencji - TensorFlow. Efektem pracy jest aplikacja z graficznym interfejsem użytkownika (GUI), która pozwala na wgranie dowolnego obrazka przedstawiającego figurę geometryczną, który następnie zostaje sklasyfikowany przez sieć.

Celem działania sieci było rozróżnienie 4 figur geometrycznych:

- koło
- kwadrat
- trójkąt
- sześciokąt

Schemat procesu postępowania podczas realizacji założonego celu został przedstawiony poniżej:



Rysunek 1 Schemat blokowy przedstawionego procesu

Poszczególne elementy diagramu zostały omówione w kolejnych częściach sprawozdania.

Wprowadzenie teoretyczne

Problem klasyfikacji obrazów przy użyciu sieci neuronowych opiera się na konwolucyjnych sieciach neuronowych.

Operacja konwolucji jest matematyczną operacją (inaczej określana nazwą ‘splot’), polegającą na działaniu określonym dla dwóch funkcji w wyniku czego otrzymywana jest inna funkcja, często postrzegana jako zmodyfikowana wersja oryginalnych funkcji. Ideą zastosowania konwolucji w modelowaniu sztucznych sieci neuronowych jest m.in. wykonanie próby odkrywania cech obrazu (ang. feature engineering). Celem zastosowania tej operacji jest filtracja obrazu (sygnału) z zastosowaniem techniki przemieszczania się w strukturze mniejszego okna. Dzięki temu sieć jest w stanie wyróżnić cechy sąsiednich komórek. Filtry, w zależności od ustawionych wartości wag umożliwiają m.in. na wydobycie

krawędzi na obrazie, rozmycie lub wyostrzenie obrazu oraz wykrycie charakterystycznych układów linii (jak w naszym przykładzie dotyczącym rozpoznawania szkiców figur geometrycznych). Jednoczesne zastosowanie kilku filtrów umożliwia wykrywanie różnych charakterystycznych cech lub obiektów.

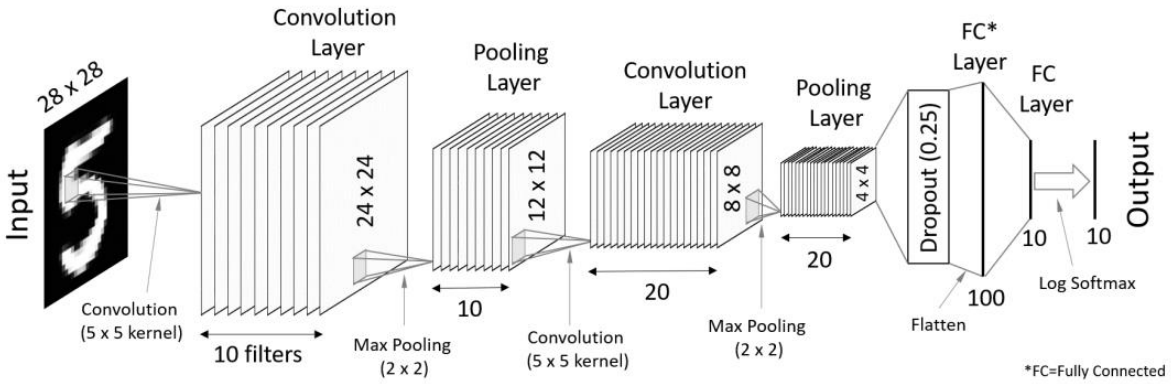
W naszym przypadku zastosowanym filtrem dla obrazków o wymiarze 28 x 28 jest macierz **5 x 5** wraz z dobranymi wagami, którą to przemieszczamy po całej powierzchni tych obrazków. Podczas przemieszczania filtru wykonywane są operacje mnożenia wartości komórek filtru z odpowiadającymi im wartościami komórek obrazu. W kolejnym kroku wszystkie wartości są sumowane. Dzięki temu otrzymujemy wartość dla nowego piksela dla przefiltrowanego obrazu.

Ogólnie znane problemy głębokich sieci neuronowych:

- zwiększenie liczby warstw w sieci skutkuje wzrostem liczby nie wyznaczonych parametrów modelu. Niezbędna jest dostatecznie duża ilość danych. W przypadku uczenia wielowarstwowej sieci z użyciem niewystarczającego zbioru danych, sieć ta relatywnie szybko jest ‘przetrenowana’ . Obecnie nowy problem stanowi również trudność eksploracji ogromnych objętościowo baz danych
- zwiększenie ilości danych w zbiorze trenującym oraz zwiększenie rozmiaru sieci skutkuje zwiększonym zapotrzebowaniem na moc obliczeniową,
- problem ‘zanikającego gradientu’ przy klasycznym treningu i zastosowaniu metod gradientowych oraz algorytmu propagacji wstecznej.

Architektury konwolucyjnej sieci neuronowej

Zastosowanie konwolucyjnych sieci neuronowych umożliwiają rozwiązanie problemu braku dostatecznej skalowalności danych wejściowych przy użyciu sieci typu MLP (Multilayer perceptron). W omawianym przez nas problemie dotyczącym przetwarzania obrazów przykładowo dla monochromatycznego obrazu o rozmiarze 255 × 255 każdemu neuronowi z warstwy ukrytej sieci MLP przypada 65 tysięcy wag, co jest problemem m.in. pod względem wydajnościowym. W celu rozwiązania tego problemu neurony są łączone lokalnie dla pewnej części obrazu, a nie całościowo. Takie rozwiązanie ma odniesienie do rzeczywistej funkcjonalności wzroku człowieka i aktywowania neuronów w przypadku wystąpienia jakiegoś zdarzenia w danym fragmencie pola widzenia. Również na podstawie dostępnych badań wykazano, że wyznaczona reprezentacja cechy lokalnych neuronów jest relatywnie zgodna z cechami reprezentowanymi przez pełną powierzchnię obrazu. Na podstawie przedstawionych własności upoważnione jest tworzone grup neuronów posiadających wspólne wagi, ale różniących się miejscem włączenia do wejściowego obrazu. Zbiór sygnałów wyjściowych z neuronów o identycznych wagach interpretuje się jako mapę aktywności danej cechy w pełnym obszarze obrazu. Filtrem nazywamy cechę, która jest reprezentowana przez pojedynczy zestaw wag.



Rysunek 2 Przykładowa topologia zastosowanej konwolucyjnej sieci neuronowej

Architektura konwolucyjnej sieci neuronowej jest złożona z następujących warstw:

- Warstwa konwolucyjna (lub splotowa, ang. Convolutional layers)

Nazewnictwo tej warstwy podkreśla fakt zamiany operacji mnożenia wektorów (wykonywanej w klasycznym neuronie) na operację splotu, dającej dobre rezultaty w przypadku wykrywania wzorców.

$$y = f(\sum_{i=1}^{N} w_i x_i + b) \longrightarrow y_j = f(W_j * x + b_j)$$

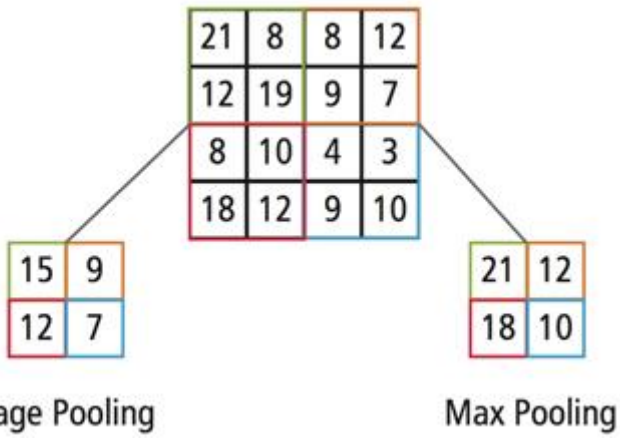
Wyjście pojedyńczego neuronu — Wyjście w przypadku zastosowania CNN

Równanie: Definicje wartości wyjściowych sieci

Warstwa jest określonym zbiorem stosowanych map filtrów. Dla każdego podregionu wykonywane są obliczenia, w efekcie czego wyznaczane są pojedyncze wartości na wyjściowej mapie obiektów. Często stosowaną funkcją aktywacji na wyjściu warstw konwolucyjnych jest funkcja ReLU, wprowadzająca nieliniowość do modelu. Analogicznie do obrazów wejściowych, mapa filtru jest również dwuwymiarowa.

- Warstwa głosująca (ang. pooling layer)

W tej warstwie redukowany jest wymiar obrazu (wyodrębnionego przez warstwy splotowe) poprzez wyznaczenie relatywnie prostych statystyk aktywacji z lokalnego zakresu mapy neuronów. Celem zmniejszenia wymiarowości jest skrócenie czasu przetwarzania. Powszechnie stosowanym algorytmem aktywacji jest maksimum aktywacji (ang. max pooling), który wyodrębnia podregiony mapy obiektów (np. obszary 2 × 2), zachowuje ich maksymalną wartość i odrzuca wszystkie inne wartości. Inną często stosowaną statystyką jest średnia aktywacyjna (ang. average pooling).



Rysunek 3 Metodyka wyznaczania wartości w warswie głosującej

- Warstwa pełna (ang**.** Dense (fully connected) layers)

Po ostatnim module splotowym występuje jedna lub kilka gęstych warstw, które dokonują klasyfikacji. W tej warstwie każdy węzeł jest połączony z każdym węzłem z poprzedniej warstwy. Ostatnia gęsta warstwa w CNN zawiera pojedynczy węzeł dla każdej klasy docelowej w modelu (wszystkie możliwe klasy, które może przewidywać model), z funkcją aktywacji (np. softmax) do generowania wartości od 0 do 1 dla każdego węzła (suma wszystkich wartości jest równa 1). Możemy interpretować wartości softmax dla danego obrazu jako względny pomiar prawdopodobieństwa, że obraz wpadnie do każdej klasy docelowej.

Trening sieci neuronowych

Charakter treningu sieci konwolucyjnej jest analogiczny do nadzorowanego treningu sieci MLP. Jest również możliwe zastosowanie innych rozwiązań, m.in. algorytm trenowania wraz z początkowym wykonaniem ‘pretreningu’ filtrów warstwy konwolucyjnej.

Pretrening (ang. Pre-training) jest techniką nienadzorowanego uczenia sieci, tj. Kolejno jej każdej ukrytej warstwy. Trenowanie rozpoczynane jest od pierwszej warstwy sieci i oczekiwane jest, aby ta warstwa jak najlepiej reprezentowała dane wejściowe. W kolejnym kroku wytrenowaną warstwę używamy jako warstwę wejściową dla następnej warstwy. Schemat należy powtarzać do ostatniej warstwy ukrytej. Wagi nie są aktualizowane w już przetrenowanych warstwach (algorytm zachłanny). Efektem pretreningu jest znaczna poprawa dostosowania wartości wag sieci do zbioru danych treningowych, w porównaniu z losową inicjalizacją wartości wag.

Kolejną stosowaną techniką jest tzw. dotrenowanie (ang. finetuning). Polega ona na dostrajaniu wag podczas treningu (wag wcześniej zainicjalizowanych podczas pretreningu do rozwiązywanego problemu). Dla wariantu nauczania z nauczycielem z pretrenowanych warstw można złożyć m.in. głęboką sieć MLP.

Forma nauki

Zastosowaną formą nauki sieci neuronowej jest uczenie nadzorowane (ang. Supervised learning). Zgodnie z tym algorytmem uczenia, oczekiwany rezultat jest dostarczany wraz z pozostałymi danymi wejściowymi w trakcie trenowania sieci. Dzięki zastosowanej formule możliwe jest wyznaczenie wartości błędu pomiędzy oczekiwanym i

otrzymanym rezultatem, a na tej podstawie wyznaczenie wartości korekty wag połączeń i ich aktualizowanie.

Funkcja kosztu

W obydwu wariantach ważne jest, aby wybieranie podzbioru odbywało się w sposób losowy.

- Zastosowany przez nas optymalizator:
  - Stochastic gradient descent (SGD)

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)}).$$

Wykonuje on aktualizację parametru dla *każdego* przykładu treningowego (estymuje gradient na podstawie tylko jednego przykładu ze zbioru treningowego)

W porównaniu z GDV (ang. Gradient descent variants), które wykonuje nadmiarowe obliczenia dla dużych zestawów danych (ponieważ przelicza gradienty dla podobnych przykładów przed każdą aktualizacją parametru), SGD eliminuje tę nadmiarowość, wykonując jedną aktualizację na raz. Z tego powodu zaletą SGD jest szybkość oraz fakt, że może być wykorzystany do nauki online. SGD dokonuje częstych aktualizacji o wysokiej wariancji, które powodują znaczne wahania funkcji celu (aktualizacje wag są szybkie i bardzo niedokładne). Skutkiem jest zmniejszenie stabilności algorytmu, jednak wprowadzony „szum" do metody redukuje znaną wadę metody bazowej – zagnieżdżenie się w minimum lokalnym.

Inną wyróżnioną modyfikacją metody gradientu prostego jest metoda Mini-Batch Gradient Descent, estymująca wartość gradientu na podstawie małego podzbioru danych trenujących. Ogólnie często przyjmuje się rozmiar takiej paczki danych (ang. Batch size) w zakresie od 10 do 100. Użycie takiej ilości danych dalej cechuje większa szybkość w porównaniu z zastosowaniem całego zbioru. Dodatkowo metoda jest stabilniejsza niż SGD. Podczas klasyfikacji ważne jest, aby w paczce danych była równa liczba przykładów każdej z klas.

*źródło: http://ruder.io/optimizing-gradient-descent/

- Kolejną użytą przez nas funkcją kosztu była funkcja 'Adam', tj. **Adaptacyjny algorytm gradientowy.** Adam jest algorytmem optymalizacyjnym, który może być stosowany zamiast klasycznej procedury gradientowego stochastycznego gradientu do aktualizacji iteracji sieci w oparciu o dane treningowe. Nazwa Adam pochodzi od oszacowania momentu adaptacyjnego 'Adam' różni się od klasycznego spadku stochastycznego.
Stochastyczne nachylenie gradientu utrzymuje pojedynczą szybkość uczenia się (określaną jako alfa) dla wszystkich aktualizacji wagi, a szybkość uczenia się nie zmienia się podczas treningu.
Szybkość uczenia się jest utrzymywana dla każdej wagi sieci (parametru) i oddzielnie dostosowywana w miarę rozwijania się uczenia.
Metoda oblicza indywidualne współczynniki uczenia adaptacyjnego dla różnych parametrów z oszacowań pierwszego i drugiego momentu gradientów.

- Ostatnią porównywaną przez nas optymalizator to Adagrad
Adagrad to algorytm optymalizacji gradientowej, który dostosowuje szybkość uczenia się do parametrów, wykonując mniejsze aktualizacje
(tj. niskie współczynniki uczenia się) dla parametrów związanych z często występującymi cechami i większych aktualizacji (tj. wysokich współczynników uczenia się) dla parametrów związanych z nieczęsto występującymi cechami. Z tego powodu jest dobrze przystosowany do radzenia sobie z rzadkimi danymi.

Jedną z głównych zalet Adagrad jest to, że eliminuje potrzebę ręcznego dostrajania tempa uczenia się.
Większość implementacji używa wartości domyślnej 0.01 i pozostawia ją na tym poziomie. Główną wadą Adagrad jest nagromadzenie kwadratów gradientów w mianowniku:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t.$$

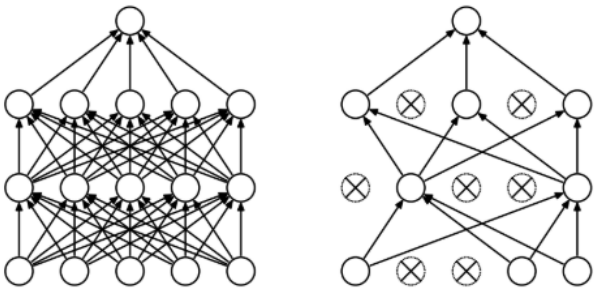
Ponieważ każdy dodany rekord jest dodatni, nagromadzona suma stale rośnie podczas treningu. To z kolei powoduje, że szybkość uczenia się kurczy się i ostatecznie staje się nieskończenie mała, w którym to momencie algorytm nie jest już w stanie zdobyć dodatkowej wiedzy.

Max-norm

Jest jednym z typów regularyzacji, dzięki któremu uniemożliwione jest zjawisko tzw. ‘eksplozji’ wag, kiedy współczynnik nauki jest zbyt duży. W przypadkach niektórych modeli umożliwia to nieszkodliwe zwiększanie wartości tego współczynnika, czego skutkiem jest znaczne przyśpieszenie procesu nauczania. Max-norm wyznacza górną granicę wartości wektora wag każdego z neuronów oraz skaluje ten wektor w dół, gdy wartość graniczna zostaje przekroczona.

Dropout

Jest stosunkowo prostą, a zarazem skuteczną techniką, której ideą jest losowe omijanie pewnych neuronów podczas treningu sieci.



(a) Standard Neural Net (b) After applying dropout.

Rysunek 4 Tolpologia prezentująca zastosowanie dropout'u

W trakcie treningu pozostawiony jest fragment ‘p’ neuronów warstwy, natomiast pozostałe ‘1 − p’ zostaje zignorowane. Realizacja metody jest wykonana poprzez nałożenie binarnej maski na wartości wyjściowe każdej warstwy:

$$y'_n = \frac{r_n * y_n}{p}$$

Zmodyfikowany wektor wyjść´ y'n uczestniczy dalej w propagacji w standardowy sposób:

$$y_{n+1} = f\left(w_n y'_n + b_n\right)$$

Metodę można interpretować następująco:

- Neurony nie powinny polegać na pozostałych neuronach tej samej warstwy, wobec czego samodzielnie starają się wyodrębniać ważne informacje z danych wejściowych. Cechy są ogólniejsze, co sprzyja prezentacji informacji kolejnym warstwom w różnych formach oraz ułatwia im wnioskowanie,

- Próbkowanie podsieci z pełnej topologii sieci. Całą sieć możemy potraktować jako większy zbiór modeli o wspólnych parametrach, natomiast rezultaty pełnej sieci jako uśredniony ´ wynik ze wszystkich modeli.

- Szum dodany do informacji w każdej warstwie sieci. Sieć uczy się w obecności szumu, wobec czego wyciągane są jedynie istotne informacje, których szum nie zdołał zakłócić.

Funkcje aktywacji:

- Softmax

$$f(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}, \quad j = 1..K$$

Inne nazewnictwo tej funkcji to znormalizowana funkcja wykładnicza. Jest ona uogólnieniem funkcji sigmoidalnej. Transformuje K-wymiarowy wektor wartości rzeczywistych do K-wymiarowego wektora wartości w zakresie (0; 1). Suma wartości tego wektora wynosi 1. Funkcja ta jest najczęściej implementowana w ostatniej warstwie sieci, wykorzystywanej do zadań klasyfikacji, kiedy to klasy wykluczają się wzajemnie. Wartość $f(x)_j$ reprezentuje prawdopodobieństwo przynależności wartości wejściowej do klasy 'j'.

- ReLU (ang. Rectified Linear Units)

$$f(x) = \max(0, x)$$

Jest przybliżeniem funkcji Softplus przez zastosowanie prostego progowanie w zerze. Skutkiem jest przyspieszenie implementacji oraz obliczeń algorytmu, w porównaniu z efektem użycia funkcji sigmoidalnej lub tanh. Udowodniono również, iż znacznie przyspiesza zbieżność stochastycznej metody gradientu prostego. Istnieje ryzyko, że wagi neuronu przyjmą stan, w którym to neuron będzie zawsze zwracał 0. Wejście w taki stan jest dla neuronu nieodwracalne. Relatywnie niska wartość współczynnika learning rate zmniejsza prawdopodobieństwo, że problem ten wystąpi.

- Softplus

$$f(x) = \sum_{i=1}^{\infty} \frac{1}{1 + e^{-(x-i+0.5)}} \approx \ln(1 + e^{x})$$

Jest to funkcja aproksymująca sumę wielu sigmoid, przesuniętych względem siebie wzdłuż osi ‘x’. Można to zinterpretować jako sumę wielu neuronów z sigmoidalna funkcją aktywacji, posiadających podobne wartości wag, jednak różne wartości biasu.

- LeakyReLU

$$f(x) = \max(ax, x)$$

Funkcja aktywacja będąca modyfikacją funkcji ReLU, która wspomaga rozwiązanie problemu neuronów, których wagi stabilizuję się w okolicy wartości zerowych (tzw. neuronu ‘umierające’). Dla zakresu, gdy x < 0 funkcje aktywacji cechuje mały dodatni gradient ($a \approx 0.01$). Dzięki temu umożliwione jest uwolnienie neuronu z tego nieodwracalnego stanu w przypadku zastosowania funkcji ReLU.

- f. Sigmoidalna

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$

Jest ciągłym przybliżeniem funkcji progowej, ograniczającym rzeczywiste wartości w zakresie od 0 do 1. Interpretacja funkcji może być następująca: ‘0’ oznacza brak aktywności neuronu i rośnie do maksymalnej częstotliwości wysyłania impulsów (do ‘1’).

- Tanh

$$f(x) = tanh(x) = 2\sigma(x) - 1$$

Funkcja tangens hiperboliczny nasyca się w takich samych przedziałach jak funkcja sigmoidalna (zasadniczo jest jej przeskalowaniem). Jednakże w fukcji tanh wyjścia neuronów o tej samej f. aktywacji są skoncentrowane wokół zera.

Zbiór danych

Dane, które posłużyły jako dane trenujące i testowe w procesie tworzenia i sieci neuronowej pochodzą z gry QuickDraw - stworzonej przez Google na potrzeby utworzenia jak największej liczby danych rysunków różnej kategorii i dostępnej pod adresem: https://quickdraw.withgoogle.com/
Dzięki polityce open-source, którą zastosowała firma można w prosty sposób wykorzystać zgromadzone w ten sposób zbiory danych rysunkowych. W tym celu pobrano dane zawierające rysunki przedstawiające następujące kształty.

- koło
- kwadrat
- trójkąt
- sześciokąt

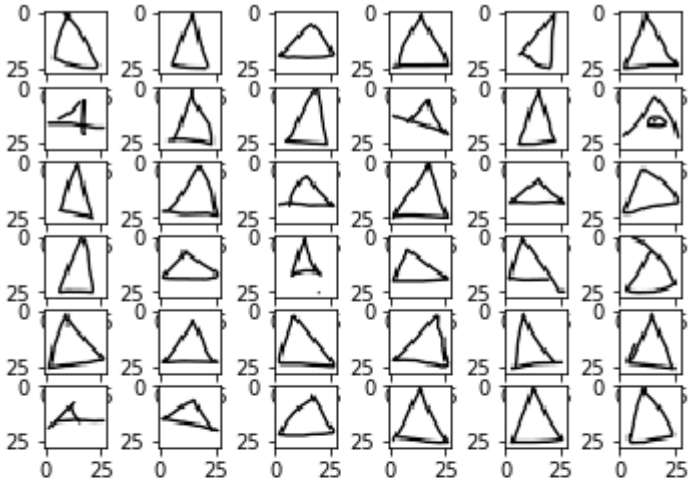
Podstawową zaletą skorzystania z danych pochodzących z QuickDraw jest ich ilość: dla każdej kategorii ilość rysunków wynosi: od 123170 do 142435. Wadą natomiast jest fakt, iż dla każdej kategorii można znaleźć rysunki niedokończone, niewyraźne lub generalnie błędne. Te ostatnie zwykle nie są rozpoznane przez sieć stworzoną przez Google, dzięki

czemu można dokonać selekcji i wziąć pod uwagę jedynie “pewne” przykłady. W tym celu pobrano dane w formacie plików binarnych, które poza macierzą definiującą obraz zawierają również dodatkowe informacje. Na potrzeby projektu w celu uzyskania jak najdokładniejszej sieci, pominięto wszystkie przykłady w których klucz “recognize” przyjmował wartość “false”.

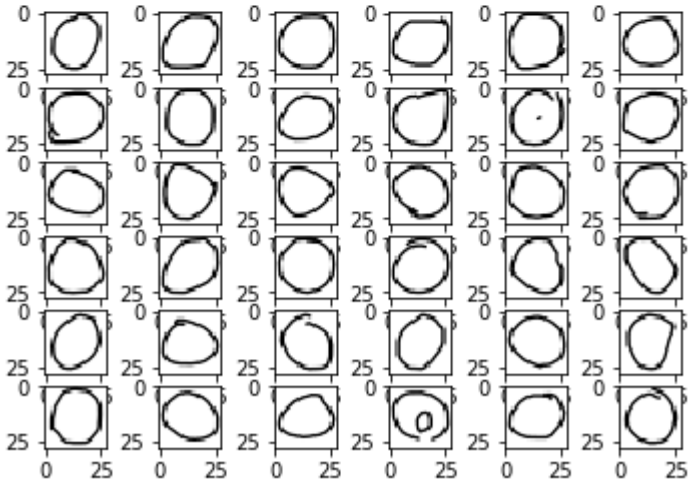
Dane przykładowego obrazka:


```
{
  "key_id":"5891796615823360",
  "word":"circle",
  "countrycode":"AE",
  "timestamp":"2017-03-01 20:41:36.70725 UTC",
  "recognized":true,
  "drawing":[[[129,128,129,129,130,130,131,132,132,133,133,133,133,...]]]
 }
```


Przykładowe rysunki przed odfiltrowaniem nieprawidłowych przykładów:

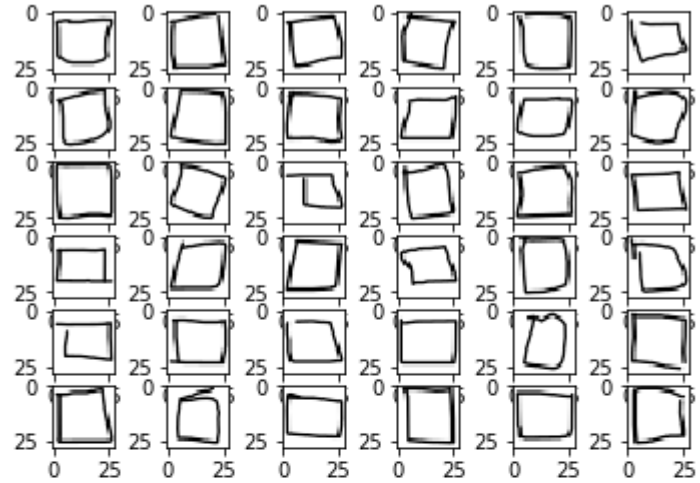


Total number of samples: 122876

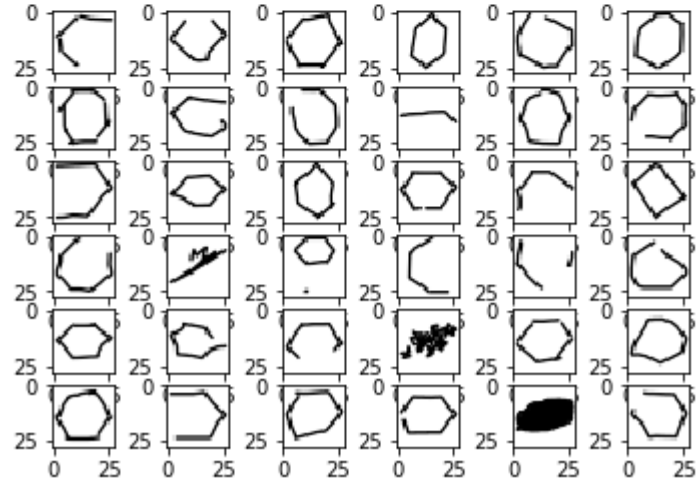


Total number of samples: 125145

```
Total number of samples: 125145
```



```
Total number of samples: 142435
```



Dane po odfiltrowaniu błędnych przykładów (nierozpoznanych przez sieć Google). Ostatecznie uzyskano dane w ilości:

- koło: 118808
- kwadrat: 120538
- trójkąt: 120500
- sześciokąt: 135803

Do trenowania skorzystano z 10 tys danych. Zostały one podzielone na dane uczące i testowe w proporcji 77:33.

Architektura sieci

Jako rozwiązanie problemu przyjęte zostało użycie konwolucyjnych sieci neuronowych. Architekturę sieci przyjęto według poniższego schematu:

- warstwy konwolucyjne (conv1, conv2)
- warstwy dokonujące max-pool’ing wykorzystującej technikę dropout
- jednej warstwy w pełni połączonej (full connect – fc)

- jednej warstwie wyjściowej (output)

Testowanie parametrów

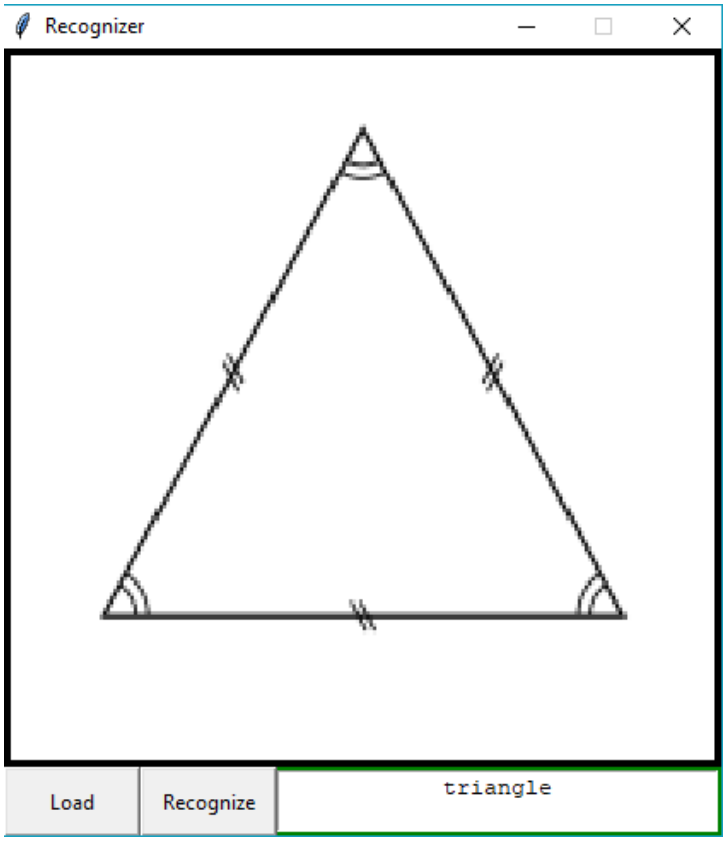
Testowanie sieci polegało na przetestowaniu jej parametrów w celu określenia optymalnych. Testom podlegały następujące parametry:

- współczynnik uczenia (learning rate)
- ilość epok
- ilość warstw konwolucyjnych
- algorytm uczenia
- funkcja aktywacji w warstwie końcowej

Wyniki poszczególnych testów zawiera tabela w rozdziale: Testy.

Graficzny Interfejs Użytkownika (GUI)

Aplikacja została napisana w języku Python przy użyciu biblioteki Tkinter. Umożliwia w wygodny i intuicyjny sposób klasyfikowanie dowolnego obrazu, który użytkownik podaje na wejściu. Przykładowe działanie aplikacji zostało przedstawione na poniższym obrazie:



Obraz 2

Testy

Tabela z wynikami uczenia i testowania

learning rate	batch size	epochs	the number of data	D	*units*	*conv1, conv2 l.filtrów*	*conv1, conv2 size*	*pool1, pool2, size*
0.001	32	3000	10 000	0.8	*1024*	*32 , 64*	*5x5, 5x5*	*3x3, 2x2*
Wyniki:	INFO:tensorflow:loss = 0.30733868, step = 51 INFO:tensorflow:global_step/sec: 17.2518 INFO:tensorflow:loss = 0.15010309, step = 151 (5.793 sec) INFO:tensorflow:loss = 0.27133906, step = 251 (4.928 sec) INFO:tensorflow:loss = 0.3739342, step = 351 (4.991 sec) INFO:tensorflow:loss = 0.18825337, step = 451 (4.795 sec) ... INFO:tensorflow:loss = 0.031888213, step = 1651 (6.620 sec) INFO:tensorflow:loss = 0.09209482, step = 1751 (7.390 sec) INFO:tensorflow:loss = 0.15178043, step = 1851 (5.911 sec) INFO:tensorflow:loss = 0.20764065, step = 1951 (6.091 sec) ... INFO:tensorflow:loss = 0.033816975, step = 2651 (4.744 sec) INFO:tensorflow:loss = 0.045167424, step = 2751 (4.670 sec) INFO:tensorflow:loss = 0.06171717, step = 2851 (5.391 sec) INFO:tensorflow:loss = 0.12565175, step = 2951 (4.538 sec) INFO:tensorflow:Saving checkpoints for 3050 INFO:tensorflow:Loss for final step: 0.063124426. {'accuracy': 0.97325, 'loss': 0.08085338, 'global_step': 3050}							
0.001	32	3000	10 000	0.8	*1024*	*10 , 20*	*5x5, 5x5*	*3x3, 2x2*
Wyniki:	INFO:tensorflow:loss = 26.441586, step = 1 INFO:tensorflow:global_step/sec: 38.8849 INFO:tensorflow:loss = 0.6678389, step = 101 (2.573 sec) INFO:tensorflow:loss = 0.43748832, step = 201 (2.268 sec) INFO:tensorflow:loss = 0.58719957, step = 301 (2.218 sec) INFO:tensorflow:loss = 0.49927074, step = 401 (2.243 sec) ... INFO:tensorflow:loss = 0.029825365, step = 1601 (2.308 sec) INFO:tensorflow:loss = 0.099383414, step = 1701 (2.284 sec) INFO:tensorflow:loss = 0.18413216, step = 1801 (2.436 sec) INFO:tensorflow:loss = 0.18608892, step = 1901 (2.147 sec) ... INFO:tensorflow:loss = 0.12396299, step = 2601 (2.709 sec) INFO:tensorflow:loss = 0.033139575, step = 2701 (2.915 sec) INFO:tensorflow:loss = 0.1635159, step = 2801 (2.700 sec) INFO:tensorflow:loss = 0.26868844, step = 2901 (2.294 sec) INFO:tensorflow:Saving checkpoints for 3000 INFO:tensorflow:Loss for final step: 0.11789802. {'accuracy': 0.96075, 'loss': 0.10765614, 'global_step': 3000} Wniosek: Rezultat gorszy niż w przypadku większej liczby filtrów, tj. 32 i 64. Zaletą jest szybsze wykonanie obliczeń, około 1,5 razy							

0.001	32	3000	10 000	0.8	*1024*	*32 , 64*	*5x5, 5x5*	*4x4, 3x3*
Wyniki:	INFO:tensorflow:loss = 31.766521, step = 1 INFO:tensorflow:global_step/sec: 34.148 INFO:tensorflow:loss = 0.5967581, step = 101 (2.929 sec) INFO:tensorflow:loss = 0.14856534, step = 201 (2.538 sec) INFO:tensorflow:loss = 0.09460824, step = 301 (2.254 sec) INFO:tensorflow:loss = 0.44611755, step = 401 (2.729 sec) ... INFO:tensorflow:loss = 0.009308245, step = 1601 (2.578 sec) INFO:tensorflow:loss = 0.029362513, step = 1701 (2.691 sec) INFO:tensorflow:loss = 0.18299319, step = 1801 (2.674 sec) INFO:tensorflow:loss = 0.13080269, step = 1901 (2.520 sec) ... INFO:tensorflow:loss = 0.04012815, step = 2601 (2.645 sec) INFO:tensorflow:loss = 0.03213796, step = 2701 (2.426 sec) INFO:tensorflow:loss = 0.07745548, step = 2801 (2.633 sec) INFO:tensorflow:loss = 0.04653777, step = 2901 (2.466 sec) INFO:tensorflow:Saving checkpoints for 3000 INFO:tensorflow:Loss for final step: 0.09366407. {'accuracy': 0.974, 'loss': 0.08048587, 'global_step': 3000} Wniosek:Lepszy rezultat niż w przypadku pool1 3x3 i pool2 2x2, jednakże gorszy niż pool1 2x2 i pool2 2x2							
0.00001	32	3000	10 000	0.8	*1024*	*32 , 64*	*5x5, 5x5*	*4x4, 3x3*
	INFO:tensorflow:loss = 16.07301, step = 1 INFO:tensorflow:global_step/sec: 37.1042 INFO:tensorflow:loss = 11.825712, step = 101 (2.695 sec) INFO:tensorflow:loss = 12.789598, step = 201 (2.304 sec) INFO:tensorflow:loss = 7.1183467, step = 301 (2.527 sec) INFO:tensorflow:loss = 8.223159, step = 401 (2.415 sec) ... INFO:tensorflow:loss = 1.0644301, step = 1601 (2.347 sec) INFO:tensorflow:loss = 2.0838003, step = 1701 (2.300 sec) INFO:tensorflow:loss = 1.5326867, step = 1801 (2.315 sec) INFO:tensorflow:loss = 2.5172691, step = 1901 (2.409 sec) ... INFO:tensorflow:loss = 0.9549405, step = 2601 (2.351 sec) INFO:tensorflow:loss = 1.0866656, step = 2701 (2.230 sec) INFO:tensorflow:loss = 1.0984383, step = 2801 (2.196 sec) INFO:tensorflow:loss = 0.7136529, step = 2901 (2.252 sec) INFO:tensorflow:Saving checkpoints for 3000 INFO:tensorflow:Loss for final step: 1.1004095. {'accuracy': 0.873875, 'loss': 0.40572217, 'global_step': 3000} Wniosek: Zbyt niska wartość współczynnika learning rate							
0.01	32	3000	10 000	0.8	*1024*	*32 , 64*	*5x5, 5x5*	*3x3, 2x2*
	INFO:tensorflow:loss = 20.675999, step = 1 INFO:tensorflow:global_step/sec: 25.6448 INFO:tensorflow:loss = 0.26067093, step = 101 (3.899 sec) INFO:tensorflow:loss = 0.24316646, step = 201 (3.706 sec) INFO:tensorflow:loss = 0.30980355, step = 301 (3.467 sec) INFO:tensorflow:loss = 0.13129035, step = 401 (3.533 sec) ... INFO:tensorflow:loss = 0.092621215, step = 1601 (3.304 sec) INFO:tensorflow:loss = 0.11742373, step = 1701 (3.892 sec) INFO:tensorflow:loss = 0.30458066, step = 1801 (3.333 sec) INFO:tensorflow:loss = 0.36364192, step = 1901 (3.182 sec) ... INFO:tensorflow:loss = 0.047625486, step = 2601 (3.197 sec) INFO:tensorflow:loss = 0.03490088, step = 2701 (3.137 sec) INFO:tensorflow:loss = 0.1548295, step = 2801 (3.120 sec) INFO:tensorflow:loss = 0.20882538, step = 2901 (3.410 sec) INFO:tensorflow:Saving checkpoints for 3000 INFO:tensorflow:Loss for final step: 0.12985042. {'accuracy': 0.969875, 'loss': 0.08920808, 'global_step': 3000}							

	Wniosek: Zbyt wysoka wartość współczynnika learning rate							
0.0001	32	3000	10 000	0.8	*32*	*32 , 64*	*5x5, 5x5*	*3x3, 2x2*
	INFO:tensorflow:loss = 41.88728, step = 1 INFO:tensorflow:global_step/sec: 36.5862 INFO:tensorflow:loss = 1.6541309, step = 101 (2.733 sec) INFO:tensorflow:loss = 1.4665015, step = 201 (2.531 sec) INFO:tensorflow:loss = 1.0184181, step = 301 (2.256 sec) INFO:tensorflow:loss = 0.7081888, step = 401 (2.284 sec) ... INFO:tensorflow:loss = 0.48136255, step = 1601 (2.495 sec) INFO:tensorflow:loss = 0.7816095, step = 1701 (2.296 sec) INFO:tensorflow:loss = 0.5577041, step = 1801 (2.294 sec) INFO:tensorflow:loss = 0.6444434, step = 1901 (2.380 sec) … INFO:tensorflow:loss = 0.68339944, step = 2701 (2.306 sec) INFO:tensorflow:loss = 0.8448539, step = 2801 (2.390 sec) INFO:tensorflow:loss = 0.45703796, step = 2901 (2.433 sec) INFO:tensorflow:Saving checkpoints for 3000 INFO:tensorflow:Loss for final step: 0.43914413. {'accuracy': 0.871125, 'loss': 0.3496554, 'global_step': 3000} Wniosek: Rezultat gorszy niż w przypadku mniejszej liczby neuronów, tj.1024							
0.001	32	3000	10 000	0.8	*2048*	*32 , 64*	*5x5, 5x5*	*3x3, 2x2*
	INFO:tensorflow:loss = 21.124092, step = 1 INFO:tensorflow:global_step/sec: 22.3107 INFO:tensorflow:loss = 0.3878967, step = 101 (4.467 sec) INFO:tensorflow:loss = 0.30771837, step = 201 (4.032 sec) INFO:tensorflow:loss = 0.19603282, step = 301 (4.097 sec) INFO:tensorflow:loss = 0.24411944, step = 401 (4.369 sec) ... INFO:tensorflow:loss = 0.07775119, step = 1601 (4.078 sec) INFO:tensorflow:loss = 0.026634116, step = 1701 (4.123 sec) INFO:tensorflow:loss = 0.1225611, step = 1801 (4.982 sec) INFO:tensorflow:loss = 0.15551135, step = 1901 (4.666 sec) ... INFO:tensorflow:loss = 0.021261089, step = 2601 (4.695 sec) INFO:tensorflow:loss = 0.009955495, step = 2701 (4.903 sec) INFO:tensorflow:loss = 0.062460423, step = 2801 (4.920 sec) INFO:tensorflow:loss = 0.075649455, step = 2901 (4.488 sec) INFO:tensorflow:Saving checkpoints for 3000 INFO:tensorflow:Loss for final step: 0.050068207. {'accuracy': 0.972875, 'loss': 0.08179901, 'global_step': 3000} Wniosek: Rezultat gorszy niż w przypadku mniejszej liczby neuronów, tj.1024							
0.001	128	3000	10 000	0.8	*1024*	*32 , 64*	*5x5, 5x5*	*3x3, 2x2*
	INFO:tensorflow:loss = 39.33452, step = 1 INFO:tensorflow:global_step/sec: 8.47692 INFO:tensorflow:loss = 0.4708563, step = 101 (11.782 sec) INFO:tensorflow:loss = 0.39662462, step = 201 (11.711 sec) INFO:tensorflow:loss = 0.26990983, step = 301 (11.237 sec) INFO:tensorflow:loss = 0.23637895, step = 401 (10.998 sec) ... INFO:tensorflow:loss = 0.15400615, step = 1501 (11.190 sec) INFO:tensorflow:loss = 0.13139966, step = 1601 (11.805 sec) INFO:tensorflow:loss = 0.064102106, step = 1701 (11.207 sec) INFO:tensorflow:loss = 0.21690843, step = 1801 (10.986 sec) ... INFO:tensorflow:loss = 0.097921506, step = 2601 (11.800 sec) INFO:tensorflow:loss = 0.11025455, step = 2701 (11.284 sec) INFO:tensorflow:loss = 0.06939028, step = 2801 (11.664 sec) INFO:tensorflow:loss = 0.05975999, step = 2901 (11.656 sec) INFO:tensorflow:Saving checkpoints for 3000							

	INFO:tensorflow:Loss for final step: 0.15774795. {'accuracy': 0.97425, 'loss': 0.083717816, 'global_step': 3000} Wynik: Rezultat lepszy o 0.1% w porównaniu z przypadkiem gdy batch size = 32 Wada: czas obliczeń został wydłużony ok 2.5 razy							
0.001	32	3000	40 000	0.95	*1024*	*32 , 64*	*5x5, 5x5*	*3x3, 2x2*
	INFO:tensorflow:loss = 17.879753, step = 1 INFO:tensorflow:global_step/sec: 26.3949 INFO:tensorflow:loss = 0.2317445, step = 101 (3.790 sec) INFO:tensorflow:loss = 0.32196242, step = 201 (3.362 sec) INFO:tensorflow:loss = 0.26066566, step = 301 (3.457 sec) INFO:tensorflow:loss = 0.108417034, step = 401 (3.424 sec) ... INFO:tensorflow:loss = 0.2102659, step = 1601 (3.357 sec) INFO:tensorflow:loss = 0.29969308, step = 1701 (3.574 sec) INFO:tensorflow:loss = 0.09425877, step = 1801 (3.527 sec) INFO:tensorflow:loss = 0.123214364, step = 1901 (3.432 sec) ... INFO:tensorflow:loss = 0.20347285, step = 2701 (3.514 sec) INFO:tensorflow:loss = 0.025033832, step = 2801 (3.349 sec) INFO:tensorflow:loss = 0.13901456, step = 2901 (3.396 sec) INFO:tensorflow:Saving checkpoints for 3000 INFO:tensorflow:Loss for final step: 0.10309763. {'accuracy': 0.966875, 'loss': 0.09800799, 'global_step': 3000} Wniosek: Zwiększona liczba danych trenujących (38 000) przy zachowaniu ilości danych testujących (2000) skutkuje gorszym rezultatem niż w przypadku mniejszego zbioru trenującego (zachowując pozostałe wartości parametrów).							
0.001	32	3000	40 000	0.9	*1024*	*32 , 64*	*5x5, 5x5*	*3x3, 2x2*
	INFO:tensorflow:loss = 34.63279, step = 1 INFO:tensorflow:global_step/sec: 24.47 INFO:tensorflow:loss = 0.8143069, step = 101 (4.088 sec) INFO:tensorflow:loss = 0.5800495, step = 201 (3.577 sec) INFO:tensorflow:loss = 0.17879683, step = 301 (3.564 sec) INFO:tensorflow:loss = 0.28327474, step = 401 (3.578 sec) … INFO:tensorflow:loss = 0.17691524, step = 1501 (3.426 sec) INFO:tensorflow:loss = 0.1910481, step = 1601 (3.409 sec) INFO:tensorflow:loss = 0.16364062, step = 1701 (3.515 sec) INFO:tensorflow:loss = 0.21363933, step = 1801 (3.396 sec) ... INFO:tensorflow:loss = 0.023291044, step = 2701 (3.498 sec) INFO:tensorflow:loss = 0.10438112, step = 2801 (3.501 sec) INFO:tensorflow:loss = 0.13728009, step = 2901 (3.375 sec) INFO:tensorflow:Saving checkpoints for 3000. INFO:tensorflow:Loss for final step: 0.07441478. {'accuracy': 0.9681875, 'loss': 0.093103, 'global_step': 3000} Wniosek: Analogicznie jak przykład wyżej, jednak rezultat lepszy niż w przypadku D = 0.95							
0.001	32	3000	40 000	0.8	*1024*	*32 , 64*	*5x5, 5x5*	*2x2, 2x2*
	INFO:tensorflow:loss = 34.50894, step = 1 INFO:tensorflow:global_step/sec: 13.8258 INFO:tensorflow:loss = 0.53108305, step = 101 (7.233 sec) INFO:tensorflow:loss = 0.19846791, step = 201 (6.926 sec) INFO:tensorflow:loss = 0.30724072, step = 301 (7.115 sec) INFO:tensorflow:loss = 0.17951885, step = 401 (7.053 sec) ... INFO:tensorflow:loss = 0.21716917, step = 1501 (7.288 sec) INFO:tensorflow:loss = 0.26237, step = 1601 (7.204 sec) INFO:tensorflow:loss = 0.29712948, step = 1701 (7.393 sec) INFO:tensorflow:loss = 0.17189181, step = 1801 (7.139 sec) ... INFO:tensorflow:loss = 0.04786882, step = 2601 (7.054 sec)							

	INFO:tensorflow:loss = 0.45375717, step = 2701 (7.438 sec) INFO:tensorflow:loss = 0.18597187, step = 2801 (7.406 sec) INFO:tensorflow:loss = 0.12372989, step = 2901 (6.946 sec) INFO:tensorflow:Saving checkpoints for 3000. INFO:tensorflow:Loss for final step: 0.033542335. {'accuracy': 0.9681563, 'loss': 0.1028255, 'global_step': 3000} Wniosek: W naszym przypadku dla macierzy głosujących o wymiarach 2x2, 2x2 otrzymano gorsze rezultaty niż w 1 przypadku (macierzy 3x3 oraz 2x2). Równocześnie czas obliczeń został wydłużony około 0,5 razy w porównaniu ze wspomnianym przykładem.							
0.001	32	200	10 000	0.8	*1024*	*32 , 64*	*5x5, 5x5*	*3x3, 2x2*
	INFO:tensorflow:loss = 21.117504, step = 1 INFO:tensorflow:global_step/sec: 27.5543 INFO:tensorflow:loss = 0.50047565, step = 101 (3.629 sec) INFO:tensorflow:Saving checkpoints for 200 INFO:tensorflow:Loss for final step: 0.45580563. {'accuracy': 0.9155, 'loss': 0.22906436, 'global_step': 200} Wniosek: Niewystarczająca liczba epok skutkuje nie wyuczeniem sieci.							
0.001	32	20 000	15 000	0.8	*1024*	*32 , 64*	*5x5, 5x5*	*3x3, 2x2*
	{'accuracy': 0.98075, 'loss': 0.061270174, 'global_step': 10 000} Wniosek: Duża liczba epok oraz większa ilość danych prawdopodobnie wpłynęła na znaczną poprawę ostatecznego wyniku.							

Poniżej w tabeli przedstawiono wyniki treningu i testowania sieci z zastosowaniem różnego rodzaju funkcji kosztu:

learning rate	batch size	epochs	No. of data	D	*units*	*conv1, conv2 l.filtrów*	*conv1, conv2 size*	*pool1, pool2, size*
0.001	32	3000	10 000	0.8	*1024*	*32 , 64*	*5x5, 5x5*	*3x3, 2x2*
F.aktywacji: ReLU f. optymalizacji: 'SGD' Wyniki:	INFO:tensorflow:loss = 19.28673, step = 1 INFO:tensorflow:global_step/sec: 17.3526 INFO:tensorflow:loss = 0.6171347, step = 101 (5.761 sec) INFO:tensorflow:loss = 0.31913868, step = 201 (4.990 sec) INFO:tensorflow:loss = 0.30332744, step = 301 (6.855 sec) INFO:tensorflow:loss = 0.13010344, step = 401 (6.175 sec) ... INFO:tensorflow:loss = 0.24762689, step = 1501 (5.419 sec) INFO:tensorflow:loss = 0.17892215, step = 1601 (5.083 sec) INFO:tensorflow:loss = 0.18717127, step = 1701 (5.493 sec) INFO:tensorflow:loss = 0.045690082, step = 1801 (5.513 sec) ... INFO:tensorflow:loss = 0.005543088, step = 2601 (5.482 sec) INFO:tensorflow:loss = 0.0936856, step = 2701 (5.654 sec) INFO:tensorflow:loss = 0.07243414, step = 2801 (5.587 sec) INFO:tensorflow:loss = 0.036477137, step = 2901 (5.117 sec) INFO:tensorflow:Saving checkpoints for 3000 INFO:tensorflow:Loss for final step: 0.057269845. {'accuracy': 0.96825, 'loss': 0.097019866, 'global_step': 3000}							
F.aktywacji: ReLU f. optymalizacji: 'Adam' Wyniki:	INFO:tensorflow:loss = 34.473675, step = 1 INFO:tensorflow:global_step/sec: 14.5856 INFO:tensorflow:loss = 0.20169273, step = 101 (6.855 sec) INFO:tensorflow:loss = 0.18353964, step = 201 (6.594 sec INFO:tensorflow:loss = 0.15045601, step = 301 (6.987 sec) INFO:tensorflow:loss = 0.10836896, step = 401 (7.053 sec) ... INFO:tensorflow:loss = 0.029423371, step = 1501 (6.169 sec) INFO:tensorflow:loss = 0.046642452, step = 1601 (5.547 sec) INFO:tensorflow:loss = 0.08637902, step = 1701 (5.941 sec) INFO:tensorflow:loss = 0.0029631103, step = 1801 (6.550 sec)							

	... INFO:tensorflow:loss = 0.10151286, step = 2701 (5.262 sec) INFO:tensorflow:loss = 0.034535654, step = 2801 (5.619 sec) INFO:tensorflow:loss = 0.00080576155, step = 2901 (5.874 sec) INFO:tensorflow:Saving checkpoints for 3000 INFO:tensorflow:Loss for final step: 0.07310441. {'accuracy': 0.9625, 'loss': 0.18037197, 'global_step': 3000} Wniosek: Wzrost błędu testowania o 0.5% w porównaniu z poprzednim przykładem
F.aktywacji: ReLU f. optymalizacji: 'Adagrad' Wyniki:	INFO:tensorflow:loss = 32.853447, step = 1 INFO:tensorflow:global_step/sec: 16.9992 INFO:tensorflow:loss = 1.2772653, step = 101 (5.881 sec) INFO:tensorflow:loss = 0.54968804, step = 201 (5.758 sec) INFO:tensorflow:loss = 0.40553415, step = 301 (7.439 sec) INFO:tensorflow:loss = 0.09632772, step = 401 (6.861 sec ... INFO:tensorflow:loss = 0.43230405, step = 1501 (5.838 sec) INFO:tensorflow:loss = 0.22375193, step = 1601 (6.443 sec) INFO:tensorflow:loss = 0.10397084, step = 1701 (5.484 sec) INFO:tensorflow:loss = 0.27837428, step = 1801 (5.475 sec) ... INFO:tensorflow:loss = 0.2355638, step = 2701 (6.568 sec) INFO:tensorflow:loss = 0.0949536, step = 2801 (5.659 sec) INFO:tensorflow:loss = 0.21946523, step = 2901 (5.509 sec) INFO:tensorflow:Saving checkpoints for 3000 INFO:tensorflow:Loss for final step: 0.145916. {'accuracy': 0.9585625, 'loss': 0.14364411, 'global_step': 3000} Wniosek: Wzrost błędu testowania o 1 % w porównaniu z pierwszy przykładem

Poniżej w tabeli przedstawiono wyniki treningu i testowania sieci z zastosowaniem różnego rodzaju funkcji aktywacji w warstwie *dense* (końcowej z 1024 neuronami).

learning rate	batch size	epochs	No. of data	D	*units*	*conv1, conv2 l.filtrów*	*conv1, conv2 size*	*pool1, pool2, size*
0.001	32	3000	10 000	0.8	*1024*	*32 , 64*	*5x5, 5x5*	*3x3, 2x2*
F.aktywacji: 'tanh' f. optymalizacji: 'SGD' Wyniki:	INFO:tensorflow:loss = 2.4624288, step = 1 INFO:tensorflow:global_step/sec: 15.2573 INFO:tensorflow:loss = 0.6948135, step = 101 (6.554 sec) INFO:tensorflow:loss = 0.455294, step = 201 (5.816 sec) INFO:tensorflow:loss = 0.45351535, step = 301 (6.268 sec) INFO:tensorflow:loss = 0.34781128, step = 401 (5.632 sec) ... INFO:tensorflow:loss = 0.48577446, step = 1501 (5.220 sec) INFO:tensorflow:loss = 0.21276787, step = 1601 (5.342 sec) INFO:tensorflow:loss = 0.15147495, step = 1701 (5.607 sec) INFO:tensorflow:loss = 0.09309605, step = 1801 (5.386 sec) ... INFO:tensorflow:loss = 0.02640666, step = 2601 (5.238 sec) INFO:tensorflow:loss = 0.2335428, step = 2701 (5.143 sec) INFO:tensorflow:loss = 0.093856625, step = 2801 (5.291 sec) INFO:tensorflow:loss = 0.026159868, step = 2901 (5.250 sec) INFO:tensorflow:Saving checkpoints for 3000 INFO:tensorflow:Loss for final step: 0.25972965. {'accuracy': 0.9563125, 'loss': 0.12430986, 'global_step': 3000} Wniosek: Wzrost błędu testowania o 1.2 % w porównaniu z użyciem funkcji aktywacji 'ReLU'							
F.aktywacji: 'sigmoid'	INFO:tensorflow:loss = 2.1977324, step = 1 INFO:tensorflow:global_step/sec: 18.2619 INFO:tensorflow:loss = 0.6493024, step = 101 (5.475 sec) INFO:tensorflow:loss = 0.5953043, step = 201 (4.996 sec) INFO:tensorflow:loss = 0.52864194, step = 301 (5.147 sec)							

f. optymalizacji: 'SGD' Wyniki:	INFO:tensorflow:loss = 0.1869964, step = 401 (5.291 sec) ... INFO:tensorflow:loss = 0.3049881, step = 1501 (5.283 sec) INFO:tensorflow:loss = 0.3100931, step = 1601 (5.004 sec) INFO:tensorflow:loss = 0.23541811, step = 1701 (5.297 sec) INFO:tensorflow:loss = 0.20520696, step = 1801 (4.996 sec) ... INFO:tensorflow:loss = 0.07216778, step = 2601 (4.993 sec) INFO:tensorflow:loss = 0.13014968, step = 2701 (4.983 sec) INFO:tensorflow:loss = 0.28614077, step = 2801 (6.026 sec) INFO:tensorflow:loss = 0.15069592, step = 2901 (7.017 sec) INFO:tensorflow:Saving checkpoints for 3000 INFO:tensorflow:Loss for final step: 0.19194351. {'accuracy': 0.9443125, 'loss': 0.16422223, 'global_step': 3000} Wniosek: Wzrost błędu testowania o 2.4 % w porównaniu z użyciem funkcji aktywacji 'ReLU'
F.aktywacji: 'softplus' f. optymalizacji: 'SGD' Wyniki:	INFO:tensorflow:loss = 22.620834, step = 1 INFO:tensorflow:global_step/sec: 18.7899 INFO:tensorflow:loss = 0.73266506, step = 101 (5.321 sec) INFO:tensorflow:loss = 0.30769348, step = 201 (4.859 sec) INFO:tensorflow:loss = 0.5570066, step = 301 (5.067 sec) INFO:tensorflow:loss = 0.13406321, step = 401 (5.327 sec) ... INFO:tensorflow:loss = 0.31266743, step = 1501 (5.440 sec) INFO:tensorflow:loss = 0.15937202, step = 1601 (5.233 sec) INFO:tensorflow:loss = 0.104498915, step = 1701 (5.258 sec) INFO:tensorflow:loss = 0.15908626, step = 1801 (5.398 sec) ... INFO:tensorflow:loss = 0.013213908, step = 2601 (5.737 sec) INFO:tensorflow:loss = 0.13123754, step = 2701 (5.445 sec) INFO:tensorflow:loss = 0.055847824, step = 2801 (5.280 sec) INFO:tensorflow:loss = 0.02506992, step = 2901 (5.918 sec) INFO:tensorflow:Saving checkpoints for 3000 INFO:tensorflow:Loss for final step: 0.110096276. {'accuracy': 0.9683125, 'loss': 0.09888154, 'global_step': 3000} Wniosek: Wartość błędu dla testowania nieznacznie w porównaniu z użyciem funkcji aktywacji 'ReLU' (o około 0.01%)
F.aktywacji: 'relu6' f. optymalizacji: 'SGD' Wyniki:	INFO:tensorflow:loss = 7.130274, step = 1 INFO:tensorflow:loss = 1.5983177, step = 101 (5.890 sec) INFO:tensorflow:loss = 1.1302364, step = 201 (6.634 sec) INFO:tensorflow:loss = 0.47782612, step = 301 (5.329 sec) INFO:tensorflow:loss = 0.26652166, step = 401 (4.951 sec) ... INFO:tensorflow:loss = 0.5232506, step = 1501 (5.806 sec) INFO:tensorflow:loss = 0.025068652, step = 1601 (5.353 sec) INFO:tensorflow:loss = 0.30094498, step = 1701 (5.003 sec) INFO:tensorflow:loss = 0.05244458, step = 1801 (4.932 sec) ... INFO:tensorflow:loss = 0.0038598087, step = 2601 (5.313 sec) INFO:tensorflow:loss = 0.05116356, step = 2701 (6.240 sec) INFO:tensorflow:loss = 0.07965459, step = 2801 (5.262 sec) INFO:tensorflow:loss = 0.24739754, step = 2901 (5.051 sec) INFO:tensorflow:Saving checkpoints for 3000 INFO:tensorflow:Loss for final step: 0.16802913. {'accuracy': 0.9565625, 'loss': 0.14688866, 'global_step': 3000} Wniosek: Wzrost błędu testowania o 1.2 % w porównaniu z użyciem funkcji aktywacji 'ReLU'
F.aktywacji: 'softsign' f. optymalizacji:	INFO:tensorflow:loss = 1.7654436, step = 1 INFO:tensorflow:global_step/sec: 18.7373 INFO:tensorflow:loss = 0.67032564, step = 101 (5.335 sec) INFO:tensorflow:loss = 0.42842638, step = 201 (6.265 sec) INFO:tensorflow:loss = 0.46843278, step = 301 (6.085 sec) INFO:tensorflow:loss = 0.23147307, step = 401 (5.571 sec) ...

'SGD' Wyniki:	INFO:tensorflow:loss = 0.18558905, step = 1501 (5.067 sec) INFO:tensorflow:loss = 0.219969, step = 1601 (5.116 sec) INFO:tensorflow:loss = 0.16993533, step = 1701 (5.045 sec) INFO:tensorflow:loss = 0.21847945, step = 1801 (5.063 sec) ... INFO:tensorflow:loss = 0.08524479, step = 2601 (5.053 sec) INFO:tensorflow:loss = 0.12566958, step = 2701 (5.197 sec) INFO:tensorflow:loss = 0.15113822, step = 2801 (5.021 sec) INFO:tensorflow:loss = 0.065280624, step = 2901 (5.295 sec) INFO:tensorflow:Saving checkpoints for 3000 INFO:tensorflow:Loss for final step: 0.14328866. {'accuracy': 0.9588125, 'loss': 0.11938826, 'global_step': 3000} Wniosek: Wzrost błędu testowania o 1 % w porównaniu z użyciem funkcji aktywacji 'ReLU'

Na podstawie wartości zamieszczonych w powyższych 2 tabelach, stworzonych w celu doboru najlepszej funkcji aktywacji i funkcji kosztu dla naszego zadania, najlepsze wartości testowania uzyskana dla zastosowania:

- Funkcja aktywacji: 'Softplus'
- Funkcja kosztu: 'SGD'

Testy przeprowadzono w dwóch seriach. W pierwszej treningowi poddawano zbiór danych bez poddawania go „czyszczeniu" z błędnych przykładów. W ten sposób maksymalna dokładność jaką udało się osiągnąć wyniosła 96%. Po odfiltrowaniu danych, uznanych za nieprawidłowe sieć osiągnęła wartość trafności 98%.

Najlepsza sieć

Najlepszą sieć uzyskano dla następujących parametrów uczących:

- learning rate: 0.001,
- liczba przetwarzanych obrazów w jednym kroku: 32,
- liczba epok: 20 000,
- liczba danych: 15 000,
- liczba neuronów w ostatniej warstwie: 1024,
- liczba filtrów w pierwszej warstwie: 32,
- liczba filtrów w drugiej warstwie: 64,
- rozmiar filtrów pierwszej warstwie: 5x5 piksele,
- rozmiar filtrów drugiej warstwy: 5x5 piksele,
- rozmiar pierwszego okna poolingu: 3x3 piksele,
- rozmiar drugiego okna poolingu: 2x2 piksele.

Podczas uczenia wykorzystano metodę największego spadku gradientu, do wprowadzania korekty parametrów filtrów. Jako funkcję aktywacji użyto funkcję ReLU (ang. Rectified Linear Units).

Testując sieć na danych nie należących do zbioru uczącego, uzyskano poprawność rozpoznawanych figur na poziomie 98%.

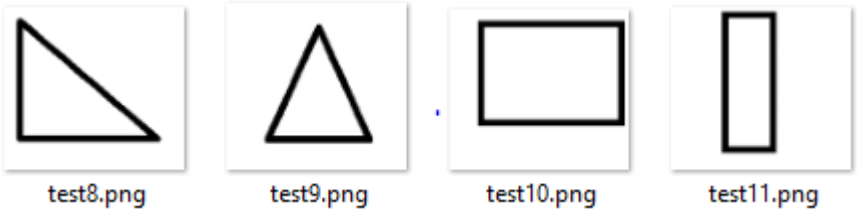
Podsumowanie

Analiza wyników testowania pokazuje że niezwykle istotnym elementem w procesie trenowania sieci jest jakość oraz ilość danych. Możliwość korzystania z dużego zbioru danych jest kluczowym elementem zarówno podczas trenowania sieci neuronowych jak i w innych dziedzinach uczenia maszynowego i analizy danych.

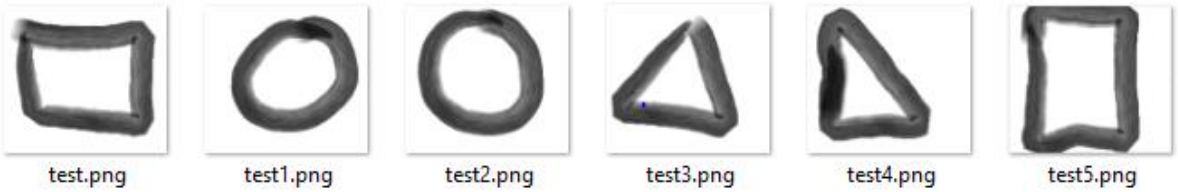
Kolejnym elementem który znacznie wpływał na jakość wyników okazała się być ilość epok.

Natomiast niewielkie znaczenie dla wyników okazały się mieć zmiany proporcji dla danych uczących i testujących. Wynika z prawdopodobnie z ich dużej ilości. Mając do dyspozycji dane w ilości 10 tys., różnica rzędu 1 tysiąca staje się bardzo mało znacząca. Innym parametrem, którego zmiany nie wywoływały większego pogorszenia się lub polepszenia wyniku jest współczynnik uczenia.

Jakość działania sieci, mimo satysfakcjonujących wyników liczbowych (98%) ma pewne ograniczenia. Związane są one bezpośrednio z wielkością analizowanych obrazów. Zbiór danych zawierał szkice o rozmiarze 28x28, dlatego też tylko obrazki o takiej wielkości sieć jest w stanie klasyfikować. Niesie to za sobą problem z rozróżnianiem od siebie kształtów takich jak koło i sześciokąt.
Wersja obrazka akceptowalnego przez sieć jest na tyle mała, że aby wyniki były satysfakcjonujące, grubość linii powinna być odpowiedniej grubości. Dla przykładu, natępujące obrazki prawdopodobnie nie zostaną odpowiednio sklasyfikowane:



Natomiast poniższe będą:



W celu uniknięcia tego typu ograniczeń, treningowi powinno poddać się obrazki o większym rozmiarze, co za tym idzie zwiększyłby się również czas treningu.

Źródła

- http://www.dsp.agh.edu.pl/_media/pl:dydaktyka:kodrzywolek_praca_magisterska_skrot.pdf
- http://ruder.io/optimizing-gradient-descent/
- https://ksopyla.com/machine-learning/siec-konwolucyjna-rozpoznawanie-cyfr-z-obrazow/
- https://www.tensorflow.org/