Uniwersytet Mikołaja Kopernika
Wydział Matematyki i Informatyki

Paweł Marcin Chojnacki
nr albumu: 260082
informatyka

Praca magisterska

Porównanie wydajności współczesnych architektur sieci neuronowych

Opiekun pracy dyplomowej
dr hab. Piotr Wiśniewski

Toruń 2018

Spis treści

Słownik pojęć

Klasyfikacja binarna

Klasyfikacja binarna polega na jak najdokładniejszym stwierdzeniu posiadania cechy lub przynależności do kategorii danego obiektu. Najczęściej używane metody do klasyfikacji binarnej to: drzewa decyzyjne Dane wejściowe należy przedstawić w formie macierzy. Każdy przykład do treningu i później klasyfikacji, musi być tych samych wymiarów. Wyjściem algorytmu klasyfikacji binarnej jest wektor z prawdopodobieństwem klasyfikacji każdego z przykładów. Wizualizacja funkcji, klasyfikującej czerwone i zielone kółka.

Regresja logistyczna

Metoda statystyczna używana do analizy zbioru danych, w którym mam więcej niż jedną zmienną determinującą wyjście. Wyjściem jest prawdopodobieństwo wystąpienia klasyfikowanego elementu. Algorytm regresji logistycznej przyjmuje na wejściu dane: n-wymiarowy wektor liczb rzeczywistych [np. Obraz], zestaw wag o tych samych wymiarach, liczba rzeczywista, bias. Do wygenerowania wyjścia, wystarczy obliczyć y = wT (wagi transponowane) * x (wejście) + b (bias). Należy jeszcze zastosować operację, która pozwoli na ustawienie parametrów w przedziale 0-1. Całe powyższe równanie użyć jako wejście do funkcji sigmoidy. Sigm(z) = 1 / 1 + e-z. Jeśli z jest duże, sigmoida będzie bliska 1, jeśli z jest liczbą ujemną, sigmoida zbliży się do 0.

Funkcja kosztu

Funkcja do trenowania modelu regresji logistycznej. Mając zestawy treningowe, można wytrenować algorytm tak, aby podawał wartości prawdopodobieństwa jak najbliższe zestawu treningowego. Chcemy ustawić wagi i bias tak, aby te parametry dawały prawidłową odpowiedź dla każdego przykładu uczącego. Funkcja kosztu ma następujący wzór:

Metoda gradientu prostego

Algorytm pozwalający znaleźć minimum funkcji. Wyobrażając sobie płaszczyznę funkcji dla wszystkich możliwych argumentów, algorytm przechodzi z losowo rozpoczętego miejsca w miejsce gdzie jest najgłębiej. Mając funkcję kosztu J(w,b) szukamy miejsca w którym błąd algorytmu jest jak najmniejszy.

Wykres obliczeniowy

(ang. Computation graph). Dekompozycja wyrażenia w pojedyncze atomowe kroki. Używany do optymalizowania funkcji. Przydaje się podczas ręcznej analizy funkcji błędu.

Funkcje aktywacji

- funkcja definiująca

Pooling

- technika polegająca na zmniejszeniu danych wejściowych. Aby zmniejszyć obraz ustala się rozmiar filtra oraz wielkość kroku. Następuje przejscie po całym obrazie (po nałożeniu filtra) i wybiera się max z całego obszaru filtra do nowej macierzy. Pozwala to ograniczyć ilość parametrów i wyodrębnić konkretne cechy obiektu.

Konwolucja

Wsteczna propagacja błędu

Dropout

Rektyfikowana jednostka liniowa (ReLU)

Softmax

Epoka

Model

Wstęp

Celem niniejszej pracy magisterskiej jest przedstawienie nowoczesnych, czyli z ostatnich pięciu lat, architektur sieci neuronowych oraz przegląd najpopularniejszych bibliotek implementujących algorytmy uczenia maszynowego i w szczególności zoptymalizowanych do pracy z uczeniem głębokim na wielu procesorach graficznych. Głębokie sieci neuronowe znane są już od lat 60tych XX wieku, algorytmy uczące również były aktywnie rozwijane do lat 80tych. Wtedy też nadzieje wojska amerykańskiego na sztuczną inteligencję wygasły powodując zimę SI przez brak funduszy na badania. Do 2012 roku była to dziedzina zarezerwowana dla doktorów matematyki i informatyki (głównie statystyków). Wtedy też pojawiły się pierwsze głębokie sieci neuronowe mające trafność blisko 80%. Od tej pory nastąpił gwałtowny rozwój start-upów związanych z Deep Learningiem. Moc obliczeniowa tanieje z każdym rokiem i kluczowe dla rozwoju dziedziny okazały się karty graficzne firmy NVidia (szczególnie te przeznaczone na rykek graczy). Właśnie przez brak ciągłości w rozwoju i nagłą modę, wszystkie obecnie używane narzędzia mają nie więcej niż 5 lat. Literatura w czasie wydania często staje się nieaktualna, dlatego wykorzystana literatura to głównie publikacje elektroniczne z serwisu arXiv.com, zazwyczaj jeszcze bez recenzji naukowej. W dalszej części pracy przedstawiane są główne problemy wydajnościowe, z którymi borykają się osoby praktykujące Deep Learning. Przez wydajność należy rozumieć zarówno czas obliczeń potrzebny do uczenia modelu rozpoznawania danego mu zagadnienia oraz dokładność z jaką już nauczony program potrafi rozpoznać niespotkane wcześniej dane. Oba te parametry są od siebie zależne, koszt poprawy jakości predykcji algorytmu o ostatnie 1-2% może wynosić setki tysięcy złotych. Na koszt składa się energia elektryczna, farma serwerów wyposażonych w karty graficzne oraz tygodnie czekania na wynik. Architektury posiadają wiele hiperparametrów, które wpływają na precyzję, po zmianie wartości dowolnego należy zacząć pracę algorytmu od początku. Następnie przedstawnionych jest kilka nowatorskich technik strojenia hiperparametrów, które ze względu na to że zostały

odkryte przez mało znanych badaczy, nie zostały jeszcze rozpowszechnione w dużych ośrodkach badawczych jak DeepMind, te techniki również są na tyle nowe, że nie ma wydawnictwa branżowego opisującego ich użycia dla praktyków. Te algorytmy pozwalają efektywnie zmniejszyć błąd klasyfikatora o 10-20%, co wiąże się z dużo wyższą wydajnością, przy znikomym nakładzie pracy. Do prezentacji działania bibliotek, architektur i algorytmów użyty został gotowy zbiór danych. Zestaw danych prezentacyjnych pochodzi z serwisu Kaggle. Jest to zbiór zdjęć z prześwietleń rentgenowskich podzielony na dwie kategorie: płuca zdrowe i płuca z zapaleniem płuc. Dane są anonimowe i zostały zweryfikowane przez Kaggle, na końcu pracy ze zdjęciami znajduje się porównanie z osiągnięciami społeczności. Badanym obszarem pozostają zagadnienia klasyfikacji obrazów. Na przykładzie zbioru zdjęć, prezentowany jest sposób wykorzystania gotowych moedeli i metody optymalizacji je pod wybraną domenę. Takie podejście pozwala uniknąć wysokich kosztów tworzenia własnych modeli (co kosztuje kilka do kilkunastu tygodni obliczeń na wielu kartach GPU). Głębokie sieci neuronowe również świetnie się sprawdzają przy zadaniach rozpoznawania mowy, przetwarzaniu języka naturalnego i systemach rekomendacji, zagadnienia zbyt obszerne by zostały opisane razem. Praca jest podzielona na etapy tworzenia modelu od dołu struktury sieci neuronowych. Rozpoczyna się od przedstawienia pojedynczych elementów i struktury matematycznej sieci. Następnie opisany jest każdy element tworzący gotową architekturę. Wszystkie przedstawione kody źródłowe zostały napisane w języku Python. Z założenia stał się językiem dominującym w uczeniu maszynowym, kosztem języka R. Python składnią jest zbliżony do pseudokodu, dodatkowo wszystkie popularne biblioteki udostępniają nakładki API dla Pythona. Książki wykorzystane w tworzeniu tego tekstu, zawierają wyłącznie kod Pythonowy (z kilkoma nawiązaniami do języków R i C++).

Rozdział 1.

Neurony

W niniejszym rozdzialę zostały przedstawione idee sztucznego neuronu McCullocha i Pittsa. Model neuronu był wczesnym wyobrażeniem neuronu mózgowego z 1943 roku, kilka lat później biolodzy odkryli że mózg wydląda i działa inaczej. Model obliczeniowy został stworzony na podstawie algorytmu logiki progowej. Nazwa została przyjęta w kręgach akademickich i biznesowych, stąd często osoby nie znające budowy mózgu ani działania sieci neuronowych mylnie nazywają zagadnienie Sztuczną Inteligencją. Sztuczne neurony znane są również jako perceptrony.

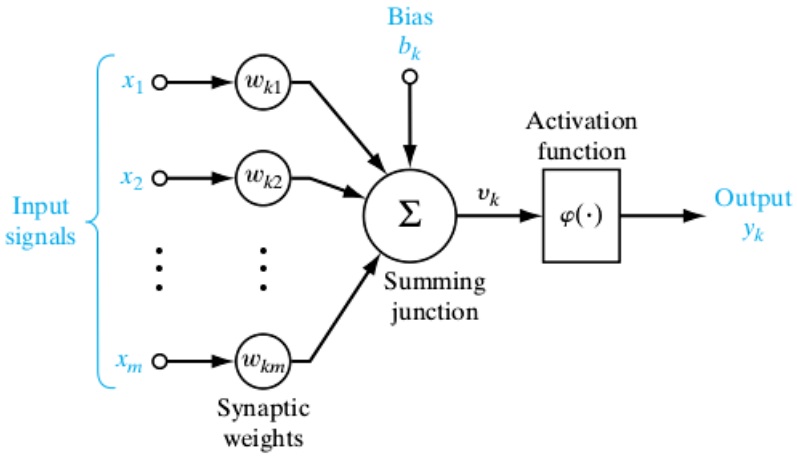
Sztuczny neuron jest funkcją matematyczną, otrzymuje jedną lub więcej wartości jako argument (dane wejściowe), następnie je sumuje z osobnymi wagami dla każdego wejścia. Dane wejściowe oraz wyjściowe mogą przyjmować wartości z ograniczonego przedziału. Na koniec wartość jest wejściem dla funkcji aktywacji neuronu. Funkcja aktywacji zwyczajowo ma kształt sygmoidy, ale stosowane są obecnie również funkcje ReLU. Aktywacja symuluje zachowanie, kiedy powyżej przekroczeniu pewnego progu (standardowo 0,5) aktywuje się dane zachowanie lub rozpoznanie cechy.

składa się funkcja f(x, w) -¿ x - wejście, w - waga. Ta prosta jednostka stanowi dziś podstawę budowy każdej sieci neuronowej. Aby funkcja zwracała oczekiwane wyniki, wagi powinny być poprawnie ustawione. Początkowo wagi ustawiano ręcznie za pomocą operatora, który wcześniej przeliczał je dla odpowiednich parametrów wejścia
wyjścia. W latach 50tych perceptron stał się pierwszym modelem umiejącym samodzielnie wyliczyć poprawnie wagi definiujące zadaną klasę na podstawie przykładów. Już kilka lat po wynalezieniu sieci neuronowych, okazało się że neurony mózgowe działają zupełnie inaczej, nomenklatura nawołująca do neuronów mózgowych jakkolwiek błędna, pozwala wprowadzić prostą abstrakcję na modele matematyczne.

Opiszę w jaki sposób stworzyć sieć neuronową oraz przedstawię wszystkie elementy potrzebne do zbudowania funkcjonalnej i wydajnej architektury sieci.

1.1. Model perceptronu prostego

Elementy z których buduje się sieci, charakteryzują się wieloma wejściami i jednym wyjściem. Wartości przekazywane na wejściu i wyjście zazwyczaj przyjmują znormalizowane wartości z przedziału xE [-1, 1] dla każdego z wejść oraz y E[-1,1] dla wyjścia. W uproszczeniu można przyjąć y = SUM (wi * xi). wi są nazywane wagami (dawniej wagami synaptycznymi) i podlegają zmianom w trakcie uczenia neuronu. Wagi te stanowią zasadniczą cechę sieci neuronowych działających jako adapcyjne systemy przetwarzania informacji.



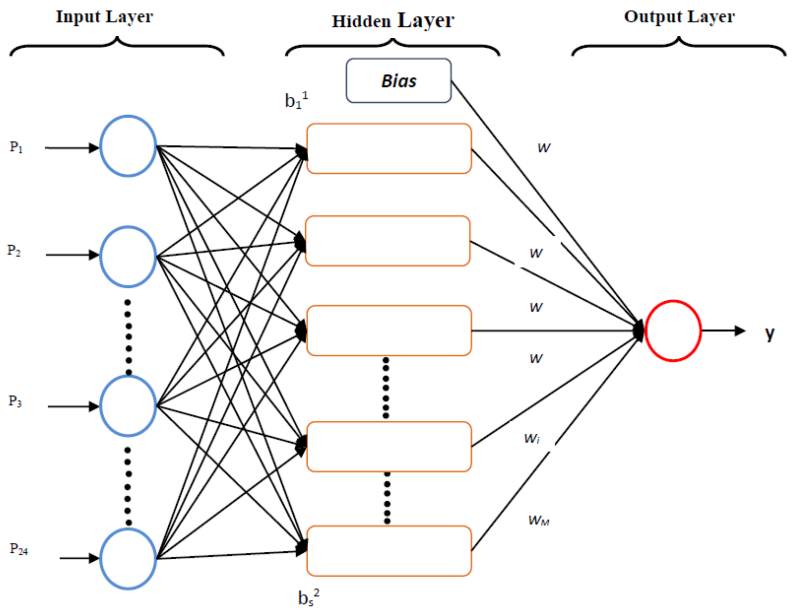
Rysunek 1.1: Prosta reprezentacja neuronu.

1.2. Co to jest sieć neuronowa?

Koncepcja sieci neuronowej to luźne połączenie wielu pojedynczych neuronów. Pozwala to na rozpoznawanie bardziej skomplikowanych wzorców niż klasyfikacja binarna.

1.3. Jak uczyć sieci neuronowe

W obecnym momencie istnieją dwie możliwości by sieć posiadła umiejętność poprawnej klasyfikacji.



Rysunek 1.2: Prosta sieć neuronowa.

Uczenie nadzorowane

Uczenie nienadzorowane

1.4. Elementy sieci neuronowych

1.4.1. Metoda gradientu prostego

1.4.2. Funkcje aktywacji

1.4.3. Wykres obliczeniowy

1.4.4. Propagacja i wsteczna propagacja błędu

1.4.5. Parametry

1.4.6. Hiperparametry

1.4.7. Jak zbudować sieć neuronową

Rozdział 2.

Deep Learning

2.1. Techniki tworzenia głębokich sieci

W tym rozdzial przybliżę trzy interesujące techniki głębokiego uczenia. Wybrałem je ze względu na ich obszerną ilość zastosowań. Następnie opiszę kilka bibliotek implementujących wsparcie dla większości architektur sieci przedstawionych w pracy. Biblioteki są wybrane na podstawie przekroju platform i języków mi znanych.

2.1.1. Sieci neuronowe

2.1.2. Convolutional Neural Networks

Inaczej znane jako

2.1.3. Recurrent Neural Networks

2.1.4. Fully Convolutional Networks

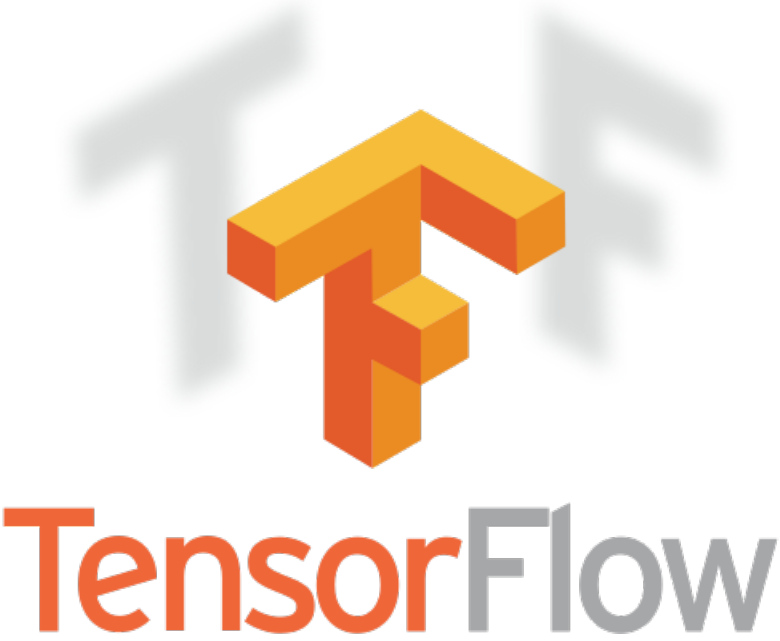
Rozdział 3.

Deep Learning

Rozdział 4.

Biblioteki implementujące uczenie głębokie

4.0.1. Tensorflow



Rysunek 4.1: Logo biblioteki Tensorflow

Pierwsza otwarta biblioteka od Google. Obecnie najczęściej używana podczas prac naukowych, o czym świadczy ilość cytowań[1]. Od 2018 roku dostępna również pod nazwą Tensorflow.js jako biblioteka javascript, udostępniające ten sam interfejs programistyczny.

Rysunek 4.2: Logo biblioteki Keras

4.0.2. Keras

Keras jest frameworkiem udostępniającym wysokopoziomowe API sieci neuronowych. Działanie opiera się na wykorzystaniu Tensorflow, CNTK lub Theano jako bibliotek wykonujących obliczenia. Głównym założeniem tego oprogramowania jest możliwość szybkiego wykonywania eksperymentów, co ma się przełożyć na lepsze badanie uczenia maszynowego.

4.0.3. PyTorch

4.0.4. Tensorflow.js [Deeplearn.js]

Znany dawniej jako DeepLearning.js Pierwszy zestaw uczenia maszynowego do użycia w przeglądarce. Działa w oparciu o WebGL. Dostarcza calej mocy Tensorflow do przeglądarki czy dowolnego interpretera kodu Javascript.

Rysunek 4.3: Logo biblioteki PyTorch

Rysunek 4.4: Logo biblioteki Tensorflow.js

4.0.5. PaddlePaddle

Chińskie oprogramowanie, mało znane w Europie / USA, za to jest to najpopularniejszy framework w Chinach. Uważam że zasługuje na wyróżnienie ze względu na unikalny charakter. Paddle jest open-source, jednak większość dokumentacji jest w języku kantońskim.

4.0.6. MXNet

4.0.7. Caffe2

4.0.8. ML.NET

Nowe oprogramowanie od Microsoftu, udostępnione na licencji [licencja], głównie polecane programistom w środowisku .NET. Docelowo ma stać się częścią .NET Core. Nastawione na łatwość integracji z oprogramowaniem biznesowym, chmurą Azure oraz łatwość obsługi przez programistów pracujących w technologiach Microsoftu.

Rysunek 4.5: Logo biblioteki Paddle

Rysunek 4.6: Logo biblioteki MXNet

4.0.9. PyBrain

4.0.10. CNTK

[Microsoft Research, 2016] - nazwa kompletna Microsoft Cognitive Toolkit. Jest to zestaw narzędzi z działu badawczego Microsoftu, zajmującego sie Deep Learningiem. Narzędzie opisuje sieci neuronowe jako szereg obliczeń na grafie skierowanym. W tym grafie węzły (liście) reprezentują wartości wejściowe oraz parametry sieci, zaś pozostałe węzły są operacjami macierzowymi na ich wejściu. Cognitive Toolkit pozwala użytkownikom w prosty sposób tworzyć i łączyć popularne modele jak (ang. fast-forward Deep Neural Networks), konwolucyjne sieci neuronowe, rekurencyjne sieci neuronowe. Implementuje algorytm spadku gradientowego, wsteczną propagację błędów z automatycznym skalowaniem na wiele urządzeń GPU rozproszonych na różnych serwerach. CNTK działa na systemach operacyjnych Windows i GNU
Linux.

Rysunek 4.7: Logo biblioteki Caffe2

Rysunek 4.8: Logo biblioteki ML.NET

Rysunek 4.9: Logo biblioteki PyBrain

Rysunek 4.10: Architektura sieci CNTK

Rozdział 5.

Architektura

5.1. Znaczenie architektury dla wydajności sieci

Sieci neuronowe można ułożyć na nieskończenie wiele sposobów. Aby uzyskać dobre rezultaty należy sprawdzić kilka architektur i jak się zachowują dla dla posiadanych zbiorów danych i zadanych hiperparametrów. W zależności od typu architektury sieci można osiągnąć zupełnie inne wyniki uczenia dla tych samych algorytmów. Architektury różnią się właściwie wszystkim, ilością warstw ukrytych ilością neuronów w warstwie ukrytej. Poglądy Tadeusiewicza na architekturę sieci neuronowych.

"Właśnie taka (warstwowa) struktura sieci wyjątkowo łatwo i wygodnie da się wytwarzać zarówno w formie modelu elektronicznego, jak i da się symulować w formie programu komputerowego. Dlatego badacze przyjęli właśnie strukturę warstwową i od tej pory stosują ją we wszystkich sztucznych sieciach neuronowych. Z pełną wiernością biologicznemu oryginałowi ma to niewiele wspólnego, ale jest praktyczne i wygodne. W związku z tym wszyscy tak postępują, nie martwiąc się ani przesłankami biologicznymi, ani dowodami wskazującymi, że architektura sieci bardziej wymyślnie dostosowanej do charakteru zadania może znacznie lepiej realizować stawiane zadania."

Tak napisał w 2007 roku, kiedy architektury sieci nie miały miały większego znaczenia ponieważ większość sieci była stosunkowo płytka (do 5 warstw ukrytych). Zbiory na których pracowano były stosunkowo niewielkie w porównaniu z dzisiejszymi zasobami skatalogowanych obrazów. Obecnie istnieją dowody (choćby coroczne zawody ImageNet, w których udowadnia się że struktura sieci ma znaczenie. Może poprawić szybkość i dokładność

uczenia). Profesor jest ekspertem w dziedzinie sieci neuronowych, jednak jego książki i poglądy są z lat 1990 - 2010. Późniejsze jego prace są dużo mniej znane.

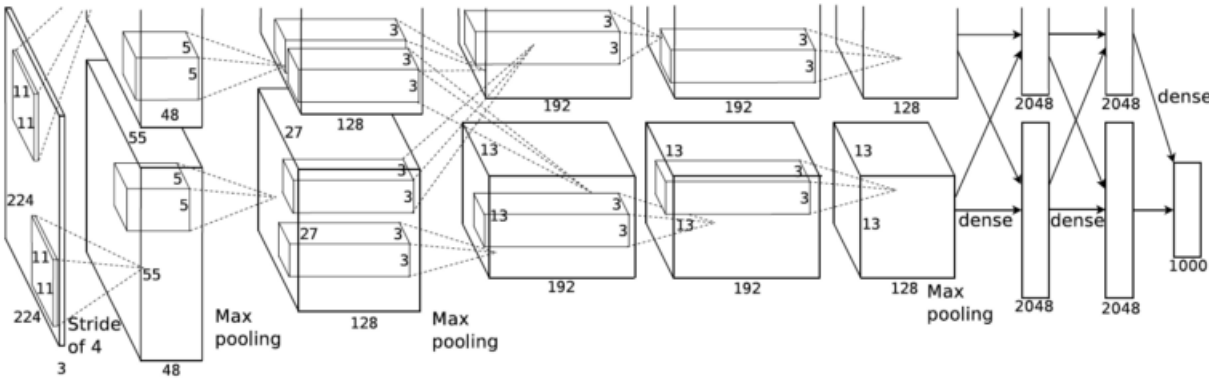
5.2. Przegląd najbardziej efektywnych architektur

5.2.1. AlexNet

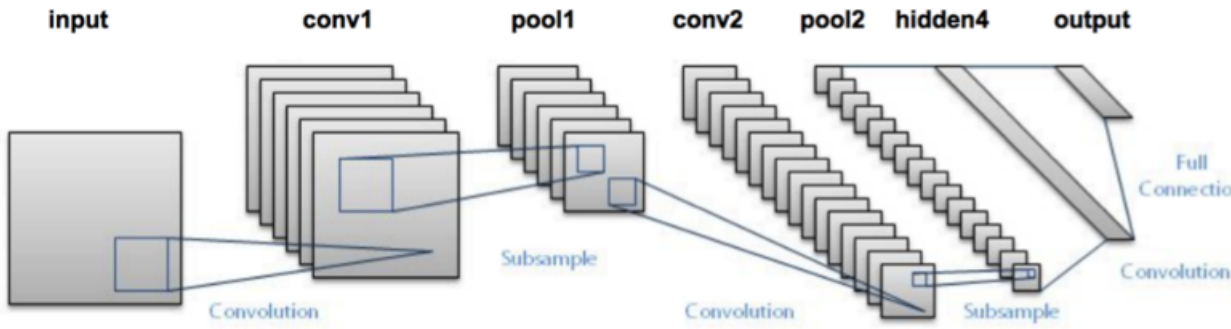
[Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, 2012] - pierwsza prawdziwa konwolucyjna sieć neuronowa (CNN), która pomogła zmienić opinię na temat uczenia głębokiego. Została zaimplementowana z użyciem biblioteki CUDA. Pierwsza sieć neuronowa, ucząca się z powodzeniem dla dużego zbioru danych. Została wytrenowana na zbiorze danych złożonym z ponad 15 milionów obrazów podzielonych na 22000 klas. W testach top-1 i top-5 uzyskała wartości kolejno 37,5% i 17%, co pozwoliło wygrać konkurs ImageNet Large Scale Visual Recognition Challenge. Sieć neuronowa zawiera 60 milionów parametrów i 650 000 neuronów. Architekturę tworzy 8 warstw, gdzie pierwsze 5 to warstwy konwolucyjne, a pozostałe 3 są warstwami w pełni połączonymi, ta ostatnia ma oczywiście 1000 wyjść z funkcji softmax. AlexNet znacząco przewyższyła swoją wydajnością poprzednich uczestników i wygrała zawody redukując błąd top-5 do 15,32%. Drugie miejsce to błąd ok 26.2% (nie była to CNN). Sieć jest głęboką modyfikacją architektury Yann'a LeCunn'a. AlexNet była zaplanowana na dwie karty graficzne, stąd rozdzielenie przepływu informacji na 2 części. Trenowanie sieci na 2 GPU było nowością na te czasy. Sieć została wytrenowana na zbiorze ImageNet. Do wyliczenia funkcji nieliniowych były używane ReLU (tutaj po raz pierwszy okazało się że ReLU działa dużo szybciej niż tanh). Sieć o której będą uczyły się dzieci na lekcjach historii. [2]

5.2.2. LeNet

[Authors, Year] - Jest to obecnie mało znacząca architektura. Pierwsze praktyczne zastosowanie sieci konwolucyjnych jeszcze w latach 90'tych. Używana była do prostych zadań: czytanie kodów pocztowych, liczb.[3]



Rysunek 5.1: Architektura sieci AlexNet



Rysunek 5.2: Architektura sieci LeNet

5.2.3. VGG Net

[Simoyan i Zisserman, 2014] - sieć złożona z 16 warstw konwolucyjnych, która charakteryzuje się małymi filtrami i duża głębokość sieci. W trakcie trenowania sieci wyjście jest ustawione na ustalony rozmiar (224 x 224 x 3). Przetwarzanie wstępne obejmuje odjęcie mediany wartości RGB dla każdego piksela. Zdjęcie jest przetwarzanie przez stos warstw konwolucyjnych, gdzie używane są filtry o bardzo małym polu widzenia (3x3) [najmniejszy możliwy rozmiar by móc rozpoznać kierunek]. Operacja Max-pooling jest wykonana na polu 4 pikseli, co pokazuje że jest to pobieranie jak najmniejszych cech z obrazu. Ukryte warstwy są wyposażone w nieliniową funkcję aktywacji ReLU. Po stosie warstw konwolucyjnych, następuje nałożenie 3 warstw w pełni połączonych (to takie duże warstwy zawierające wszystkie cechy?). Pierwsze dwie mają 4096 kanałów, trzecia już tylko 1000 (po jednym kanale na klasę obiektu). Obecnie architektura ta jest dość popularnym wyborem dla wyodrębniania cech ze zdjęć. Konfiguracje wag dla zbioru obrazów z ImageNet są dostępne online. W tej sieci problemem jest

140 milionów parametrów, którymi czasem trzeba zarządzać. [5]

5.2.4. GoogleNet / Inception

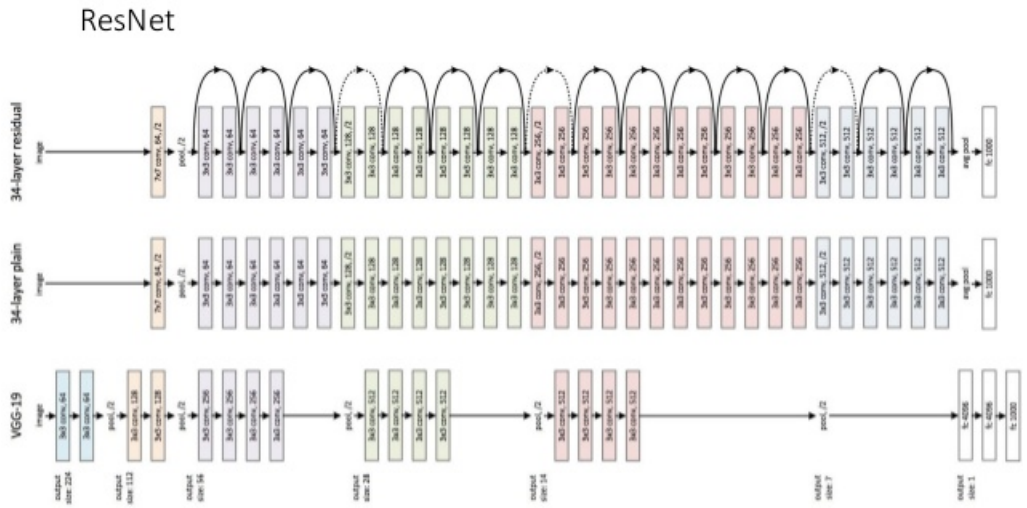
[Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, 2014] - produkt jak nazwa wskazuje wyszedł z firmy Google. Charakteryzuje się przełomową poprawą użycia zasobów wewnętrznych. ImageNet jest zaprojektowana na jak największą szerokość i głębokość jednocześnie utrzymując stałe zużycie mocy obliczeniowej. Optymalizacja jakości opiera się na zasadzie Hebbian'a (jest to teoria neuronauki, która mówi o adaptacji neuronów w trakcie nauki). Błędy klasyfikacji wynoszą top-1: 17,2% i top-5: 3,58% (dla v3). Sieć składa się łącznie z 22 warstw. Sieć jest modułowa, gdzie każdy moduł pełni rolę wielopoziomowego ekstraktora cech przeliczającego konwolucje na macierzach rozmiarów 1x1, 3x3, 5x5. Zaletą tej architektury są bardzo małe rozmiary wag (mniej niż 100MB dla v3). Kolejną ważną zaletą jest ilość parametrów, tylko 4 miliony co jest wynikiem 12 razy lepszym od AlexNet. Jest to jedna z najbardziej złożonych sieci względem architektury modułowej. W wersji naiwnej każda warstwa posiada 4 ekstraktory cech (3 konwolucyjne i jedną 3x3 max pooling), następnie wartości te są składane i wysyłane do warstwy wyżej. Do poprawienia wydajności obliczeniowej pozbyto się warstwy "w pełni połączonej". [6]

5.2.5. ResNet

[Kaiming He et al, 2015] - architektura oparta na mikro modułach. Charakteryzuje się możliwością odrzucania połączeń i posiada ogromny moduł na normalizację zbioru iteracji (batch normalization). Normalizacja jest pomysłem zaczerpniętym z rekurencyjnych sieci neuronowych. Ta technika pozwala stworzyć sieć ze 152 warstwami ukrytymi przy zachowaniu złożoności mniejszej niż VGG. [7]

5.2.6. ResNeXt

[Saining Xie, Ross Girshick, Piotr Dollar Zhuowen Tu Kaiming He, 2017] - modularyzowalna [7]



Rysunek 5.3: Architektura sieci ResNet

5.2.7. RCNN – obrazek jest zastępczy nie ma nic w internecie interesującego

[Authors, Year] - [4]

5.2.8. YOLO - You Only Look Once

[Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, 2016] - You Only Look Once jest systemem wykrywania obiektów w czasie rzeczywistym. Obraz jest dzielony na części oddzielonymi od sieibie prostokątami, ..??? Cały przepływ danych do wykrycia obiektów jest jedną siecią, dlatego może być zoptymalizowany [**?**]

5.2.9. SqueezeNet

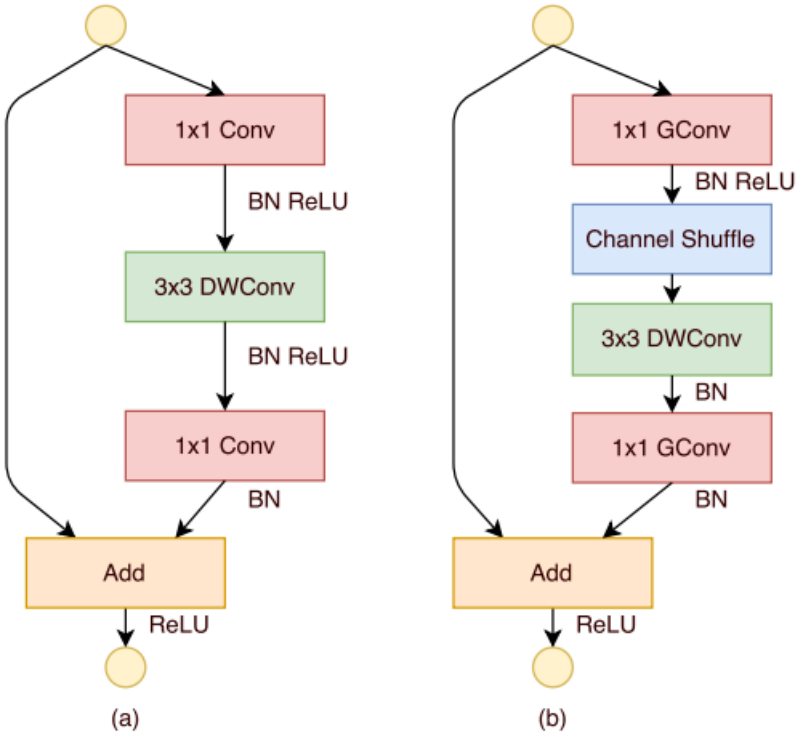
[Authors, Year] - [**?**]

5.2.10. SegNet

[Authors, Year] - [**?**]

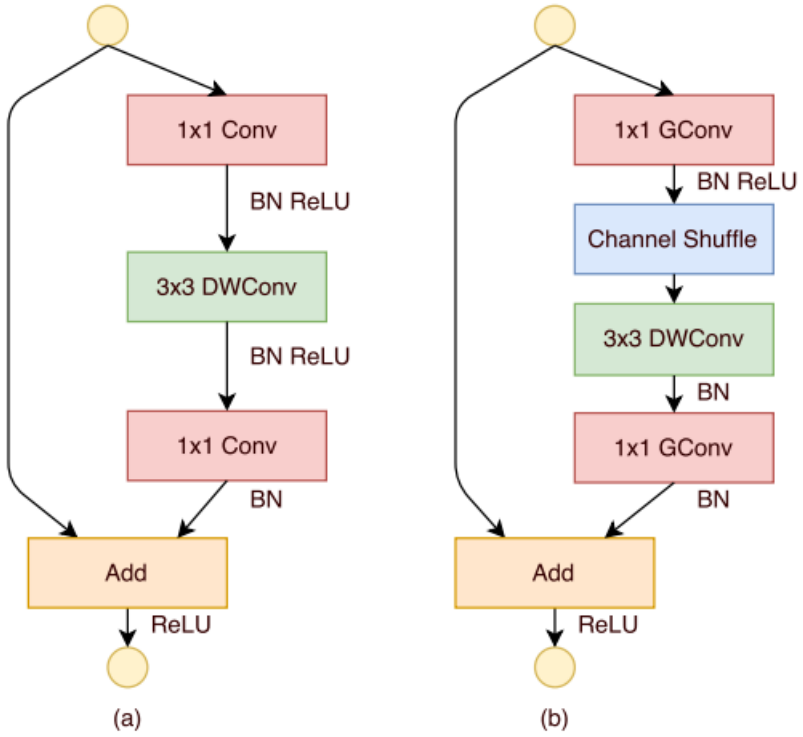
5.2.11. Generative Adversarial Network

[Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair†, Aaron Courville, Yoshua Bengio‡, 2014] - w

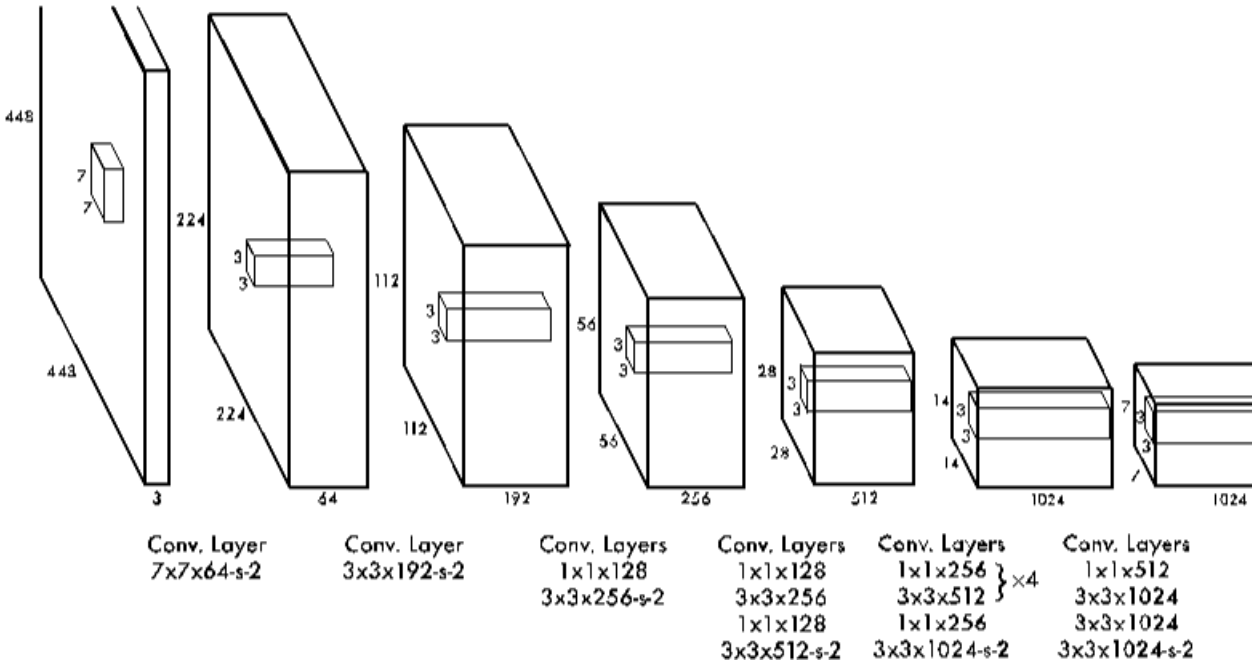


Rysunek 5.4: Architektura sieci ResNeXt

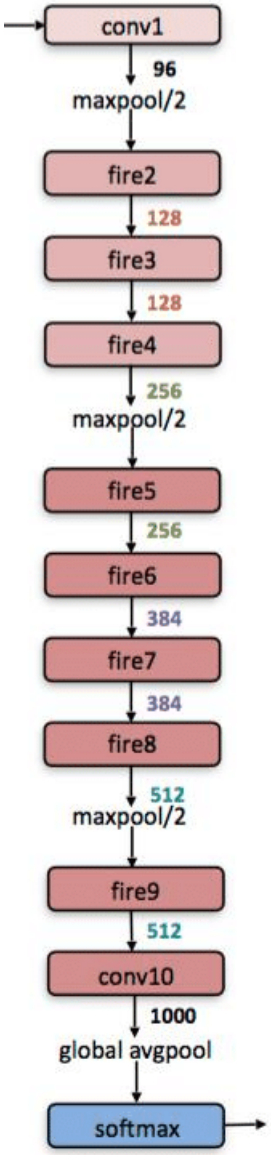
skrócie GAN. Są architekturą sieci neuronowych skomponowanych z dwóch sieci przeciwstawionych wobec siebie. Zaprojektowane na uniwersytecie w Montrealu (gdzie powstało większość przełomów dotyczących neuronów) przez największe obecnie autorytety w dziedzinie. GAN obudził duze nadzieje na szybkie i "twórcze" działania algorytmów, jego najczęstszym zastosowaniem jest tworzenie komputerowych "dzieł sztuki". Jak to działa? Jedna sieć generuje kandydatów, a druga ich ocenia wytworzone obiekty. W ten sposób obie sieci uczą się od siebie wzajemnie. - [1]



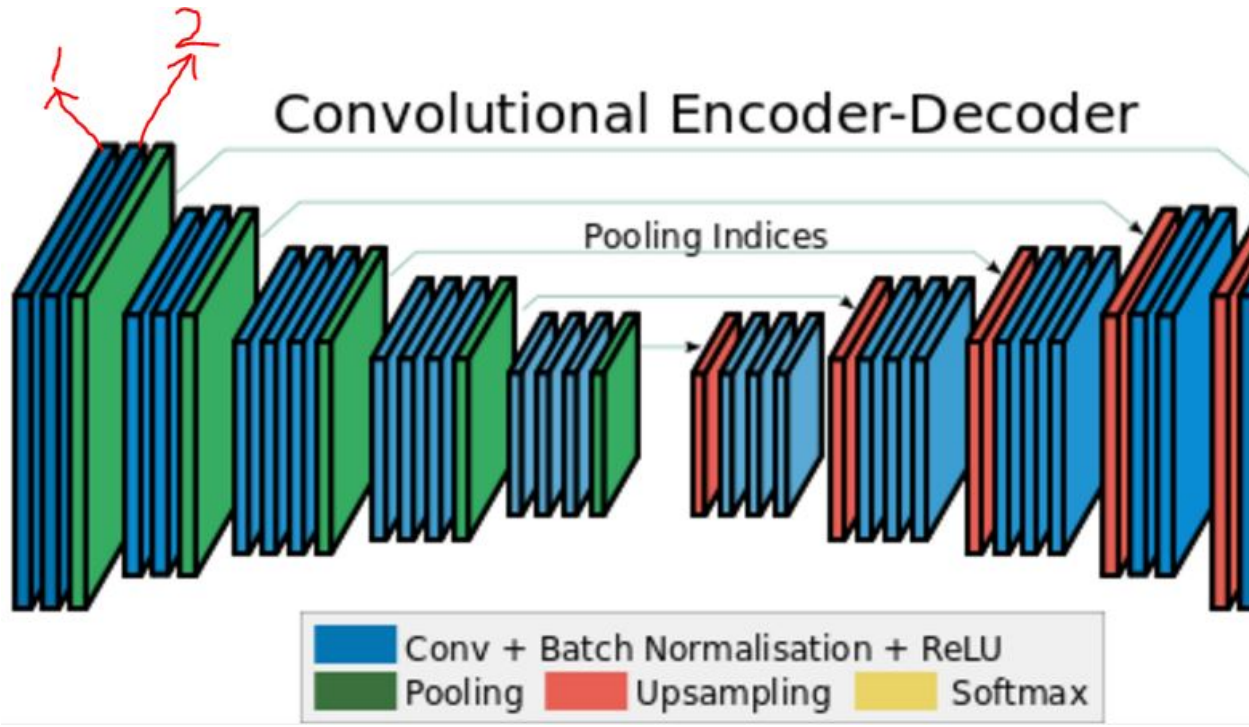
Rysunek 5.5: Architektura sieci RCNN



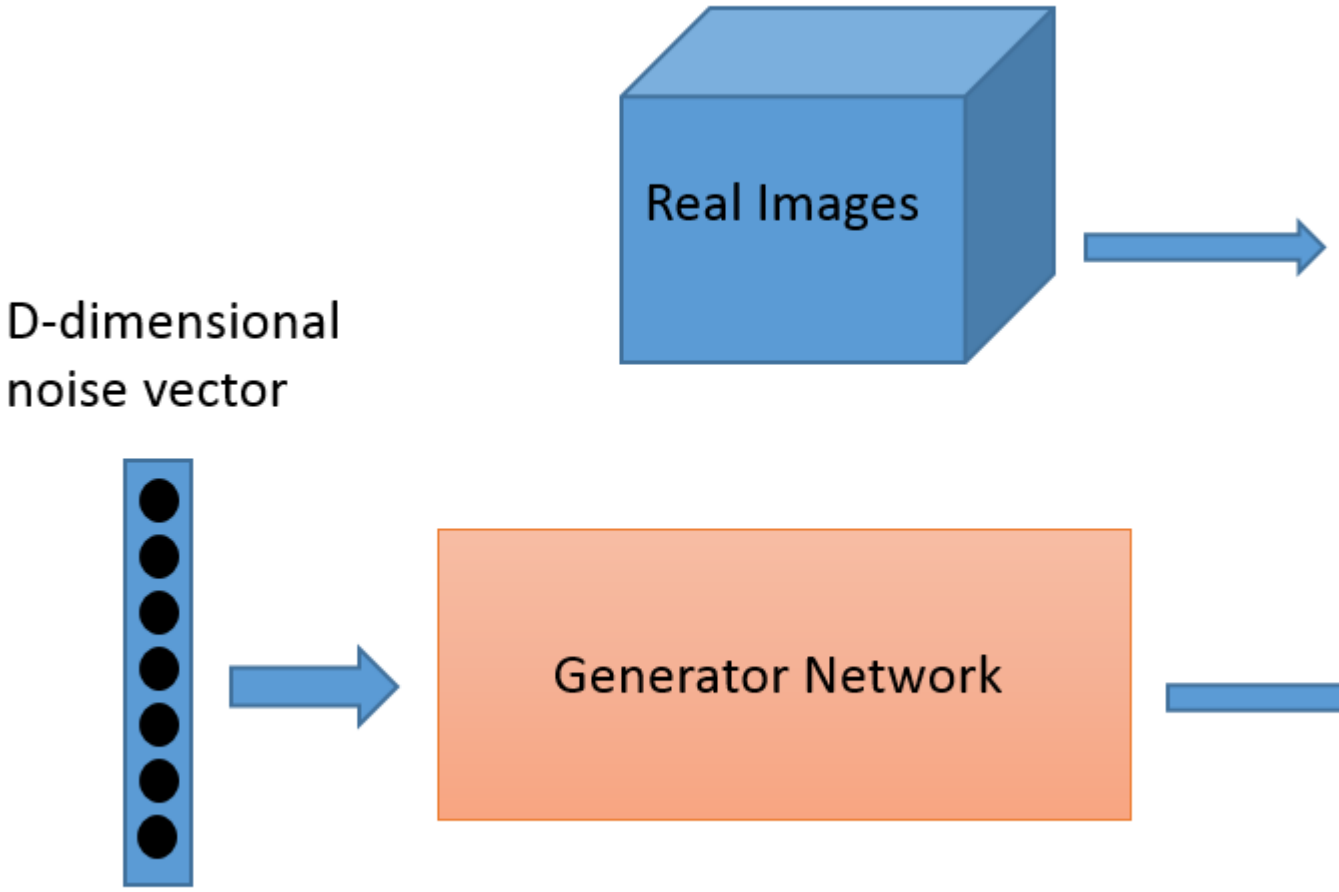
Rysunek 5.6: Architektura sieci YOLO



Rysunek 5.7: Architektura sieci SqueezeNet



Rysunek 5.8: Architektura sieci SegNet



Rysunek 5.9: Architektura sieci GAN

Rozdział 6.

Testy wydajności

Rozdział 7.

Podsumowanie

Spis rysunków

Bibliografia

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[3] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[4] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[5] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[6] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[7] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431, 2016.