Polska Akademia Nauk

Instytut Biocybernetyki i Inżynierii Biomedycznej

Praca doktorska

Proces gojenia ścięgna Achillesa oceniany przez fuzję danych z wykorzystaniem głębokich sieci neuronowych

Autor: mgr inż. Norbert Kapiński

Kierujący pracą: dr hab. inż. Antoni Grzanka

Promotor pomocniczy: dr Jakub Zieliński

Warszawa, wrzesień 2018

Streszczenie

The abstract will go here....
W tym miejscu można umieścić abstrakt pracy. W przeciwnym wypadku należy usunąć/zakomentować ninijeszy fragment kodu.

Spis treści

Spis rysunków

Spis tabel

Rozdział 1

Wstęp

Logistic regression — 1958

Hidden Markov Model — 1960

Stochastic gradient descent — 1960

Support Vector Machine — 1963

k-nearest neighbors — 1967

Artificial Neural Networks — 1975

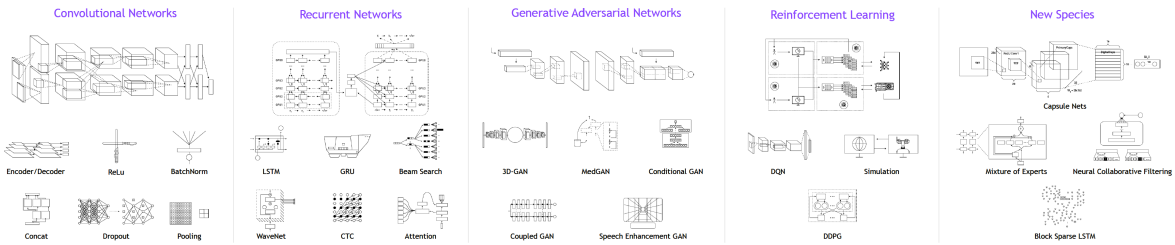
Expectation Maximization — 1977

Decision tree — 1986

Q-learning — 1989

Random forest — 1995

4% badań dotyczy oceny postępu w leczeniu [NVIDIA]



Rysunek 1.1: Podział przedstawiający różne rodzaje współczesnych głębokich sieci neuronowych.

Rozdział 2

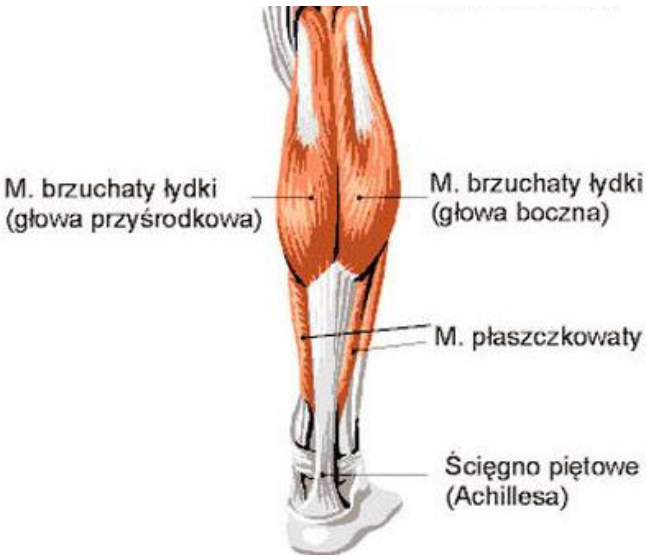
Cel i przebieg pracy

Rozdział 3

Monitorowanie procesu gojenia ścięgna Achillesa

3.1 Ścięgno Achillesa

Ścięgno Achillesa, nazywane również ścięgnem piętowym, jest największym i najsilniejszym ścięgnem występującym w ciele ludzkim. Stanowi wspólne zakończenie mięśnia trójgłowego łydki, w którego skład wchodzą dwie głowy mięśnia brzuchatego i mięsień płaszczkowaty. Całość struktury zlokalizowana jest w tylnym, powierzchownym przedziale łydki, co zostało przedstawione na Rysunku 3.1. Z obu głów (brzuścców)



Rysunek 3.1: Lokalizacja mięśnia trójgłowego łydki wraz ze ścięgnem Achillesa.

mięśnia brzuchatego łydki wyrasta jedno szerokie, płaskie ścięgno, które jest początkiem części brzuchatej ścięgna Achillesa. Następnie ścięgno to łączy się z włóknami pochodzącymi od mięśnia płaszczkowatego, które układają się stycznie do wcześniej powstałej struktury. Wówczas kształt ulega stopniowemu zwężeniu i zaokrągleniu, aż do punktu o minimalnej szerokości (około 4 cm nad przyczepem dolnym [1]). W rejonie samego przyczepu dolnego znajdującego się na tylnej powierzchnia kości piętowej, ścięgno ponownie jest płaskie i szerokie.

W kolejnych podsekcjach szczegółowo omówiona została anatomia ścięgna Achillesa, jego biomechanika, potencjalne urazy wraz z czynnikami im sprzyjającymi oraz proces gojenia i możliwości jego wspomagania. Wszystkie te aspekty są istotne z uwagi na możliwości monitorowania procesów fizjologicznych występujących w ścięgnie.

3.1.1 Anatomia

Średnia długość ścięgna Achillesa to 15 cm (11 - 26 cm). Średnia szerokość w rejonie początku wynosi 6.8 cm (4,5 - 8, 6 cm). Następnie, stopniowo ścięgno ulega zwężeniu do punktu o minimalnej szerokości 1.8 cm (1,2 - 2,6 cm). W rejonie samego przyczepu struktura ponownie się rozszerza i jej szerokość wynosi średnio 3.4 cm (2,0 - 4,8 cm) [2-3]. Zewnętrzną część ścięgna Achillesa stanowi ościęgno utworzone z tkanki łącznej włóknistej. Achil -Histologia -Unaczynienie (krew, nerwy)

3.1.2 Biomechanika

Zadaniem ścięgien jest transfer siły mięśniowej do układu szkieletowego.

3.1.3 Urazy i czynniki im sprzyjające

3.1.4 Leczenie, fazy gojenia i rehabilitacja

3.2 Zastosowanie rezonansu magnetycznego

3.3 Zastosowanie ultrasonografii

3.4 Zastosowanie badań biomechanicznych

3.5 Inne metody

Rozdział 4

Konwolucyjne sieci neuronowe

Konwolucyjne sieci neuronowe (ang. *Convolutional Neural Networks*, CNN) są biologicznie inspirowanymi sztucznymi sieciami neuronowymi. Tak jak opisano we wstępie, należą one do zbioru *głębokich sieci neuronowych* (ang. *Deep Neural Networks*, DNN), które są zkolei podzbiorem *systemów uczących się* (ang. *Machine Learning systems*) tj. algorytmów nie wymagających eksplisite programowania do ustalenia swoich parametrów.

Pierwsze matematyczne formalizmy dotyczące CNN zostały zapropozonwane już w latach 40-tych XX wieku, natomiast podłoże do inspiracji biologicznych dały badania nad korą wzrokową kotów Hubel'a i Wiesel'a z roku 1968. Dzięki tym pracom oraz badaniom kolejnych neurofizjologów (m.in. zob. []) ustalono, że kora wzrokowa zawiera złożone układy komórek, które odpowiadają za przetwarzanie informacji z wybranów regionów pola widzenia, sumarycznie pokrywając je w całości. Komórki kory wzrokowej działają zatem jak lokalne filtry przestrzeni wejściowej zaprojektowane, tak aby wydobyć istotne cechy z naturalnych obrazów. Dla przykładu reagują na orientację linii, kształty i kolory.

W sieciach konwolucyjnych procesy zachodzące w korze wzrokowej są modelowane, prowadząc do utworzenia bazy filtrów, za pomocą których istotne informacje obrazowe mogą być wyekstrahowane i następnie przetworzone. Fakt ten wykorzystywany jest w problemach dotyczących przetwarzania cyfrowych obrazów medycznych pochodzących np. z MR, TK czy USG.

Cyfrowy obraz zapisywany jest w postaci macierzy punktów o współrzędnych (x,y), gdzie x to kolumna macierzy, a y wiersz. W przypadku obrazów trójwymiarowych

dochodzi jeszcze składowa z, a zamiast macierzy użyty jest tensor. Dodatkowo dla każdego punktu kodowana jest informacja o wartości funkcji obrazu $I(x,y)$. Na tej podstawie dzielimy obrazy na:

- binarne – kodowane są jedynie dwie możliwe wartości I (0 lub 1);
- monochromatyczne – kodowana jest informacja o natężeniu jednej barwy (najczęściej są to odcienie szarości lub brązu tzw. sepia);
- kolorowe – kodowane są wartości natężenia składowych.

Do zakodowania informacji o wartości funkcji I w obrazie binarnym potrzebny jest jeden bit na punkt. Odcienie obrazu monochromatycznego i pojedyncze składowe punktu obrazu kolorowego kodowane są na 8–16 bitach, w zależności od zakresu wartości występujących w danych.

Obrazy medyczne przetwarzane w tej pracy należą do grupy obrazów monochromatycznych. Należy jednak wiedzieć, ze w praktyce sygnałem wejściowym dla sieci konwolucyjnych są często obrazy kolowe, reprezentowane odpowiednim *modelem przestrzennym barw* stanowiącym zestawienie składowych np. RGB, YUV, HSV, HLS (zob. [barwy])

Podstawową operacją wykorzystywaną w sieciach konwolucyjnych do wydobycia istotnych cech obrazowych jest *splot* maski K filtru z kolejnymi fragmentami funkcji I. K jest najczęściej macierzą kwadratową o wymiarze N i (z uwagi na rosnącą wraz z N złożonością obliczeniową) ogranicza się do wymiarów $N = 1, 3, 5, 7, 9, 11$. Splot zdefiniowany jest następująco:

$$I'(x,y) = \sum_n \sum_k I(x-n, y-k)\, K(n,k)\,, \tag{4.1}$$

gdzie I' jest nową funkcją obrazową powstałą po filtracji, a n i k to kolejne współrzędne maski filtru w odniesienia do jego punktu centralnego. W realizacji splotu dla wartości brzegowych, dla współrzędnych poza sygnałem przyjmuje się z reguły wartości 0. Inne metody to: odbicie obrazu poza jego granicami; powtórzenie obrazu bez odbicia; powielenie brzegowych wartości; modyfikacji maski filtru na brzegu obrazu, tak by maska nie wychodziła poza obraz.

Operacje, takie jak splot maski z funkcją I polegające na modyfikacji danego piksela biorąc pod uwagę piksel i jego otoczenie nazywane są *przekształceniami kontekstowymi*.

Przy ich użyciu można bardzo efektywnie wydobywać charakterystyczne cechy funkcji I, takie jak:

- *krawędzie* – ciągi punktów o gwałtownych zmianach $I(x,y)$;
- *rogi* – przecięcia dwóch krawędzi;
- *grzbiety* lub *doliny* – lokalne maksimum lub odpowiednio minimum funkcji I;
- *skupiska* (ang. *blobs*) – obszary jednorodnych wartości funkcji I, różniących się znacząco od najbliższego otoczenia;
- *tekstury* – charakterystyczne przestrzenne ułożenie wartości funkcji I w powtarzające się wzory.

Z powyższych cech można sformułować wektor cech w, który wykorzystywany jest do wnioskowania końcowego na temat obiektów znajdujących się w obrazach. Do tego celu służą grupy metod wśród których wyszczególnić można:

- *segmentację* – podział obrazu na spójne fragmenty, najczęściej wiążące się z wyodrębnieniem *obiektu* z *tła* na podstawie ustalonego progu lub miary np.:

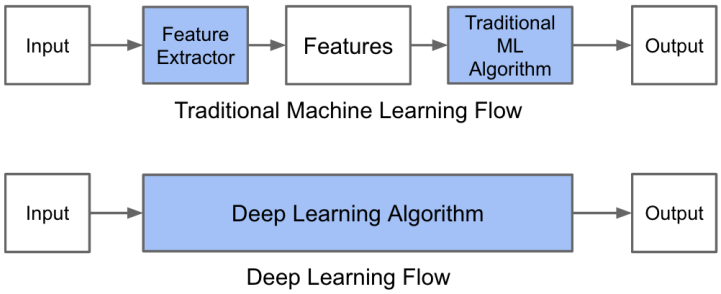
$$I(x,y) \geqslant T_p \Rightarrow (x,y) \in \text{"}obiekt\text{"}$$
$$I(x,y) < T_p \Rightarrow (x,y) \in \text{"}tlo\text{"},$$

  przy czym metody ustalania wartości progowej T_p lub wielu wartości T_p^1,...,T_p^n są obszarem szerokich badań (zob. [Prog]).

- *klasyfikację* – przyporządkowanie obiektu do odpowiedniej klasy (np. tkanka zdrowa lub patologiczna).
- *detekcję* – binarne rozróżnienie traktujące o tym czy obiekt znajduje się w obrazie czy nie.
- *śledzenie* – detekcja lub też klasyfikacja obiektów w kolejnych krokach czasowych.

Schemat stosowany jeszcze do niedawna (zob. Rys. 4.1) w większości badań wykorzystujących uczenie się maszyn uwzględniał wyliczenie wektora cech i wnioskowanie końcowe jako oddzielne kroki[1].

[1] Wyjątki typu SVM



Rysunek 4.1: Porównanie schematów przetwarzania danych z wykorzystaniem metod głębokiego uczenia się i innych algorytmów.

W algorytmach głębokiego uczenia, zarówno ekstrakcja cech jak i ostateczne wnioskowanie na ich podstawie realizowane jest w jednym kroku, co nazywane jest paradygmatem *end-to-end learning*. W kolejnej sekcji omówiono dokładniej jak wyglądała ewolucja tego podejścia do obecnej postaci.

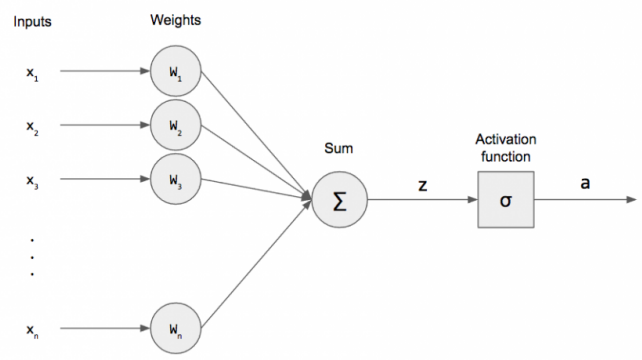
4.1 Zarys historyczny

Pierwszy formalny model neuronu został zaproponowany przez Warrena McCulloch i Waltera Pitts w roku 1943 [Pitts]. Była to bramka logiczna, której wyjście stawało się aktywne w momencie, gdy liczba aktywnych wejść przekroczyła pewien zdefiniowany próg. Taka zależność sygnału wyjściowego y od sygnałów wejściowych $x_1 ... x_n$ została potem nazwana *funkcją aktywacji neuronu*, którą zapisujemy jako:

$$y = f\left(x_1, x_2, ..., x_n\right) \tag{4.2}$$

W modelu neuronu McCulloch-Pitts można było modyfikować parametr progu, nie istniała natomiast możliwość uczenia się takiej architektury. Dlatego w 1957 zaproponowano sztuczną sieć neuronowej zawierającą wiele neuronów z ważonymi połączeniami między sobą [Perc]. Sieć nazwano perceptronem, co było implikacją zamiłowania jego twórcy Franka Rosenblatta do aplikacji związanych z percepcją, zwłaszcza mowy czy pisma. Schemat sieci pokazano na Rys. 4.2.

Zastosowanie wektora wag $w = (w_1,...,w_n)$ dało możliwość uczenia się poprzez adaptacyjną zmianę wartości jego elementów. W modelu Rosenblatta zastosowano ponadto



Rysunek 4.2: Topologia perceptronu.

progową funkcję aktywacji z progiem T_p:

$$y(z) = \begin{cases} 0, & n < T_a, \\ 1, & n \geqslant T_a, \end{cases} \tag{4.3}$$

gdzie z to suma ważona wyjść poszczególnych neuronów:

$$y = \sum_{i=1}^{n} w_i x_i \tag{4.4}$$

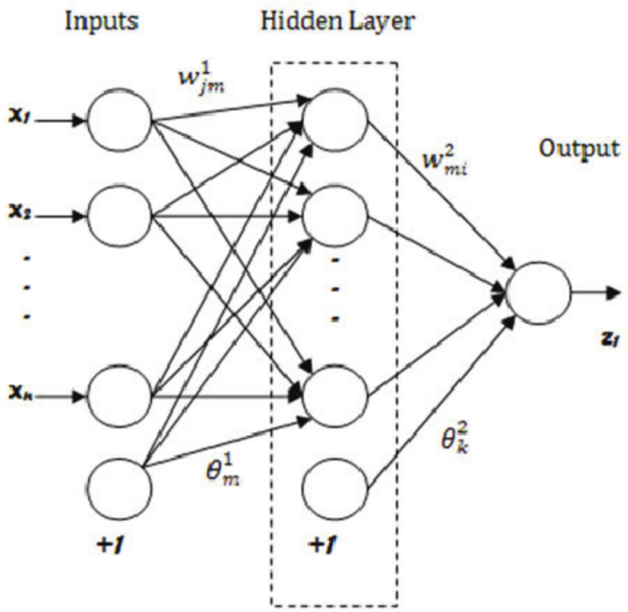
W dwuwymiarowej przestrzeni sygnałów wejściowych x_1, x_2 działanie perceptronu można opisać jako wyliczenie funkcji liniowej rozdzielającej obserwacje x_1 od x_2. W trzech wymiarach będzie to płaszczyzna, a w n-wymiarach, hiperpłaszczyzna.

Powyżej opisane działanie jest daleko idącym uproszczeniem funkcjonowania biologicznego neuronu (opisz coś więcej o neuronie). Przez blisko dekadę nowa architektura była obiektem badań i doczekała się sukcesów na polach takich jak rozpoznawanie tekstu, czy mowy [].

Pomimo tego, liniowy charakter perceptronu wprowadzał duże ograniczenia w możliwościach jego zastosowania. W 1969 roku Marvin Minsky i Seymour Papert w książce *Perceptrons* opublikowali listę problemów, których nie można było rozwiązać z użyciem perceptronu. Do najszerzej dyskutowanych należał problem związany z brakiem możliwości modelowania funkcji XOR.

Po latach intensywnych prac, część z opisanych przez Minsky-Papert problemów udało się rozwiązać dopiero w 1986, za sprawą pracy Davida Rumelharta, Geoffa Hintona i Ronalda Williams traktującej o perceptronach wielowarstwowych. Schemat takiej

sieci zaprezentowano na Rys. 4.3



Rysunek 4.3: Topologia perceptronu wielowarstwowego.

Spośród najważniejszych innowacji wprowadzonych w perceptronie wielowarstwowym wyszczególnić można zastosowanie w praktyce nowego algorytmu uczenia się sieci, który został opisany dalej w kolejnej sekcji jak również nowej, sigmoidalnej funkcji aktywacji:

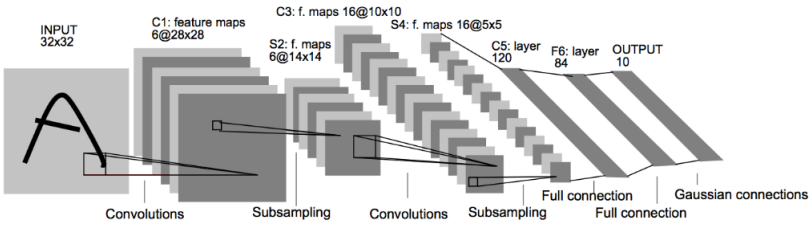
$$y(x) = \frac{1}{1 + e^{-x}} \tag{4.5}$$

Z wykorzystaniem sieci wielowarstwowych możliwe stało się modelowanie funkcji XOR jak i innych problemów nieliniowych o praktycznym wymiarze.

Kolejny przełom nastąpił w 1989 roku kiedy to Yann LeCunn, były uczeń Geaoffa Hintona, zaprezentował swoje wyniki dotyczące klasyfikacji odręcznego pisma z użyciem sieci wielowarstwowych [LeCun et al., 1989a]. Finalnie, badania te doprowadziły do przedstawienia w 1998 roku pierwszej sieci konwolucyjnej nazwanej LeNet [LeNet]. Architekturę tej sieci przedstawiono na Rys. 4.4.

Sieć składała się z 7 warstw i zawierała około 60,000 parametrów. Oryginalnie, sygnał wejściowy sieci stanowił obrazek o wymiarach 32×32.

W architekturze LeNet uwidoczniły się dwie podstawowe składowe współczesnych sieci konwolucyjnych tj. *ekstraktor cech* – zawierający filtry, których wartości parametrów masek są optymalizowane pod kątem wyboru cech charakterystycznych dla zada-



Rysunek 4.4: Topologia sieci LeNet.

nego zbioru danych oraz *klasyfikator* – zawierający neurony z wagami, których wartości są optymalizowane w celu końcowego rozróżnienia zbioru danych na podstawie wyekstrachowanego wczęsniej wektora cech. Oba z tych komponentów były uwzględnione w jednym procesie uczenia się, co było zgodne z paradygmatem end-to-end learning.

W porównaniu do poprzednich architektur, sieć konwolucyjną mozna opisać jako topologię perceptronu wielowarstwowego poprzedzoną ciągiem bloków służących do automatycznej ekstrakcji cech obrazowych. Najczęściej bloki te składają się z *warstw konwolucyjnych*, w których pogrupowane są filtry używane do ekstrakcji cech obrazowych różnego poziomu (np. krawędzie, kształty, obiekty), z opisanych wcześniej warstw aktywacji oraz warstwy typu *pooling* realizujące nieliniową redukcję wymiarowości (np. max-pool wybiera największą wartość z danego obszaru obrazu[2]). Warstwy wykorzystane do klasyfikatora są *warstwami w pełni połączonymi* (ang. *fully connected*, w skr. *FC*), a więc neuron z danej warswy jest połączony z każdym neuronem z warstwy następnej. Dal przykładu LeNEt miała 2 warstwy typu FC.

Pojedynczy splot filtru z obrazem wejściowym nazywany jest *cechą* [OTR-19]. Cechy są bardzo łatwe do wizualizacji dlatego sieci konwolucyjne mogą być łatwiejsze do interpretacji niż inne algorytmy sztucznej inteligencji. W ostatnich latach powstało wiele algorytmów do wizualizacji wyników działania sieci konwolucyjnych np. Saliency Maps [OTR - 20], GradCam [OTR - 21] albo metody bazujące na skierowanych acyklicznych grafach [OTR - 22]. Grupy cech z wybranego obszaru obrazu wejściowego, połączone w reprezentacje na róznych warstwach konwolucyjnych tworzą *mapy cech*. Dla przykładu w LeNet każda z warstw zawierała po odpowiednio: 6, 6, 16, 16 i 120 map cech o różnych wymiarach.

[2]operacja max-pool jest najczęściej stosowaną operacją do nieliniowej redukcji wymiarowości, o innych można przeczytać w [poolingOps]

Złożoność budowy sieci konwolucyjnych oraz duża liczba parametrów spowodowała szereg wyzwań związanych z praktycznym ich wykorzystaniem m.in. poprawną klasyfikacją zbioru danych. Najważniejsze z wyzwań zostały opisane w kolejnej sekcji.

4.2 Szkolenie głębokich sieci neuronowych

Wiekszość algorytmów szkolenia głębokich sieci neuronowych obejmuje zadanie *optymalizacji*, rozumiane jako minimalizację, bądź maksymalizację *funkcji celu* $f(x)$ przez zmianę x. W literaturze można też znaleźć inne nazwy funkcji celu takie jak *kryterium*, *funkcja kosztów*, *funkcja strat*, *funkcja błędów* (za [DL-IANGoodfellow]).

Podczas zadania optymalizacji bardzo często wykorzystuje się *pochodną funkcji* oznaczaną jako $f'(x)$ lub $\frac{\delta y}{\delta x}$, gdyż niesie ona informacje o nachyleniu funkcji w punkcie x. W praktyce funkcje celu są wielowymiarowe dlatego wykorzystywane są *pochodne cząstkowe* informujące o nachyleniu w poszczególnych wymiarach. Wektor zawierający pochodne cząstkowe funkcji nazywany jest *gradientem* i oznaczany jako $\bigtriangledown f(x)$. Dodatkowo czasami istnieje konieczność znalezienia wszystkich pochodnych cząstkowych funkcji, której wejście i wyjście stanowią wektory. Taka macież nosi nazwę *macierzy Jacobiego*, natomiast macierz *drugich pochodnych* cząstkowych (tj. pochodnych pochodnych) *macierzą Hessego*.

Szereg metod optymalizacyjnych niegradientowych zostało opracowanych (zob. []). Jednak to właśnie metody optymalizacji bazujące na wartości gradientu są najczęściej używane w głębokich sieciach neuronowych z uwagi na szybkie znajdowanie lokalnych minimów. Lokalne minimum nie jest zazwyczaj najlepszym możliwym rozwiązaniem, ale wiele badań wykazało następujące fakty (por. [WhyGradientMethodsoinDL]):

1. Dla sieci neuronowych o dużych rozmiarach większość lokalnych minimów charakteryzuje się podobnymi wynikami w kontekście osiągnięć zadanych problemów na zbiorze testowym.

2. Prawdopodobieństwo znalezienia lokalnego minimum, którego implikacją będą niskie rezultaty sieci maleje wraz ze wzrostem rozmiaru sieci.

3. Próba znalezienia globalnego minimum bardzo często prowadzi do problemu nadmiarowego dopasowania, który został omówiony w kolejnej sekcji.

Podsumowując, metody gradientowe są wydajne obliczeniowo i prowadzą do znalezienia wielu satysfakcjonujących rozwiązań, które mogą być wykorzystane do rozwiązania praktycznych problemów. Metody gradientowe w kolejnych krokach iteracji obierają kolejne wartości funkcji f przesuwając się w kierunku spadku gradientu:

$$x' = x - \epsilon \bigtriangledown f(x), \tag{4.6}$$

gdzie ϵ to *szybkość uczenia się*, parametr określający wielkość kroku.

Funkcja celu w przypadku praktycznych zadań optymalizacyjnych, wykorzystujących głębokie uczenie się jest zazwyczaj *funkcją złożoną*, a zatem efekt jej działania jest równoważny operacją wykonywanym przez szereg funkcji. Do obliczeń pochodnych funkcji złożonych z funkcji, których pochodne są znane stosuje się tzw. *regułę łańcuchową*. Przypuśćmy, że y = g(x) i z = f(g(x)) = f(y), x i y to wektory. Wówczas regułę łańcuchową można zapisać jako:

$$\frac{\delta z}{\delta x_i} = \sum_j \delta z / \delta y_j \delta y_j / \delta x_i, \tag{4.7}$$

co w zapisie wektorowym równoważne jest z równaniem:

$$\bigtriangledown_x z = (\delta y / \delta x)^T \bigtriangledown_y z, \tag{4.8}$$

gdzie $\frac{\delta y}{\delta x}$ to macierz Jacobiego. Regułę tak zapisaną prosto uogólnie się do zmiennych tensorowych (zob. IanGoodf-205).

Można zatem gradient zmiennej x otrzymać przez pomnożenie macierzy Jacobiego przez gradient $\bigtriangledown_y z$. W tym celu w metodach głębokiego uczenia się wykorzystuje się algorytm *propagacji wstecznej*, który oblicza regułę łańcuchową w wydajnej kolejności stosując działania w grafie, którego przykładową strukturę stanowi perceptron wielowarstwowy. Przykład zastosowania algorytmu wstecznej propagacji do perceptronu wielowarstwowego znajduje się [], a oryginalna praca na temat algorytmu można znaleźć w [].

Proces szkolenia się sieci ma na celu najlepsze możliwe przybliżenie docelowej klasyfikacji bazując na danych przykładach, czyli *zbiorze uczącym* U. Algorytmy optymalizacyjne używane do szkolenia głębokich sieci neuronowych zazwyczaj działają pośrednio, optymalizując pewną miarę wydajności P, która jest zdefiniowana na *zbiorze testowym*, zawierającym przykłady inne niż w U. Często bierze się również pod uwagę jeszcze do-

datkowy podzbiór, rozłączny z U i T - *zbiór walidacyjny*, który ma pomóc w wyborze najlepszej architektury sieci oraz najlepszych algorytmów i wartości parametrów odpowiedzialnych za jej szkolenie.

W procesie szkolenia zmniejszana jest funkcja kosztów f w oparciu na o U, a celem jest poprawa P. Algorytmy optymalizacyjne wykorzystujące cały zbiór U do liczenia wartości gradientu nazywane są *pakietowymi* lub *deterministycznymi*, gdyż przetważają jednocześnie wszystkie przykłady szkoleniowe. Te, które używają jednego przykładu na raz są nazywane *stochastycznymi*. W praktyce przy szkoleniu głębokich sieci stosowane są algorytmy *minipakietowe*, wykorzystujące więcej niż jeden przykład, ale mniej niż cały zbiór. Z reguły są to liczby z przedziału 8-256. Takie podejście zapewnia kompromis między szybkością obliczeń i dokładnością esymacji wartości gradientu.

Przykładem algorytmu minipakietowego jest *stochastyczny spadek gradientu* (ang. *stochastic gradient descent*, SGD). Bazuje on na założeniu, że nieobciążoną estymację gradientu można otrzymać wyciągając średnią gradinetu na minipakiecie m przykładów. Kolejne kroki algorytmu można zapisać nastepująco:

1. Wybierz wrtość parametru szybkości uczenia się ϵ_k.
2. Próbkuj minipakiet złożony z m przykładów ze zbioru szkoleniowego.
3. Oblicz estymację gradientu $g = frac1m \bigtriangledown \sum_{i=1}^{m} L(x_i, y_i, f)$, gdzie L to funkcja strat na jeden przykład o wejściej wartości próbki x_i i oczekiwanym wyjściu y_i.
4. Zastosuj aktualizację wartości funkcji celu równą $\epsilon_k g$.
5. Jeżeli kryterium stopu nie zostało spełnione, wówczas wróć do kroku 2.

Przy czym kryterium stopu jest najczęściej określone liczbą iteracji lub satysfakcjonującą wartością funkcji f. Kwestia optymalnego wyboru parametru ϵ_k również silnie zależy od problemu i najczęściej stosowane są metody empiryczne, przy czym zazwyczaj ϵ_k maleje wraz ze zbliżaniem się do satysfakcjonującego rozwiązania.

Kwestia strategii nadawania parametrom wartości początkowych w procesie szkolenia się sieci jest silnie dyskutowana w literaturze (zob. []) i jej kompleksowy obraz wykracza poza zakres tej pracy. Warto jednak wspomnieć o algorytmach z adaptacyjnyą szybkością uczenia się, gdyż jest to jeden z najtrudniejszych do ustawienia hiperparametrów, a jednocześnie bardzo istotny. Są to m.in.:

- Adaptive Gradient Algorithm (AdaGrad) [] – indywidualnie adaptuje szybkość uczenia się wszystkich parametrów modelu, skalując je odwrotnie proporcjonalnie do pierwiastka kwadratowego sumy wszystkich historycznych kwadratów gradientów.
- Root Mean Square Propagation (RMSProp) [] – jest to modyfikacja AdaGrad, która zmienia akumulację gradientu w wykładniczo ważoną ruchomą średnią.
- Adaptive moments algorithm (Adam) [] – W porównaniu z RMSProp, Adam poza momentem pierwszego rzędu (tj. średnią) wykorzystuje również moment drugiego rzędu (tj. wariancję). Dokładniej rzecz biorąc, w algorytmie liczona jest wykładnicza ruchoma średnia gradientu i kwadratu z gradientu, a parametry β_1 i β_2 kontrolują zakres liczenia średnich.

W praktyce ten ostatni jest w momencie pisania tej pracy najczęściej rekomendowanym algorytmem jako domyślna metoda szkolenia głębokich sieci neuronowych dlatego poniżej zamieszczono dokładny opis procedury:

1. Wybierz wartość początkową ϵ, β_1, β_2 oraz ustaw początkowe wartości zmiennych momentu 1 i 2 stopnia $s = 0$ i $r = 0$, warość kroku czasowego $t = 0$ i stałą σ używaną do stabilizacji numerycznej.
2. Próbkuj minipakiet złożony z m przykładów ze zbioru szkoleniowego.
3. Oblicz estymację gradientu $g = frac1m \bigtriangledown \sum_{i=1}^{m} L(x_i, y_i, f)$ i zwiększ t o 1
4. aktualizuj estymację pierwszego momentu. $s = \beta_1 s$ + (1-β_1)g
5. aktualizuj estymację drugiego momentu. $r = \beta_2 r$ + (1-β_2)$g \bigodot g$
6. skoryguj obciążenie momentu pierwszego rzędu $s = \frac{s}{1-\beta_1^t}$
7. skoryguj obciążenie momentu drugiego rzędu $r = \frac{r}{1-\beta_2^t}$
8. Zastosuj aktualizację wartości funkcji celu równą -$\epsilon \frac{s}{\sqrt{r}+\sigma}$.
9. Jeżeli kryterium stopu nie zostało spełnione, wówczas wróć do kroku 2.

Ocena procesu szkolenia się sieci polega na obliczeniu odpowiednich miar i współczynników odzwierciedlających przybliżenie zbioru T lub/i W przez znalezione rozwiązanie. W powszechnym problemie klasyfikacji, ocenianym również w tej pracy korzysta się z następujących parametrów:

- *Fałszywie pozytywna klasyfikacja* (*FP* od ang. *False Positive*) – liczba obserwacji zaklasyfikowanych jako pozytywne, a należących do klasy obserwacji negatywnych.
- *Fałszywie negatywna klasyfikacja* (*FN* od ang. *False Negative*) – liczba obserwacji zaklsyfikowanych jako fałszywie negatywne, a należących do klasy obserwacji pozytywnych.
- *Prawdziwie pozytywna klasyfikacja* (*TP* od ang. *True Positive*) – liczba wyników poprawnie zaklasyfikowanych jako pozytywne.
- *Prawdziwie negatywna klasyfikacja* (*TN* od ang. *True Negative*) – liczba wyników poprawnie zaklasyfikowanych jako negatywne.
- *Dokładność klasyfikacji* (ang. *Accuracy*) – $ACC = \frac{TP+TN}{TP+TN+FP+FN}$.
- *Czułość klasyfikacji* (ang. *Sensitivity*) – $TPR = \frac{TP}{TP+FN}$.
- *Swoistość klasyfikacji* (ang. *Specificity*) – $TNR = \frac{TN}{TN+FP}$.
- *Precyzja klasyfikacji* (ang. *Precision*) – $PPV = \frac{TP}{TP+FP}$.

W problemach empirycznych dąży się do maksymalizowania bądź minimalizowania powyższych współczynników. Prawidłowe podejście polega również na wybraniu możliwie efektywnej architektury sieci. Z tymi zadanimi związane są problemy nadmiernego dopasowania i redukcji wymiarowości, które zostały opisane w kolejnych podsekcjach.

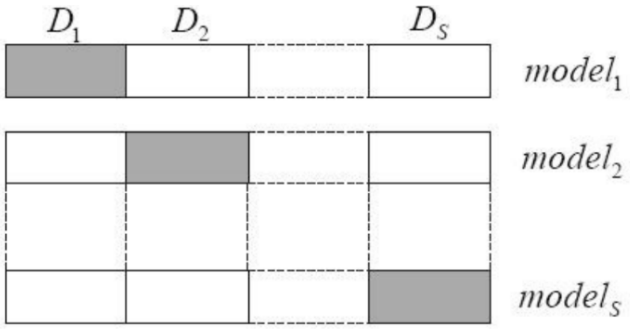
4.2.1 Problem nadmiernego dopasowania

Dążenie do najlepszego możliwego przybliżenia zbioru U wprowadza niepożądane zjawisko zwane *nadmiernym dopasowaniem*. Wtedy to dokładność klasyfikacji zbioru U jest wysoka lub nawet bezbłędna, natomiast znacznie niższa jest dokładność klasyfikacji zbioru testowego T i walidacyjnego W. W praktyce oznacza to, że model staje się mało użyteczny, gdyż nowe dane nie są poprawnie klasyfikowane.

Z uwagi na ten fakt ogólnym dążeniem w procesie uczenia się sieci jest osiągnięcie *maksymalnej generalizacji* klasyfikacji. Sieć o wysokim współczynniku generalizacji lepiej klasyfikuje ogół zadanych wekto- rów wejściowych niż sieć, która ma niski współczynnik generalizacji i jest nadmiernie dopasowana do zbioru U.

W celu uzyskania generalizacji należy wybrać taki model klasyfikatora, który wystarczy do zachowania poprawnej klasyfikacji w mocy. Empirycznie ustalane są zatem maksymalnie ogólne, dostateczne warunki poprawnej klasyfikacji. Dzięki generalizacji wzrasta prawdopodobieństwo, że przykład z poza zbioru U będzie poprawnie klasyfikowany przez algorytm sieci.

W celu osiągnięcia kompromisu pomiędzy maksymalnie dobrą klasyfikacją zbioru U i wysoką generalizacją sieci neuronowej można zastosować metodę oceny krzyżowej (ang. cross-validation). Metoda ta polega na podziale zbioru uczącego na s segmentów D, z których każdy w innej iteracji służy jako zbiór testujący i walidacyjny, a pozostałe segmenty pełnią rolę zbioru uczącego. Podział zobrazowany jest na Rysunku 4.5.



Rysunek 4.5: Reprezentacja graficzna oceny krzyżowej.

Stosując metodę oceny krzyżowej dla różnych modeli sieci można stwierdzić, który z nich spełnia najlepiej kompromis między dobrą klasyfikacją zbioru U i wysoką generalizacją.

Kombinacja predykcji wielu różnych modeli jest bardzo wydajną metodą do polepszenia generalizacji i zmniejszenia błędu klasyfikacji na zbiorach testowych oraz walidacyjnych (zob. [AlexNet 1, 3]). Jednak współczesne sieci neuronowe, których przykłady zostały opisane w dalszych sekcjach mogą zawierać miliony parametrów, których optymalizacja jest wymagająca obliczeniowo. Z uwagi na ten fakt trening s segmentów jest mało wydajny i stosuje się go w ograniczonym zakresie np. obierając $s = 5$ lub $s = 10$.

W 2012, w [AlexNet-10] zaproponowano technikę zwaną *dropout*, której główna idea bazuje na zerowaniu wyjścia neuronów sieci z prawdopodobieństwem 0,5 przy każdej iteracji treningu sieci. Neurony, które są w ten sposób tymczasowo dezaktywowane nie mają wpływu w danej iteracji na predykcję sieci i nie są uwzględniane przy wstecznej propagacji gradientu. Podejście to można porównać do treningu w każdej iteracji różnych modeli sieci. Dla przykładu w [AlexNet] wykazano, że metoda dropout wymaga

jedynie 2 razy więcej iteracji do przybliżenia zbioru U, przy tym uzyskuje znacznie lepszą generalizację.

Kluczowym składnikiem potrzebnym do treningu sieci i maksymalizacji generalizacji jest odpowiedni rozmiar zbioru danych. W praktyce jest to problem szeroko dyskutowany, gdyż zwłaszcza w danych medycznych istnieje szereg ograniczeń związanych z dostępem i akwizycją odpowiedniego materiału badawczego (np. ograniczenia: prawne, związane z prywatnością, czy z etyką). W przypadku, gdy zgromadzenie odpowiedniego zbioru danych jest niemożliwe pewnym rozwiązaniem problemu jest zastosowanie metod jego sztuczego powiększania (ang. *data augmentation*).

W przypadku obrazów stosuje się metody afinicznych przekształceń. Są to najczęściej rotacje, odbicia lub skalowanie. W określonych przypadkach używane są również nielioowe przekształcenia. Np. w [Ronneberger et al., arXiv:1505.04597v1, 2015] wykorzystano m.in. deformacje do powiększenia zbioru 30 obrazów mikroskopowych przedstawiających macierz komórkową i uzyskano znacząco lepsze wyniki niż istniejące w 2015 algorytmy typu state-of-the-art.

W przypadku danych medycznych należy szczególnie zwrócić uwagę, aby powiększony zbiór zawierał dane przypominające w rzeczywistości występujące przypadki np. nieduże obroty występujące u pacjentów skanowanych rezonansem magnetycznym lub niewielkie skalowania rozmiaru organów. Szeroką dyskusję prowadzi się również na temat wykorzystania sztucznie generowanych zbiorów danych o czym więcej można przeczytać w pracach [Sztuczna generacja danych medycznych].

4.2.2 Problem redukcji wymiarowości

Rozmiar wektora cech wejściowych we współczesnych problemach rozwiązywanych przez algorytmy sztucznej inteligencji często osiąga liczby liczone w tysiącach (zob. [przykłady dużych problemów FB, Google]). Również w powszechnie używanych głębokich sieciach neuronowych ekstraktor cech tworzy wektor o rozmiarach tego rzędu.

Duży rozmiar wektora cech wejściowych prowadzi do problemu nazwanego *przekleństwem wymiarowości* (ang. *curse of dimensionality*). Określenie te zostało poraz pierwszy sformułowane przez Richarda Bellmana w latach 50-tych XX wieku. Naukowca zajmującego się teorią sterowania (zob. [teoria sterowania]), który podczas swojej pracy obserwował algorytmy doskonale działające w 3 wymiarach, a prezentujące znacząco gorsze wyniki w hiperprzestrzeni.

Problemy z zastosowaniem algorytmów szucznej inteligencji w tym głębokich sieci neuronowych w dużej liczbie wymiarów mają dwie główne przyczyny: (1) w miarę wzrostu liczby wymiarów cech liczba obserwacji w zbiorze trenującym potrzebnych do wiarygodnego oszacowania funkcji wyjściowej rośnie wykładniczo; (2) nie wszystkie cechy są jednakowo znaczące w kontekście rozróżnienia danych.

Problem (2) jest szczególnie istotny w dość prostych algorytmach takich jak algorytm K najbliższych sąsiadów, gdzie do poprawnego działania należy policzyć dystans pomiędzy sąsiednimi obserwacjami. Uwzględniając dużą liczbę nieistotnych cech jako argumenty funkcji dystansu uzyskuje się wyniki uniemożliwiające poprawną klasyfikację zbioru.

W przypadku głębokich sieci neuronowych zastosowanie wag zmniejsza znaczenie problemu (2), gdyż wpływ poszczególnych cech może być regulowany poprzez wartość w. W przypadku problemu wynikającego z (1) można wyróżnić dwa podejścia stosowane do jego niwelacji:

- wybór podzbioru istotnych cech o liczności $n' << n$,
- przekształcenie oryginalnych n zmiennych na nowy zbiór n' cech, gdzie ponownie $n' << n$.

/*rozwinąć i wprowadzić definicję korelacji oraz entropii - zapytać Kubę jakby ten akapit widział*/

Wybór podzbioru istotnych cech polega na określeniu minimalnego podzbioru, dla którego rozkład prawdopodobieństwa różnych klas obiektów jest jak najbliższy oryginalnemu rozkładowi uzyskanemu z wykorzystaniem wszystkich cech. W tym celu stosuje się np.: miary siły związku [], testy wykorzystujące entropię względną [], badania korelacji wzajemnych [], teorię zbiorów przybliżonych [] itp.

W kontekście przekształcenia zbioru cech najczęściej wykorzystywaną metodą jest algorytm *analizy składowych głównych*) (ang. *Principal Component Analysis*, w skr. PCA) opracowany przez Karla Pearsona w 1901 r (zob. Pearson1901). Istotą PCA jest ortogonalne przekształcenie początkowych skorelowanych cech w nowy zbiór nieskorelowanych zmiennych. Każdy z nowych wektorów jest kombinacją liniową pewnych składowych głównych (odnoszących się do oryginalnych cech). Nowe nieskorelowane zmienne, tzw. *składowe główne*, powstają z przekształcenia oryginalnych zmiennych skorelowanych, w taki sposób aby w maksymalnym stopniu wyjaśniać całkowitą wariancję w próbie cech oryginalnych. Wariancje składowych głównych są wartościami

własnymi macierzy kowariancji oryginalnych zmiennych. Dla przykładu pierwsza składowa główna redukuje największą część zróżnicowania, druga kolejną której nie redukowała poprzednia itp.

Procedure PCA można zapisać następująco:

1. oblicz macierz kowariancji: Sx = $X^T X$, gdzie X to macierz danych zawierająca obserwacje w wierszach. macierz Sx jest symetryczna i pozwala ocenić: (1) wariancje zmiennych (elementy na głównej przekątnej); (2) zależności pomiędzy zmiennymi (elementy poza główną przekątną)
2. dokonaj rozkładu: Sx = KLK^T
3. utwórz nowe zmienne wykonując operację: Y = XK

/*przykład numeryczny https://www.itl.nist.gov/div898/handbook/pmc/section5/pmc552.htm */

Oprócz PCA można również wyróznić nastepujące algorytmy:

1. t-SNE
2. ISOMAP: Isomap Homepage
3. ICA: What is Independent Component Analysis ?
4. LSA: Popular in NLP domain :: LSA @ CU Boulder
5. Sammon mapping: Implementation in python tompollard/sammon
6. Self Organizing Maps: Matlab implementation SOM Toolbox

   https://www.quora.com/What-are-different-unsupervised-feature-selection-methods-other-than-principle-component-analysis-PCA

4.3 Przykłady współczesnych topologii

W ostatnich latach wprowadzono wiele rozwiązań zarówno w warstwie sprzętowej jak i oprogramowania, które przyczyniły się do wzrostu liczby zastosowań głębokich sieci neuronowych wtym sieci konwolucyjnych.

W głównej mierze prace te skupiły się na rozwiązaniach dedykowanych do optymalizacji fazy szkolenia omówionej w poprzedniej sekcji jak również *fazy wnioskowania*, gdzie wytrenowana sieć użyta jest d przetwarzania kolejnych obserwacji.

W fazie szkolenia największy problem obliczeniowy wynika z konieczności ciągłej aktualizacji parametrów. Natomiast w fazie wnioskowania wynika z propagacji sygnału w sieci. Ponieważ liczby parametrów we współpcześnie wykorzystywanych topologiach wahają się od milionów jak w [AlexNet] do nawet bilionów [GoogleBrain 23-TTCF],problemy te stanowią poważne wąskie gardło w pracach badawczych i rozwojowych.

Najczęściej wykorzystywanymi operacjami w szkoleniu sieci jest mnożenie i dodawanie tensorów, dlatego wiodące firmy sprzętowe takie jak Intel, czy NVIDIA, zdecydowały się przedstawić dedykowane tym celom akceleratory sprzętowe np. NVIDIA Tensor Processing Unit (w skr. TPU) [] czy Intel Lakecrest []. W chwili pisania tej pracy umożliwiają one nawet do 120 tera operacji tensorowych na sekundę (w skr. TTOPs).

Faza wnioskowania jest znacznie mniej wymagająca obliczeniowo lecz w praktyce często wymaga wykonywania w czasie rzeczywistym lub zbliżonym do niego. Rozwiązania takie jak [24-TTCF] minimalizują czasy przetwarzania wykorzystując np. optymalizację reprezentacji liczb zmiennoprzecinkowych lub przetwarzanych macierzy.

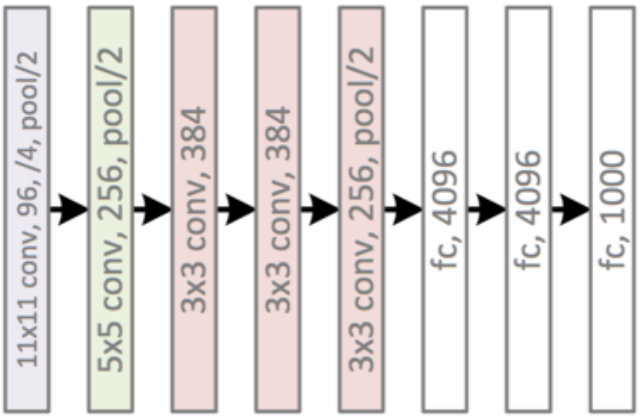
Dodatkowo, mnogość rozwiązań takich jak Caffe, Caffe2, TensorFlow, Theano, PyTorch czy MXNet, dedykowanych do wspierania obliczeń z udziałem głebokich sieci neuronowych, przyspiesza znacząco rozwój dziedziny i prace nad współczesnymi topologiami (zob. [porównania]). Wybrane przykłady takich architektur zostały omówione w kolejnych podsekcjach.

4.3.1 AlexNet

Sieć AlexNet, której nazwa pochodzi od imienia głównego twórcy tej architektury Alexa Krizhevsky, zawiera blisko 60 milionów parametrów i 650 tysięcy neuronów. Architekturę zaprezentowano na Rys. 4.6

W skład topologii wchodzi pięć warstw konwolucyjnych i trzy typu fully-connected. Po pierwszej, drugiej i piątej warstwie konwolucyjnej występują operacje typu max-pool z jądrem o wymiarach 2×2 [3].

[3]autorzy pracy podają też przykłady użycia jąder o wymiarze 2×3, które nakładają się w przestrzeni funkcji obrazowej



Rysunek 4.6: Topologia architektury AlexNet.

Pierwsza warstwa konwolucyjna przyjmuje na wejściu dane o wymiarze 227×227×3, na których wykonywana jest operacja splotu z 96 filtrami z jądrem splotu o wymiarach 11×11×3 i krokiem 4. W rezultacie (uwzględniając również operację max-pool) objętość wynikowa przekazywana do kolejnej warstwy ma wymiar 27×27×96. W drugiej warstwie konwolucyjnej wykonywana jest operacja splotu z 256 filtrami z jądrem o wymiarach 5×5×96. Wymiar objętości wynikowej zostaje ponownie zredukowany poprzez operacje max-pool do 13×13×256. Kolejne 3 warstwy konwolucyjne są połączone bezpośrednio ze sobą. Trzecia warstwa zawieraja 384 filtry o wymiarze 3×3×256, w skład czwartej wchodzą 384 filtry o wymiarze 3×3×384, a w piątej znajdują się 256 filtry ponownie o wymiarze 3×3×384. Końcowe dwie warstwy typu FC zawierają po 4096 neuronów, a ostatnia zawiera tyle neuronów ile klas występuje w ostatecznym podziale - w oryginalnej pracy było to 1000 [AlexNet].

W celu lepszego zrozumienia przetwarzania sygnału wejściowego przez sieć poniżej przedstawiono przykład algorytmu wykorzystywanego dla pierwszej warstwy konwolucyjnej opisywanej topologii:

1. Z danych wejściowych o wymiarze [227×227×3] wybierany jest co czwarty blok (zarówno wzdłuż wysokości jak i szerokości) o wymiarach [11×11×3]. Punkty krawędziowe, które stanowią margines potrzebny do wyliczenia splotu są zazwyczaj pomijane. W rezultacie otrzymywanych jest 217 punktów w każdym rzędzie i w kolumnie, w których mieści się [55×55] tj. 3025 bloków.

2. Zarówno 11×11×3 = 363 wagi znajdujące się w 96 filtrach jak i wartości 363 punktów obrazowych znajdujących sie 3025 blokach są przedstawiane w postaci macierzy A o wymiarach [96×363] i B o wymiarach [363×3025].

3. liczony jest iloczyn skalarny w postaci $A^{\intercal}B = C$, gdzie nowa, wyjściowa macierz C ma wymiar [96×3025].

4. Resultat w postaci macierzy C ponownie przewymiarowywany jest na postać [55×55×96].

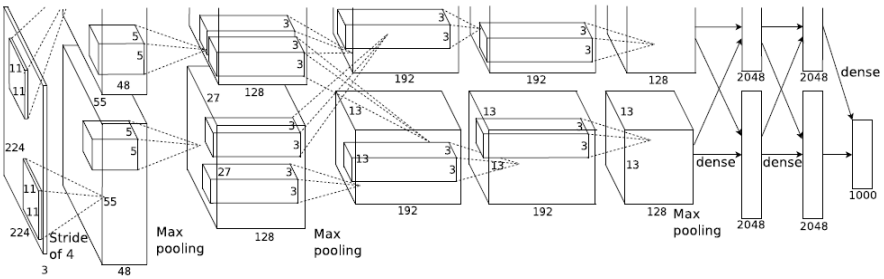
W architekturze jako funkcję aktywacji neuronów wykorzystano ReLU, co znacząco przyspieszyło trening sieci. Dla przykładu uzyskano 6-krotne przyspieszenie treningu dla danych CIFAR-10 [CIFAR] w stosunku do tej samej topologii wykorzystującej funkcję aktywacji tanh.

Ponieważ funkcja ReLU nie posiada górnego ograniczenia neurony teoretycznie mogą posiadać nieograniczone wartości funkcji aktywacji. W celu polepszenia kontrastu pomiędzy neuronami i wydobycia tych, które na tle innych się wyróżniają, zastosowano normalizację:

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=max(0,i-n/2)}^{min(N-1,i+n/2)} (a^i_{x,y})^2 \right)^{\beta} \tag{4.9}$$

gdzie n to liczba filtrów znajdujących się w tej samej przestrzennej lokalizacji, N to suma wszystkich filtrów w warstwie, $a^i_{x,y}$ to wartość funkcji aktywacji neuronu po splocie funkcji obrazowej na pozycji (x, y) z filtrem i, a b to wynik normalizacji. Po zastosowaniu tej techniki autorom pracy udało się zredukować błąd klasyfikacji top-5 o wartość 1,2 punkta procentowego.

W kontekście zwiększenia efektywności treningu zastosowano powiększenie rozmiaru danych poprzez rotacje i modyfikacje funkcji obrazowej z wykorzystaniem czynników głównych (zob [AlexNet]), co zmniejszyło błąd top-1 o 1%. Zastosowano również technikę dropout opisaną w 4.2.1. Ostatecznie wprowadzono także trening z wykorzystaniem wielu GPU (zob. Rys. 4.7).



Rysunek 4.7: Topologia architektury AlexNet z podziałem na dwa akceleratory GPU.

Topologia z podziałem na 2 karty zwiększyła dwukrotnie sumaryczną pamięć i pozwoliła na koalokację parametrów sieci. /*Normalizacja*/ Praca Alexa Krizhevsky, Ilya Sutskever i Geoffrey'a Hinton zapoczątkowała wzrost zainteresowania technikami głębokiego uczenia się, co doprowadziło do publikacji kolejnych podobnych architektur. Do najbardziej znanych należą ZFNet z 2013 roku [ZFNet], gdzie m.in. zastosowano zmniejszenie wymiaru jądra stosowanego w filtrach pierwszej warstwy konwolucyjnej do 7×7 oraz VGGNet z 2014 roku, gdzie zastosowano większą liczbę warstw konwolucyjnych z mniejszym wymiarem jądra splotu. Innowacyjnym pomysłym, który został zaprezentowany również w 2014 było pojawienie się nowych modułów w sieci GoogleNet, która dokładniej została opisana w kolejnej podsekcji.

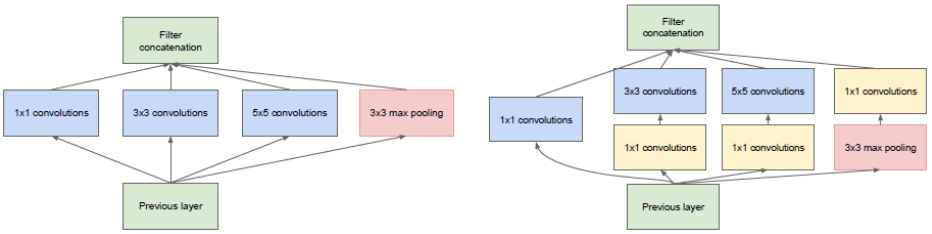
4.3.2 GoogLeNet

Architekturę o nazwie GoogLeNet zaprezentowano w 2014 r. w pracy [GoogleNet]. Nazwa architektury pochodzi od nazwy zwycięzkiego zespołu startującego w ILSVRC14, składającego się z pracowników firmy Google. Oryginalnie topologia składała się z 22 warstw i zawierała około 5 mln parametrów (12 razy mniej niż w przypadku sieci AlexNet).

Redukcję liczby parametrów przy jednoczesnym podwyższeniu dokładności klasyfikacji i lokalizacji obiektów (zob. [ILSCV]) udało się uzyskać poprzez poszukiwania konstrukcji optymalnych lokalnych topologii i ich połączeń. Mianowicie, wiadomo że duża część funkcji aktywacji neuronów przyjmuje wartość 0 lub jest redundatna z powodu wysokiej korelacji między sobą (zob. [Arora z GoogleNet]). Matematyka dotycząca przetwarzania *macierzy rzadkich*, tj. gdzie przeważająca liczba elementów przyjmuje wartość 0, jest dobrze znana np. [3-GoogleNet]. Jednak implementacje bibliotek do obliczeń związanych z algebrą liniową są zooptymalizowane pod kątem *macierzy gęstych*, gdzie przeważająca liczba elementów przyjmuje wartości różne od 0 (zob. [16, 9 googleNet]).

Ideą modułu incepcji zaproponowanego przez twórców GoogLeNet jest aproksymacja rzadkich macierzy z użyciem komponentów o gęstej strukturze. Takie komponenty nazwano *modułami incepcji* (ang. *inception modules*). Przykłądy modułów incepcji pokazano na Rys. 4.8

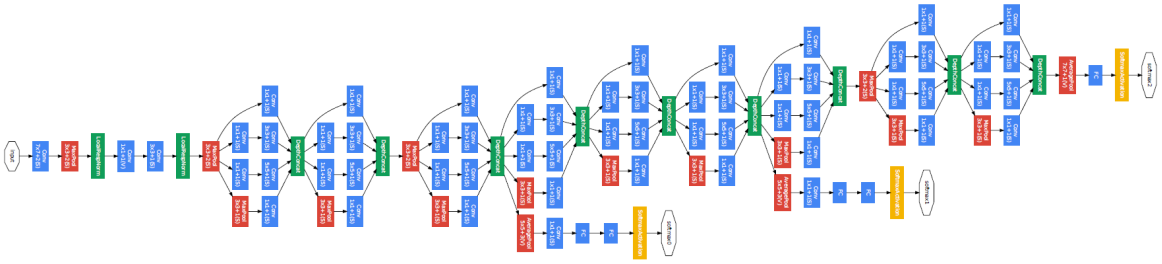
Rys. 4.8 (a) przedstawia najwną formę modułu incepcji, gdzie grupowane są operacje filtrów z jądrem o wymiarach 5×5, 3×3, 1×1 oraz operacja max-pool. Rys. 4.8 (b)



Rysunek 4.8: Topologia architektury AlexNet z podziałem na dwa akceleratory GPU.

prezentuje koncepcję zooptymalizowaną obliczeniowo gdzie filtry 1×1 służą do redukcji wymiarowości i używane są bezpośrednio przed splotami z bardziej wymagającymi obliczeniowo splotami z jądrami o wymiarach 5×5 i 3×3.

Złożenie różnego rodzaju modułów dało topologię zaprezentowaną na Rys.



Rysunek 4.9: Topologia architektury GoogleNet

Dokładne zestawienie parametrów znajduje się w Tabeli 4.1

Ważną cechą sieci GoogleNet jest brak warstw typu FC na zakończeniu, gdzie w przypadku sieci AlexNet znajdowało się około 90% parametrów. Końcowe wnioskowanie jest realizowane na podstawie wartości średniej z dwuwymiarowych map cech.

Dla lepszego zrozumienia idei redukcji wymiarowości realizowanej przez moduły incepcji, podobnie jak w przypadku sieci AlexNet, przeanalizowane zostanie działanie pierwszego modułu w topologii z Rys. 4.9. Moduł zawiera 128 filtrów z jądrami o wymiarach 3×3 i 32 filtry z jądrami o wymiarach 5×5. Dane na wejściu modułu mają 192 kanały (zob. Tabela 4.1). Dla przykładu, rząd wielkości obliczeń operacji splotów 32 filtrów 5×5 wynosi 25×32×192=153 600 i dalej wzrastałby z głębokością sieci. W celu zapobiegnięcia nadmiarowi obliczeń stosowana jest redukcja z użyciem 16 filtrów z jądrem o wymiarach 1×1. W efekcie rząd wielkości obliczeń spada do 16×192 + 25×32×16=15 876, co pozwala na dalsze budowanie wielowarstwowych struktur.

Topologia GoogLeNet jest wciąż rozwijana. Po pierwszej prezentacji pojawiły się

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Tabela 4.1: Parametry architektury GoogleNet

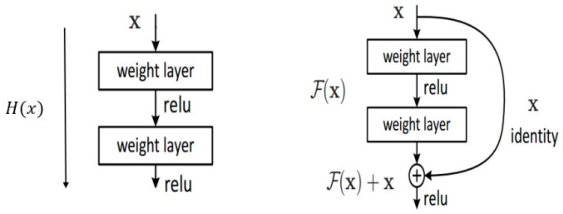
kolejne modernizacje wprowadzające dodatkowe faktoryzacje modułów jak w Inception-v2 [Inception-v2], lub normalizacje wartości wynikowych poszczególnych warstw jak w Inception-v3 [Inception-v3]. Kolejny innowacyjny pomysł, bazujący na dodatkowych połączeniach między blokami, został wprowadzony w 2015 roku w sieci ResNet, która została opisana w kolejnej podsekcji.

4.3.3 ResNet

Jednym z najbardziej oczywistych pomysłów na polepszenie dokładności działania sieci neuronowych jest zwiększenie liczby warstw. Jednak wraz ze wzrostem liczby warstw, trening takich architektur z użyciem tradycyjnych metod gradientowych (takich jak algorytm wstecznej propagacji błędu) staje się mniej wydajny. Problem wynika z faktu, że zmiana wartości sygnału na wyjściu sieci w odpowiedzi na sygnał wejściowy jest mniejsza wraz ze wzrostem liczby warstw [ResNet]. W takiej sytuacji gradient wyliczany na podstawie sygnału będącego różnicą pomiędzy sygnałem wejściowym a wejściowym może przyjmować wartości bliskie 0 uniemożliwiając progres uczenia się. Problem zanikającego gradientu (ang. *vanishing gradient problem*) rozwiązywany jest poprzez zastosowanie normalizacji oraz nieliniowych funkcji aktywacji, których przykłady zostały opisane w sekcji 4.3.1 oraz szeroko w literaturze np. w [ResNet 2,3,4].

Dzięki tym mechanizmom algorytm treningu głębokich sieci neuronowych w większej liczbie przypadków zbiega do użytecznego minimum lokalnego.

W momencie znalezienia takiego minimum dodanie kolejnych warstw i parametrów sieci jest redundatne, a nawet prowadzi do pogorszenia wyników treningu sieci, co związane jest z trudnościami optymalizacji przestrzeni wieloparametrycznych [Opt]. Zjawiskio to nosi nazwę degradacji treningu (ang. *degradation problem*). Twórcy architektury ResNet zaproponowali rozwiązanie tego problemu poprzez implementację bloków rezydualnych (ang. *Residuum Units*) zawierających dodatkowe, skrótowe połączenia (ang. *skip conections*) pomiędzy wejściem a wyjściem bloków. Porównanie schematów funkcjonalnych nowych bloków i wcześniej istniejącego rozwiązania stosowanego np. w AlexNet został przedstawiony na Rys. 4.11.



Rysunek 4.10: Schemat funkcjonalny pojedynczego bloku w architekturze ResNet.

Ogólną postać równania bloku rezydualnego można zapisać następująco:

$$\begin{aligned} y_l &= h(x_l) + F(x_l, W_l), \\ x_{l+1} &= f(y_l), \end{aligned} \tag{4.10}$$

gdzie x_l i x_{l+1} stanowią sygnał wejściowy i wyjściowy l-tego bloku. F stanowi funkcję rezydualną optymalizowaną podczas treningu sieci, $h(x_l)$ stanowi funkcję przekształcenia sygnału x_l przekazywanego skrótowym połączeniem, f jest funkcją ReLU, a W stanowi macierz wag.

Funkcja $h(x_l)$ jest funkcją tożsamościową, a zatem $h(x_l) = x_l$. Żeby uzasadnić ten wybór należy rozważyć propagację gradientu wewnątrz sieci składającej się z bloków rezydualnych. Dla każdego L-tego bloku zachodzi równanie:

$$x_L = x_l + \sum_{i=l}^{L-1} F(x_i, W_i) \tag{4.11}$$

Korzystając z reguły łańcuchowej [ResNet-wyj-9] można zapisać równanie na gradient

funkcji kosztu ε:

$$??\frac{\partial \varepsilon}{\partial x} = \frac{\partial \varepsilon}{\partial x_L}\frac{\partial x_L}{\partial x_l} = \frac{\partial \varepsilon}{\partial x_L}\left(1 + \frac{\partial}{\partial x_l}\sum_{i=l}^{L-1} F(x_i, W_i)\right) \tag{4.12}$$

z czego wynika, że gradient może być podzielony na dwie addytywne składowe: (1) $w = \frac{\partial \varepsilon}{\partial x_L}$ propagowaną bez wpływu na warstwy zawierające wagi i (2) $\lambda = \frac{\partial \varepsilon}{\partial x_L}\frac{\partial}{\partial x_l}\sum_{i=l}^{L-1} F(x_i, W_i)$ propagowaną przez nie.

Przykład propagacji gradientu w sieci składającej się z trzech bloków wyglądałby następująco:

$$\frac{\partial \varepsilon}{\partial x_0} = \frac{\partial \varepsilon}{\partial x_3} * (w_2 + \lambda_2) * (w_1 + \lambda_1) * (w_0 + \lambda_0) \tag{4.13}$$

Przyjmując, że wartości w są zazwyczaj znormalizowane do przedziału (-1;1) można rozważyć 4 istotne przypadki:

1. $\lambda = 0$ – nie ma skrótowych połączeń, co odpowiada płaskiej strukturze sieci. Ponieważ wartości w są z przedziału (-1;1) dodawanie kolejnych warstw wzmacnia wcześniej omówiony efekt zanikającego gradietu.

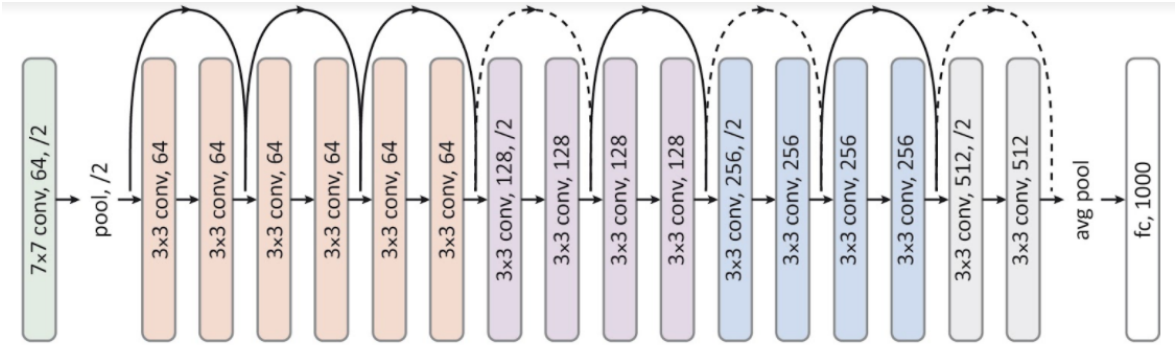
2. $\lambda > 1$ – z każdą warstwą, sumaryczna wartość gradientu zwiększa się inkrementalnie, co nazywane jest *problemem eksplozji gradientu* (ang. *exploding gradient problem*).

3. $\lambda < 1$ – przy założeniu, że $w + \lambda < 1$, dla sieci skłądających się z wielu warstw występuje problem zaniku gradientu, jak w przypadku 1. Natomiast, gdy $w + \lambda > 1$ podobnie jak w przypadku 2 może występować problem eksplozji gradienu.

4. $\lambda = 1$ – wartości w są inkrementowane dokładnie o 1, co eliminuje problemy podane w przypadkach 1, 2 i 3 i stanowiło motywacje dla twórców architektury ResNet dla wyboru funkcji tożsamościowej $h(x_l)$.

Dokładny opis matematyczny funkcjonowania bloków rezydualnych wraz z dowodami znajduje się w [ResNet-wyj]. Przykład topologii sieci składającej się z 8 bloków i łącznie 18 warstw tzw. ResNet-18, przedstawiono na Rys.

Pierwsza warstwa konwolucyjna zawiera filtry z jądrem splotu o wymiarach 7×7. W kolejnych zastosowano wymiar 3×3. Zastosowanie mniejszych wymiarów jąder splotu niż w AlexNet oraz podobnie jak w przypadku sieci GoogLeNet wyliczenie na końcu wartości średniej z dwuwymiarowych map cech zredukowało liczbę parametrów.



Rysunek 4.11: Topologia architektury ResNet-18.

Architektura ResNet-18 jest najmniejszą z pojawiających się w literaturze przykładów. W praktyce, z powodzeniem wykorzystywano topologie składające się nawet z 1202 warstw [ResNet]. W 2016 roku zaprezentowano hybrydę sieci GoogleNet i ResNet [InceptionResNet]. Pracowano również nad bardziej złożonymi blokami, co w konsekwencji doporwadziło w 2017 roku do zaprezentowania architektury ResNetX, która w wielu testach klasyfikacji różnych zbiorów okazała się być lepsza niż poprzednicy [ResNetX]. Ciekawy przegląd dotyczący historii tych prac można znaleźć w [https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035].

Sieć ResNet i jej warianty dla wielu testowych zbiorów danych takich jak ImageNet, CIFAR czy COCO osiągnęły dokładność klasyfikacji porównywalną z możliwościami ludzkiego obserwatora. Dalszy progres był możliwy m.in. dzięki zastosowaniu synergii wielu modeli, co zostało opisane w kolejnej podsekcji.

4.3.4 Złożenia

Uczenie złożeń sieci (ang. *esemble learning*) polega na wykorzystywaniu kilku modeli bazowych i wybranej metody ich synergii. Koncepcja nie jest nowa i stosowana była już przed etapem kiedy to metody głębokiego uczenia się zyskały na popularności. Dobry przegląd wcześniejszych prac bazujących na tradycyjnym podejściu do uczenia maszynowego można znaleźć w [Ensemble 1-3].

W kontekście głębokiego uczenia się stosowane są różnea metody kombinacji modeli bazowych [Ensemble]. Jako często stosowane przykłady można podać: uśrednianie, głosowanie, klasyfikacja Bayesa, generalizację stosów. Zostaną one omówione w kolejnych

paragrafach:

Uśrednianie

Uśrednianie jest prostą metodą kombinacji wyników predykcji. Najczęściej stosowane jest uśrednianie bez wag, gdzie suma wyników predykcji modeli bazowych podzielona jest przez liczbę tych modeli. Uśredniać można bezpośrednio wyniki finalnej predykcji jak również prawdopodobieństwa przynależności do odpowiednich klas, które są np. wynikiem funkcji softmax.

Główną zaletą uśredniania jest redukcja wariancji. Jest ona tym większa im bardziej nieskorelowane są wyniki predykcji modeli bazowych. Pomimo prostoty tego rodzaju koncepcja odnosiła już sukcesy m.in. w lasach losowych (Breiman, 2001).

Zastosowanie uśredniania przy silnie odstających od średniej najgorszych predykcjach znacząco obniża dokładność całego złożenia. Dlatego przy silnie nieheterogenicznych modelach bazowych dających bardzo różne wyniki często poszukiwane są inne metody.

Głosowanie

W głosowaniu stosuje się mechanizm zliczania przewidzianych przez modele bazowe etykiet. Etykieta, która została wybrana przez największą liczbę modeli bazowych jest obierana jako wynik ostatecznej predykcji. Jest to tzw. *głosowanie większościowe*.

W porównaniu do uśredniania, głosowanie jest mniej czułe na predykcje pojedynczych modeli. Wykorzystuje jednak jedynie informacje o przewidzianych etykietach, co utrudnia konstrukcje bardziej wyszukanych rozwiązań.

Klasyfikacja Bayesa

W przypadku tej metody, każdy model bazowy j postrzegany jest jako hipoteza h_j. Każda z hipotez posiada wagę proporcjonalną do prawdopodobieństwa zdarzenia, w którym dany zbiór danych trenujących zostałby wybrany z ogółu danych gdyby dana hipoteza była prawdziwa. Jest to tzw. optymalna klasyfikacja Bayesa, którą można

zapisać następującycm równaniem:

$$y = armax_{c_j \in C} \sum_{h_i \in H} P(c_j \mid h_i) P(T \mid h_i) P(h_i) \tag{4.14}$$

gdzie y to przewidziana etykieta, C jest zbiorem wszystkich możliwych klas, H to przestrzeń hipotez, a T to zbiór danych trenujących.

W praktyce z uwagi na dużą złożoność obliczeniową optymalną klasyfikację Bayesa wykorzystuje się rzadko. Cześćiej stosowane są techniki BPA (od ang. *Bayesian parameter averaging*), BMA (od ang. *Bayesian model averaging*), czy teć BMC (od ang. *Bayesian model combination*) (zob. [BPA, BMA, BMC]), które aproksymują optymalną metodę.

Generalizacja stosów

Idea generalizacji stosów oryginalnie została zaproponowana w [Wolpert, 1992]. Wykorzystana została koncepcja meta-uczenia, a zatem konstrukcja nadrzędnego klasyfikatora, którego zadniem jest wybór optymalnego wektora wag a dla stosu s predykcji dla danych x:

$$s(x) = \sum_{i=1}^{m} a_i s_i(x) \tag{4.15}$$

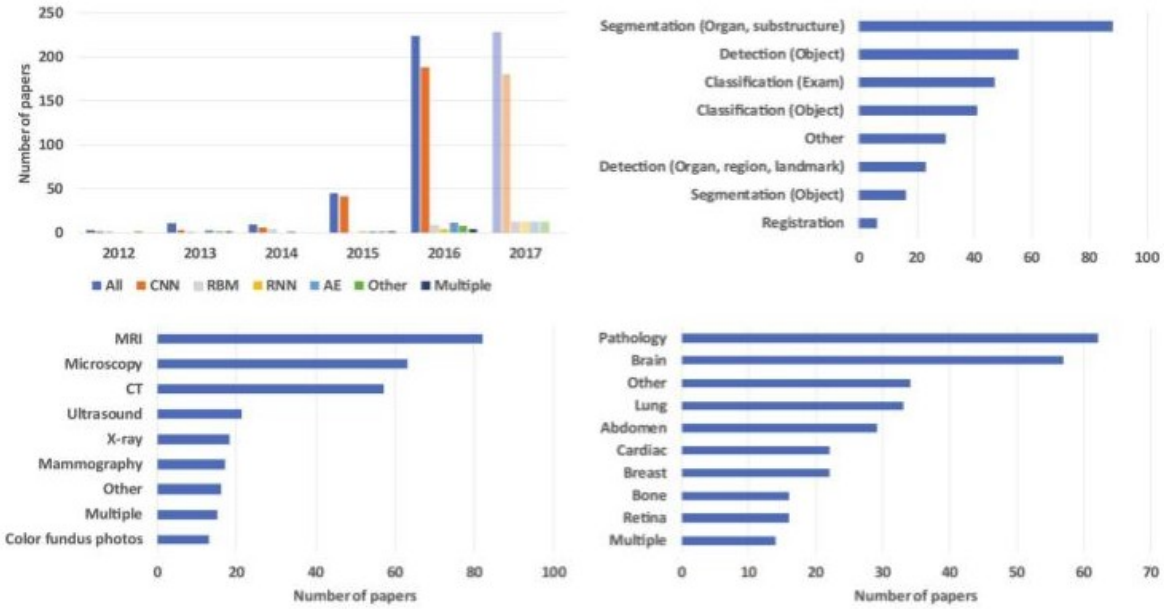
W praktyce predykcje z modeli bazowych składowane są na stosie, a następnie klasyfikator nadrzędny wykorzystuje je jako dane do swojego treningu poprawnych wartości a wykorzystując jako odniesienie znane, poprawne etykiety.

4.4 Zastosowania w medycynie

W 1994 roku ukazała się pierwsza praca, która w praktyce wykorzystywała mechanizmy związane z głębokim uczeniem do przetwarzania obrazów medycznych [Zhang, 1994]. Użyte wówczas sieci nazywano sieciami typu *shift-invariant*. Zastosowanie ich pozwoliło na eliminacje 55% FP otrzymywanych przy wcześniejszych metodach stosowanych do detekcji skupisk mikrozwapnień w mommografach. *Shift-invariant* oznaczało, że przesunięcie obrazu wejściowego nie powodowało zmian w klasyfikacji, co jest istotną cechą z uwagi na implementację toru akwizycji danych w praktyce radiologicznej.

Po roku 2012 nastąpił znaczący wzrost zainteresowania metodami głębokiego ucze-

nia się w medycynie. Obrazuje to praca z 2017, w której przytoczono statystyki medycznych publikacje zawierających słowa kluczowe związane z głębokim uczeniem się [arXiv:1702.05747v2]. Najważniejsze dane przedstawiono a Rys. 4.12.

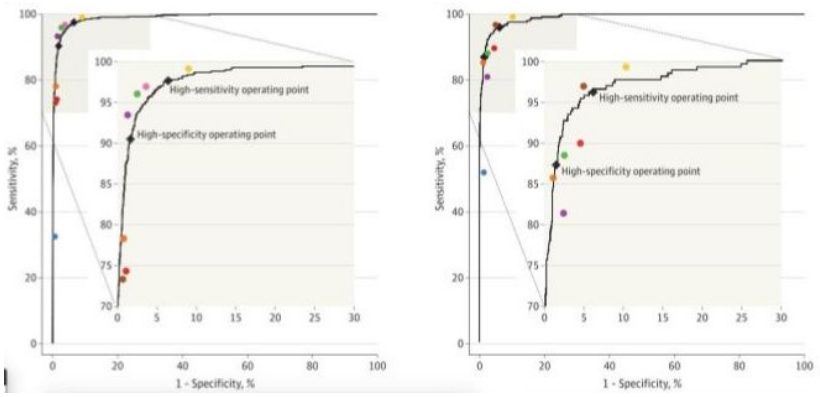


Rysunek 4.12: Statystyki dotyczące publikacji medycznych zawierających słowa kluczowe związane z głębokim uczeniem się.

Widoczny wzrost liczby publikacji nastąpił począwszy od 2015, co związane było z kilkuletnią adaptacją nowych metod w dziedzinie przetwarzania obrazów medycznych i gromadzeniem odpowiednich danych. Rok 2016 i 2017 były pod pewnym względem przełomowe gdyż pojawiało się coraz więcej prac naukowych, w których przedstawiano rezultaty dokładności klasyfikacji medycznych zbiorów danych na poziomie dorównującym ekspertom dziedzinowym.

Dla przykładu, w Listopadzie 2016 ukazał się praca grupy Google Research, Mountain View, Kalifornia [https://www.ncbi.nlm.nih.gov/pubmed/27898976], gdzie zastosowano sieć GoogLeNet w wersi inception-v3 do zautomatyzowanej detekcji retinopatii cukrzycowej i cukrzycowego obrzęku plamki w obrazach dna oka. Wyniki porównano z panelem skjładającym się z 7 ekspertów, okulistów. Porównanie przedstawiono na Rys. 4.13.

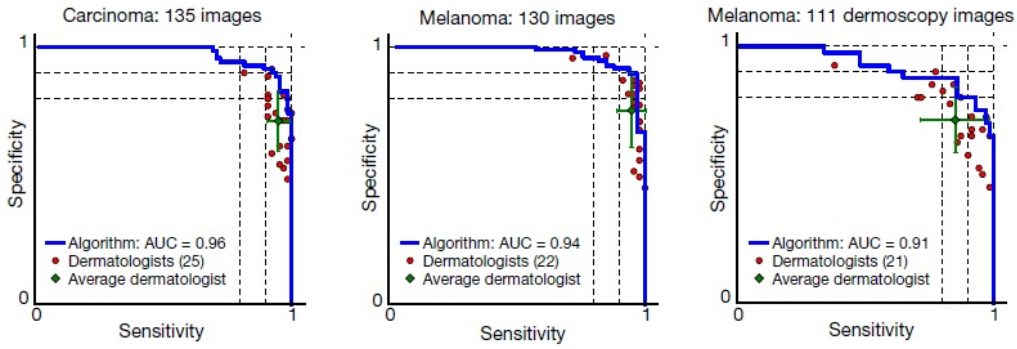
Na wykresach dla dwóch zadań klasyfikacyjnych umieszczono krzywe reprezentujące zależność swoistości od czułości dla algorytmu automatycznego oraz 7 punktów oznaczające wynik oceny każdego z ekspertów-okulistów. Ogółem mniej niż połowa



Rysunek 4.13: Porównanie automatycznej klasyfikacji retinopatii cukrzycowej i cukrzycowego obrzęku plamki z oceną panelu ekspertów

ekspertów uzyskałą lepszy wynik niż algorytm sztucznej inteligencji.

Kolejna ciekawa praca pojawiła się w czasopiśmie nature w styczniu 2017 i traktowała o automatycznej detekcji raka skóry na zdjęciach [Esteva, 2017]. Autorzy wykorzystali dane składające się z 129,450 obrazów klinicznych, na których zobrazowano 2,032 różne schorzenia skóry. Ponownie do klasyfikacji wykorzystano sieć GoogleNet w wersji inception-v3. Wyniki klasyfikacji automatycznej porównano z oceną przeprowadzoną przez 21 certyfikowanych dermatologów. Przykład porównania zaprezentowano na Rys. 4.14.



Rysunek 4.14: Porównanie automatycznej klasyfikacji 3 chorób skóry z oceną ekspertów dermatologów

Tym razem umieszczono odwrotną zależność (tj. czułość od swoistości), czerwonymi punktami oznaczono wynik oceny ekspertów, a zielonym krzyżykiem wynik uśrednienia oceny eksperckiej. W każdym przypadku średnia ocena była gorsza od automatycznje klasyfikacji.

Obrazy medyczne nie są jedynymi danymi, które z powodzeniem są przetwarzane za pomocą metod głębokiego uczenia się. W lipcu 2017, przez grupę ze Stanford University została opublikowana praca dotycząca klasyfikacji arytmii na podstawie szeregów czasowych zapisanych na elektrokardiogramach [Rajpurkar, 2017]. Autorzy wykorzystali dane z 64,121 elektrokardiogramów, próbkowanych z częstotliwością 200 Hz, pochodzących od 29,163 pacjentów. Zaprojektowano dedykowaną, 34-warstwową sieć konwolucyjną do detekcji 12 różnych dysfunkcji pracy serca, pracy prawidłowej i szumów (łącznie 14 klas). Wyniki klasyfikacji porównano z oceną prowadzoną przez 3 kardiologów. Średnia dokładność klasyfikacji automatycznej wyniosła 80% , natomiast manualnej 72%.

Naturalnie, podobnych przykładów zostało opublikowanych dużo więcej. Architektura AlexNet z sukcesem była użyta do detekcji polipów w kolonospokopii [AlexNet-kolo, 2017]. Sieć ResNet sprawdziła się w badaniach w Mayo Clinic Rotschester, dotyczących radiogenomiki i rozróżnienia zmian w mózgu bez konieczności biopsji [Mayo]. Złożenia natomiast z sukcesem zostały zaaplikowany w pracach dotyczących detekcji raka płuc, gdzie modele bazowe analizowały różne skale problemu [LungChalenge]. W wielu pracach raportuje się dokładność klasyfikacji automatycznej znaczącą przewyższające możliwości dziedzinowych ekspertów np. [https://doi.org/10.1016/j.cell.2018.03.040, FNP].

Przytoczone prace pokazują, że dla szczególnych przypadków pewien element pracy eksperta zajmującego się danymi medycznymi (np. radiologa) może być z sukcesem wspomagany przez algorytmy głębokiego uczenia się. Należy jednak podkreślić, że jest również szereg problemów wiążących się z wykorzystaniem sztucznej inteligencji w medycynie. Do najważniejszych należą:

1. Gromadzenie dużych zbiorów danych z odpowiednimi etykietami.

2. Wykorzystanie heterogenicznych danych pochodzących np. z wielu urządzeń lub modalności.

3. Kalibracja i szacowanie niepewności wyników modelu.

4. Unifikacja modeli wykonujących podobne zadania.

5. Minimalizacja liczby parametrów modelu przy zachowaniu satysfakcjonującego poziomu dokładności.

Dyskusja na temat tych problemów wciąż jest tematem wielu paneli dyskusyjnych i debat konferencyjnych np. [NVIDIA 1-2]. Najbardziej zaawansowane prace dotyczą problemu gromadzenia dużych zbiorów danych medycznych, co wymaga bliskiej współpracy ekspertów medycznych z ekspertami od uczenia maszynowego. Często konieczna jest również modyfikacja bądź tworzenie dedykowanych programów do akwizycji danych medycznych. Jako przykłady takich inicjatyw można wymienić programy Stanford Medicine [MedicalImageNet], Harward School of Medicine [10 mln images] czy Masachuset Hospital [NVIDIA 2018]. Ponadto w roku 2018 na konferencji NVIDIA GTC w San Jose (Kalifornia) Amerykańskie Stowarzyszenie Radiologii i stowarzyszenie MICCAI (od ang. *Medical Image Computing and Computer Assisted Intervention*) ogłosiły porozumienie, co do wspólnej współpracy mającej na celu eliminacje barier legislacyjnych związanych ze współpracą przy pozyskiwaniu odpowiednich danych dla wykorzystania algorytmów uczenia maszynowego.

Autor tej rozprawy jest świadom ograniczeń jakie są związane z wykorzystaniem algorytmów głębokiego uczenia się. Jednocześnie, duża liczba sukcesów, które pojawiły się w ostatnich latach w aplikacjach medycznych stanowi silną motywację dla autora do przeprowadzenia własnych badań zaprezentowanych w następnym rozdziale.

Rozdział 5

Nowa metoda oceny procesu gojenia ścięgna Achillesa

5.1 Metodyka

5.2 Rozróżnienie ścięgna zdrowego i po zerwaniu

5.3 Obliczanie krzywych gojenia

5.3.1 Topologia sieci

5.3.2 Redukcja wymiarowości

5.3.3 Miara wygojenia

Rozdział 6

Wyniki i walidacja

6.1 Ocena procesu gojenia z użyciem nowej metody

6.2 Porównanie z wynikami z rezonansu magnetycznego

6.3 Porównanie z wynikami ultrasonografii

6.4 Porównanie z wynikami badań biomechanicznych

Rozdział 7

Podsumowanie

Bibliografia

[1] Witold Pokorski and Graham G. Ross. Flat directions, string compactification and three generation models. 1998.

Dodatek A

AchillesDL: System komputerowego wspomagania oceny gojenia ścięgien i więzadeł