

Politechnika Gdańsk
Wydział Elektroniki, Telekomunikacji i Informatyki

Rozprawa doktorska

Szybkie ortogonalne sieci neuronowe
w zagadnieniach klasyfikacji
i rozpoznawania wzorców

Bartłomiej Stasiak

Promotor: prof. dr hab. Michał Jacymirski

Gdańsk 2009

Podziękowania

- Panu Profesorowi Michałowi Jacymirskiemu za inspirację, istotną „ortogonalizację” kierunków moich badań, wszechstronną opiekę i nadzór merytoryczny
- Pani Profesor Lilianie Byczkowskiej-Lipińskiej, Panu Profesorowi Piotrowi Szcześniakowi oraz Pani Profesor Marii Pietruszce za inspirację, opiekę i zapewnienie dogodnych warunków do pracy naukowej w Instytucie Informatyki PŁ
- Mojej wspaniałej Rodzinie oraz pani Uli
- Paniom Asi, Halince, Eli, Monice, Joli oraz Panu Tomkowi
- Darkowi, Kamilowi, Piotrkowi, Marcinowi i wielu innym kolegom i koleżankom z Instytutu Informatyki PŁ za wymianę myśli i inspirację

Wszystkim tu wymienionym, a i wielu niewymienionym również, pragnę serdecznie podziękować przede wszystkim za cierpliwość, wszechstronną pomoc i zrozumienie, bez których powstanie tej pracy byłoby niemożliwe.

Spis treści

1 Wstęp	1
1.1 Przegląd zastosowań szybkich liniowych przekształceń ortogonalnych i sieci neuronowych w problemach klasyfikacji	3
1.2 Cele i tezy pracy	9
2 Liniowe przekształcenia ortogonalne i ich szybkie realizacje	10
2.1 Sygnały jako elementy przestrzeni funkcyjnych	10
2.2 Reprezentacje analityczne sygnałów	13
2.3 Przykłady dyskretnych przekształceń ortogonalnych	20
2.4 Szybkie dwuetapowe algorytmy dyskretnych przekształceń ortogonalnych	23
2.4.1 Szybka transformata cosinusowa drugiego rodzaju (FCT2) .	25
2.4.2 Algorytm FCT2 z mnożnikami tangensowymi (mFCT2) . .	26
2.4.3 Algorytm szybkiej transformaty cosinusowej czwartego roduzu z mnożnikami tangensowymi (mFCT4)	27
2.5 Podsumowanie	29
3 Jednokierunkowe wielowarstwowe sieci neuronowe	30
3.1 Podstawowe modele sieci jednokierunkowych	31
3.2 Adaptacja	37
3.2.1 Algorytm wstecznej propagacji błędu	38
3.2.2 Algorytm gradientów sprzężonych	43
3.3 Klasyfikacja, a własności generalizacyjne sieci	50
3.4 Podsumowanie	56
4 Szybkie sieci neuronowe	57
4.1 Założenia podstawowe	58
4.2 Szybkie ortogonalne sieci neuronowe	59
4.2.1 Uczenie sieci typu FONN	60
4.3 Podsumowanie	66

5 Przykłady konstrukcji i rezultaty adaptacji sieci typu FONN	67
5.1 Sieci typu FONN vs szybkie sieci neuronowe	68
5.2 Rekurencyjne uczenie sieci typu FONN	70
5.3 Transformata Hartley'a, Fouriera oraz widmo amplitudowe Fouriera w realizacji neuronowej	74
5.3.1 Szybka transformata Hartley'a	76
5.3.2 Szybka transformata Fouriera	77
5.3.3 Szybka transformata Fouriera - sieć oparta na algorytmie z mnożnikami tangensowymi	78
5.3.4 Widmo amplitudowe transformaty Fouriera	80
5.3.5 Adaptacja – rezultaty testów i wnioski	82
5.4 Dwuwymiarowe sieci typu FONN	84
5.4.1 Transformata cosinusowa – metoda „wiersze-kolumny”	85
5.4.2 Transformata cosinusowa – metoda bezpośrednia	91
5.4.3 Transformata Fouriera	104
5.4.4 Sieć oparta na algorytmie z mnożnikami tangensowymi	113
5.4.5 Widmo amplitudowe dwuwymiarowej transformaty Fouriera	116
5.5 Podsumowanie	117
6 Zastosowanie sieci typu FONN w problemach klasyfikacji i rozpoznawania wzorców	120
6.1 Analiza sygnałów jednowymiarowych	120
6.1.1 Ekstrakcja cech	121
6.1.2 Klasyfikacja w oparciu o widmo amplitudowe	125
6.1.3 Rozpoznawanie rodzaju muzyki	128
6.2 Klasyfikacja i rozpoznawanie obrazów	135
6.2.1 Klasyfikacja i kategoryzacja obrazów	136
6.2.2 Klasyfikacja obrazów niezależna od rotacji, translacji i skalowania	138
6.2.3 Rozpoznawanie obiektów na zdjęciach	151
6.3 Możliwości przyspieszenia adaptacji	155
6.3.1 Dynamiczna rozbudowa struktury sieci	156
6.3.2 Równoległa realizacja obliczeń	158
6.4 Podsumowanie	163
7 Wnioski i podsumowanie pracy	165

1 Wstęp

Niniejsza praca dotyczy nowego narzędzia analizy danych jakim jest szybka ortogonalna sieć neuronowa (ang. *fast orthogonal neural network*, FONN). Użycie liczby mnogiej w temacie rozprawy uzasadnia fakt, że można mówić w zasadzie o całej klasie narzędzi opartych na wspólnych założeniach konstrukcyjnych i funkcjonalnych. Sama nazwa „szybka ortogonalna sieć neuronowa”, której autor jest współtwórcą, nawiązuje bezpośrednio do zaproponowanej na początku pierwszej dekady XXI w. koncepcji stworzenia sieci neuronowej o architekturze opartej na schemacie obliczeniowym szybkich algorytmów transformat ortogonalnych.

Z uwagi na nowatorstwo tematu najważniejszą część pracy stanowi opis podstaw konstrukcji oraz metod uczenia szybkich ortogonalnych sieci neuronowych, zaproponowanych przez autora (rozdział 4). W szczególności podkreślone zostało rozwinięcie koncepcji przedstawionej w pracach [59][131], z uwzględnieniem nowych typów neuronów i sposobu ich adaptacji. Uniwersalność i elastyczność zaproponowanych rozwiązań, uniezależniająca proces nauczania sieci od jej architektury została zademonstrowana poprzez konstrukcję sieci w oparciu o różne typy przekształceń trygonometrycznych, w tym dwuwymiarowych.

Drugim zasadniczym elementem pracy jest wykazanie przydatności szybkiej ortogonalnej sieci neuronowej w zastosowaniach praktycznych. Zagadnienie to wydawać się może dość proste z uwagi na mnogość zastosowań zarówno liniowych przekształceń ortogonalnych, jak i sieci neuronowych do analizy danych. Z drugiej jednak strony samo sformułowanie zagadnienia demonstracji możliwości nowego narzędzia ogólnego przeznaczenia stanowi niejako odwrócenie typowego schematu, w którym wychodzimy od konkretnego problemu, dla którego dobieramy optymalną metodę rozwiązania. W wyniku naturalnej, w tej sytuacji, konieczności zawężenia zakresu rozważań do określonej klasy problemów, z wielu potencjalnych obszarów zastosowań wybrany został zbiór zagadnień związanych z klasyfikacją i rozpoznawaniem wzorców.

Uzasadnienie wyboru tematu wynika z przedstawionej w następnym podrozdziale analizy istniejącej literatury. Już teraz jednak warto podkreślić, że dwa czynniki miały tu największe, zasadnicze znaczenie. Po pierwsze, konieczność de-

korelacji i znaczącej redukcji ilości danych występująca zwłaszcza w przypadku klasyfikacji obrazów, dźwięku czy innych multimediów często prowadzi do zastosowania liniowych przekształceń ortogonalnych w procesie ekstrakcji cech i na wcześniejszych etapach przetwarzania w systemach rozpoznawania wzorców. Po drugie, sieci neuronowe są jednym z podstawowych narzędzi wykorzystywanych w charakterze klasyfikatora, w odniesieniu do bardzo szerokiego spektrum problemów, stanowiąc istotną alternatywę dla innych metod. Na szczególne podkreślenie zasługuje fakt, że łączne zastosowanie liniowych metod redukcji danych z klasyfikatorem neuronowym wyznacza częsty schemat systemów rozpoznawania, stanowiąc tym samym naturalny punkt odniesienia dla rozwiązań opartych na zaproponowanej sieci typu FONN.

Struktura dalszej części pracy jest następująca: W następnych podrozdziałach przedstawiony został aktualny stan badań oraz zastosowania metod klasyfikacji opartej na sieciach neuronowych i przekształceniach ortogonalnych oraz sformułowane zostały cele i tezy pracy. Rozdział 2 przedstawia podstawy teoretyczne przekształceń ortogonalnych i zasady konstrukcji szybkich algorytmów obliczeniowych przekształceń trygonometrycznych, a w rozdziale 3 opisano podstawy działania sieci neuronowych, ze szczególnym uwzględnieniem sieci typu feed-forward trenowanych algorymem propagacji wstecz (ang. *backpropagation*). W rozdziale tym szczególnie podkreślone zostały metody zastosowania omawianych sieci w problemach klasyfikacji i rozpoznawania wzorców. W rozdziale 4 przedstawiono istniejący model sieci neuronowej o architekturze opartej na szybkich algorytmach przekształceń ortogonalnych. W rozdziale tym wprowadzono również nową koncepcję szybkich ortogonalnych sieci neuronowych bezpośrednio wykorzystujących ortogonalność operacji bazowych oraz zaprezentowano metody ich konstrukcji i uczenia. Zaproponowany nowy rodzaj sieci neuronowej został zaimplementowany i zweryfikowany eksperymentalnie w rozdziale 5. Rozdział ten prezentuje możliwości realizacji szeregu znanych szybkich przekształceń ortogonalnych przez szybkie ortogonalne sieci neuronowe. Zawarto w nim własne wyprowadzenia kilku szybkich algorytmów (w szczególności dwuetapowego, jednolitego algorytmu do obliczania dwuwymiarowej transformaty Fouriera), wykorzystanych następnie jako podstawa architektury sieci FONN. Rozdział 6 przedstawia przykłady zastosowań szybkiej ortogonalnej sieci neuronowej w problemach klasyfikacji i rozpoznawania wzorców. W rozdziale 7 przedstawiono wnioski i dokonano podsumowania wszystkich uzyskanych wyników.

1.1 Przegląd zastosowań szybkich liniowych przekształceń ortogonalnych i sieci neuronowych w problemach klasyfikacji

Liniowe przekształcenia ortogonalne stanowią użyteczne i szeroko stosowane narzędzia przetwarzania i analizy danych, znajdujące zastosowanie m. in. w kompresji danych i klasyfikacji wzorców. Jednym z podstawowych przykładów przekształceń liniowych służących do redukcji wymiaru i dekorelacji danych jest metoda PCA [97][64] (ang. *principal component analysis*), określana też jako KLT (ang. *Karhunen–Loëve transform*) [67][80][135] lub przekształcenie Hotellinga [54]. Metoda ta, oparta na obliczaniu wektorów własnych macierzy kowariancji, pozwala na określenie ortogonalnego przekształcenia liniowego maksymalizującego wariancję danych wejściowych względem kolejnych współrzędnych przestrzeni wyjściowej. Tak otrzymane przekształcenie zapewnia możliwość optymalnej, w sensie metody najmniejszych kwadratów, redukcji wymiaru wektora danych, stanowiąc tym samym podstawę zarówno efektywnych algorytmów kompresji [1][20][23][65], jak i ekstrakcji cech na potrzeby klasyfikacji [137][70][156][114][56][98]. W pracach [155][34] wykazano także związek między PCA, a klasteryzacją danych opartą na metodzie k-średnich [87]. Pokrewnymi metodami, dedykowanymi do zastosowań klasyfikacyjnych są: liniowa analiza dyskryminacyjna (ang. *linear discriminant analysis*, LDA) i liniowy dyskryminator Fishera (ang. *Fisher's linear discriminant*, FLD) [36]. Metody te ukierunkowane są na maksymalizację separacji klas, umożliwiając tym samym efektywną generację wektorów cech, bądź bezpośrednią klasyfikację danych [12][82][81][159].

Podstawową cechą powyższych metod statystycznych jest ich bezpośrednia zależność od analizowanych danych. Parametry przekształcenia liniowego, bądź kombinacji liniowej są za każdym razem obliczane na podstawie zbioru wektorów wejściowych, co wiąże się z jednej strony z możliwością optymalnego „dopasowania” do danych, ale niestety również z wysokim kosztem obliczeniowym i koniecznością przechowywania pełnej informacji o uzyskanej macierzy przekształcenia. W praktycznych zastosowaniach wykorzystuje się zatem często przekształcenia liniowe definiowane przez macierze o ustalonych z góry współczynnikach zapewniających określone własności uzyskiwanej reprezentacji. Klasycznym przykładem jest tu przekształcenie Fouriera [38] w jego podstawowych odmianach: ciągłe przekształcenie Fouriera, szereg Fouriera, dyskretnie przekształcenie Fouriera (ang. *discrete Fourier transform*, DFT) [130][113]. Przekształcenie to umożliwia dekompozycję sygnału na jego składowe harmoniczne, co — z samego tylko punktu widzenia

klasyfikacji — w radykalny sposób rozszerza możliwości interpretacyjne w stosunku do większości znanych typów sygnałów [84][130][134][28]. Cytując lorda Kelvina: „twierdzenie Fouriera jest nie tylko jednym z najpiękniejszych wyników współczesnej analizy, ale można o nim powiedzieć, że dostarcza niezastąpionego instrumentu przy rozważaniu niemal każdego zawiłego problemu w fizyce współczesnej” [84]. Prace Fouriera stały się inspiracją do poszukiwań innych przekształceń o podobnych własnościach, jak np. przekształcenie Hartley'a [47], dedykowane do przetwarzania sygnałów rzeczywistych¹.

Metody analitycznej reprezentacji sygnałów były intensywnie badane i coraz szerzej stosowane jako ważne narzędzie analizy danych, szczególnie począwszy od końca pierwszej połowy ubiegłego wieku. Jednak dopiero opracowanie efektywnych obliczeniowo szybkich algorytmów pozwoliło na pokonanie ograniczeń związanych z wydajnością stosowanego ówcześnie sprzętu, wyznaczając nowe standardy cyfrowego przetwarzania sygnałów. W przełomowej pracy z 1965 roku James Cooley i John Tukey zaprezentowali algorytm obliczania dyskretnego przekształcenia Fouriera redukujący złożoność obliczeniową z $O(N^2)$ do $O(N \log N)$ względem rozmiaru wektora wejściowego [24]. Warto tu wspomnieć o tym, że podobny, rekurencyjny sposób redukcji złożoności obliczeniowej był stosowany już znacznie wcześniej przez różnych autorów [43], począwszy od Gaussa, który wykorzystywał go do interpolacji trajektorii asteroid [50], jednak dopiero praca [24] wraz z przykładową implementacją na maszynie IBM 7094 przyczyniła się do jego szerokiej popularyzacji. Zaproponowany algorytm, określany obecnie jako szybka transformata Fouriera o podstawie 2 (ang. *fast Fourier transform*, FFT), doczekał się wkrótce licznych modyfikacji oraz umożliwił uzyskanie podobnej redukcji złożoności dla innych transformat, takich jak transformata Hartley'a (ang. *fast Hartley transform*, FHT) [16]. Podobnie transformata cosinusowa, zaproponowana w pracy [4], dzięki implementacji opartej na algorytmie FFT mogła być obliczana w szybki sposób [4][62]. Pod koniec lat siedemdziesiątych własne warianty obliczeniowe dla transformaty cosinusowej zaproponowali m.in. Haralick [46], Tseng i Miller [136], Narasimha [91] i Chen et al. [22]. Szybkie algorytmy transformat cosinusowych i sinusowych oparte na bezpośredniej faktoryzacji macierzy przekształcenia, a zatem nie wymagające obliczeń na liczbach zespolonych, zostały podane przez Wanga [143].

W ogólnym przypadku bezpośrednie algorytmy transformat cosinusowych i sinusowych [22][144][143][35] cechuje zwykle nieco mniejsza złożoność obliczeniowa w stosunku do algorytmów opartych na FFT czy innych znanych szybkich transformatach [4][91][46][86]. Oprócz samej redukcji ilości operacji arytmetycznych

¹w przeciwieństwie do przekształcenia Fouriera zdefiniowanego dla sygnałów zespolonych

w konstrukcji poszczególnych szybkich algorytmów rozważane są również inne czynniki, takie jak możliwość wykonania obliczeń *in-place* [76][27][55]. W aspekcie bezpośredniej realizacji sprzętowej istotnego znaczenia nabierają cechy strukturalne szybkich algorytmów, co doprowadziło do powstania algorytmów dwuetapowych [151][150][154]. Wśród nich wyróżnić można algorytmy o różnej strukturze obu etapów [151][150] oraz jednolite dwuetapowe algorytmy o jednakowej strukturze obu etapów [154][60].

Przytoczone wyżej przekształcenia cosinusowe i sinusowe, chociaż blisko związane z przekształceniem Fouriera, mają jednak nieco odmienne własności, takie jak np. własność koncentracji mocy na niewielkiej liczbie współczynników bliską optymalnej w odniesieniu do pewnych klas sygnałów [61], co rzutuje na ich zastosowania praktyczne. Typowym przykładem jest tu dyskretna transformata cosinusowa drugiego rodzaju DCT2 [4] (ang. *discrete cosine transform*), stanowiąca asymptotyczne przybliżenie transformaty KLT dla sygnałów Markowa pierwszego rzędu o współczynniku korelacji bliskim jedności [103][21][45][108]. Najbardziej znane zastosowania DCT2 to przede wszystkim standardy kompresji obrazów i sekwencji video takie jak JPEG (ang. *Joint Photographic Experts Group*) [142][57], czy grupa standardów MPEG (ang. *Moving Picture Experts Group*) [145]. Dyskretna transformata cosinusowa czwartego rodzaju (DCT4) w wariancie określonym jako „zmodyfikowana transformata cosinusowa” (ang. *modified discrete cosine transform*, MDCT), jest natomiast szczególnie często stosowana w przetwarzaniu dźwięku, m.in. w popularnych standardach kompresji dźwięku: MP3 (MPEG-1, warstwa 3 [17]), AAC (MPEG-2/MPEG-4 [17]), Vorbis (Ogg Vorbis [139]).

Jakkolwiek szeroko wykorzystywane w zadaniach związanych z kompresją, rozważane transformaty mają — ze względu na własność redukcji danych — zasadnicze znaczenie również we wstępnej obróbce sygnałów na potrzeby klasyfikacji i rozpoznawania wzorców. Znajdują one zastosowanie zwłaszcza w analizie danych multimedialnych np. w zadaniach rozpoznawania obiektów na obrazach [68][157][11], włącznie z obrazami medycznymi [112][158] i identyfikacją biometryczną [96][31][63]. W wielu dziedzinach stosowane są specjalne rodzaje transformat zapewniające określone własności otrzymywanej reprezentacji. Przykładem może tu być transformata Fouriera-Mellina i inne metody oparte na podobnych założeniach, zapewniające niezależność od wybranych przekształceń afiucznych obrazu [110][8][13][102][32][89], czy przekształcenie adaptacyjne, takie jak SA-DCT (ang. *shape-adaptive discrete cosine transform*) dedykowane przetwarzaniu wyodrębnionych obiektów na obrazie [111][2].

Osobną grupę zastosowań stanowi analiza i rozpoznawanie dźwięku, gdzie z uwagi na charakterystykę danych podstawowym narzędziem jest analiza oparta

na przekształceniu Fouriera i metodach pokrewnych: krótkoczasowej transformacji Fouriera (ang. *short time Fourier transform*, STFT), analizie cepstralnej, mel-cepstralnej, transformacie cosinusowej 4 rodzaju i in. [28][73]. Dziedzina klasyfikacji dźwięku obejmuje bardzo wiele różnorodnych zagadnień, od klasycznego problemu rozpoznawania mowy [101], poprzez klasyfikację instrumentów i gatunków muzyki [138], na rozpoznawaniu i interpretacji dźwięków otoczenia przez roboty i urządzenia mobilne [25] kończąc.

Redukcja wymiaru przestrzeni danych i ekstrakcja cech jest ważnym celem, realizowanym w większości systemów rozpoznawania wzorców, niezależnie od charakteru i modalności przetwarzanej informacji. Otrzymujemy w ten sposób jednak tylko pewną pośrednią reprezentację ułatwiającą osiągnięcie celu nadzawanego, jakim jest właściwa klasyfikacja.

Zagadnienia klasyfikacji i rozpoznawania wzorców stanowią jedną z najliczniejszych — obok aproksymacji, predykcji, filtracji sygnałów, kompresji, asocjacji, identyfikacji układów i sterowania — grupę zastosowań sztucznych sieci neuronowych [94][132][15][30]. Droga do osiągnięcia obecnej, mocno ugruntowanej, pozycji sieci neuronowych w tej dziedzinie była dosyć długa, jeśli za jej początek przyjmamy klasyczną pracę McCullocha i Pittsa z 1943 roku [88]. Atrakcyjna idea adaptacyjnej maszyny działającej w sposób analogiczny do funkcjonowania układu nerwowego żywych organizmów znalazła kontynuację w pracach wielu autorów, spośród których wyróżnić należy koncepcję uczenia asocjacyjnego zaproponowaną przez Hebbę [49]. Na przełomie lat 1950/1960 Rosenblatt zaproponował model *perceptronu* [104] stanowiącego klasyfikator gwarantujący zbieżność iteracyjnego procesu nauki w przypadku liniowo separowalnych klas. Z tego okresu pochodzi też koncepcja sieci Madaline (ang. *many adaptive linear element*) zaproponowana przez Widrowa [148][147]. Pomimo dużych nadziei pokładanych w sieciach tego typu, problemy związane z jednej strony z uproszczeniami przyjętego modelu klasyfikacji, z drugiej zaś z trudnością skutecznego trenowania sieci o bardziej złożonej strukturze, w tym wielowarstwowych, doprowadziły praktycznie do zaprzestania badań w tej dziedzinie na okres kilkunastu lat, z kilkoma ważnymi wyjątkami, do których można zaliczyć prace Grossberga [44], Fukushimy [39], Kohonena [72] czy Hopfielda [53]. Niewątpliwym wpływem na to miała zniechęcająca analiza możliwości ówczesnych sieci neuronowych przedstawiona przez Minsky'ego i Paperta w 1969 roku [90], w której podkreślili oni m.in. fakt niecelowości budowania wielowarstwowych sieci liniowych. Sytuacja uległa zmianie w drugiej połowie lat osiemdziesiątych, kiedy uświadomiono sobie, że wielowarstwowa sieć z nielinową funkcją aktywacji (ang. *multilayer perceptron*, MLP) jest w stanie pokonać ograniczenia dotyczczących sieci liniowych. Opracowanie przez kilku autorów, w tym przez

Rumelharta, Hintona i Williamsa [106] skutecznych metod propagacji wstecznej błędu opartych na uogólnionej regule delta umożliwiło efektywne uczenie wielowarstwowych sieci typu MLP, stanowiących obecnie jeden z najpopularniejszych rodzajów sztucznych sieci neuronowych.

Biorąc pod uwagę mnogość potencjalnych dziedzin zastosowań oraz fakt, że do realizacji całości, bądź istotnej części procesu klasyfikacji można wykorzystać również wiele innych rodzajów sieci neuronowych o zupełnie odmiennych własnościach, np. sieci typu RBF (ang. *radial basis function network*), ART (ang. *adaptive resonance theory*), sieci Kohonena, sieci rekurencyjne Hopfielda lub BAM (ang. *bidirectional associative memory*) [132][94][107], przedstawienie wyczerpującego przeglądu literatury w tym zakresie jest niemożliwe. Należy jednak zauważyć, że w praktyce częstym schematem proponowanych systemów rozpoznawania wzorców jest połączenie dekorelacji danych i ekstrakcji cech realizowane za pomocą wstępnych przekształceń liniowych z właściwą klasyfikacją realizowaną przez jednokierunkową (ang. *feed-forward*) siecią typu MLP, adaptowaną algorytmami adaptacji wstecznej (ang. *backpropagation*). Ten typ schematu klasyfikacji wykorzystywany był do analizy i rozpoznawania obrazu [95][99][18], w tym m.in. twarzy [14][69], w diagnostyce medycznej [18][66][109], do klasyfikacji dźwięku m.in. pochodzącego z transmisji telewizyjnych [79], generowanego przez naturalne instrumenty muzyczne [73] i na potrzeby aparatów słuchowych [5], do diagnostyki uszkodzeń w urządzeniach elektronicznych [6] oraz do klasyfikacji danych pochodzących z różnych czujników i urządzeń pomiarowych, takich jak sonar [10][9], spektrometr satelitarny MODIS [146], czy reflektometr optyczny [71].

Warto zauważyć, że niektórzy autorzy nie stosują wybranej transformaty wyłącznie jako skutecznego ekstraktora cech, ale wykorzystują jej pewne specyficzne własności oraz możliwość zastosowania szybkich procedur obliczeniowych. Dla przykładu, w artykule [14] przedstawiono metodę przyspieszenia procesu wykrywania twarzy na obrazie bazującą na podejściu zaproponowanym we wcześniejszych pracach [105][129]. Sieć neuronowa typu MLP trenowana jest do wykrywania twarzy na podobrazach o wymiarach 25×25 pikseli. Następnie podejmowana jest próba wykrycia twarzy na całym obrazie kolejno w każdej możliwej lokalizacji. Ponieważ zbiór wartości wyjściowych każdego neuronu ukrytego dla wszystkich możliwych lokalizacji odpowiada funkcji korelacji wzajemnej analizowanego obrazu i macierzy wag neuronu, możliwe jest alternatywnie wykonanie mnożenia ich reprezentacji w dziedzinie częstotliwości. Użycie szybkiej transformaty Fouriera (FFT) zapewnia redukcję złożoności obliczeniowej.

Niemniej jednak w większości przytoczonych wyżej zastosowań wybrane przekształcenie liniowe stanowi po prostu część deterministycznego algorytmu oblicza-

nia cech, podczas gdy elementem adaptacyjnym pozostaje sam neuronowy klasyfikator. Poprawność działania całego systemu zależy zatem w sposób kluczowy od jakości procesu ekstrakcji cech, który pomimo stosowania skutecznych narzędzi redukcji danych, musi być zawsze starannie zaprojektowany w oparciu o wiedzę i doświadczenie eksperta dziedzinowego². Dodatkowo, nieprzewidziana zmiana warunków akwizycji danych może spowodować, że ekstraktor cech nie spełni swojej funkcji, czy wręcz zacznie odfiltrowywać i usuwać informację istotną z punktu widzenia dyskryminacji klas i żaden, nawet najbardziej elastyczny i adaptacyjny klasyfikator nie będzie w stanie tej sytuacji naprawić.

Potencjalnym rozwiązaniem tego problemu jest implementacja ekstraktora cech w postaci sieci neuronowej. Jedną z zalet takiego podejścia jest możliwość zintegrowania go z (również neuronowym) klasyfikatorem do postaci pojedynczej, hybrydowej sieci neuronowej operującej na surowych danych wejściowych co, dzięki zwiększonej możliwości adaptacji, może zapewnić większą autonomię całego systemu rozpoznawania. W tej rozprawie poddano analizie takie rozwiązanie, przy czym jako jego podstawę przyjęto model sieci neuronowej o strukturze opartej na grafie dwuetapowych, jednolitych, szybkich algorytmów przekształceń trygonometrycznych zaproponowany w pracach [59][131]. W ten sposób możliwe jest z jednej strony skorzystanie z własności odpowiedniego przekształcenia trygonometrycznego, z drugiej zaś radykalne zredukowanie złożoności obliczeniowej, tak samo jak w przypadku użycia klasycznych szybkich algorytmów transformaty Fouriera, Hartley'a, czy cosinusowej. Połączenie efektywnych obliczeniowo szybkich algorytmów transformat ortogonalnych z możliwościami adaptacji, właściwymi dla sieci neuronowych stwarza zatem interesujące możliwości zastosowań, zwłaszcza do bezpośredniego rozwiązywania problemów klasyfikacyjnych wymagających przetwarzania dużych ilości wielowymiarowych danych, takich jak obrazy, czy dźwięk.

Z drugiej strony metody adaptacji szybkich sieci neuronowych zaproponowane w pracy [131] są dość mocno związane z konkretną architekturą połączeń neuronowych i — co za tym idzie — z konkretnym przekształceniem ortogonalnym, co może stanowić utrudnienie w analizie różnych rodzajów szybkich sieci. Ponadto, wykorzystany w pracy [131] klasyczny model neuronu wraz z algorytmem adaptacji nie wykorzystuje ortogonalności operacji bazowych grafu szybkiego algorytmu wyznaczającego architekturę sieci, co powoduje użycie większej niż niezbędna ilości wag neuronowych. Te dwa fakty stały się przesłankami do zaproponowania przez autora niniejszej pracy nowego typu neuronu, neuronu ortogonalnego (ang. *basic operation orthogonal neuron*, BOON) oraz sieci złożonej z takich neuronów, czyli

²Warto zauważyć, że znaczna część literatury poświęconej rozpoznawaniu wzorców koncentruje się na określaniu najbardziej skutecznych cech dla poszczególnych zagadnień

szynkowej ortogonalnej sieci neuronowej (ang. *fast orthogonal neural network* [120], FONN). Analiza zaproponowanych rozwiązań oraz wykazanie możliwości ich zastosowania do zadań klasyfikacji i rozpoznawania wzorców składają się na zasadnicze cele i tezy pracy.

1.2 Cele i tezy pracy

W niniejszej rozprawie została przedstawiona koncepcja dalszej, w stosunku do pracy [131] redukcji stopnia złożoności szybkiej sieci neuronowej dzięki wykorzystaniu ortogonalności operacji bazowych szybkich algorytmów przekształceń trygonometrycznych. Zaprezentowany został nowy typ neuronu liniowego wraz z metodami jego uczenia umożliwiającymi łatwą konstrukcję szybkich ortogonalnych sieci neuronowych wykorzystujących gradientowe metody adaptacji i wstępnią propagację błędu. Omówione zostały sposoby dalszej redukcji liczby wag sieci przy zachowaniu zdolności do realizacji szybkich przekształceń ortogonalnych. Skuteczność szybkich ortogonalnych sieci neuronowych w zadaniach klasyfikacji i rozpoznawania wzorców została zaprezentowana na zbiorach danych testowych o różnym charakterze. Rozważone zostały sposoby wykorzystania sieci w charakterze ekstraktora cech zintegrowanego z klasyfikatorem oraz metoda określania funkcji celu. Dzięki uzyskanej elastyczności w zakresie architektury i nauczania sieci podkreślona została łatwość połączenia szybkiej ortogonalnej sieci neuronowej z sieciami neuronowymi innych typów.

Cele pracy:

1. Opracowanie, implementacja i weryfikacja metod konstrukcji i nauczania szybkich ortogonalnych sieci neuronowych.
2. Zademonstrowanie skuteczności szybkich ortogonalnych sieci neuronowych w klasyfikacji wzorców.

Tezy pracy:

1. Wykorzystanie ortogonalności operacji bazowych umożliwia uzyskanie istotnej poprawy efektywności nauczania w stosunku do istniejących szybkich sieci neuronowych, przy zachowaniu możliwości realizacji znanych szybkich przekształceń ortogonalnych.
2. Szybkie ortogonalne sieci neuronowe posiadają własności kwalifikujące do zastosowań w zadaniach klasyfikacji danych i rozpoznawaniu wzorców.

2 Liniowe przekształcenia ortogonalne i ich szybkie realizacje

Mówiąc o przekształceniu ortogonalnym będziemy zasadniczo mieć na myśli przekształcenie liniowe $\mathbb{R}^N \rightarrow \mathbb{R}^N$ lub $\mathbb{C}^N \rightarrow \mathbb{C}^N$ definiowane pewną macierzą o szczególnych własnościach, a mianowicie *macierzą ortogonalną*. Właściwe zdefiniowanie tego pojęcia wymaga zwięzłego wprowadzenia teoretycznego przedstawionego w kolejnych dwóch podrozdziałach¹. Sformalizujemy w nich pojęcie ortogonalności oraz przytoczymy pewne podstawowe definicje i twierdzenia dotyczące konstrukcji przekształceń ortogonalnych w dość ogólnym kontekście [130][115]. Z uwagi na wykorzystanie elementów teorii sygnałów, w stosunku do danych poddawanych przekształceniom będziemy używać pojęcia *sygnału*, przy czym za model sygnału przyjmujemy tu rzeczywistą lub zespoloną funkcję jednej, bądź dwóch zmiennych niezależnych. W większości przypadków nasze rozważania zawężimy przy tym do analizy sygnałów dyskretnych, czyli takich, które możemy reprezentować w postaci skończonych, bądź nieskończonych ciągów liczb rzeczywistych bądź zespolonych.

2.1 Sygnały jako elementy przestrzeni funkcyjnych

Przyjmując założenie, że sygnały reprezentowane są w postaci funkcji, można posłużyć się pojęciem przestrzeni sygnałów X ; jej elementami będą funkcje modelujące sygnały. Tak jak w przestrzeniach funkcyjnych, również tutaj można zdefiniować pojęcie normy sygnału. Określenie normy wymaga, aby spełnione były aksjomaty przestrzeni liniowej nad pewnym ciałem F . W zależności od ciała F możemy zatem mówić o rzeczywistej lub zespolonej unormowanej przestrzeni liniowej sygnałów. Ponadto w przestrzeni takiej w naturalny sposób określone jest pojęcie odległości sygnałów, w postaci metryki indukowanej przez normę:

$$\forall_{\mathbf{x}, \mathbf{y} \in X} \quad (\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\| : \quad (2.1)$$

¹Częściowo wykorzystano w nich fragmenty pracy magisterskiej autora [115]

Można pokazać, że dla poprawnie określonej normy $\|\cdot\|$ funkcjonał jest poprawnie określona metryką w takiej przestrzeni.

W przestrzeniach liniowych możemy mówić o liniowej niezależności jej elementów. Powiemy, że zbiór $\{\mathbf{x}_i : i = 1; 2; \dots; N\}$ przestrzeni liniowej X nad ciałem F jest liniowo niezależny, jeśli:

$$\sum_{i=1}^N \mathbf{x}_i = \quad \Rightarrow \forall_{i=1,2,\dots,N} \quad i = 0 ; \quad (2.2)$$

gdzie $i \in F$, dla $i = 1; 2; \dots; N$, a jest elementem zerowym przestrzeni X .

Przestrzeń liniową, w której każdy zbiór liniowo niezależny zawiera co najwyżej N elementów, nazywamy przestrzenią N -wymiarową. Każdy liniowo niezależny, N -elementowy zbiór w N -wymiarowej przestrzeni liniowej nazywamy bazą tej przestrzeni. Ważnym pojęciem, przydatnym również w przestrzeniach sygnałów jest pojęcie zbieżności ciągów elementów tych przestrzeni. Powiemy, że ciąg $\{\mathbf{x}_i\}$ elementów przestrzeni X jest zbieżny do $\mathbf{x} \in X$ w sensie metryki jeśli

$$\lim_{i \rightarrow \infty} (\mathbf{x}_i; \mathbf{x}) = 0 : \quad (2.3)$$

Łatwo pokazać, że każdy ciąg zbieżny w przestrzeni X jest *ciągiem Cauchy'ego*, tj.

$$\forall_{\varepsilon > 0} \exists_{n_0 \in \mathbb{N}} \forall_{n, m > n_0} \quad (\mathbf{x}_n; \mathbf{x}_m) < " : \quad (2.4)$$

Przestrzeń liniową z metryką, w której zachodzi zależność odwrotna, tj. w której każdy ciąg Cauchy'ego jest zbieżny w sensie metryki do pewnego elementu tej przestrzeni, nazywamy *przestrzenią zupełną*. Unormowana przestrzeń zupełna nazywana jest *przestrzenią Banacha*. Rozważmy dla przykładu zbiór X sygnałów określonych w przedziale domkniętym $[0; T]$. Przyjmując normę tej przestrzeni w postaci

$$\|\mathbf{x}\| = \int_0^T |x(t)| dt ; \quad (2.5)$$

przy założeniu, że dla każdego sygnału $\mathbf{x} = x(t)$ powyższa całka istnieje, otrzymujemy przestrzeń sygnałów *bezwzględnie całkowalnych* $L(0; T)$. Jeśli dla zbioru X normę zdefiniujemy jako

$$\|\mathbf{x}\| = \sqrt{\int_0^T |x(t)|^2 dt} ; \quad (2.6)$$

wówczas określmy tak uzyskaną przestrzeń jako *przestrzeń sygnałów całkowalnych z kwadratem*, inaczej *przestrzeń sygnałów o ograniczonej energii* $L^2(0; T)$. W obu

powyższych przestrzeniach ciągi spełniające warunek (2.4) są zbieżne, zatem przestrzenie te – jako unormowane i zupełne – są przestrzeniami Banacha.

Innymi ważnymi przykładami przestrzeni Banacha są przestrzenie, których elementami są ciągi skończone bądź nieskończone. W pierwszym wypadku mówimy o przestrzeniach \mathbb{R}^n oraz \mathbb{C}^n wszystkich n -elementowych ciągów liczb rzeczywistych (odpowiednio: zespolonych), w których norma dowolnego elementu $\mathbf{x} = (x_1; x_2; \dots; x_n)$ jest zdefiniowana jako

$$\|\mathbf{x}\| = \sqrt{\sum_{i=1}^n |x_i|^2}; \quad (2.7)$$

gdzie $x_i \in \mathbb{R}$ (odpowiednio: $x_i \in \mathbb{C}$), dla $i = 1; 2; \dots; n$. Dla przestrzeni ℓ^2 ciągów nieskończonych natomiast przyjmujemy normę

$$\|\mathbf{x}\| = \sqrt{\sum_{i=1}^{\infty} |x_i|^2}; \quad (2.8)$$

gdzie $\mathbf{x} = (x_1; x_2; x_3; \dots)$.

Dla dalszych rozważań konieczne jest również wprowadzenie w przestrzeni sygnałów pojęcia iloczynu skalarnego. Stanowi to punkt wyjścia do poszukiwania alternatywnych, analitycznych metod przedstawienia sygnału. Przestrzeń liniową X , w której zdefiniowano odwzorowanie: $\langle \cdot, \cdot \rangle : X \times X \rightarrow \mathbb{C}$, spełniające aksjomaty iloczynu skalarnego nazywamy *przestrzenią unitarną*, jeżeli norma tej przestrzeni została określona jako

$$\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}; \quad (2.9)$$

gdzie $\mathbf{x} \in X$.

Zupełną przestrzeń unitarną, czyli przestrzeń Banacha z iloczynem skalarnym spełniającym warunek (2.9) nazywamy *przestrzenią Hilberta*.

Przestrzeń sygnałów o ograniczonej energii $L^2(0; T)$ z iloczynem skalarnym

$$\langle \mathbf{x}, \mathbf{y} \rangle = \int_0^T x(t)y^*(t)dt; \quad (2.10)$$

gdzie gwiazdka (*) oznacza sprzężenie, jest przestrzenią Hilberta. Również przestrzeń \mathbb{C}^n z iloczynem skalarnym

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^n x_i y_i^*; \quad (2.11)$$

oraz przestrzeń ℓ^2 z iloczynem skalarnym

$$\langle \mathbf{x}; \mathbf{y} \rangle = \sum_{i=1}^{\infty} x_i y_i^* ; \quad (2.12)$$

stanowią przykłady przestrzeni Hilberta.

Przestrzeń sygnałów bezwzględnie całkowalnych $L(0; T)$ natomiast nie jest przestrzenią Hilberta, gdyż nie można w niej poprawnie zdefiniować odwzorowania spełniającego założenia iloczynu skalarnego.

Wprowadzenie do przestrzeni sygnałów iloczynu skalarnego umożliwia określenie wielu ważnych pojęć i własności, m. in. pojęcie kąta między elementami tej przestrzeni i ich wzajemnej ortogonalności. Mówimy, że dwa sygnały $\mathbf{x}, \mathbf{y} \in X$ są *ortogonalne*, jeśli ich iloczyn skalarny jest równy zeru, tj. $\langle \mathbf{x}; \mathbf{y} \rangle = 0$. Przyjmując definicję cosinusa kąta między elementami \mathbf{x} i \mathbf{y} jako:

$$\cos = \frac{\langle \mathbf{x}; \mathbf{y} \rangle}{\|\mathbf{x}\| \|\mathbf{y}\|} ; \quad (2.13)$$

można zauważyć, iż warunek ortogonalności jest równoważny stwierdzeniu, że cosinus kąta między elementami \mathbf{x} i \mathbf{y} wynosi 0. Jeżeli każde dwa elementy pewnego zbioru $\{\mathbf{x}_i\}$ przestrzeni unitarnej X są ortogonalne, wówczas zbiór ten określamy jako *zbiór ortogonalny*. Jeśli ponadto zachodzi:

$$\forall_i \|\mathbf{x}_i\| = 1 ; \quad (2.14)$$

wówczas zbiór ten nazwiemy *zbiorem ortonormalnym*.

2.2 Reprezentacje analityczne sygnałów

Początkowa postać sygnału poddawanego analizie nie zawsze jest korzystna pod względem łatwości osiągnięcia postawionego celu przetwarzania. Typową sytuacją jest zaszumienie sygnału lub po prostu występowanie w nim dużej ilości składników nieistotnych z punktu widzenia przeprowadzanej analizy. Przydatne staje się wówczas przedstawienie sygnału w pewnej postaci analitycznej. Istota takiej zmiany reprezentacji sprowadza się do przyporządkowania każdemu sygnałowi z danej przestrzeni X skończonego, bądź nieskończonego ciągu identyfikującego go jednoznacznie w tej przestrzeni. Jest to tzw. reprezentacja dyskretna, w odróżnieniu od reprezentacji ciągłej, w której sygnałowi przyporządkowany zostaje element pewnej przestrzeni funkcji ciągłych.

W przypadku reprezentacji dyskretnej, jeśli X^N jest N -wymiarową przestrze-

nią sygnałów, interesuje nas określenie takiego przekształcenia $\phi : X^N \rightarrow \mathbb{C}^N$, że dla $\mathbf{x} \in X^N$ współrzędne wektora wynikowego $(\mathbf{x}) \in \mathbb{C}^N$, są jednocześnie współrzędnymi sygnału wejściowego \mathbf{x} względem pewnej zadanej bazy tej przestrzeni. Ten ostatni warunek można zapisać w postaci

$$\mathbf{x} = \sum_{i=1}^N {}_i \mathbf{x}_i ; \quad (2.15)$$

gdzie zbiór $\{\mathbf{x}_i : i = 1; \dots; N\}$ stanowi bazę przestrzeni X^N , a wektor $\boldsymbol{\alpha} = [{}_1; {}_2; \dots; {}_N]^T$ (gdzie symbol T oznacza transpozycję) jest reprezentacją sygnału \mathbf{x} w zbiorze \mathbb{C}^N . Równoważnie:

$$\mathbf{x} = \mathbf{A}\boldsymbol{\alpha} ; \quad (2.16)$$

gdzie \mathbf{A} jest macierzą przekształcenia, której kolumny stanowią wektory $\{\mathbf{x}_i\}$, tj. $\mathbf{A} = [\mathbf{x}_1; \mathbf{x}_2; \dots; \mathbf{x}_N]$.

Główny problem polega tu na tym, w jaki sposób mając daną bazę \mathbf{x}_i i sygnał \mathbf{x} wyznaczyć wektor $\boldsymbol{\alpha}$ spełniający zależność (2.15). Zagadnienie to jest prostsze przy założeniu, że przestrzeń X jest przestrzenią unitarną. Możemy wówczas w miejsce równości (2.15) przyrównać iloczyny skalarne obu jej stron z elementami bazy:

$$\langle \mathbf{x}; \mathbf{x}_j \rangle = \left\langle \sum_{i=1}^N {}_i \mathbf{x}_i; \mathbf{x}_j \right\rangle = \sum_{i=1}^N \langle \mathbf{x}_i; \mathbf{x}_j \rangle {}_i ; \quad (2.17)$$

gdzie $j = 1; \dots; N$.

Otrzymamy wówczas układ N równań z niewiadomymi ${}_i$. Możemy zapisać go w postaci macierzowej jako

$$\mathbf{b} = \mathbf{B}\boldsymbol{\alpha} ; \quad (2.18)$$

gdzie

$$\mathbf{b} = \begin{bmatrix} \langle \mathbf{x}; \mathbf{x}_1 \rangle \\ \langle \mathbf{x}; \mathbf{x}_2 \rangle \\ \vdots \\ \langle \mathbf{x}; \mathbf{x}_N \rangle \end{bmatrix} ; \quad \mathbf{B} = \begin{bmatrix} \langle \mathbf{x}_1; \mathbf{x}_1 \rangle & \langle \mathbf{x}_1; \mathbf{x}_2 \rangle & \dots & \langle \mathbf{x}_1; \mathbf{x}_N \rangle \\ \langle \mathbf{x}_2; \mathbf{x}_1 \rangle & \langle \mathbf{x}_2; \mathbf{x}_2 \rangle & \dots & \langle \mathbf{x}_2; \mathbf{x}_N \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \mathbf{x}_N; \mathbf{x}_1 \rangle & \langle \mathbf{x}_N; \mathbf{x}_2 \rangle & \dots & \langle \mathbf{x}_N; \mathbf{x}_N \rangle \end{bmatrix} ;$$

Z liniowej niezależności elementów $x_1; x_2; \dots; x_N$ wynika, że $\det \mathbf{B} \neq 0$, zatem układ (2.18) ma jednoznaczne rozwiązanie postaci

$$\boldsymbol{\alpha} = \mathbf{B}^{-1} \mathbf{b} ; \quad (2.19)$$

Zauważmy, że postać macierzy \mathbf{B} uprości się znacznie, jeśli postawimy warunek, aby baza przestrzeni była zbiorem ortogonalnym (mówimy wówczas o *bazie ortogonalnej*). Jedyne niezerowe elementy będą wówczas występować na głównej przekątnej. Jeśli natomiast przyjmiemy za bazę zbiór ortonormalny (czyli *bazę ortonormalną*), wówczas macierz \mathbf{B} stanie się macierzą jednostkową. W takiej sytuacji otrzymujemy natychmiast rozwiązanie układu (2.18) jako

$$\boldsymbol{\alpha} = \begin{bmatrix} \langle \mathbf{x}; \mathbf{x}_1 \rangle \\ \langle \mathbf{x}; \mathbf{x}_2 \rangle \\ \vdots \\ \langle \mathbf{x}; \mathbf{x}_N \rangle \end{bmatrix} : \quad (2.20)$$

Zauważmy, że ortonormalność bazy oznacza tym samym, że kolumny macierzy przekształcenia \mathbf{A} są ortonormalne². Macierz o wyrazach rzeczywistych posiadającą tę szczególną własność nazywamy *macierzą ortogonalną*, a macierz o wyrazach zespolonych – *macierzą unitarną*. Analogicznie, przekształcenie liniowe definiowane taką macierzą określamy jako *przekształcenie ortogonalne*. Na podstawie (2.20) i (2.18) łatwo zauważyc, że transpozycja macierzy ortogonalnej równa jest jej odwrotnością:

$$\mathbf{A}^T = \mathbf{A}^{-1} : \quad (2.21)$$

Stąd otrzymujemy, że

$$\mathbf{A}\mathbf{A}^T = \mathbf{A}^T\mathbf{A} = \mathbf{I} : \quad (2.22)$$

gdzie \mathbf{I} jest macierzą jednostkową. W szczególnym przypadku, jeśli macierz \mathbf{A} jest ponadto symetryczna, wówczas jest ona odwrotnością sama dla siebie, spełniając:

$$\mathbf{A}\mathbf{A} = \mathbf{I} : \quad (2.23)$$

W przypadku macierzy unitarnej we wzorach (2.21), (2.22) operację transpozycji należy zastąpić przez sprzężenie hermitowskie.

A zatem w przypadku ortonormalnej bazy $\{\mathbf{x}_i\}$ przestrzeni N -wymiarowej X^N , określenie współrzędnych dowolnego elementu \mathbf{x} tej przestrzeni względem bazy $\{\mathbf{x}_i\}$ upraszcza się do obliczenia odpowiednich iloczynów skalarnych (2.20). Warto zauważyc, że w przypadku, gdy dana baza przestrzeni nie jest ortonormalna, możemy dokonać jej ortonormalizacji za pomocą procedury Grama-Schmidta [130].

Skończenie-wymiarowa reprezentacja sygnału jest wygodna z punktu widzenia bezpośredniej realizacji za pomocą układów cyfrowych. Stanowi ona jednakże często pewne uproszczenie, a bliższym rzeczywistości modelem przestrzeni sygna-

²Można pokazać, że wówczas również wiersze mają tę własność

łów jest przestrzeń nieskończonego wymiaru. W przestrzeni takiej zagadnienie analitycznej reprezentacji jej elementów przedstawia się nieco inaczej. Możemy tu bowiem mówić jedynie o N -wymiarowej podprzestrzeni X^N rozpiętej na bazie $\{\mathbf{x}_i; i = 1; \dots; N\}$. Jeżeli element \mathbf{x} należy do przestrzeni, ale nie jest jednocześnie elementem podprzestrzeni X^N , możemy wówczas tylko reprezentować go za pomocą elementu $\hat{\mathbf{x}} \in X^N$ z pewną dokładnością. Dochodzimy w ten sposób do problemu najlepszej aproksymacji. Problem ten można sformułować jako żądanie minimalizacji wyrażenia:

$$\|\mathbf{x} - \sum_{i=1}^N \alpha_i \mathbf{x}_i\| : \quad (2.24)$$

Dla jego rozwiązania możemy w przestrzeniach unitarnych posłużyć się twierdzeniem o rzucie [130].

Twierdzenie 2.2.1 (o rzucie ortogonalnym) *Niech X będzie przestrzenią unitarną, a X^N jej N -wymiarową podprzestrzenią rozpiętą na ortonormalnej bazie $\{\mathbf{x}_i; i = 1; \dots; N\}$. Dla każdego $\mathbf{x} \in X$ istnieje dokładnie jeden element $\hat{\mathbf{x}} \in X^N$ dany wyrażeniem:*

$$\hat{\mathbf{x}} = \sum_{i=1}^N \langle \mathbf{x}; \mathbf{x}_i \rangle \mathbf{x}_i : \quad (2.25)$$

taki, że:

1. dla każdego $\tilde{\mathbf{x}} \in X^N$ różnego od $\hat{\mathbf{x}}$ jest spełniona nierówność:

$$\|\mathbf{x} - \hat{\mathbf{x}}\| < \|\mathbf{x} - \tilde{\mathbf{x}}\|,$$
2. element $\mathbf{x} - \hat{\mathbf{x}}$ jest ortogonalny do każdego elementu należącego do X^N .

Element $\hat{\mathbf{x}}$ nazywamy rzutem ortogonalnym elementu \mathbf{x} na przestrzeń X^N , przy czym z p.1 powyższego twierdzenia wynika, że stanowi on poszukiwaną najlepszą aproksymację elementu \mathbf{x} . Jakość tej aproksymacji możemy stwierdzić rozważając normę błędu aproksymacji danego jako $\|\mathbf{x} - \hat{\mathbf{x}}\|$. Z uwagi na p.2 twierdzenia o rzucie możemy tu zastosować twierdzenie Pitagorasa dla przestrzeni unitarnych, otrzymując:

$$\|\mathbf{x} - \hat{\mathbf{x}}\|^2 = \|\mathbf{x}\|^2 - \|\hat{\mathbf{x}}\|^2 : \quad (2.26)$$

Należy zauważyć, że reprezentacja $\hat{\mathbf{x}}$ elementu \mathbf{x} nie jest wzajemnie jednoznaczna. W szczególności dany element $\hat{\mathbf{x}}$ może być aproksymacją wielu elementów $\mathbf{x} \in X$ różniących się normą elementu różnicowego $\|\mathbf{x} - \hat{\mathbf{x}}\|$. W celu poprawy jakości aproksymacji możemy zwiększać liczebność bazy $\{\mathbf{x}_i\}$ poprzez dodawanie do niej kolejnych elementów ortonormalnych. W ten sposób możemy skonstruować nieskończony ciąg elementów ortogonalnych $\{\mathbf{x}_n\}$ i nieskończony ciąg elementów

aproksymujących $\{\hat{\mathbf{x}}_n\}$, taki że:

$$\hat{\mathbf{x}}_n = \sum_{i=1}^n \mathbf{x}_i : \quad (2.27)$$

Należy tu rozważyć kwestię, czy tak zdefiniowany ciąg jest zbieżny do elementu \mathbf{x} , tj. czy możemy za pomocą elementów \mathbf{x}_i aproksymować \mathbf{x} z dowolną dokładnością.

Można pokazać, że ciąg $\{\hat{\mathbf{x}}_n\}$ jest ciągiem Cauchy'ego w przestrzeni X . Korzystając z uogólnionego wzoru Pitagorasa otrzymamy:

$$\|\hat{\mathbf{x}}_n\|^2 = \left\| \sum_{i=1}^n \mathbf{x}_i \right\|^2 = \sum_{i=1}^n \|\mathbf{x}_i\|^2 = \sum_{i=1}^n |\mathbf{x}_i|^2 : \quad (2.28)$$

Ponieważ \mathbf{x}_i są ortonormalne, możemy to zapisać w postaci:

$$\|\hat{\mathbf{x}}_n\|^2 = \sum_{i=1}^n |\mathbf{x}_i|^2 : \quad (2.29)$$

Stąd i z równości (2.26) wynika:

$$\|\hat{\mathbf{x}}_n\|^2 = \|\mathbf{x}\|^2 - \sum_{i=1}^n |\mathbf{x}_i|^2 : \quad (2.30)$$

a zatem:

$$\sum_{i=1}^n |\mathbf{x}_i|^2 \leq \|\mathbf{x}\|^2 : \quad (2.31)$$

Przechodząc z wyrażeniami (2.30) i (2.31) do granicy otrzymujemy:

$$\|\hat{\mathbf{x}}_n\|^2 = \|\mathbf{x}\|^2 - \sum_{i=1}^{\infty} |\mathbf{x}_i|^2 : \quad (2.32)$$

$$\sum_{i=1}^{\infty} |\mathbf{x}_i|^2 \leq \|\mathbf{x}\|^2 : \quad (2.33)$$

Nierówność (2.33) nosi nazwę *nierówności Bessela*.

Weźmy teraz pod uwagę dowolne wyrazy $\hat{\mathbf{x}}_m, \hat{\mathbf{x}}_k$ ciągu $\{\hat{\mathbf{x}}_n\}$ dla $m < k$ i zauważmy, że

$$\|\hat{\mathbf{x}}_k - \hat{\mathbf{x}}_m\|^2 = \left\| \sum_{i=m+1}^k \mathbf{x}_i \right\|^2 = \sum_{i=m+1}^k |\mathbf{x}_i|^2 : \quad (2.34)$$

Stąd i na podstawie nierówności Bessela łatwo zauważyc, że ciąg $\{\hat{\mathbf{x}}_n\}$ jest ciągiem Cauchy'ego.

A zatem, aby zapewnić zbieżność ciągu elementów aproksymujących $\{\hat{\mathbf{x}}_n\}$ wy-

starczy postawić warunek, aby rozważana przestrzeń X była przestrzenią Hilberta. Mając zapewnioną zbieżność ciągu, należy jeszcze stwierdzić, że jego granicę istotnie stanowi aproksymowany element \mathbf{x} . Przyjmijmy następującą definicję [130]:

definicja 2.2.1 *Niech X będzie przestrzenią Hilberta, a zbiór $\{\mathbf{x}_n : n = 1; 2; \dots\}$ nieskończonym ortogonalnym zbiorem elementów tej przestrzeni. Zbiór ten nazywamy zbiorem zupełnym jeśli nie istnieje nie należący do niego różny od zera element przestrzeni X ortogonalny do każdego elementu \mathbf{x}_n .*

Rozwiążanie kwestii zbieżności ciągu $\{\hat{\mathbf{x}}_i\}$ do elementu aproksymowanego wynika z następującego twierdzenia [130]:

Twierdzenie 2.2.2 *Niech X będzie przestrzenią Hilberta, a zbiór $\{\mathbf{x}_n : n = 1; 2; \dots\}$ nieskończonym ortonormalnym zbiorem elementów tej przestrzeni. Dla każdego $\mathbf{x} \in X$ ciąg $\{\hat{\mathbf{x}}_n\}$ dany jako:*

$$\hat{\mathbf{x}}_n = \sum_{i=1}^n c_i \mathbf{x}_i ; \text{ gdzie } c_i = \langle \mathbf{x}; \mathbf{x}_i \rangle ; \quad (2.35)$$

jest zbieżny do elementu \mathbf{x} wtedy i tylko wtedy, gdy zbiór $\{\mathbf{x}_n\}$ jest zbiorem zupełnym.

Powyższy warunek zbieżności możemy zapisać w postaci:

$$\mathbf{x} = \sum_{i=1}^{\infty} c_i \mathbf{x}_i ; \text{ gdzie } c_i = \langle \mathbf{x}; \mathbf{x}_i \rangle ; \quad (2.36)$$

Tak określony szereg nazywamy szeregiem Fouriera elementu \mathbf{x} wyznaczonym względem zbioru $\{\mathbf{x}_n\}$. Liczby c_i nazywamy współczynnikami Fouriera tego szeregu. Szereg Fouriera stanowi poszukiwaną analityczną reprezentację sygnału \mathbf{x} . Należy podkreślić, że warunkiem koniecznym i wystarczającym zachodzenia równości (2.36) jest zupełność zbioru $\{\mathbf{x}_n\}$. Ponadto, jeżeli zbiór $\{\mathbf{x}_n\}$ jest zupełny, nierówność Bessela staje się równością:

$$\sum_{i=1}^{\infty} |c_i|^2 = \|\mathbf{x}\|^2 ; \quad (2.37)$$

Równość (2.37) nazywamy *równością Parsevala*.

Można zauważyc, że w ten sposób rozszerzyliśmy pojęcie reprezentacji analitycznej sygnału za pomocą odwzorowania (por. założenia na stronie 14) na przypadek przestrzeni nieskończonego wymiaru. W tym wypadku definiujemy odwzorowanie

$$: X \rightarrow \ell^2 ; \quad (2.38)$$

przyporządkowujące każdemu elementowi $x \in X$, gdzie X jest przestrzenią, w której istnieje zbiór ortonormalny zupełny³, ciąg jego współczynników Fouriera. Jak można pokazać, odwzorowanie jest liniowe, a ponadto zachowuje normę i iloczyn skalarny. W konsekwencji, każda ośrodkowa przestrzeń Hilberta jest izometryczna z przestrzenią ℓ^2 . Fakt ten ma istotne znaczenie praktyczne, z uwagi na uproszczenie metod analizy i modeli obliczeniowych, wynikające z operowania na reprezentacji sygnałów w postaci ciągów. Dodatkowo, w zależności od konkretnych zagadnień i rodzaju sygnałów wybór odpowiedniej ortonormalnej bazy przestrzeni umożliwia ograniczenie reprezentacji sygnału, do postaci ciągu o skończonej liczbie elementów, z zachowaniem wszystkich jego charakterystyk ważnych z punktu widzenia przeprowadzanej analizy. Otrzymujemy w ten sposób możliwość aproksymacji sygnału, stanowiącą m.in. podstawę konstrukcji efektywnych metod ekstrakcji cech na potrzeby klasyfikacji i rozpoznawania wzorców.

W rozważanych wyżej przestrzeniach sygnałów można określić różne ortonormalne zbiory zupełne umożliwiające reprezentację elementów tych przestrzeni za pomocą szeregów Fouriera. Dla przykładu, w przestrzeni $L^2(0; T)$ zbiór zespolonych funkcji wykładniczych $\{e^{jn\frac{2\pi}{T}t}\}$:

$$e_n(t) = \frac{1}{\sqrt{T}} e^{jn\frac{2\pi}{T}t}; \quad (2.39)$$

gdzie $n = 0; \pm 1; \dots$, a j jest jednostką urojoną, jest zbiorem ortonormalnym zupełnym umożliwiającym konstrukcję szeregu Fouriera dla sygnału $x(t) \in L^2(0; T)$:

$$x(t) = \sum_{n=-\infty}^{\infty} n \frac{1}{\sqrt{T}} e^{jn\frac{2\pi}{T}t}; \quad (2.40)$$

gdzie współczynniki n są zdefiniowane jako:

$$n = \langle x(t); \frac{1}{\sqrt{T}} e^{jn\frac{2\pi}{T}t} \rangle = \frac{1}{\sqrt{T}} \int_0^T x(t) e^{-jn\frac{2\pi}{T}t} dt; \quad (2.41)$$

Włączając współczynnik $1=\sqrt{T}$ do współczynnika n i przyjmując:

$$\chi_n = \frac{1}{T} \int_0^T x(t) e^{-jn\frac{2\pi}{T}t} dt; \quad (2.42)$$

wyrażenie (2.40) można zapisać w postaci:

$$x(t) = \sum_{n=-\infty}^{\infty} \chi_n e^{jn\frac{2\pi}{T}t}; \quad (2.43)$$

³Warto zauważyć, że warunek ten jest spełniony dla ośrodkowych przestrzeni Hilberta

Szereg (2.43) nazywamy zespolonym szeregiem Fouriera.

Inne przykłady zbiorów ortonormalnych zupełnych możemy otrzymać na podstawie m.in. wielomianów Legendre'a, Czebyszewa lub Laguerre'a, a także funkcji Haara lub Walsha [130]. W dalszych rozważaniach skoncentrujemy się jednak na szeregu Fouriera zdefiniowanym zależnością (2.40). W szczególności, rozważać będziemy przekształcenie znane w dziedzinie cyfrowego przetwarzania sygnałów jako DFT (ang. *discrete Fourier transform*), stanowiące odpowiednik wyrażenia (2.42) dla sygnałów dyskretnych, reprezentowanych w postaci ciągów skończonych. Oprócz niego w następnym podrozdziale przedstawimy również rodzinę dyskretnych przekształceń sinusowych i cosinusowych, a w kolejnym – podamy przykłady konstrukcji szybkich algorytmów do ich obliczania.

2.3 Przykłady dyskretnych przekształceń ortogonalnych

Dyskretne przekształcenie Fouriera (DFT) sygnału reprezentowanego w postaci N -elementowego ciągu rzeczywistego lub zespolonego $x(n)$, dla $n = 0; 1; \dots; N - 1$, definiujemy następująco [84]:

$$X(k) = \text{DFT}_N\{x(n)\} = \sum_{n=0}^{N-1} x(n) e^{-j2\pi \frac{nk}{N}} ; \quad (2.44)$$

gdzie $k = 0; 1; \dots; N - 1$; zaś przekształcenie odwrotne IDFT jako:

$$x(n) = \text{IDFT}_N\{X(k)\} = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{j2\pi \frac{nk}{N}} ; \quad (2.45)$$

Warto zauważyć, że zarówno w przypadku przekształcenia Fouriera, jak i dla zdefiniowanych niżej przekształceń sinusowych i cosinusowych ich macierzowa reprezentacja nie jest, ściśle rzecz biorąc ortogonalna. Ponieważ kolumny ich macierzy przekształcenia są ortogonalne, ale nie ortonormalne, wystarczy zatem je przeskalać ze współczynnikiem zależnym od rozmiaru przekształcenia. Dla przykładu, ortogonalna postać DFT przedstawia się następująco:

$$X(k) = \text{DFT}_N\{x(n)\} = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x(n) e^{-j2\pi \frac{nk}{N}} ; \quad (2.46)$$

Ponieważ to dodatkowe skalowanie nie zmienia istoty przekształcenia, będziemy je pomijać w definicjach kolejnych transformat, zachowując zgodność z konwencją

przyjętą w literaturze. Wynik przekształcenia DFT określamy mianem *widma*, a ciąg $X(k)$ nazywamy ciągiem współczynników widmowych. Dodatkowo definiujemy widmo amplitudowe DFT:

$$|X(k)| = \left| \sum_{n=0}^{N-1} x(n) e^{-j2\pi \frac{nk}{N}} \right| ; \quad (2.47)$$

oraz fazowe:

$$\varphi(k) = \arg \left\{ \sum_{n=0}^{N-1} x(n) e^{-j2\pi \frac{nk}{N}} \right\} ; \quad (2.48)$$

gdzie $|\cdot|$ i $\arg(\cdot)$ oznaczają odpowiednio amplitudę i argument liczby zespolonej.

Rozszerzając dziedzinę transformaty przekonujemy się że jest ona w istocie okresowa, tj:

$$X(k + mN) = X(k) ; \quad (2.49)$$

gdzie $m \in \mathbb{Z}$. Z tego względu niekiedy jako ciąg wyjściowy rozpatruje się równoważnie ciąg $X(k)$ dla $k = -N/2; -N/2 + 1; \dots; -1; 0; 1; \dots; N/2 - 1$. Warto również zauważyć, że w przypadku rzeczywistego ciągu wejściowego $x(n)$ widmo transformaty Fouriera jest symetryczne. W szczególności, widmo amplitudowe jest parzyste:

$$|X(-k)| = |X(k)| ; \quad (2.50)$$

gdzie $k \in \mathbb{Z}$; zaś widmo fazowe – nieparzyste:

$$\varphi(-k) = -\varphi(k) : \quad (2.51)$$

A zatem stosując DFT do sygnałów rzeczywistych połowa z N obliczonych współczynników może być odrzucona jako nadmiarowa. Cechy tej⁴ nie posiadają przekształcenia cosinusowe i sinusowe, które są określone dla ciągów liczb rzeczywistych [103][4][131].

Niech $x(n)$, gdzie $n = 0; 1; \dots; N - 1$ będzie teraz ciągiem rzeczywistym. Dyskretne przekształcenie cosinusowe (DCT_N^{II}) i sinusowe (DST_N^{II}) drugiego rodzaju definiujemy [131] jako:

$$L_N^{II}(k) = DCT_N^{II}\{x(n)\} = \sum_{n=0}^{N-1} x(n) C_{4N}^{k(2n+1)} ; \quad (2.52)$$

$$Q_N^{II}(k) = DST_N^{II}\{x(n)\} = \sum_{n=0}^{N-1} x(n) S_{4N}^{(k+1)(2n+1)} ; \quad (2.53)$$

⁴będącej wadą z punktu widzenia sygnałów rzeczywistych, z uwagi na nadmiarowość obliczeń

gdzie $k = 0; 1; \dots; N - 1$, a współczynniki pomocnicze C_N^r i S_N^r są równe:

$$\begin{aligned} C_N^r &= \cos\left(2 \frac{\pi k}{N}\right) ; \\ S_N^r &= \sin\left(2 \frac{\pi k}{N}\right) ; \end{aligned} \quad (2.54)$$

Przekształceniem odwrotnym do transformaty cosinusowej drugiego rodzaju jest transformata cosinusowa trzeciego rodzaju (DCT^{III}) i — analogicznie — dla DST^{II} odwrotnym przekształceniem jest transformata sinusowa trzeciego rodzaju (DST^{III}):

$$L_N^{III}(k) = \text{DCT}_N^{III}\{x(n)\} = \sum_{n=0}^{N-1} p_n x(n) C_{4N}^{(2k+1)n} ; \quad (2.55)$$

$$Q_N^{III}(k) = \text{DST}_N^{III}\{x(n)\} = \sum_{n=0}^{N-1} p_n x(n) S_{4N}^{(2k+1)(n+1)} ; \quad (2.56)$$

gdzie

$$p_n = \begin{cases} \frac{1}{N} , & \text{dla } n = 0 ; \\ \frac{2}{N} , & \text{dla } n \neq 0 : \end{cases} \quad (2.57)$$

Natomiast przekształcenie cosinusowe czwartego rodzaju (DCT^{IV}):

$$L_N^{IV}(k) = \text{DCT}_N^{IV}\{x(n)\} = \sum_{n=0}^{N-1} x(n) C_{8N}^{(2k+1)(2n+1)} ; \quad (2.58)$$

gdzie $k = 0; 1; \dots; N - 1$, jest symetryczne, czyli jest przekształceniem odwrotnym samo dla siebie. Taka sama własność dotyczy przekształcenia sinusowego czwartego rodzaju (DST^{IV}):

$$Q_N^{IV}(k) = \text{DST}_N^{IV}\{x(n)\} = \sum_{n=0}^{N-1} x(n) S_{8N}^{(2k+1)(2n+1)} ; \quad (2.59)$$

Zatem w przypadku transformat czwartego rodzaju ich macierz przekształcenia spełnia, z dokładnością do współczynnika skalującego $\sqrt{2/N}$, równość (2.23).

Kolejnym symetrycznym przekształceniem jest dyskretna transformata Hartley'a (DHT) zdefiniowana jako:

$$H_N(k) = \text{DHT}_N\{x(n)\} = \sum_{n=0}^{N-1} x(n) (C_N^{kn} + S_N^{kn}) ; \quad (2.60)$$

która z racji bezpośrednich powiązań z przekształceniem Fouriera:

$$\begin{aligned} \frac{H_N(k) + H_N(N-k)}{2} &= \operatorname{Re}(DFT_N\{x(n)\}) ; \\ \frac{H_N(k) - H_N(N-k)}{2} &= -\operatorname{Im}(DFT_N\{x(n)\}) ; \end{aligned} \quad (2.61)$$

gdzie Re i Im oznaczają odpowiednio część rzeczywistą i urojoną liczby zespolonej, bywa stosowana zamiennie do DFT w przypadku przetwarzania sygnałów rzeczywistych. Transformatę Hartley'a przedstawiamy tutaj razem z definicjami przekształceń cosinusowych i sinusowych, z uwagi na to, że można ją traktować jako pewien wariant transformaty sinusowej lub cosinusowej pierwszego rodzaju [131].

Wymienione wyżej transformaty posiadają również warianty przeznaczone do przetwarzania sygnałów dwuwymiarowych, takich jak obrazy. Przykładowo, dla wejściowego dyskretnego sygnału dwuwymiarowego (obrazu) $x(n; m)$ dwuwymiarową transformatę Fouriera definiujemy następująco:

$$X_{N \times M}(p; q) = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} x(n; m) e^{-j2\pi \frac{pn}{N}} e^{-j2\pi \frac{qm}{M}} ; \quad (2.62)$$

gdzie $p; n = 0; 1; \dots; N-1$; oznacza numer wiersza, $q; m = 0; 1; \dots; M-1$ oznacza numer kolumny, a N i M definiują odpowiednio wysokość i szerokość obrazu wejściowego. Podobnie, wersję dwuwymiarową transformaty cosinusowej II rodzaju, 2D-DCT^{II} definiujemy jako:

$$L_{N \times M}^{II}(p; q) = \text{DCT}_{N \times M}^{II}\{x(n; m)\} = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} x(n; m) C_{4N}^{(2n+1)p} C_{4M}^{(2m+1)q} ; \quad (2.63)$$

2.4 Szybkie dwuetapowe algorytmy dyskretnych przekształceń ortogonalnych

Złożoność obliczeniowa ogólnego przekształcenia liniowego definiowanego macierzą $N \times N$ odpowiada złożoności mnożenia macierzowego i wynosi $O(N^2)$. Podstawowym sposobem przyspieszenia obliczeń jest wykorzystanie pewnych własności konkretnej macierzy przekształcenia w celu jej faktoryzacji na ciąg macierzy rzadkich. Zastosowanie strategii *divide et impera* prowadzi do rekurencyjnego wykorzystania algorytmu faktoryzacji, którego rezultatem jest $O(\log N)$ macierzy rzadkich. Mnożenie przez każdą z nich wymaga $O(N)$ operacji arytmetycznych, a zatem złożoność całego wynikowego szybkiego algorytmu wynosi $O(N \log N)$.

Za podstawę konstrukcji szybkich ortogonalnych sieci neuronowych została wybrana rodzina szybkich dwuetapowych, jednolitych algorytmów transformat cosinusowych i sinusowych [60][154]. Ich istotną zaletą jest regularność i przejrzystość struktury obliczeń, którą wygodnie można reprezentować w postaci skierowanego grafu przepływu sygnału⁵. Zasadnicza część każdego algorytmu obejmuje wzór rozkładu N -punktowego przekształcenia na dwa $N=2$ -punktowe, oraz procedurę podstawową opartą na wzorze rozkładu z uwzględnieniem własności symetrii danej transformaty.

Typowo, ciąg wejściowy jest rozdzielany na dwa podciągi z wykorzystaniem wyłącznie operacji dodawania/odejmowania, następnie oba podciągi są poddawane przekształceniom $N=2$ -punktowym, a ich wyniki są ostatecznie scalane zgodnie z procedurą podstawową, z udziałem zarówno operacji dodawania jak i mnożenia. Cały ten proces jest rekurencyjnie powielany dla przekształceń $N=2$ -punktowych, dając w efekcie algorytm dwuetapowy, w którym pierwszy etap jest wyznaczony przez operację rozdzielania ciągów wejściowych, wykonywaną we wszystkich krokach rekurencji, a etap drugi jest rezultatem rekurencyjnego stosowania procedury podstawowej. Dodatkowo, ciąg wejściowy i wyjściowy mogą być poddane jednorazowej permutacji elementów.

Jednolitość algorytmu, wynikająca z odpowiedniego sformułowania procedury podstawowej, wyraża się podobieństwem strukturalnym obu etapów. W szczególności, w obu etapach wykorzystywane są dwupunktowe *operacje podstawowe* równoważne mnożeniu dwuelementowych wektorów przez macierze 2×2 . Operacje podstawowe zgrupowane są w warstwy, z których każda połączona jest tylko z warstwą poprzednią i następną. Typowo ilość warstw w każdym z dwóch etapów odpowiada ilości kroków rekurencji algorytmu.

W kolejnym podrozdziale zostały zaprezentowane przykłady procedury podstawowej i grafu przepływu dla szybkiej transformaty cosinusowej drugiego i czwartego rodzaju. Dla obu transformat przedstawiono pewną modyfikację szybkiego algorytmu określana jako *algorytm z mnożnikami tangensowymi*, cechujący się zmniejszoną ilością operacji arytmetycznych wykonywanych w drugim etapie. Wyprowadzenia wzorów, również dla innych rodzajów transformat ortogonalnych, zostały podane m.in. w [131], a w przypadku algorytmów z mnożnikami tangensowymi — w pracach [153][58]. Na ich podstawie autor niniejszej rozprawy dokonał opracowania szybkiego dwuetapowego algorytmu dwuwymiarowej transformaty cosinusowej drugiego rodzaju oraz — również dwuwymiarowej — transformaty Fouriera. Szczegóły wyprowadzenia tych algorytmów zostały przedstawione w rozdziale 5, a przykłady ich zastosowania do klasyfikacji obrazów — w rozdziale 6.2.

⁵Ta forma będzie preferowana w niniejszej pracy zamiast reprezentacji macierzowej

2.4.1 Szybka transformata cosinusowa drugiego rodzaju (FCT2)

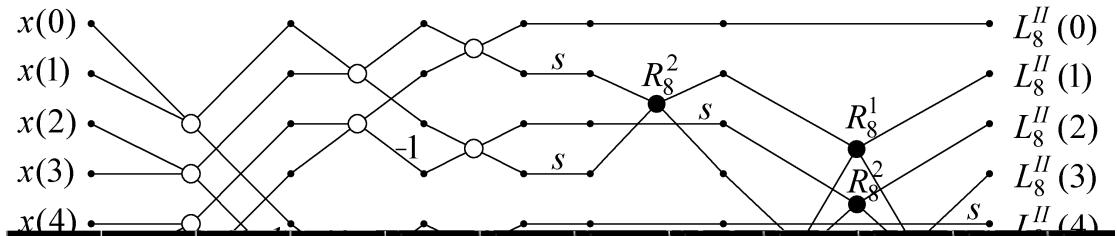
Procedurę podstawową szybkiej transformaty cosinusowej drugiego rodzaju (2.52) dla wejściowego N -elementowego ciągu $x(n)$, gdzie $n = 0; 1; \dots; N - 1; N = 2^m$; $m \in \mathbb{N}$, zdefiniujemy jako [58]:

$$\begin{aligned} L_N^{II}(0) &= L_1(0) ; \\ L_N^{II}(N=2) &= \sqrt{2}=2 \cdot L_2(0) ; \\ L_N^{II}(k) &= C_{4N}^k L_1(k) + S_{4N}^k L_2(N=2-k) ; \\ L_N^{II}(N-k) &= -S_{4N}^k L_1(k) + C_{4N}^k L_2(N=2-k) ; \\ k &= 1; 2; \dots; N=2-1 ; \end{aligned} \quad (2.64)$$

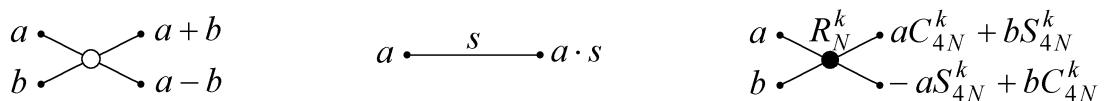
gdzie $L_1(k) = \text{DCT}_{N/2}^{II}\{a(n)\}; L_2(k) = \text{DCT}_{N/2}^{II}\{b(n)\}$, współczynniki C_{4N}^k i S_{4N}^k są zdefiniowane wzorami (2.54) a ciągi $a(n)$ i $b(n)$ są utworzone z ciągu wejściowego $x(n)$ następująco:

$$\begin{aligned} a(n) &= x(2n) + x(2n+1) ; \\ b(n) &= (-1)^n (x(2n) - x(2n+1)) ; \\ n &= 0; 1; \dots; N=2-1 ; \end{aligned} \quad (2.65)$$

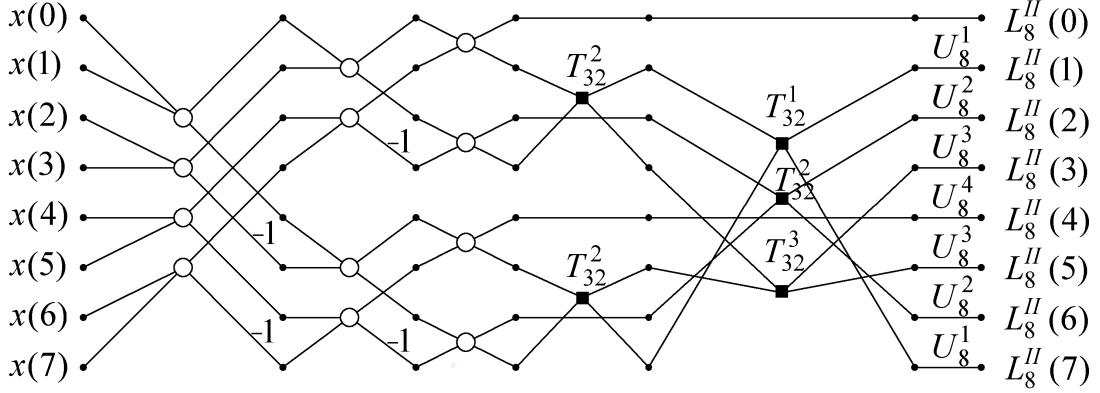
Wzory (2.64) i (2.65) stosuje się rekurencyjnie co prowadzi do jednorodnego, dwuetapowego algorytmu FCT2 przedstawionego w postaci grafu skierowanego na rys. 2.1, 2.2.



Rysunek 2.1: Graf algorytmu FCT2 dla $N = 8$, $s = \sqrt{2}=2$



Rysunek 2.2: Operacje podstawowe algorytmu FCT2

Rysunek 2.3: Graf przepływu algorytmu mFCT2 dla $N = 8$

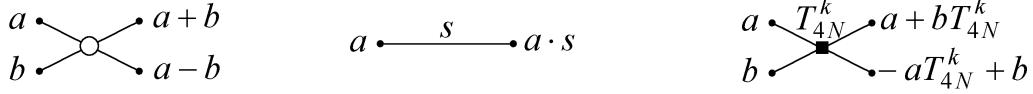
2.4.2 Algorytm FCT2 z mnożnikami tangensowymi (mFCT2)

Mnożąc i dzieląc wzory (2.64) przez C_{4N}^k uzyskujemy dodatkową optymalizację [58] wynikającą ze zgrupowania współczynników C_{4N}^k do postaci pojedynczego mnożenia wykonywanego na ostatnim etapie transformaty (rys. 2.3, 2.4). Wartości wykorzystywanych w tym mnożeniu współczynników U_N^k definiowane są rekurencyjnie:

$$\begin{aligned}
 U_N^0 &= 1 ; \\
 U_N^{N/2} &= \sqrt{2}=2 ; \\
 U_4^1 &= \sqrt{2}=2 \cdot C_{16}^1 ; \\
 U_K^k &= U_{K/2}^k C_{4K}^k ; \\
 U_K^{K/2-k} &= U_{K/2}^k C_{4K}^{K/2-k} ; \\
 U_K^{K/4} &= \sqrt{2}=2 \cdot C_{4K}^{K/4} ; \\
 U_N^{N/2+k} &= U_N^{N/2-k} ; \\
 U_N^{N-k} &= U_N^k ; \\
 k &= 1;2;\dots;K=4-1 ; \\
 K &= 8;16;\dots;N ;
 \end{aligned} \tag{2.66}$$

a współczynniki T_{4N}^k występujące w drugim etapie algorytmu (rys. 2.4) definiuje się jako:

$$T_K^k = \tan (2 \cdot k=K) : \tag{2.67}$$



Rysunek 2.4: Operacje podstawowe algorytmu mFCT2

2.4.3 Algorytm szybkiej transformaty cosinusowej czwartego rodzaju z mnożnikami tangensowymi (mFCT4)

Procedurę podstawową szybkiej transformaty cosinusowej czwartego rodzaju (2.58) dla wejściowego N -elementowego ciągu $x(n)$, gdzie $n = 0; 1; \dots; N - 1; N = 2^m$; $m \in \mathbb{N}$ zdefiniujemy jako [153]:

$$\begin{aligned}
 L_N^{IV}(k) &= C_{8N}^{2k+1} L_1(k) + S_{8N}^{2k+1} L_2(N=2-1-k) ; \\
 L_N^{IV}(i) &= -S_{8N}^{2k+1} L_1(k) + C_{8N}^{2k+1} L_2(N=2-1-k) ; \\
 L_2^{IV}(0) &= x(0) C_{16}^1 + x(1) S_{16}^1 ; \\
 L_2^{IV}(1) &= x(0) S_{16}^1 - x(1) C_{16}^1 ; \\
 k &= 0; 1; \dots; N=2-1 ; \\
 i &= N-1-k ;
 \end{aligned} \tag{2.68}$$

gdzie

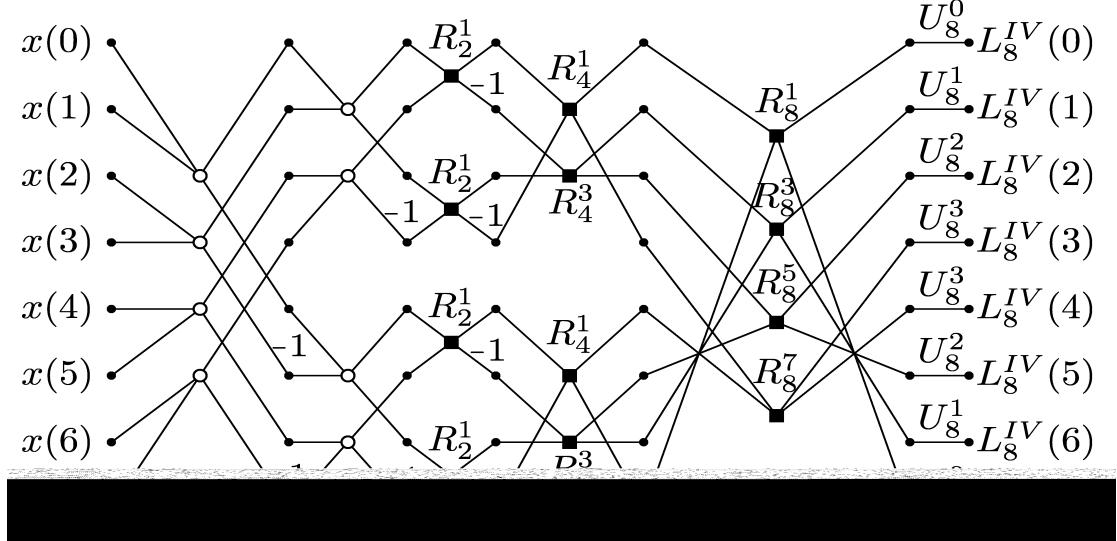
$$\begin{aligned}
 L_N^{IV}(k) &= \text{DCT}_N^{IV}\{x(n)\} ; \\
 L_1(k) &= \text{DCT}_{N/2}^{IV}\{x(2n) + x(2n+1)\} ; \\
 L_2(k) &= \text{DCT}_{N/2}^{IV}\{(-1)^n(x(2n) - x(2n+1))\} ;
 \end{aligned}$$

Podobnie jak w przypadku mFCT2 algorytm z mnożnikami tangensowymi (mFCT4) otrzymuje się następująco [153]:

$$\begin{aligned}
 L_N^{IV}(k) &= C_{8N}^{2k+1} [L_1(k) + T_{8N}^{2k+1} L_2(N=2-1-k)] ; \\
 L_N^{IV}(i) &= C_{8N}^{2k+1} [-T_{8N}^{2k+1} L_1(k) + L_2(N=2-1-k)] ; \\
 L_2^{IV}(0) &= C_{16}^1 [x(0) + x(1) T_{16}^1] ; \\
 L_2^{IV}(1) &= C_{16}^1 [x(0) T_{16}^1 - x(1)] ; \\
 k &= 0; 1; \dots; N=2-1 ; \\
 i &= N-1-k ;
 \end{aligned} \tag{2.69}$$

gdzie współczynniki T_{8N}^k dane są wzorem (2.67).

Wzory (2.69) są stosowane rekurencyjnie, prowadząc do otrzymania jednorodnego algorytmu mFCT4, przedstawionego w postaci skierowanego grafu (rys. 2.5, 2.6).

Rysunek 2.5: Graf przepływu algorytmu mFCT4 dla $N = 8$

$$\begin{array}{c} a \\ b \end{array} \xrightarrow{\quad} \begin{array}{c} a+b \\ a-b \end{array} \quad a \xrightarrow{s} a \cdot s \quad \begin{array}{c} a \\ b \end{array} \xrightarrow{R_N^k} \begin{array}{c} a+b \\ -aT_{8N}^k + b \end{array}$$

Rysunek 2.6: Operacje podstawowe algorytmu mFCT4

Współczynniki C_{8N}^{2k+1} są zgrupowane na ostatnim etapie przetwarzania w postaci pojedynczych mnożników U_N^k . Ich wartości oblicza się rekurencyjnie:

$$\begin{aligned} U_K^k &= U_{K/2}^k \cos((2k+1)\pi/(4K)) ; \\ U_K^{K/2-1-k} &= U_{K/2}^k \cos((K-2k-1)\pi/(4K)) ; \\ U_2^0 &= \cos(\pi/8) ; \\ k &= 0; 1; \dots; K=4-1 ; \\ K &= 4; 8; \dots; N ; \end{aligned} \tag{2.70}$$

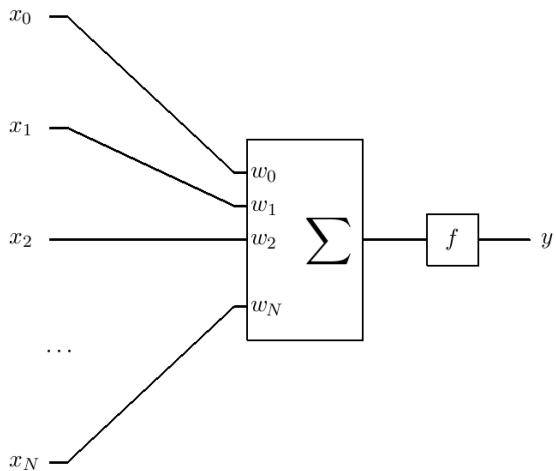
Rozważając strukturę połączeń pierwszego etapu we wszystkich zaprezentowanych dotąd grafach przepływu zauważamy, iż żaden z nich nie definiuje algorytmu *in-place*. Istotnie, numery wyjść wszystkich operacji podstawowych w pierwszych dwóch warstwach różnią się od numerów wejść (przykładowo: pierwsza operacja pierwszej warstwy ma wejścia o indeksach 0, 1, natomiast wyjścia o indeksach 0, 4). Z drugiej strony, taka struktura pierwszego etapu zapewnia możliwość podania elementów ciągu wejściowego w kolejności naturalnej. W rozdziale 5.3 zaprezentowano podejście alternatywne, zapewniające możliwość obliczeń *in-place*, jednak kosztem konieczności wykonania dodatkowej permutacji wejściowej.

2.5 Podsumowanie

W tym rozdziale przytoczono niezbędne podstawy teoretyczne umożliwiające sformułowanie pojęcia ortogonalności oraz wykorzystanie go do konstrukcji reprezentacji analitycznej sygnałów w przestrzeniach Hilberta skończonego i nieskończonego wymiaru. Przedstawiono przykłady dyskretnych przekształceń ortogonalnych oraz ich szybkich schematów obliczeniowych opartych na znanych dwuetapowych, jednolitych algorytmach. Zaprezentowane schematy obliczeniowe zostały w rozdziale 4 wykorzystane jako podstawa architektury szybkich ortogonalnych sieci neuronowych. Kolejne przykłady dwuetapowych, jednorodnych algorytmów, opracowane przez autora i zastosowane w konstrukcji szybkich sieci ortogonalnych do analizy i rozpoznawania obrazu, zostały zaprezentowane w rozdziale 5.

3 Jednokierunkowe wielowarstwowe sieci neuronowe

Podstawowym elementem konstrukcyjnym większości znanych sztucznych sieci neuronowych jest neuron, modelowany zazwyczaj w postaci sumatora uzupełnionego o tzw. funkcję aktywacji (rys. 3.1). Sumator oblicza sumę ważoną swoich wejść, przy czym wartości wag podlegają zmianie podczas procesu uczenia (treningu), co stanowi podstawę adaptacji neuronu i umożliwia mu rozwiązanie postawionego zadania. Wybór funkcji aktywacji oraz algorytmu adaptacji decydują o rodzaju neuronu i tym samym o rodzaju całej sieci neuronowej.



Rysunek 3.1: Ogólny model neuronu

Innym elementarnym kryterium podziału istniejących sieci neuronowych jest schemat połączeń między neuronami i sposób przetwarzania sygnału. Pod tym względem wyróżniamy zasadniczo sieci jednokierunkowe (ang. *feed-forward*), w których sygnał jest podawany na wejście i propagowany w przód do wyjścia, oraz sieci rekurencyjne zawierające pętle sprzężenia zwrotnego. Z uwagi na fakt, iż szybkie ortogonalne sieci neuronowe stanowią pewien szczególny przypadek sieci jednokierunkowych, ten rodzaj architektury zostanie bliżej omówiony w następnym podrozdziale.

3.1 Podstawowe modele sieci jednokierunkowych

Klasyczny neuron [88] jest skrajnie uproszczonym modelem neuronu biologicznego, który generuje impuls „na wyjściu” po przekroczeniu wartości progowej przez sumę pobudzeń wejściowych. To założenie prowadzi do przyjęcia funkcji aktywacji f typu skokowego:

$$y = f(u(\mathbf{x})) = \begin{cases} 1 & , \text{ dla } u(\mathbf{x}) \geq 0 ; \\ 0 & , \text{ dla } u(\mathbf{x}) < 0 ; \end{cases} \quad (3.1)$$

gdzie

$$u(\mathbf{x}) = \mathbf{w}^T \mathbf{x} = \sum_{j=0}^N w_j x_j ; \quad (3.2)$$

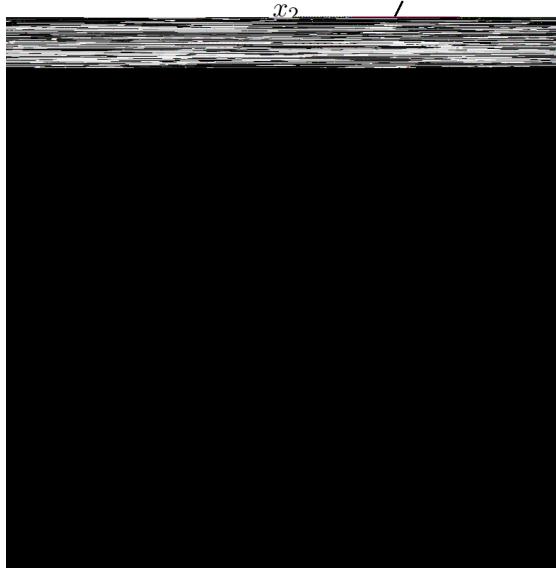
wektor $\mathbf{w} = [w_0; w_1; \dots; w_N]^T$ reprezentuje wagi neuronu, zaś $\mathbf{x} = [1; x_1; x_2; \dots; x_N]^T$ jest N -elementowym wektorem wejściowym uzupełnionym dodatkową współrzędną o ustalonej wartości $x_0 = 1$. Ten dodatkowy element, określany mianem *biasu*, umożliwia zmianę progu aktywacji neuronu, powyżej którego wyjście przyjmie wartość 1. Jak widać na rys. 3.1 i we wzorze (3.2) wejście, na które podawany jest bias również posiada swoją wagę w_0 , zatem próg aktywacji może być adaptowany w procesie nauki. Należy zauważyć, że bias jest elementem opcjonalnym¹, jednak z powodów, które zostaną omówione za chwilę, jego użycie poprawia zwykle własności funkcjonalne sieci.

Innym elementarnym typem funkcji aktywacji jest funkcja liniowa (w najprostszym przypadku tożsamościowa: $y(u) = u$). Neuron z liniową funkcją aktywacji określamy po prostu jako neuron liniowy. Warto zauważyć, że oba rozwiązania mogą być stosowane łącznie, jak w przypadku neuronu typu adaline [149], gdzie pomimo zastosowania skokowej funkcji aktywacji, adaptacja jest oparta na sygnale wyjściowym sumatora, a zatem na funkcji liniowej względem wejść i wag neuronu.

Neuron liniowy jest w stanie zrealizować każde odwzorowanie liniowe $\mathbb{R}^N \rightarrow \mathbb{R}$, stanowiąc tym samym, w połączeniu z funkcją aktywacji (3.1), najprostszy klasyfikator. Istotnie, jeśli dwie klasy są separowalne liniowo w \mathbb{R}^N , to znaczy jeśli istnieje hiperpłaszczyzna rozdzielająca te klasy, wówczas istnieje taki wektor wag neuronu, że dla wszystkich obiektów z klasy pierwszej sygnał wyjściowy sumatora $u(x)$ będzie mniejszy od zera, a dla wszystkich obiektów z drugiej klasy większy. A zatem, biorąc pod uwagę (3.1), uzyskujemy binarną odpowiedź na pytanie o przynależność dowolnego obiektu, reprezentowanego przez N -elementowy wektor, do jednej z klas. Równanie hiperpłaszczyzny separującej otrzymujemy przy-

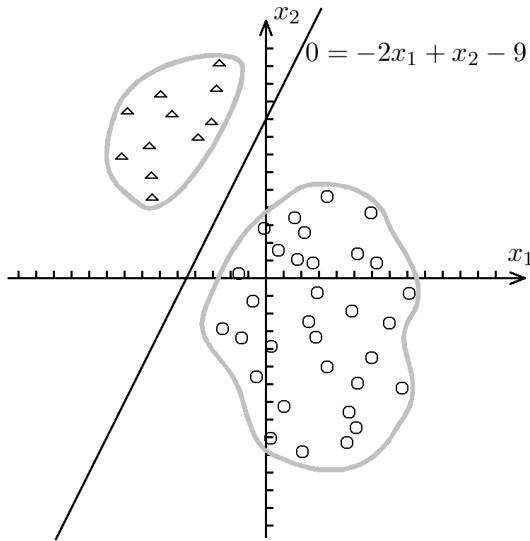
¹Możemy zatem pod tym względem wyróżnić dwa rodzaje neuronów: z biasem i bez biasu

równując wyjście $U(x)$ do zera. W szczególnym przypadku, gdy hiperpłaszczyzna ta jest podprzestrzenią liniową wymiaru $N - 1$, wówczas waga w_0 jest równa zeru, zaś wagi $w_1; \dots; w_N$ stanowią wektor normalny do tej hiperpłaszczyzny (rys. 3.2). Problem klasyfikacyjny tego rodzaju możemy zatem rozwiązać za pomocą neuronu



Rysunek 3.2: Przykład klasyfikacji liniowej w przestrzeni \mathbb{R}^2 ($w_0 = 0$)

bez biasu. W ogólnym przypadku obszar zajmowany przez daną klasę może obejmować również element zerowy $[0; 0; \dots; 0]^T$ przestrzeni wejściowej i wówczas tylko użycie biasu, wyznaczającego przesunięcie hiperpłaszczyzny separującej względem odpowiedniej podprzestrzeni liniowej, umożliwia właściwą klasyfikację (rys. 3.3).



Rysunek 3.3: Przykład klasyfikacji liniowej w przestrzeni \mathbb{R}^2 ($w_0 = -9$)

Poszczególne neurony w obrębie sieci neuronowej organizowane są zazwyczaj

w warstwy, przy czym każdy neuron w warstwie zwykle otrzymuje na wejściu ten sam wektor² \mathbf{x} . Funkcję realizowaną przez pojedynczą warstwę zawierającą K neuronów liniowych możemy zatem przedstawić w postaci mnożenia macierzowego:

$$\mathbf{y} = \mathbf{Ax} ; \quad (3.3)$$

gdzie \mathbf{y} jest teraz K -elementowym wektorem wyjściowym, zaś macierz $\mathbf{A}_{K \times (N+1)}$ zawiera wagi wszystkich neuronów. Oznaczając j -tą wagę i -tego neuronu przez w_{ij} możemy przedstawić zależność (3.3) w postaci:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_K \end{bmatrix} = \begin{bmatrix} w_{10} & w_{11} & \cdots & w_{1N} \\ w_{20} & w_{21} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{K0} & w_{K1} & \cdots & w_{KN} \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_N \end{bmatrix} ; \quad (3.4)$$

gdzie y_i oznacza wyjście i -tego neuronu.

Na podstawie powyższej reprezentacji łatwo zauważyc, że dodanie kolejnej warstwy neuronów liniowych nie zwiększy możliwości sieci w sensie ilości i rodzaju odwzorowań wejście-wyjście możliwych do wyuczenia. Dodatkowa warstwa byłaby równoważna dodatkowej macierzy \mathbf{B} , przez którą mnożony byłby wektor \mathbf{y} . Jednak z uwagi na łączność mnożenia macierzowego wektor wyjściowy \mathbf{z} dwuwarstwowej sieci liniowej możemy przedstawić jako:

$$\mathbf{z} = \mathbf{By} = \mathbf{BAx} = \mathbf{Cx} ; \quad (3.5)$$

gdzie $\mathbf{C} = \mathbf{BA}$, co jest równoważne zależności (3.3). Pomijamy tutaj dla uproszczenia kwestię biasu drugiej warstwy, jednak nie zmienia to faktu, że dwuwarstwowa sieć liniowa jest zasadniczo równoważna sieci jednowarstwowej.

Inaczej sprawia się ma w przypadku nieliniowej funkcji aktywacji. Zazwyczaj przyjmuje się funkcję sigmoidalną, stanowiącą pewną analogię do opisanej wyżej funkcji skokowej. Istotną zaletą funkcji sigmoidalnej (tak samo jak i funkcji liniowej) w stosunku do funkcji skokowej jest jej ciągłość i różniczkowalność w całej dziedzinie. Zauważmy tu, iż ogólnie rzecz biorąc, ciągłość i różniczkowalność funkcji aktywacji umożliwia konstrukcję również ciągłej i różniczkowalnej funkcji celu, co daje szansę na zastosowanie do jej minimalizacji efektywnych algorytmów gradientowych. Ma to szczególne znaczenie w przypadku neuronów i sieci

²Mówimy tu o połączeniu typu *każdy z każdym*, ponieważ każda współrzędna wektora wejściowego jest połączona z każdym neuronem warstwy

nieliniowych³. Najczęściej spotykane rodzaje funkcji sigmoidalnych, to funkcja sigmoidalna unipolarna:

$$f_u(x) = \frac{1}{1 + e^{-\beta x}} ; \quad (3.6)$$

oraz funkcja sigmoidalna bipolarna:

$$f_b(x) = \tanh(-x) : \quad (3.7)$$

Parametr β decyduje o stromości nachylenia krzywej wykresu funkcji, przy czym dla $\beta \rightarrow 0$ otrzymujemy funkcję stałą, zaś dla $\beta \rightarrow \infty$ funkcję skokową. W praktyce najczęściej stosuje się ustaloną wartość parametru $\beta = 1$. Obie funkcje są monotoniczne, rosnące w całej dziedzinie, przy czym:

$$\lim_{x \rightarrow -\infty} f_u(x) = 0 ; \quad (3.8)$$

$$\lim_{x \rightarrow \infty} f_u(x) = 1 ; \quad (3.9)$$

$$\lim_{x \rightarrow -\infty} f_b(x) = -1 ; \quad (3.10)$$

$$\lim_{x \rightarrow \infty} f_b(x) = 1 : \quad (3.11)$$

Istotną cechą obu funkcji jest łatwość obliczenia pochodnych. Istotnie, z zależności:

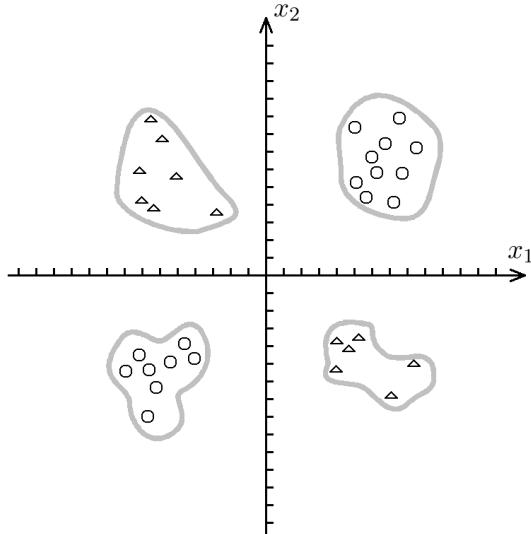
$$\frac{df_u(x)}{dx} = f_u(x)(1 - f_u(x)) ; \quad (3.12)$$

$$\frac{df_b(x)}{dx} = (1 - f_b^2(x)) ; \quad (3.13)$$

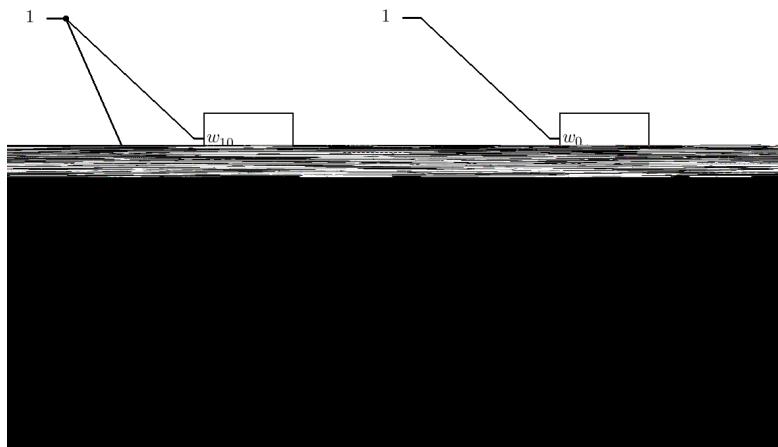
wynika, że pochodna funkcji sigmoidalnej w punkcie x wyraża się łatwo za pomocą wartości samej funkcji w tym punkcie, co pozwala na uproszczenie i skrócenie czasu obliczeń wykonywanych w procesie adaptacji sieci.

Wykorzystanie neuronów nieliniowych (tj. neuronów z nieliniową funkcją aktywacji) znacząco rozszerza zbiór zadań, w tym także problemów klasyfikacyjnych, możliwych do rozwiązania za pomocą sieci. W szczególności, sieć wielowarstwowa zbudowana z neuronów nieliniowych nie jest już równoważna pojedynczej warstwie i posiada nowe, korzystne własności, jak wynika z analizy klasycznego problemu typu XOR (rys. 3.4) [90][94]. Problem klasyfikacyjny XOR odpowiada funkcji logicznej o tej samej nazwie: przypisz do pierwszej klasy te obiekty, dla których tylko jedno z wejść spełnia pewien warunek (w tym wypadku – jest dodatnie). Jeśli oba wejścia spełniają warunek, bądź oba go nie spełniają, wówczas obiekt

³Dla sieci liniowej zwykle można łatwo (choć nie zawsze efektywnie obliczeniowo) określić wartości wag, rozwiązując odpowiedni układ równań liniowych



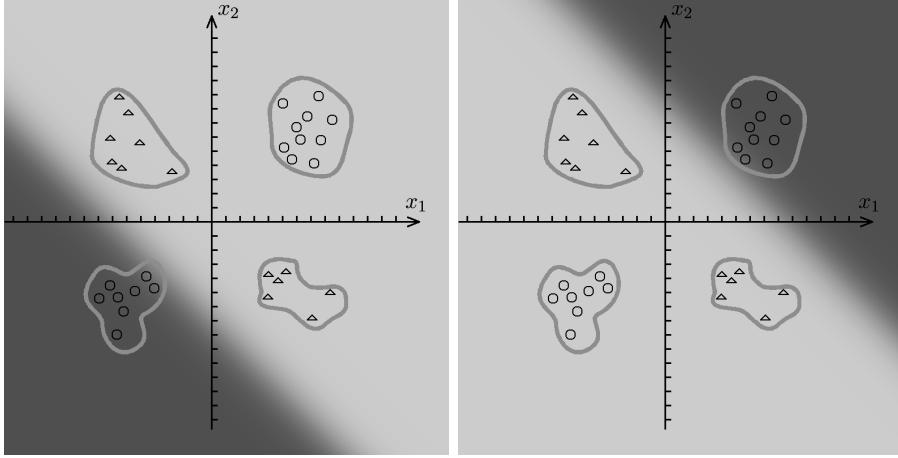
Rysunek 3.4: Problem klasyfikacyjny typu XOR



Rysunek 3.5: Struktura sieci o jednej warstwie ukrytej z dwoma neuronami sigmoidalnymi

należy do klasy drugiej.

Jak łatwo zauważać problem ten nie jest rozwiązywalny za pomocą pojedynczej hiperpłaszczyzny separującej i klasyfikator liniowy nie będzie tu w stanie dokonać właściwej klasyfikacji. Wystarczy jednak zastosować dwuwarstwową sieć z dwoma neuronami sigmoidalnymi w pierwszej warstwie i pojedynczym neuronem w warstwie drugiej (rys. 3.5). Jest to najprostszy przykład wielowarstwowego perceptronu, czyli sieci typu MLP. Jego pierwszą warstwę nazywamy *warstwą ukrytą* (ogólnie: wszystkie warstwy, których wyjścia nie są bezpośrednio „widoczne” na wyjściu całej sieci określa się jako ukryte), warstwa druga zaś, to warstwa wyjściowa. Domyślnie przyjmuje się zazwyczaj schemat połączeń „każdy z każdym”, czyli każde wyjście warstwy poprzedniej jest połączone ze wszystkimi wejściami



Rysunek 3.6: Problem XOR – kolorem tła zostało zaznaczone wyjście odpowiednio pierwszego i drugiego neuronu warstwy ukrytej

warstwy następnej. Przez „neuron sigmoidalny” rozumiemy neuron z sigmoidalną funkcją aktywacji, przy czym rodzaj sigmoidy nie ma w tym wypadku istotnego znaczenia. Podobnie jest w przypadku warstwy wyjściowej, w której neurony mogą być nawet liniowe. Rodzaj funkcji aktywacji warstwy wyjściowej ma większe znaczenie w innych typach zadań, np. w aproksymacji, gdzie musimy uwzględnić fakt, że sigmoida — w przeciwnieństwie do funkcji liniowej — jest ograniczona. Dla ustalenia uwagi założymy, że w omawianym przypadku oba neurony warstwy ukrytej oraz neuron w warstwie wyjściowej mają bipolarną funkcję aktywacji.

Przykładowe wartości wag neuronów warstwy ukrytej są następujące:

$$\begin{bmatrix} w_{10} & w_{11} & w_{12} \\ w_{20} & w_{21} & w_{22} \end{bmatrix} = \begin{bmatrix} 6 & 1 & 1 \\ 6 & -1 & -1 \end{bmatrix} : \quad (3.14)$$

Na rys. 3.6 wartości wyjściowe obu neuronów ukrytych z przedziału $[-1; 1]$ przedstawiono za pomocą odcieni szarości, przy czym kolor ciemnoszary odpowiada wartośćm ujemnym, a jasnoszary dodatnim. Przyjmując wagi neuronu ostatniej warstwy: $[w_0 \ w_1 \ w_2]^T = [-1; 1; 1]^T$ łatwo zauważyc, że jego wartość wyjściowa będzie dodatnia tylko dla pewnego pasa w przestrzeni \mathbb{R}^2 symetrycznego względem prostej $x_2 = -x_1$ (i zawierającego tą prostą), co umożliwia bezpośrednie zdefiniowanie odpowiedniej funkcji kryterialnej.

Przedstawione powyżej przykłady pokazują możliwość dokonania klasyfikacji przez sieć neuronową o danej architekturze, w sensie istnienia stanu sieci zapewniającego właściwą dyskryminację klas. W następnym podrozdziale rozważymy w jaki sposób sieć może ten stan osiągnąć, czyli przedstawimy podstawy algorytmów adaptacji wag w odniesieniu do sieci typu MLP.

3.2 Adaptacja

Jedną z interesujących cech sieci neuronowej jest uczenie się na podstawie dostarczonych danych, bez konieczności jawnej specyfikacji występujących w nich zależności i reguł. Z tego względu sieć neuronowa może być postrzegana jako czarna skrzynka, która generuje odpowiednie rezultaty w oparciu o określone informacje wejściowe, przy czym nie wiemy zazwyczaj *dla czego* tak się dzieje, tzn. jaka wewnętrzna reprezentacja odpowiada za dane zachowanie sieci. Jakkolwiek możliwe jest również zastosowanie sieci neuronowych do uzyskania informacji o charakterze symbolicznym, w postaci regułowej [7][3], jednak najczęściej interesuje nas po prostu otrzymanie właściwej odpowiedzi na zadany sygnał wejściowy.

Podstawowe rozróżnienie sposobów adaptacji sieci dotyczy tego, czy dysponujemy jawną postacią pożądanego wektora wyjściowego \mathbf{d} . Jeśli tak, mówimy wówczas o treningu z *nauczycielem*, podczas którego prezentowane są pary wektorów $(\mathbf{x}^{(m)}; \mathbf{d}^{(m)})$, $m = 1; 2; \dots; M$, gdzie M jest liczbą przykładów uczących. Naszym celem jest minimalizacja rozbieżności między faktyczną odpowiedzią sieci $\mathbf{y}^{(m)}$ na wektor wejściowy $\mathbf{x}^{(m)}$, a zadanym wektorem docelowym $\mathbf{d}^{(m)}$. Jest to typowa sytuacja dla zadań klasyfikacji i rozpoznawania wzorców, kiedy dysponujemy informacją o faktycznej przynależności rozpoznawanych obiektów do klas. W przeciwnym wypadku, jeśli nie znamy z góry właściwej odpowiedzi, mówimy o treningu „bez nauczyciela”, którego celem może być np. analiza klastrowa, umożliwiająca wyodrębnienie grup (skupień) wśród danych wejściowych.

W przypadku klasyfikacji możemy uogólnić przykłady z poprzedniego podrozdziału na większą liczbę klas: przyjmuje się w tym celu zazwyczaj ilość wyjść sieci K równą ilości klas, a docelowa wartość k -tego wyjścia jest równa:

$$d_k^{(m)} = \begin{cases} h & , \text{ jeśli wzorzec } x^{(m)} \text{ należy do } k\text{-tej klasie;} \\ l & , \text{ w przeciwnym wypadku;} \end{cases} \quad (3.15)$$

przy czym za wartości h i l można w najprostszym przypadku przyjąć 1 i 0, odpowiednio. W przypadku neuronów wyjściowych z funkcją aktywacji w postaci unipolarnej sigmoidy korzystne jest przyjęcie nieco innych wartości, np. 0.75 i 0.25, z uwagi na to, że liczby 0 i 1 leżą na krańcach przedziału możliwych wartości wyjściowych⁴. Próba ich osiągnięcia w procesie adaptacji może więc prowadzić do „wysycenia” neuronów, tj. do ustalenia się bardzo dużych wartości wag, przy których pochodna funkcji aktywacji jest bliska zeru, co znacząco spowalnia naukę.

Uwzględniając (3.15) ostateczna decyzja o przynależności m -tego wzorca do

⁴Analogiczna sytuacja zachodzi w przypadku sigmoidy bipolarnej dla wartości θ_h i θ_l równych 1 i -1, odpowiednio

klasy k_0 jest podejmowana jeśli:

$$k_0 = \arg \max_k \{y_k^{(m)}\} : \quad (3.16)$$

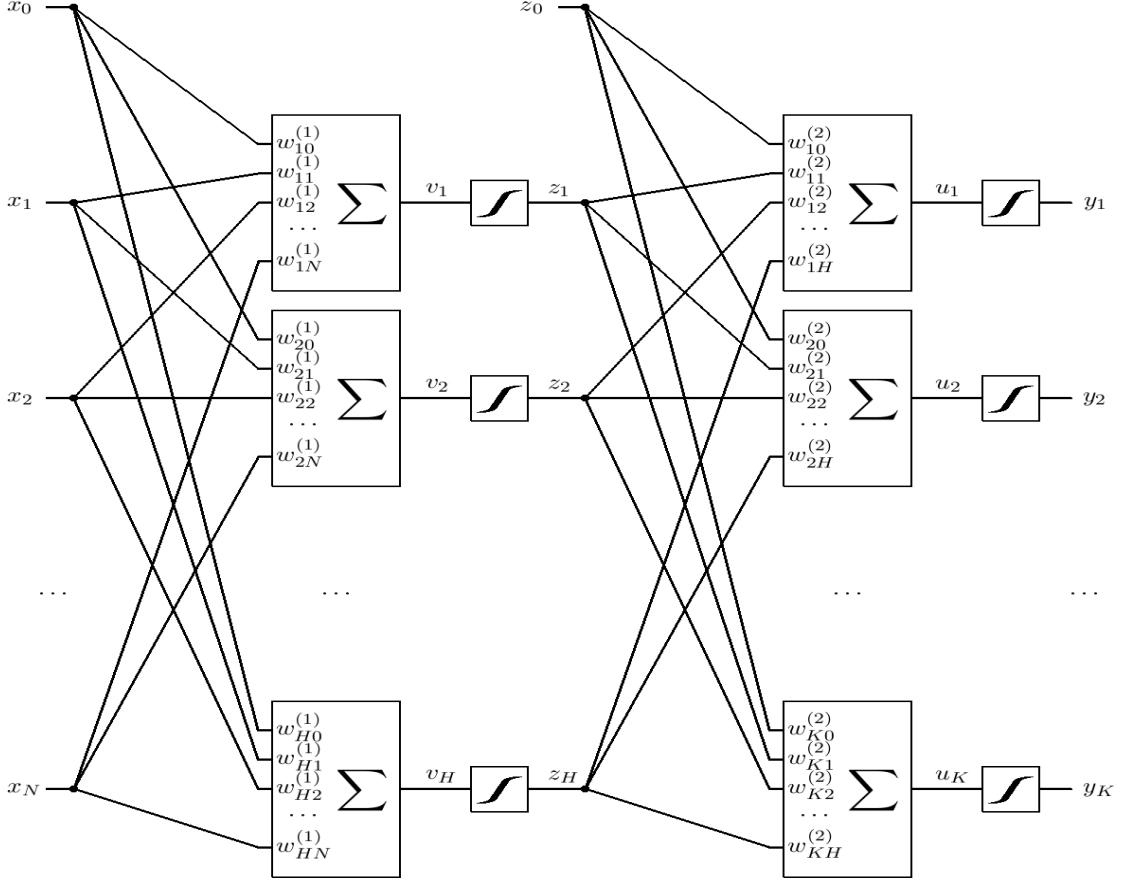
Metody uczenia sieci możemy też podzielić ze względu na tryb adaptacji wag: po prezentacji każdego przykładu uczącego, bądź jednorazowo, po prezentacji wszystkich dostępnych przykładów. W pierwszym przypadku mówimy o treningu typu *on-line*, w drugim zaś o treningu typu *off-line*. Zgodnie z przyjętą w literaturze nomenklaturą zbiór wszystkich przykładów uczących określimy jako zbiór uczący lub treningowy, a prezentację wszystkich przykładów ze zbioru treningowego nazwiemy epoką. A zatem możemy powiedzieć, że adaptacja wag w trybie *off-line* zachodzi jednokrotnie, a w trybie *on-line* M -krotnie w ciągu całej epoki. Ilość epok koniecznych do uzyskania zadowalających wyników procesu uczenia zwykle waha się w granicach od kilku do kilkuset, w zależności od zadania, dostępnego zbioru treningowego i efektywności zastosowanych metod adaptacji. Warto zauważać, że w przypadku nauki *on-line* kolejność prezentacji przykładów powinna być różna w różnych epokach (w praktyce kolejność przykładów jest zwykle losowana w każdej epoce).

Należy podkreślić, że szczególne uwarunkowania i celu adaptacji istotnie zależą od rodzaju sieci i jej przeznaczenia. Zazwyczaj problem postawiony do rozwiązania umożliwia konstrukcję pewnej funkcji celu, która następnie jest minimalizowana w procesie adaptacji sieci. Sam algorytm adaptacji jest już uzależniony od typu neuronów i architektury połączeń pomiędzy nimi. Spośród klasycznych rozwiązań opracowanych dla różnych rodzajów sieci można tu wymienić m.in.: regułę perceptronu, regułę Widrowa-Hoffa, regułę Hebb'a, regułę Grossberga, regułę Oji, regułę Kohonena, czy regułę Kosko.

Jednym z najistotniejszych algorytmów adaptacji sieci neuronowej jest algorytm wstecznej propagacji błędu umożliwiający skuteczne uczenie wielowarstwowych sieci jednokierunkowych typu MLP w oparciu o metody gradientowe. Omówimy go w tym miejscu z uwagi na to, że podobna metodologia została zastosowana do opracowania algorytmu uczenia szybkich ortogonalnych sieci neuronowych.

3.2.1 Algorytm wstecznej propagacji błędu

Przyjmijmy następujące oznaczenia (rys. 3.7): niech N , K oznaczają odpowiednio liczbę wejść i wyjść sieci, H jest liczbą neuronów ukrytych, zaś M liczbą wzorców uczących. Funkcję celu stanowić będzie błąd średniokwadratowy zdefiniowany dla



Rysunek 3.7: Ogólny schemat sieci typu MLP

m -tego wzorca jako:

$$E^{(m)}(\mathbf{w}) = \frac{1}{K} \sum_{k=1}^K E_k^{(m)}(\mathbf{w}) ; \quad (3.17)$$

gdzie

$$E_k^{(m)}(\mathbf{w}) = \frac{1}{2} \left(y_k^{(m)} - d_k^{(m)} \right)^2 : \quad (3.18)$$

W przypadku nauki *off-line* funkcja celu będzie średnią z funkcji błędu dla poszczególnych wzorców:

$$E(\mathbf{w}) = \frac{1}{M} \sum_{m=1}^M E^{(m)}(\mathbf{w}) = \frac{1}{MK} \sum_{m=1}^M \sum_{k=1}^K \frac{1}{2} \left(y_k^{(m)} - d_k^{(m)} \right)^2 : \quad (3.19)$$

Czynnik $1=(MK)$ nie zmienia istoty wyprowadzenia wzorów na propagację wsteczną błędu i w literaturze bywa pomijany. W praktyce jednak warto go uwzględnić, aby uzyskiwane wartości błędu były porównywalne niezależnie od ilości wzorców uczących i od liczby wyjść sieci.

Oznaczmy i -te wyjście warstwy ukrytej uzyskane dla m -tego wzorca uczącego

jako $Z_i^{(m)}$, gdzie:

$$Z_i^{(m)} = f \left(\sum_{n=0}^N w_{in}^{(1)} x_n^{(m)} \right) ; \quad (3.20)$$

a k -te wyjście warstwy wyjściowej uzyskane dla m -tego wzorca uczącego jako $y_k^{(m)}$, gdzie:

$$y_k^{(m)} = f \left(\sum_{i=0}^H w_{ki}^{(2)} Z_i^{(m)} \right) = f \left(\sum_{i=0}^H w_{ki}^{(2)} f \left(\sum_{n=0}^N w_{in}^{(1)} x_n^{(m)} \right) \right) ; \quad (3.21)$$

Przyjmujemy tu konwencję, że indeks górnny w nawiasie oznacza w przypadku wag: numer warstwy, natomiast w przypadku symboli oznaczających sygnały: numer wzorca uczącego. Jako funkcję aktywacji f przyjmuje się zazwyczaj funkcję sigmoidalną, chociaż w ogólnym przypadku może to być dowolna funkcja ciągła i różniczkowalna.

Rozważmy teraz pochodną funkcji (3.19) względem j -tej wagi l -tego neuronu warstwy wyjściowej $w_{lj}^{(2)}$:

$$\frac{\partial E}{\partial w_{lj}^{(2)}} = \frac{1}{M} \sum_{m=1}^M \frac{\partial E^{(m)}}{\partial w_{lj}^{(2)}} = \frac{1}{MK} \sum_{m=1}^M \frac{\partial E_l^{(m)}}{\partial w_{lj}^{(2)}} ; \quad (3.22)$$

Pochodną funkcji błędu l -tego neuronu wyjściowego dla m -tego wzorca możemy natomiast przedstawić następująco:

$$\frac{\partial E_l^{(m)}}{\partial w_{lj}^{(2)}} = \left(y_l^{(m)} - d_l^{(m)} \right) \frac{\partial y_l^{(m)}}{\partial w_{lj}^{(2)}} = \left(y_l^{(m)} - d_l^{(m)} \right) \frac{\partial f(u_l^{(m)})}{\partial u_l^{(m)}} Z_j^{(m)} ; \quad (3.23)$$

A zatem poszukiwana pochodna względem wagi związanej z j -tym wejściem wyraża się iloczynem sygnału podanego na to wejście, $Z_j^{(m)}$ oraz pewnego czynnika zależnego od wyjścia neuronu (aktualnego i żądanego). Czynnik ten zdefiniujemy jako:

$$l^{(2)(m)} = \frac{\partial f(u_l^{(m)})}{\partial u_l^{(m)}} \left(y_l^{(m)} - d_l^{(m)} \right) ; \quad (3.24)$$

gdzie dwójka w indeksie górnym odnosi się do numeru warstwy. Zauważmy, że tak określony czynnik jest iloczynem pochodnej funkcji aktywacji neuronu i sygnału różnicowego $y - d$. Uwzględniając (3.24) i (3.23) w (3.22) mamy:

$$\frac{\partial E}{\partial w_{lj}^{(2)}} = \frac{1}{MK} \sum_{m=1}^M l^{(2)(m)} Z_j^{(m)} ; \quad (3.25)$$

Analogicznie możemy obliczyć pochodną względem n -tej wagi j -tego neuronu warstwy ukrytej:

$$\frac{\partial E}{\partial W_{jn}^{(1)}} = \frac{1}{M} \sum_{m=1}^M \frac{\partial E^{(m)}}{\partial W_{jn}^{(1)}}. \quad (3.26)$$

Zauważmy jednak, że pochodna ta zależy tym razem od wyjść *wszystkich* neuronów ostatniej warstwy:

$$\frac{\partial E^{(m)}}{\partial W_{jn}^{(1)}} = \frac{1}{K} \sum_{k=1}^K \frac{\partial E_k^{(m)}}{\partial W_{jn}^{(1)}} = \frac{1}{K} \sum_{k=1}^K \left(y_k^{(m)} - d_k^{(m)} \right) \frac{\partial y_k^{(m)}}{\partial W_{jn}^{(1)}} : \quad (3.27)$$

Jej składnik związany z pojedynczym, k -tym neuronem wyjściowym wyrazimy jako:

$$\frac{\partial y_k^{(m)}}{\partial W_{jn}^{(1)}} = \frac{\partial f(u_k^{(m)})}{\partial u_k^{(m)}} W_{kj}^{(2)} \frac{\partial z_j^{(m)}}{\partial W_{jn}^{(1)}} = \frac{\partial f(u_k^{(m)})}{\partial u_k^{(m)}} W_{kj}^{(2)} \frac{\partial f(v_j^{(m)})}{\partial v_j^{(m)}} x_n^{(m)} : \quad (3.28)$$

Wstawiając (3.28) do (3.27) otrzymujemy:

$$\frac{\partial E^{(m)}}{\partial W_{jn}^{(1)}} = \frac{1}{K} \sum_{k=1}^K \left(y_k^{(m)} - d_k^{(m)} \right) \frac{\partial f(u_k^{(m)})}{\partial u_k^{(m)}} W_{kj}^{(2)} \frac{\partial f(v_j^{(m)})}{\partial v_j^{(m)}} x_n^{(m)} : \quad (3.29)$$

Zauważmy ponownie, że uzyskana pochodna względem n -tego wejścia neuronu ukrytego jest iloczynem tego wejścia i pewnego czynnika :

$$\frac{\partial E}{\partial W_{jn}^{(1)}} = \frac{1}{MK} \sum_{m=1}^M \sum_{j=1}^K \frac{\partial E^{(m)}}{\partial W_{jn}^{(1)}} x_n^{(m)} ; \quad (3.30)$$

gdzie:

$$\sum_{j=1}^K \frac{\partial E^{(m)}}{\partial W_{jn}^{(1)}} = \frac{\partial f(v_j^{(m)})}{\partial v_j^{(m)}} \sum_{k=1}^K W_{kj}^{(2)} : \quad (3.31)$$

Tak określona delta jest znów iloczynem pochodnej funkcji aktywacji rozważanego neuronu i czynnika, który można określić jako *uogólniony sygnał różnicowy*, wyrażonego sumą delt neuronów następnej warstwy wymnożonych przez wagi łączące te neurony z neuronem rozważanym.

Uogólniając powyższe rozumowanie możemy obliczyć wartości wektora gradientu $\mathbf{g} = \nabla_{\mathbf{w}} E$ dla sieci o dowolnej liczbie warstw. Uzyskany algorytm nosi nazwę algorytmu propagacji wstecznej z uwagi na kolejność obliczania czynników – od ostatniej warstwy neuronowej do pierwszej. Uwzględniając fakt, iż do obliczenia wartości dla bieżącej warstwy wykorzystujemy już obliczone wartości

delt z warstwy następnej, możemy mówić o propagacji sygnału δ , określonego też jako sygnał błędu⁵, wstecz od warstwy ostatniej począwszy. Jest to analogia do propagacji sygnału wejściowego x w przód, od warstwy pierwszej do ostatniej.

Obie te fazy występują naprzemiennie podczas uczenia sieci. Najpierw sygnał wejściowy propagowany jest w przód, dzięki czemu otrzymujemy wartości wyjściowe ostatniej warstwy oraz warstw ukrytych, a następnie na tej podstawie obliczany jest sygnał błędu dla ostatniej warstwy, który jest propagowany wstecz, umożliwiając zarazem obliczenie składników wektora gradientu. Wykorzystanie tak obliczonego gradientu do adaptacji wag sieci zamyka cykl, po czym ponownie następuje faza propagacji w przód z już zmodyfikowanymi wartościami wag.

Wektor gradientu zapewnia informację o kierunku najszybszego wzrostu funkcji celu w wielowymiarowej przestrzeni wag sieci, a wektor przeciwny do niego — o kierunku najszybszego spadku jej wartości. Zgodnie z podstawowym algorytmem wykorzystującym tę informację czyli algorymem największego spadku, mając w i -tym kroku dany wektor wag w_i obliczamy wartości wag w kroku $i + 1$ jako:

$$w_{i+1} = w_i + p ; \quad (3.32)$$

gdzie $p = -\nabla_w E$. Współczynnik p jest tzw. *krokiem* algorytmu i powinien być tak dobrany, aby spełnić warunek:

$$E(w_i + p) < E(w_i) : \quad (3.33)$$

Należy zauważyć, że algorytm najszybszego spadku nie stanowi najbardziej efektywnej metody adaptacji sieci neuronowych. Szybszą zbieżność można zapewnić stosując metodę gradientów sprzężonych, metody oparte o wyższe pochodne funkcji błędu, takie jak algorytm zmiennej metryki, czy algorytm Levenberga-Marquardta lub metody heurystyczne (Quickprop, RPROP) [94]. Metody hesjańskie, w szczególności algorytm Levenberga-Marquardta, uchodzą za najbardziej efektywne w większości zastosowań. Ich wykorzystanie może jednak nastąpić pewnych trudności w przypadku bardzo dużych rozmiarów sieci (rzędu kilku tysięcy wag). Mając na uwadze potencjalne zastosowania sieci w analizie surowych danych multimedialnych, narzucających duży rozmiar wektora wejściowego i — w konsekwencji — również wektora wag, autor zdecydował się na implementację algorytmu gradientów sprzężonych charakteryzującego się lepszą skalowalnością pod tym względem [94].

⁵czego nie należy mylić z funkcją błędu, odpowiadającą zdefiniowanej wcześniej funkcji celu

3.2.2 Algorytm gradientów sprzężonych

Metoda gradientów sprzężonych została zaproponowana w 1952 przez Hestenesa i Stiefela [51] do iteracyjnego rozwiązywania układów równań liniowych. Podobnie jak inne metody iteracyjne jest ona szczególnie przydatna dla układów określonych przez rzadką macierz współczynników o dużym rozmiarze, przy czym podstawowy wariant algorytmu gradientów sprzężonych wymaga, aby macierz ta była symetryczna i dodatnio-określona.

A zatem, dla układu równań:

$$\mathbf{A}\mathbf{w} = \mathbf{b} ; \quad (3.34)$$

gdzie wektor \mathbf{w} jest poszukiwanym rozwiązaniem, macierz $\mathbf{A} = \mathbf{A}_{N \times N}$ musi spełniać warunki:

$$\mathbf{A}^T = \mathbf{A} ; \quad (\text{warunek symetrii}) \quad (3.35)$$

$$\forall_{\mathbf{w} \neq \mathbf{0}} \mathbf{w}^T \mathbf{A} \mathbf{w} > 0 ; \quad (\text{warunek dodatniej określoności}) \quad (3.36)$$

Rozwiązania $\hat{\mathbf{w}}$ poszukuje się poprzez minimalizację formy kwadratowej:

$$f(\mathbf{w}) = \frac{1}{2} \mathbf{w}^T \mathbf{A} \mathbf{w} - \mathbf{b}^T \mathbf{w} ; \quad (3.37)$$

Jak łatwo pokazać, gradient funkcjonału f wyraża się jako:

$$f'(\mathbf{w}) = \frac{1}{2} \mathbf{A}^T \mathbf{w} + \frac{1}{2} \mathbf{A} \mathbf{w} - \mathbf{b} ; \quad (3.38)$$

a zatem wobec symetrii macierzy \mathbf{A} mamy:

$$f'(\mathbf{w}) = \mathbf{A} \mathbf{w} - \mathbf{b} ; \quad (3.39)$$

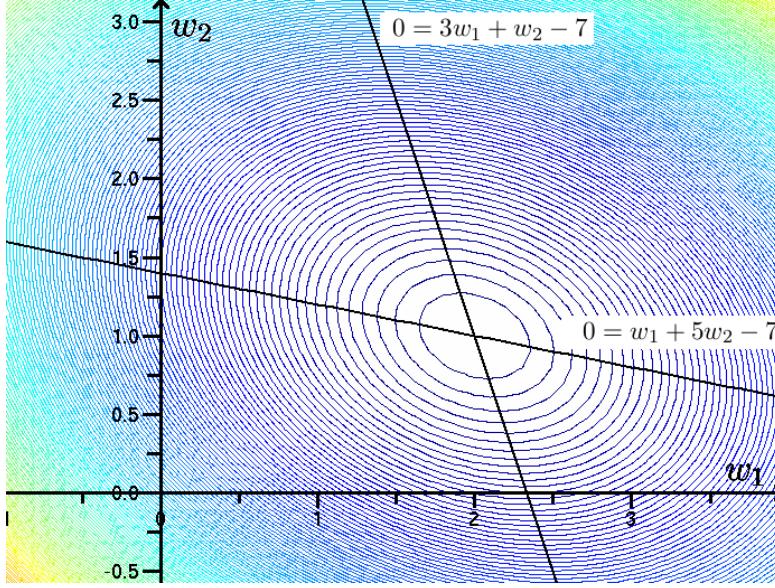
Przyrównując pochodną $f'(\mathbf{w})$ do zera otrzymujemy rozwiązywany układ.

Przykładowo, rozważmy układ zdefiniowany w \mathbb{R}^2 równaniem (3.34) dla danych:

$$\mathbf{A} = \begin{bmatrix} 3 & 1 \\ 1 & 5 \end{bmatrix} ; \quad \mathbf{b} = \begin{bmatrix} 7 \\ 7 \end{bmatrix} ; \quad (3.40)$$

Rozwiązaniem tego układu jest punkt $\hat{\mathbf{w}} = [2; 1]^T$ przecięcia prostych o współczynnikach określonych przez macierz \mathbf{A} i wektor \mathbf{b} . Jak widać na rys. 3.8 punkt ten stanowi zarazem minimum funkcji (3.37) zdefiniowanej dla wartości (3.40).

Minimalizacja funkcji (3.37) jest procesem iteracyjnym, a więc w każdym i -tym



Rysunek 3.8: Graficzna reprezentacja układu równań i odpowiadającej mu formy kwadratowej w \mathbb{R}^2

kroku mając dane przybliżone rozwiązanie \mathbf{w}_i określamy następne przybliżenie:

$$\mathbf{w}_{i+1} = \mathbf{w}_i + \alpha_i \mathbf{p}_i ; \quad (3.41)$$

przy czym kierunek \mathbf{p}_i nie jest tym razem wektorem przeciwnym do wektora gradientu \mathbf{g} funkcji f w punkcie \mathbf{w}_i , jak było to w metodzie najszybszego spadku. Należy bowiem zauważyć, że nawet w przypadku optymalnego wyboru skalarnego współczynnika α_i minimalizującego wartość f w kierunku \mathbf{p}_i metoda najszybszego spadku często jest mało efektywna dla macierzy \mathbf{A} o mocno zróżnicowanych wartościach własnych, generujących funkcję f w postaci „długiej i wąskiej doliny”. Dzieje się tak dlatego, że — w rozważanym przypadku — nowy wektor gradientu jest ortogonalny do poprzedniego, dając w efekcie „zygzakową” ścieżkę poszukiwań złożoną z wektorów tworzących zbiór zależny liniowo.

W metodzie gradientów sprzężonych wymaga się, aby nowy kierunek poszukiwań był **\mathbf{A} -ortogonalny**, inaczej: sprzężony⁶ względem macierzy \mathbf{A} , do wszystkich poprzednich kierunków poszukiwań. Dwa wektory $\mathbf{p}_1, \mathbf{p}_2$ określa się jako sprzężone względem macierzy \mathbf{A} jeśli:

$$\mathbf{p}_1^T \mathbf{A} \mathbf{p}_2 = 0 : \quad (3.42)$$

Jak łatwo zauważać, dla symetrycznej i dodatnio-określonej macierzy \mathbf{A} lewa strona równania (3.42) definiuje iloczyn skalarny, równy zwykłemu, tj. euklidesowemu iloczynowi skalarnemu dla macierzy jednostkowej. Pojęcie **\mathbf{A} -ortogonalności**

⁶To określenie nie ma nic wspólnego ze liczbami zespolonymi sprzężonymi

jest rozszerzeniem definicji wektorów ortogonalnych, czyli spełniających warunek:

$$\langle \mathbf{p}_1, \mathbf{p}_2 \rangle = 0 ; \quad (3.43)$$

na przypadek iloczynu skalarnego zdefiniowanego przez macierz \mathbf{A} :

$$\langle \mathbf{p}_1, \mathbf{p}_2 \rangle_{\mathbf{A}} = \mathbf{p}_1^T \mathbf{A} \mathbf{p}_2 ; \quad (3.44)$$

Konstrukcja ciągu \mathbf{A} -ortogonalnych — w miejsce ortogonalnych — kierunków poszukiwań umożliwia obliczenie wartości współczynnika i , która jest optymalna w sensie globalnym, tzn. zapewnia jednorazowe przesunięcie w kierunku \mathbf{p}_i o ostateczną wartość. Odpowiada to zastosowaniu kierunków ortogonalnych zgodnie z regułą najszybszego spadku w szczególnym przypadku macierzy \mathbf{A} , której wszystkie wartości własne są takie same⁷. Ponieważ zbiór N wektorów sprzężonych tworzy bazę przestrzeni \mathbb{R}^N , zatem liczba kroków potrzebnych do znalezienia minimum funkcji (3.37) jest równa wymiarowi przestrzeni (rys. 3.9).

Konstrukcję ciągu sprzężonych wektorów kierunkowych $\{\mathbf{p}_i\}$ rozpoczyna się od wektora gradientu $\mathbf{p}_1 = \mathbf{g}_0$ w punkcie startowym \mathbf{w}_0 , tak samo jak w metodzie najszybszego spadku. Nie zmniejszając ogólności rozumowania można przyjąć $\mathbf{w}_0 = 0$ (w innym wypadku wystarczy rozważyć układ $\mathbf{Aw} = \mathbf{c}$, gdzie $\mathbf{c} = \mathbf{b} - \mathbf{Aw}_0$). Do określenia kolejnego kierunku (i wszystkich następnych) oblicza się również wektor gradientu w kolejnym punkcie, ale modyfikuje się go tak, aby uzyskać wektor sprzężony do wszystkich poprzednich kierunków. Wymagana modyfikacja ma postać:

$$\mathbf{p}_{k+1} = \mathbf{g}_k - \mathbf{u}_k ; \quad (3.45)$$

gdzie

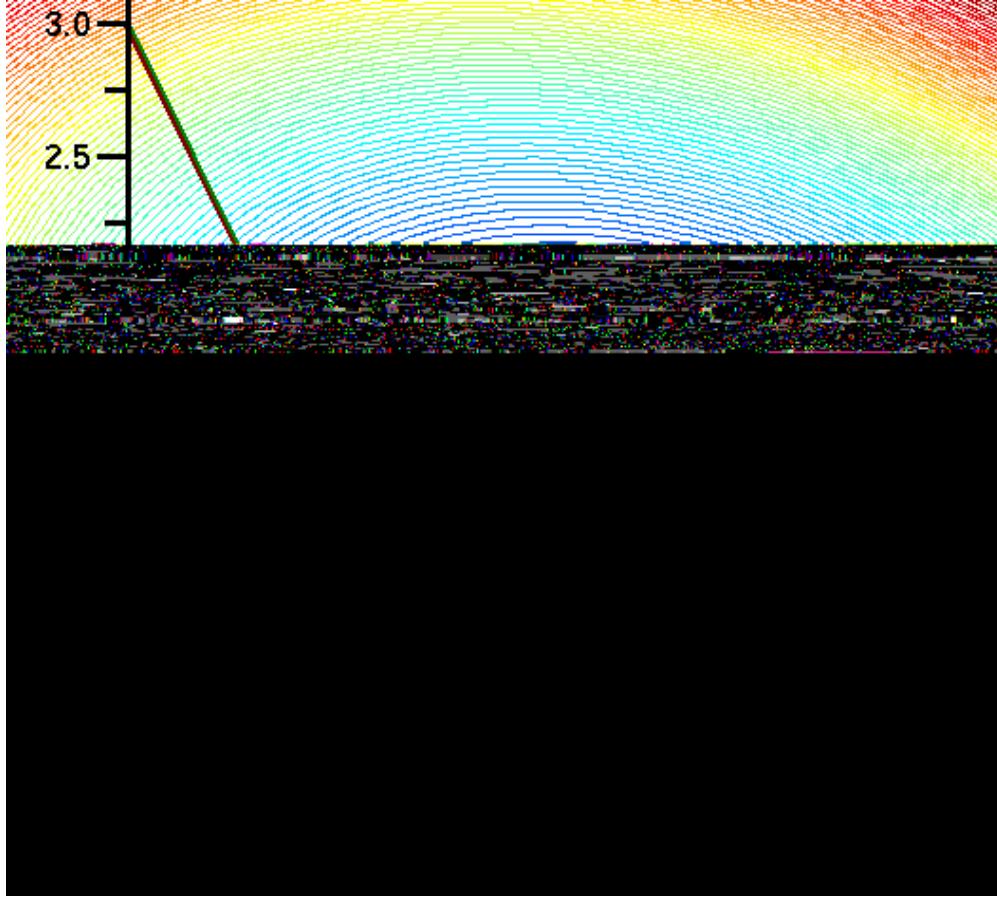
$$\mathbf{u}_k = \sum_{i=1}^k \frac{\langle \mathbf{p}_i, \mathbf{g}_k \rangle_{\mathbf{A}}}{\|\mathbf{p}_i\|_{\mathbf{A}}^2} \mathbf{p}_i = \sum_{i=1}^k \frac{\langle \mathbf{p}_i, \mathbf{g}_k \rangle_{\mathbf{A}}}{\|\mathbf{p}_i\|_{\mathbf{A}}^2} \mathbf{p}_i ; \quad (3.46)$$

Zauważmy, że norma $\|\mathbf{p}_i\|_{\mathbf{A}}$ jest tu poprawnie zdefiniowana dla dodatnio-określonej symetrycznej macierzy \mathbf{A} jako $\sqrt{\langle \mathbf{p}_i, \mathbf{p}_i \rangle_{\mathbf{A}}}$.

Wektorowi \mathbf{u}_k możemy łatwo nadać interpretację przez analogię do zwykłego iloczynu skalarnego $\langle \mathbf{p}_i, \mathbf{g}_k \rangle$, rozumianego jako miara długości rzutu wektora \mathbf{g}_k na wektor \mathbf{p}_i . Odjęcie wektora o tak określonej długości i kierunku równym kierunkowi \mathbf{p}_i powoduje ortogonalizację wektora \mathbf{p}_{k+1} , a w przypadku iloczynu skalarnego $\langle \cdot, \cdot \rangle_{\mathbf{A}}$ — jego \mathbf{A} -ortogonalizację (rys. 3.10).

Pokażemy teraz w sposób formalny, że procedura zdefiniowana wyrażeniami (3.45),

⁷Przykład z rys. 3.8 można sprowadzić do takiego przypadku skalując dziedzinę tak, aby elipsy z wykresu funkcji f stały się okręgami



Rysunek 3.9: Porównanie kierunków poszukiwania minimum dla metody najszybszego spadku (kolor zielony) i dla metody gradientów sprzężonych (kolor brązowy)

(3.46) istotnie prowadzi do konstrukcji zbioru wektorów sprzężonych, czyli, że

$$\forall_{i \neq j} \mathbf{p}_j^T \mathbf{A} \mathbf{p}_i = 0 ; \quad (3.47)$$

dla dowolnych $i, j \in \{1, \dots, N\}$. Równoważnie:

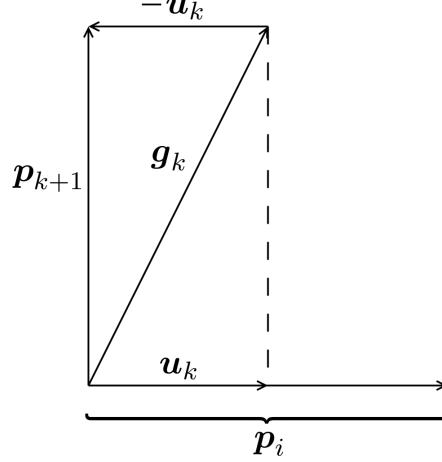
$$\forall_{j \leq k} \mathbf{p}_j^T \mathbf{A} \mathbf{p}_{k+1} = 0 ; \quad (3.48)$$

dla dowolnego $k \in \{1, \dots, N-1\}$.

Dla $k = 1$ wektor \mathbf{p}_{k+1} jest równy:

$$\mathbf{p}_2 = \mathbf{g}_1 - \frac{\mathbf{p}_1^T \mathbf{A} \mathbf{g}_1}{\mathbf{p}_1^T \mathbf{A} \mathbf{p}_1} \mathbf{p}_1 . \quad (3.49)$$

W tym wypadku warunek (3.48) wymaga sprawdzenia tylko dla $j = 1$, dla którego



Rysunek 3.10: Otrzymywanie wektora sprzężonego

zachodzi:

$$\mathbf{p}_1^T \mathbf{A} \mathbf{p}_2 = \mathbf{p}_1^T \mathbf{A} \left(\mathbf{g}_1 - \frac{\mathbf{p}_1^T \mathbf{A} \mathbf{g}_1}{\mathbf{p}_1^T \mathbf{A} \mathbf{p}_1} \mathbf{p}_1 \right) = \mathbf{p}_1^T \mathbf{A} \mathbf{g}_1 - \frac{\mathbf{p}_1^T \mathbf{A} \mathbf{g}_1}{\mathbf{p}_1^T \mathbf{A} \mathbf{p}_1} \mathbf{p}_1^T \mathbf{A} \mathbf{p}_1 = 0 : \quad (3.50)$$

Założymy teraz indukcyjnie, że warunek (3.48) jest spełniony dla każdego k , gdzie $k \in \{1, \dots, N-1\}$ i rozważmy dowolne $j \leq k$. Podstawiając (3.45), (3.46) w (3.48) otrzymujemy:

$$\mathbf{p}_j^T \mathbf{A} \mathbf{p}_{k+1} = \mathbf{p}_j^T \mathbf{A} \left(\mathbf{g}_k - \sum_{i=1}^k \frac{\mathbf{p}_i^T \mathbf{A} \mathbf{g}_k}{\mathbf{p}_i^T \mathbf{A} \mathbf{p}_i} \mathbf{p}_i \right) = \mathbf{p}_j^T \mathbf{A} \mathbf{g}_k - \sum_{i=1}^k \frac{\mathbf{p}_i^T \mathbf{A} \mathbf{g}_k}{\mathbf{p}_i^T \mathbf{A} \mathbf{p}_i} \mathbf{p}_j^T \mathbf{A} \mathbf{p}_i : \quad (3.51)$$

Zauważmy, że z założenia indukcyjnego wynika, iż wyrażenie $\mathbf{p}_j^T \mathbf{A} \mathbf{p}_i$ ma wartość różną od zera tylko dla $i = j$. Mamy zatem:

$$\mathbf{p}_j^T \mathbf{A} \mathbf{p}_{k+1} = \mathbf{p}_j^T \mathbf{A} \mathbf{g}_k - \frac{\mathbf{p}_j^T \mathbf{A} \mathbf{g}_k}{\mathbf{p}_j^T \mathbf{A} \mathbf{p}_j} \mathbf{p}_j^T \mathbf{A} \mathbf{p}_j = 0 ; \quad (3.52)$$

co kończy dowód.

Ponieważ tak skonstruowany zbiór N wektorów sprzężonych $\{\mathbf{p}_i\}$, jest liniowo niezależny, a więc tworzy bazę przestrzeni \mathbb{R}^2 . Rozwiążanie $\hat{\mathbf{w}}$ układu (3.34) można zatem przedstawić jako:

$$\hat{\mathbf{w}} = \sum_{i=1}^N i \mathbf{p}_i ; \quad (3.53)$$

definiując tym samym naszą ścieżkę poszukiwań. Kwestią pozostałą do rozwiązania są wartości współczynników i , czyli współczynników kroku algorytmu iteracyjnego.

Weźmy dowolne $k \in \{1, \dots, N\}$ i podstawmy (3.53) do (3.34):

$$\mathbf{b} = \mathbf{A}\hat{\mathbf{w}} = \sum_{i=1}^N {}_i\mathbf{A}\mathbf{p}_i ; \quad (3.54)$$

a następnie wymóżmy lewostronnie przez \mathbf{p}_k^T otrzymując:

$$\mathbf{p}_k^T \mathbf{b} = \sum_{i=1}^N {}_i\mathbf{p}_k^T \mathbf{A}\mathbf{p}_i = {}_k\mathbf{p}_k^T \mathbf{A}\mathbf{p}_k ; \quad (3.55)$$

przy czym ostatnia równość wynika z \mathbf{A} -ortogonalności wektorów $\{\mathbf{p}_i\}$. Stąd ostatecznie otrzymujemy:

$${}_k = \frac{\mathbf{p}_k^T \mathbf{b}}{\mathbf{p}_k^T \mathbf{A}\mathbf{p}_k} ; \quad (3.56)$$

Przedstawiona powyżej podstawowa wersja metody gradientów sprzężonych może być również stosowana, z pewnymi modyfikacjami, w przypadkach gdy macierz \mathbf{A} nie jest symetryczna, dodatnio-określona ani nawet kwadratowa. Z punktu widzenia zastosowań do adaptacji sieci neuronowych typu MLP najbardziej przydatne jest jednak uogólnienie na przypadek problemów nieliniowych, zaproponowane w 1964r. przez Fletchera i Reevesa [37]. Jedną z podstawowych różnic w stosunku do przypadku liniowego jest brak możliwości bezpośredniego obliczenia współczynnika kroku ${}_i$, który musi być określony na drodze minimalizacji funkcji celu wzdłuż aktualnego kierunku poszukiwań \mathbf{p}_i . Na potrzeby niniejszej pracy z wielu możliwych sposobów minimalizacji funkcji rzeczywistych jednej zmiennej wybrana i zaimplementowana została metoda bazująca na aproksymacji wielomianem trzeciego stopnia [94]. Wielomian ten jest określany w oparciu o wartość funkcji celu oraz pochodnej kierunkowej funkcji celu w dwóch punktach odległych o pewien współczynnik kroku h , modyfikowany podczas procesu adaptacji.

W praktycznych implementacjach wzory (3.45), (3.46), (3.56) można istotnie uprościć⁸, w szczególności jeżeli chodzi o sumowanie we wzorze (3.46), tak aby nie było konieczności pamiętania wszystkich wcześniejszych kierunków poszukiwań. W tym celu definiuje się pewien współczynnik ${}_k$, który w kolejnych iteracjach kumuluje informację o wszystkich poprzednich kierunkach. Istnieje kilka metod obliczania tego współczynnika [41], m.in. metoda Fletcher-Reevesa:

$${}_k = \frac{\mathbf{g}_{k+1}^T \mathbf{g}_{k+1}}{\mathbf{g}_k^T \mathbf{g}_k} ; \quad (3.57)$$

⁸Dotyczy to również przedstawionego wyżej przypadku liniowego

oraz metoda Polaka-Ribi  re'a

$$_k = \frac{\mathbf{g}_{k+1}^T (\mathbf{g}_{k+1} - \mathbf{g}_k)}{\mathbf{g}_k^T \mathbf{g}_k} : \quad (3.58)$$

Do implementacji wybrano metodę Polaka-Ribi  re'a, która wprawdzie charakteryzuje si  nieznacznie gorszą stabilno  , jednak zwykle zapewnia szybszą zbieżno  . Ostatecznie, algorytm minimalizacji nieliniowej funkcji celu w oparciu o metodę gradientów sprzężonych przedstawia si  następuj o :

1. Wybierz punkt startowy w_0
 2. Oblicz pochodną g_0 w punkcie w_0
 3. Przypisz $k := 0$; $p_0 := g_0$
 4. Znajdź \mathbf{p}_k minimalizujące $f(w_k + \mathbf{p}_k)$
 5. Oblicz $w_{k+1} = w_k + \mathbf{p}_k$
 6. Oblicz $g_{k+1} = -f'(w_{k+1})$
 7. Oblicz $\alpha_k = \frac{g_{k+1}^T(g_{k+1} - g_k)}{g_k^T g_k}$
 8. Oblicz $\mathbf{p}_{k+1} = g_{k+1} + \alpha_k \mathbf{p}_k$
 9. Przypisz $k := k + 1$
 10. Jeżeli nie spełniony warunek stopu, to wróć do punktu 4
 11. Zakończ zwracając wartość w_k

Punkt startowy w_0 może być wybierany przez losowanie. W przypadku adaptacji szybkich ortogonalnych sieci neuronowych możliwy jest również wybór wynikający z rodzaju przekształcenia wyznaczającego architekturę sieci (p. rozdział 4). Jako warunek stopu przyjąć można wiele różnych kryteriów, typowo stosowanych w uczeniu sieci neuronowych, jak np.:

- norma wektora gradientu mniejsza od pewnej wartości progowej,
 - błąd (wartość funkcji celu) mniejszy od pewnej wartości progowej,
 - osiągnięto z góry założoną ilości epok,
 - wynik klasyfikacji powyżej pewnej wartości progowej,
 - przekroczono limit czasu,
 - brak dalszych postępów nauki.

W praktycznej realizacji przedstawionego algorytmu należy dodatkowo uwzględnić konieczność jego okresowego „restartu”, czyli wyzerowania współczynnika α , co odpowiada ponownemu przyjęciu kierunku najszybszego spadku. Wynika to po-

pierwsze z faktu, że w idealnym przypadku liniowym liczba kroków algorytmu jest równa wymiarowi przestrzeni, więc zwiększenie ich liczby jest bezcelowe. Ponadto w praktyce dochodzą tu problemy związane z kumulacją błędów zaokrągleń oraz z rozbieżnościami między modelem liniowym, a faktycznie minimalizowaną nieliniową funkcją celu.

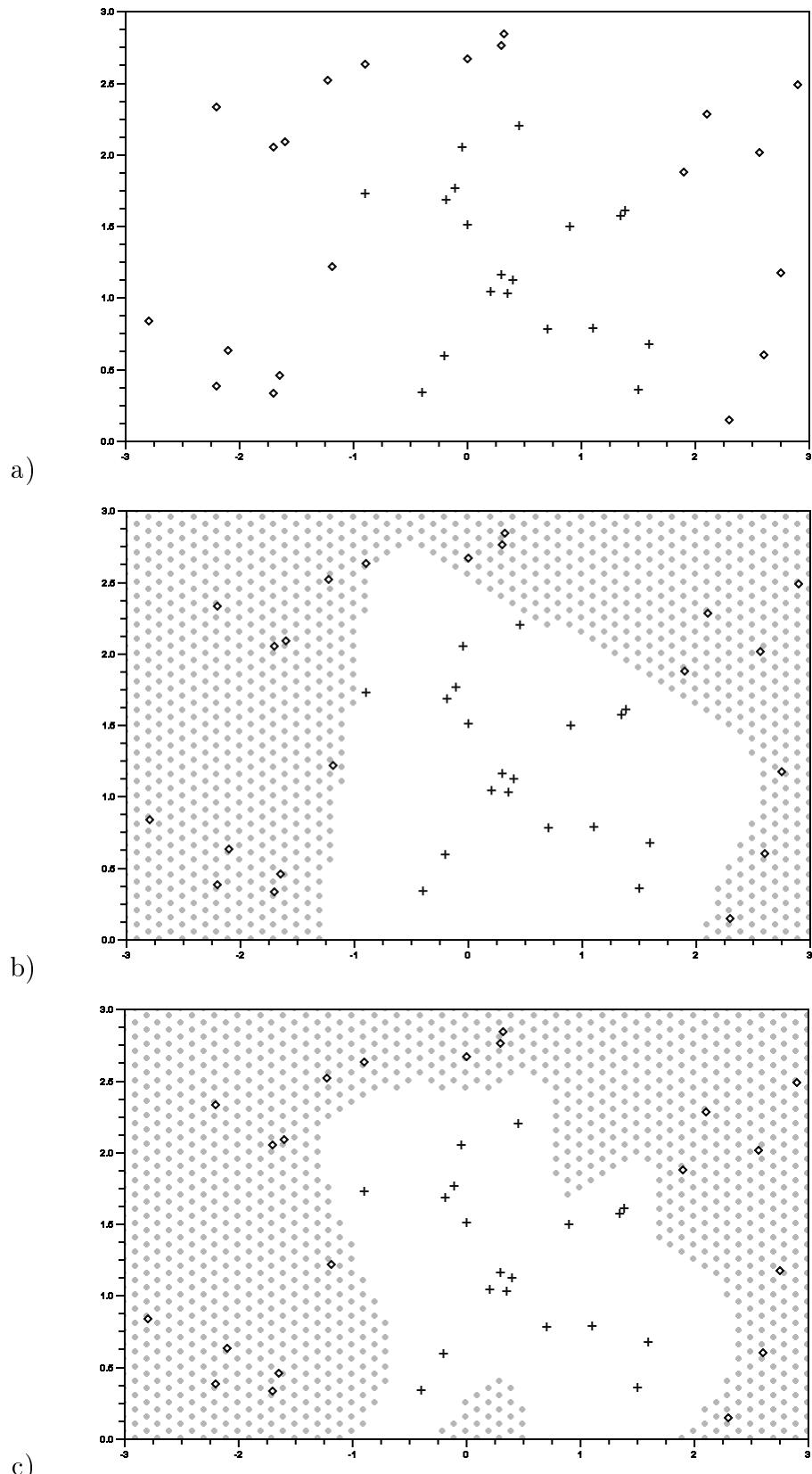
3.3 Klasyfikacja, a własności generalizacyjne sieci

Jedną z cech upodabniających sposób działania sztucznej sieci neuronowej do własności ośrodkowego układu nerwowego człowieka jest — obok uczenia się i zapamiętywania informacji — zdolność ich *uogólniania*. Wyraża się ona udzielaniem przez sieć poprawnej odpowiedzi na wyjściu po podaniu na wejście wcześniej nie prezentowanego wzorca, który jest tylko w pewnym sensie podobny do wzorców znanych. Uznając zdolność do właściwego zachowania się w nowych sytuacjach za miarę inteligencji, cecha ta być może najlepiej uzasadnia użycie określenia „sztuczna inteligencja” w stosunku do sieci neuronowych.

Kwestia owego *podobieństwa* wzorców ma tutaj znaczenie podstawowe i ściśle zależne od dziedziny zastosowań. Czy dźwięk fletu podobny jest do dźwięku oboju? Czy kształt litery odwróconej o 90° jest podobny do oryginału? Czy rower jest podobny do motocykla? Wszystko zależy od sposobu w jaki sformułujemy problem. Nasuwa się zatem pytanie – jakimi środkami dysponujemy aby wpływać na poziom szczegółowości i uogólniania podczas projektowania i treningu sieci?

Pierwszym, najoczywistszym i najbardziej intuicyjnym środkiem jest sama konstrukcja zbioru uczącego. Sieć pełniąca funkcję klasyfikatora uczy się przydzielając właściwe etykiety klas przykładom ze zbioru treningowego, a więc za podobne będzie uznawać wszystkie wzorce zbioru treningowego umieszczone w tej samej klasie.

Drugim bardzo ważnym czynnikiem jest konstrukcja samej sieci, a w szczególności jej złożoność, przy czym pojęcie złożoności sieci można tutaj rozumieć w odniesieniu do jej możliwości zapamiętywania i reprezentowania wzorców. Wracając do przykładów z podrozdziału 3.1, sieć z dwoma nieliniowymi neuronami w warstwie ukrytej ma większą złożoność od pojedynczego neuronu, co przekłada się na możliwość nauczenia się bardziej skomplikowanego sposobu podziału przestrzeni wejściowej. Typową, często stosowaną w praktyce metodą regulowania złożoności sieci typu MLP jest odpowiedni dobór liczby neuronów ukrytych (rys. 3.11). Złożoność sieci jest zatem ściśle powiązana z ilością jej wag poddawanych adaptacji, chociaż nie musi to być relacja jeden-do-jednego, a inne czynniki, takie jak typ neuronów i struktura ich połączeń odgrywają tu też istotną rolę. Dla przy-



Rysunek 3.11: Zależność wyniku klasyfikacji wzorców w przestrzeni \mathbb{R}^2 od liczby neuronów ukrytych: a) dane do klasyfikacji; b) 6 neuronów ukrytych; c) 30 neuronów ukrytych

kładu, zwiększanie liczby neuronów, a zatem i liczby wag w wielowarstwowej sieci złożonej wyłącznie z neuronów liniowych może nie mieć wpływu na jej możliwości reprezentacji danych uczących.

Metoda obiektywnego określenia złożoności sieci została zaproponowana przez Vapnika i Chervonenkisa w postaci tzw. miary $VC\text{-}dim$ [140][141], definiowanej jako maksymalna liczba punktów w przestrzeni wejściowej, które mogą być zaklasyfikowane na wszystkie możliwe sposoby przez dany klasyfikator. Przykładowo, pojedynczy neuron liniowy o dwóch wejściach z biasem ma wartość $VC\text{-}dim$ równą trzy, ponieważ łatwo stwierdzić, iż przy dowolnym, zadanym z góry, przyporządkowaniu trzech (niewspółliniowych) punktów w \mathbb{R}^2 do jednej z dwóch klas, może on dokonać poprawnej klasyfikacji. Zwiększenie liczby punktów do czterech uniemożliwia natomiast właściwą klasyfikację w pewnych przypadkach, takich jak np. rozważany w podrozdziale 3.1 problem XOR (przy założeniu, że każdy z czterech zbiorów na rys. 3.4 jest reprezentowany pojedynczym punktem).

Zwiększając złożoność sieci neuronowej należy mieć zawsze na uwadze to, że z jednej strony zwiększamy w ten sposób jej możliwości zapamiętywania wzorców, z drugiej jednak ryzykujemy ograniczeniem jej zdolności generalizacyjnych. W takiej sytuacji sieć może łatwo zacząć uczyć się „na pamięć”, traktując jako istotne również drobne fluktuacje i szумy występujące w danych treningowych. Sieć o dużej złożoności jest w stanie szczegółowo odwzorować dane zawarte w zbiorze treningowym, co przekłada się na skuteczne minimalizowanie wartości błędu podczas nauki E_{train} i poprawną klasyfikację. Jednak dopiero testowanie wytrenowanej sieci na nowych danych, które nie były używane podczas uczenia (czyli tzw. zbiorze testowym), może przynieść odpowiedź na pytanie o uzyskane zdolności generalizacyjne. Błąd uzyskany na zbiorze testowym E_{test} stanowi tym samym błąd generalizacji i — w gruncie rzeczy — to właśnie jego minimalizacja jest głównym zadaniem każdego klasyfikatora, wyznaczającym rzeczywistą przydatność w zastosowaniach praktycznych.

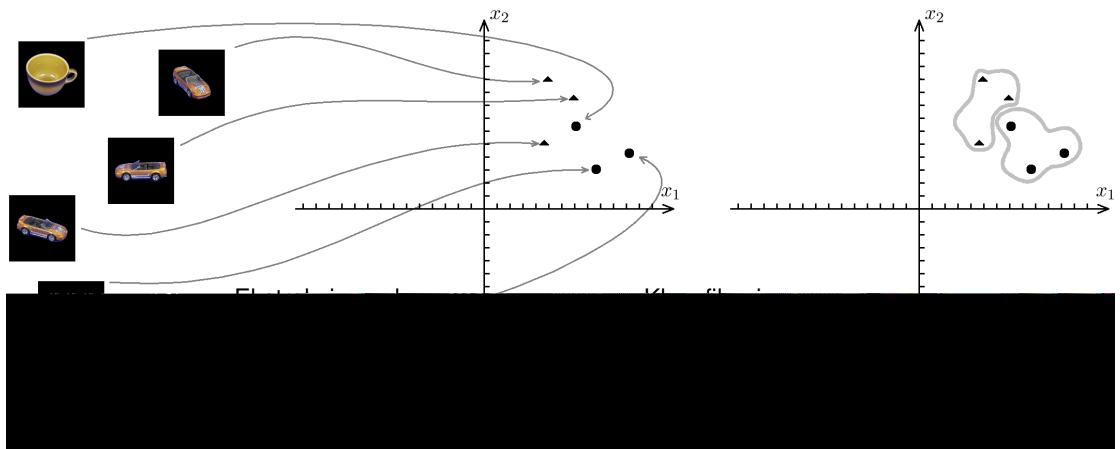
Należy zauważyć, że istotna jest tu również liczebność zbioru treningowego, a więc — stosunek jego liczebności do złożoności sieci. Mała liczba wzorców treningowych zostanie dobrze zapamiętana przez złożoną sieć, ale poziom generalizacji może być niski. Z drugiej strony, prosta sieć może dobrze uogólniać wzorce treningowe, ale jeśli jest ich zbyt wiele — nie będzie w stanie się ich dobrze nauczyć nawet w fazie trenowania (błąd E_{train} będzie wysoki). Zgodnie z oszacowaniem podanym przez Vapnika dla prawdopodobieństwa 1 — prawdziwa jest następująca relacja:

$$E_{test} \leq E_{train} + \sqrt{\frac{h(\log(2M/h) + 1) - \log(\epsilon)}{M}} ; \quad (3.59)$$

gdzie h jest miarą $VC\text{-}dim$ klasyfikatora, a M jest liczbą wzorców uczących.

Dodatkowym czynnikiem rzutującym na uzyskiwaną wartość błędu generalizacji jest przyjęte kryterium stopu algorytmu adaptacji. Wynika to z dobrze znanego faktu, że błąd generalizacji E_{test} zazwyczaj spada w trakcie procesu nauki, ale tylko do pewnego momentu, po którym następuje jego ponowny wzrost. Ponieważ błąd treningowy E_{train} zwykle stale maleje w trakcie całego procesu adaptacji, nie jesteśmy w stanie tylko na jego podstawie określić kiedy błąd E_{test} osiągnął minimum. Pod tym względem najlepsze kryterium stopu uzyskuje się wykorzystując trzeci zbiór danych, tzw. zbiór walidacyjny, który nie jest bezpośrednio używany do adaptacji, ale błąd na tym zbiorze E_{val} jest stale monitorowany i adaptacja jest przerywana w momencie, gdy osiągnie on minimum. Warto tu też zauważyć, że w przypadku klasyfikacji sama wartość błędu nie w pełni odzwierciedla osiągniętą skuteczność rozpoznawania. W szczególności, możliwa jest sytuacja kiedy wyższy błąd współwystępuje z lepszym wynikiem klasyfikacji oraz vice versa (np. dla wzorca z pierwszej klasy i parametrami $\lambda = 0.25$, $h = 0.75$ wyjście sieci może być równe $[0.9; 0.7; 0.7; \dots]^T$ w pierwszym przypadku, i odpowiednio $[0.15; 0.25; 0.25; \dots]^T$ w drugim). W związku z tym jako kryterium stopu można również przyjąć maksymalny wynik klasyfikacji na zbiorze walidacyjnym.

Trzecim środkiem służącym precyzowaniu zadania klasyfikacyjnego jest konstrukcja cech reprezentujących rozpoznawane obiekty. Należy bowiem pamiętać,



Rysunek 3.12: Etapy procesu rozpoznawania wzorców: a) reprezentacja bezpośrednią; b) przestrzeń cech; c) rezultat klasyfikacji

że typowy system rozpoznawania wzorców zawiera przynajmniej dwa podstawowe elementy (rys. 3.12):

- ekstraktor cech, którego zadaniem jest odwzorowanie danych opisujących bezpośrednio obiekty świata rzeczywistego w przestrzeń cech, w której każdy obiekt reprezentowany jest przez jeden punkt (wektor),

- właściwy klasyfikator.

Niektórzy autorzy wyróżniają w procesie rozpoznawania wzorców znacznie więcej etapów. Dla przykładu, system rozpoznawania obrazów może zawierać następujące elementy [74]:

- akwizycja danych,
- rejestracja danych,
- przetwarzanie wstępne,
- segmentacja,
- normalizacja,
- ekstrakcja cech,
- klasyfikacja,
- przetwarzanie końcowe.

Nie ulega jednak kwestii to, że ekstrakcja cech i klasyfikacja pełnią tu najistotniejsze funkcje.

Ekstrakcja cech jest kluczowym zabiegiem wpływającym na zagadnienie podobieństwa obiektów. Wracając do pytań postawionych na stronie 50, jeśli w pierwszym przypadku wyekstrahowaną cechą będzie natężenie dźwięku, jeśli w drugim przypadku cechy będą invariantne względem obrotu obrazu na płaszczyźnie, a w trzecim – jeśli ekstrakcja cech będzie obejmowała np. duże rozmycie gaussowskie zacierające szczegóły, wówczas we wszystkich trzech przypadkach oba obiekty będą traktowane jako podobne. Niezależnie od pozostałych czynników, tj. od doboru wzorców uczących i od złożoności klasyfikatora, rower zostanie utożsamiony z motocyklem, flet z obojem, a litera „Z” z literą „N”.

Projektując system rozpoznawania możemy w pewnym zakresie operować wszystkimi trzema wyżej wymienionymi czynnikami w celu poprawy jego własności generalizacyjnych. Przykładowo, mając możliwość łatwej akwizycji dużych ilości danych możemy zastosować bardziej złożoną sieć neuronową (np. z większą liczbą neuronów ukrytych) i ograniczać błąd generalizacji przez rozszerzanie zbioru treningowego.

Najciekawszy jednak wydaje się związek między metodą ekstrakcji cech, a sposobem konstrukcji zbioru uczącego. Istnieje tu bowiem pewna zależność, wyrażająca się tym, że w im bardziej precyzyjny sposób zdefiniujemy cechy, tym mniej liczny może być zbiór wzorców uczących. Dla przykładu, jeśli niezależność od

rozmiaru rozpoznawanego obiektu na obrazie będzie zapewniona przez ekstraktor cech, wówczas zbiór uczący może zawierać np. tylko po jednym zdjęciu każdego obiektu. W przeciwnym wypadku, każdy obiekt powinien być reprezentowany wielokrotnie z różnymi współczynnikami skali⁹. Naturalnie, najlepszą skuteczność rozpoznawania zwykle osiągamy starannie dopasowując algorytm ekstrakcji cech do konkretnego zastosowania. Wymaga to często dobrej znajomości danej dziedziny i dużego wysiłku ze strony projektanta, a w efekcie ekstraktor cech staje się „najbardziej intelligentnym” składnikiem całego systemu. Jak wiadomo, dobrze skonstruowany algorytm ekstrakcji cech powinien mieć niską złożoność obliczeniową i generować zwarte i jednocześnie wyczerpujące (w sensie kompletności opisu problemu) wektory cech. Jeśli te warunki są spełnione, wówczas większość klasyfikatorów wziętych „z półki” zwykle daje satysfakcyjne, a często nawet zbliżone wyniki.

Ceną jaką płacimy za wysoką skuteczność jest wąska specjalizacja i ograniczenie autonomii systemu rozpoznawania. System zaprojektowany do klasyfikacji typów limfocytów w obrazie mikroskopowym będzie raczej kiepsko rozpoznawał twarze i vice versa. To dość oczywiste. Można tu jednak odwołać się do przykładu najdoskonalszego znanego systemu klasyfikacji jakim jest mózg człowieka. Zwróćmy uwagę, iż swoje niebywałe zdolności rozpoznawania wzorców uzyskuje on w zasadzie na podstawie wyłącznie „zbioru treningowego”, czyli bodźców przekazywanych przez organy zmysłów. W tej nauce, rozpoczynającej się jeszcze w okresie prenatalnym, pewną rolę odgrywają elementy „zewnętrznej” ekstrakcji cech („popatrz – motocykl ma silnik, a rower nie”), wydaje się jednak, że jest to tylko rola pomocnicza, dodatkowa. Nasz „wbudowany ekstraktor cech”, jeśli można użyć tak drastycznego uproszczenia, posiada ogromny potencjał adaptacyjny, umożliwiający nam nabycie zdolności rozróżniania nowych wzorców praktycznie przez całe życie.

Większość analogii i odniesień do naszej wiedzy na temat funkcjonowania ludzkiego mózgu ma wciąż, przynajmniej w przypadku praktyczne realizowanych systemów rozpoznawania wzorców, charakter dosyć umowny. Jednak powyższe rozważania stały się dla autora niniejszej pracy pewną inspiracją do poszukiwań bardziej ogólnych rozwiązań, mniej zależnych od zdefiniowanych z góry algorytmów obróbki danych, a w większym stopniu opartych na adaptacji do problemów definiowanych bezpośrednio przez nieprzetworzone wzorce. Ponadto, popularyzacja urządzeń takich jak cyfrowe kamery i aparaty fotograficzne, oraz wzrost mocy obliczeniowej i pojemności nośników sprawiają, że akwizycja, przechowywanie i przetwarzanie danych o dużej objętości, w tym multimedialnych, stają się

⁹ zakładając, że naszym celem jest rozpoznawanie obiektów niezależnie od ich wielkości

coraz łatwiejsze, a jednocześnie rosną też oczekiwania użytkowników odnośnie narzędzi wspomagających analizę i zarządzanie tymi danymi. To pozwala sądzić, że jednym z istotnych kierunków rozwoju dziedziny rozpoznawania wzorców będzie w najbliższej przyszłości opracowywanie coraz bardziej ogólnych i coraz bardziej adaptacyjnych systemów, symulujących i przybliżających w swoim działaniu sposób postrzegania świata przez człowieka.

3.4 Podsumowanie

Rozdział niniejszy przedstawia pewien zbiór zagadnień dotyczących sztucznych sieci neuronowych, stanowiący niezbędną podstawę do wprowadzonego w kolejnym rozdziale tematu szybkich sieci ortogonalnych. Z uwagi na swoją istotność dla dalszego ciągu pracy pewne problemy zostały przeanalizowane dość szczegółowo. Omówiono tu w szczególności podstawy funkcjonowania sztucznych neuronów i złożonych z nich sieci jednokierunkowych w aspekcie klasyfikacji wzorców. Własne wyprowadzenie znanego algorytmu propagacji wstecznej zostało zaprezentowane w szczegółach, z uwagi na zastosowanie tej samej metodyki do adaptacji sieci ortogonalnych (rozdział 4). Przedstawiono podstawy teoretyczne i algorytm gradientów sprzężonych oraz uzasadniono jego wykorzystanie do uczenia sieci ortogonalnych. Na koniec, w sekcji 3.3 przedstawiono rozważania dotyczące własności generalizacyjnych sieci, oparte po części na własnych obserwacjach, refleksjach i doświadczeniach autora związanych z uczeniem sieci neuronowych o różnorodnej architekturze.

4 Szybkie sieci neuronowe

Punktem wyjścia do rozważań na temat szybkich sieci neuronowych jest problem złożoności sieci, poruszony w poprzednim rozdziale. Należy podkreślić, że popularne podejście praktyczne polegające na regulacji złożoności sieci typu MLP za pomocą modyfikacji liczby neuronów ukrytych wymaga nieraz wielu żmudnych eksperymentów. Bardziej wyrafinowane i skuteczne sposoby oparte na wrażliwościowych metodach redukcji połączeń międzyneuronalnych, takie jak OBD (ang. *optimal brain damage*, [26]), czy OBS (ang. *optimal brain surgeon*, [48]), obarczone są dużą złożonością obliczeniową wynikającą z konieczności aproksymacji macierzy hesjanu oraz powtarzania procedury weryfikacji przydatności poszczególnych połączeń naprzemiennie z douczaniem/korygowaniem zredukowanej architektury.

Koncepcja budowy sieci neuronowej o „szybkiej architekturze” połączeń międzyneuronalnych¹ zaproponowana w pracach [59][131] jest oparta na odmiennym podejściu. Zakłada się tam, że struktura połączeń między neuronami jest z góry zredukowana, zgodnie ze schematem faktoryzacji pewnego przekształcenia ortogonalnego. W ten sposób otrzymujemy liniową sieć wielowarstwową o rzadkiej strukturze połączeń odpowiadających niezerowym elementom rzadkich macierzy uzyskanych w wyniku tej faktoryzacji. Sieć taka z racji mniejszej liczby wag, rzędu $O(N \log N)$, w stosunku do $O(N^2)$ dla sieci o schemacie połączeń „każdy z każdym”², jest zdolna do realizacji tylko pewnej podklasy przekształceń liniowych $\mathbb{R}^N \rightarrow \mathbb{R}^N$. Jednakże wiemy, że podklastra ta zawiera również wybrane przekształcenie ortogonalne, którego szybki algorytm został użyty jako podstawa konstrukcji architektury połączeń projektowanej sieci. Co więcej, znając szczegóły szybkiego algorytmu jesteśmy w stanie z góry podać wszystkie wartości wag sieci, tak aby realizowała ona wybrane przekształcenie bez potrzeby nauki. Przyjmując ten stan sieci za punkt startowy algorytmu adaptacji możemy modyfikować własności przekształcenia pod kątem danego problemu na drodze treningu, tak jak adaptujemy zwykłą sieć neuronową do postawionego przed nią zadania. Naturalnie, możliwa jest tu również, stosowana zwykle w przypadku większości typów sieci neurono-

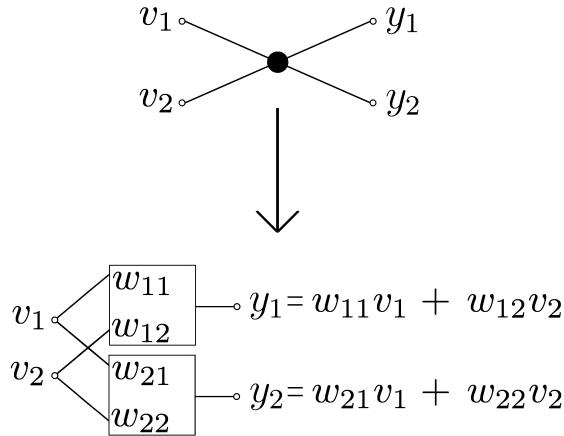
¹dla sieci tego typu będziemy używać określenia *szybka sieć neuronowa*

²odpowiadającej zwykłej, tj. gęstej macierzy przekształcenia

wych, inicjalizacja wag wartościami losowymi. Warto zauważyć, że podstawową formą weryfikacji empirycznej, czy dana sieć została skonstruowana prawidłowo jest sprawdzenie, czy jest ona w stanie wyuczyć się właściwego dla niej przekształcenia ortogonalnego z dowolną dokładnością, startując z losowego punktu w przestrzeni wag.

4.1 Założenia podstawowe

Prostota idei budowy szybkiej sieci neuronowej zgodnie z powyższym opisem, wyraża się tym, że graf przepływu szybkiego algorytmu (e.g. grafy z rys. 2.1, 2.3, 2.5), w zasadzie bezpośrednio wyznacza architekturę połączeń między jej neuronami. Istotna jest jednakże kwestia samych neuronów, które — jak to zostało zaproponowane w pracy [59] — zastępują operacje bazowe w grafie przepływu, zgodnie z rys. 4.1.



Rysunek 4.1: Reprezentacja neuronowa pojedynczej operacji bazowej grafu przepływu

Jak widać, pojedyncza operacja bazowa reprezentowana jest przez dwa zwykłe neurony o dwóch wejściach bez biasu, z liniową funkcją aktywacji. Biorąc pod uwagę sposób obliczania wyjść przez operację bazową (rys. 2.2, 2.4, 2.6) można stwierdzić, że *współczynniki* operacji bazowych o ustalonych z góry wartościach stają się teraz *wagami* o wartościach modyfikowanych w procesie adaptacji. Przyjmując interpretację operacji bazowej w postaci pary neuronów, grafy z rys. 2.1, 2.3, 2.5 stanowią — bez konieczności wprowadzania dodatkowych zmian — gotowy do implementacji schemat szybkiej sieci neuronowej.

Struktura szybkiego algorytmu transformaty cosinusowej drugiego rodzaju została wykorzystana w pracy [131] do konstrukcji sieci zdolnych nauczyć się wielu przekształceń ortogonalnych, m.in również przekształcenia sinusowego, Hartley'a

i Fouriera. Podano tam również podstawowy sposób adaptacji sieci w oparciu o metody gradientowe i przedstawiono jej zastosowanie do kompresji obrazów (por. także [128]).

W następnym podrozdziale zostanie przedstawiony nowy rodzaj szybkiej sieci neuronowej zaproponowany przez autora. Jego treść częściowo wykorzystuje znacznie rozszerzony i uzupełniony materiał opublikowany w pracy [120].

4.2 Szybkie ortogonalne sieci neuronowe

Porównując rys. 4.1 z rys. 2.2, 2.4 i 2.6 łatwo dostrzec, że cztery wagi występujące łącznie w obu neuronach z pary na rys. 4.1 są nadmiarowe w stosunku do obliczeń wykonywanych przez operacje bazowe. Dla przykładu, drugi etap algorytmu FCT2 wymaga zaledwie dwóch różnych współczynników S_K^k , C_K^k na jedną operację bazową, zaś w przypadku algorytmów z mnożnikami tangensowymi w operacjach bazowych drugiego etapu występuje tylko jeden nietrywialny współczynnik T_K^k . Co więcej, operacje bazowe pierwszego etapu posiadają wyłącznie trywialne współczynniki, jako że w gruncie rzeczy realizują jedynie operacje dodawania/odejmowania.

Poszczególne typy operacji bazowych można przedstawić w reprezentacji macierzowej. Najbardziej ogólną postać ma przekształcenie wykonywane przez parę neuronów:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = P_4 \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \text{ gdzie } P_4 = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix}; \quad (4.1)$$

zaś operacje z dwoma współczynnikami oraz z jednym można przedstawić odpowiednio jako:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = P_2 \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \text{ gdzie } P_2 = \begin{bmatrix} u & w \\ -w & u \end{bmatrix}; \quad (4.2)$$

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = P_1 \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \text{ gdzie } P_1 = \begin{bmatrix} 1 & t \\ -t & 1 \end{bmatrix}; \quad (4.3)$$

Wyrazy u , w oraz t są tutaj *wagami* reprezentującymi współczynniki C_K^k , S_K^k oraz T_K^k , odpowiednio. Wykorzystanie w konstrukcji szybkiej sieci neuronów opartych o macierz P_2 umożliwia dwukrotną redukcję liczby wag w stosunku do sieci przedstawionej w [131], a w przypadku zastosowania algorytmu z mnożnikami tangensowymi i użycia macierzy P_1 redukcja jest czterokrotna. Co istotne, w obu przypadkach skonstruowana sieć neuronowa zachowuje wciąż możliwość nauczenia się odpowiedniego przekształcenia ortogonalnego.

Zauważmy, że kolumny i wiersze każdej macierzy P_2 są zawsze ortogonalne, a ponadto równe co do normy. Nie są wprawdzie ortonormalne, więc nie możemy określić samych macierzy jako ortogonalnych, w świetle definicji ze strony 15. Jednak, biorąc pod uwagę wartości elementów macierzy P_2 dane wzorami rozkładu odpowiednich transformat:

$$\begin{aligned} u &= C_{4N}^k ; \\ w &= S_{4N}^k ; \end{aligned} \quad (4.4)$$

widzimy, że można byłoby wyrazić wagi operacji bazowej w postaci funkcji tylko jednej zmiennej :

$$\begin{aligned} u &= \cos(\) ; \\ w &= \sin(\) ; \end{aligned} \quad (4.5)$$

wskutek czego kolumny/wiersze macierzy P_2 stałyby się wektorami ortonormalnymi. Sieć neuronowa skonstruowana przy tym założeniu³ byłaby zdolna uczyć się wyłącznie przekształceń ortogonalnych, co łatwo zauważać pamiętając, że iloczyn macierzy ortogonalnych jest również macierzą ortogonalną. Podejście takie zaoferowałoby jednak koniecznością obliczania funkcji trygonometrycznych w procesie uczenia, zwiększaając istotnie złożoność obliczeniową i czas adaptacji. Z tego względu do praktycznych zastosowań wybrano — równoważną pod względem ilości adaptowanych wag — sieć opartą na algorytmie z mnożnikami tangensowymi, w której neurony wykorzystują macierz P_1 . Niemniej jednak, każdą szybką sieć, w której operacje bazowe definiowane są przez macierze o ortogonalnych kolumnach/wierszach będziemy określać jako *szybką ortogonalną sieć neuronową* (ang. *fast orthogonal neural network*, FONN).

4.2.1 Uczenie sieci typu FONN

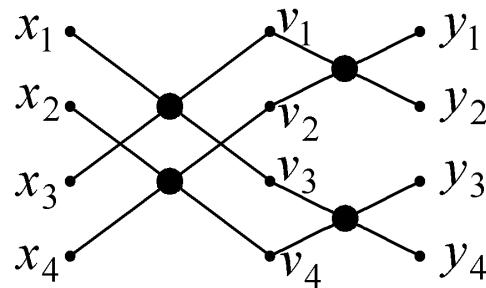
Zasadniczym elementem opisu sieci neuronowej, obok struktury połączeń, jest definicja samych neuronów, oraz algorytm ich adaptacji. W przypadku szybkich ortogonalnych sieci neuronowych pojęcie neuronu rozumianego jako „adaptowalna operacja bazowa” szybkiego algorytmu, wymaga doprecyzowania. Pewnym problemem jest tu fakt, iż klasyczny neuron przedstawiony w rozdziale 3 zwykle posiada wiele wejść, ale tylko jedno wyjście, natomiast operacje bazowe w wykorzystywanych tu szybkich dwuetapowych algorytmach mają dwa wyjścia. Reprezentując operację bazową jako parę neuronów, tak jak w pracach [59][131], omijamy ten pro-

³Taka sieć została ostatnio zaimplementowana i wykorzystana w kompresji danych [100]

blem. Jednak wykorzystanie macierzy P_2 i P_1 implikuje wzajemną zależność obu wartości wyjściowych. Możemy zatem mówić o dwóch neuronach „ortogonalnych” względem siebie lub zdefiniować nowy typ neuronu o dwóch wyjściach. Przyjmując tę drugą możliwość, będziemy od tej pory określać neuronową realizację operacji bazowej jako *ortogonalny neuron operacji bazowej* (ang. *basic operation orthogonal neuron*, BOON). Neuron typu BOON jest zatem liniowym neuronem o dwóch wejściach i dwóch wyjściach, posiadającym jedną lub dwie wagę, które determinują wartości wyjściowe zgodnie z wzorami (4.3), (4.2), odpowiednio. Przez analogię do reprezentacji wag w postaci macierzy typu P_1 lub P_2 będziemy niekiedy określać neurony wykorzystujące te macierze jako neurony typu BOON-1 i BOON-2, odpowiednio. Wykorzystanie macierzy P_4 odpowiada natomiast użyciu pary klasycznych neuronów zgodnie z rys. 4.1, a zatem jest równoważne szybkiej „nie-ortogonalnej” sieci przedstawionej w podrozdziale 4.1.

Algorytmu adaptacji szybkiej sieci neuronowej, zaproponowanego w pracy [131] nie da się bezpośrednio przenieść na grunt sieci typu FONN. Podstawowy problem stanowi tu fakt, że w neuronach ortogonalnych pojedyncza waga decyduje o dwóch wartościach wyjściowych, co jest dosyć nietypową sytuacją w przypadku jednokierunkowych sieci neuronowych. Opracowanie skutecznego algorytmu adaptacji sieci typu FONN stanowi zatem istotny cel niniejszej pracy. Dodatkowo, wzory definiujące sposób adaptacji szybkiej sieci neuronowej podane w pracy [131] cechuje dość duży stopień komplikacji, wynikający ze złożonego sposobu obliczania indeksów operacji bazowych, uzależnionego od konkretnej architektury sieci. Proponowany algorytm adaptacji neuronów typu BOON jest natomiast niezależny od wybranego schematu połączeń pomiędzy neuronami, co umożliwia użycie go do adaptacji sieci opartych na różnych rodzajach przekształceń ortogonalnych.

Rozważmy przykładową dwuwarstwową sieć przedstawioną na rys. 4.2 i założmy, że wszystkie cztery występujące w niej operacje bazowe implementowane są w oparciu o macierz P_2 (wzór (4.2)). Możemy przedstawić wyjście sieci w po-



Rysunek 4.2: Dwie warstwy sieci typu FONN

staci jawną zależność funkcyjną względem wejść:

$$\begin{aligned} y_1 &= u_1^{(2)} v_1 + w_1^{(2)} v_2 ; & v_1 &= u_1^{(1)} x_1 + w_1^{(1)} x_3 ; \\ y_2 &= -w_1^{(2)} v_1 + u_1^{(2)} v_2 ; & v_2 &= u_2^{(1)} x_2 + w_2^{(1)} x_4 ; \\ y_3 &= u_2^{(2)} v_3 + w_2^{(2)} v_4 ; & \text{gdzie } v_3 &= -w_1^{(1)} x_1 + u_1^{(1)} x_3 ; \\ y_4 &= -w_2^{(2)} v_3 + u_2^{(2)} v_4 ; & v_4 &= -w_2^{(1)} x_2 + u_2^{(1)} x_4 ; \end{aligned} \quad (4.6)$$

i gdzie wagi $u_k^{(l)}$ i $w_k^{(l)}$ odnoszą się do k -tej operacji w l -tej warstwie. Naszym celem jest minimalizacja funkcji błędu danej analogicznie jak w definicjach (3.17), (3.18) jako:

$$E = \frac{1}{2} \sum_{i=1}^N (y_i - d_i)^2 ; \quad (4.7)$$

gdzie $N = 4$, zaś d_i reprezentuje oczekiwana wartość i -tego wyjścia (rozważamy tu dla uproszczenia pojedynczy wzorzec uczący).

Rozważmy pochodne funkcji błędu względem wag neuronów drugiej warstwy. Przedstawiając funkcję błędu w postaci:

$$\begin{aligned} E &= \frac{1}{2} \left(u_1^{(2)} v_1 + w_1^{(2)} v_2 - d_1 \right)^2 + \frac{1}{2} \left(-w_1^{(2)} v_1 + u_1^{(2)} v_2 - d_2 \right)^2 + \\ &\quad + \frac{1}{2} \left(u_2^{(2)} v_3 + w_2^{(2)} v_4 - d_3 \right)^2 + \frac{1}{2} \left(-w_2^{(2)} v_3 + u_2^{(2)} v_4 - d_4 \right)^2 ; \end{aligned} \quad (4.8)$$

łatwo otrzymujemy, że:

$$\begin{aligned} \frac{\partial E}{\partial u_1^{(2)}} &= (y_1 - d_1) v_1 + (y_2 - d_2) v_2 ; \\ \frac{\partial E}{\partial w_1^{(2)}} &= (y_1 - d_1) v_2 - (y_2 - d_2) v_1 ; \\ \frac{\partial E}{\partial u_2^{(2)}} &= (y_3 - d_3) v_3 + (y_4 - d_4) v_4 ; \\ \frac{\partial E}{\partial w_2^{(2)}} &= (y_3 - d_3) v_4 - (y_4 - d_4) v_3 ; \end{aligned} \quad (4.9)$$

Wynik ten jest analogiczny do wyrażenia (3.25) uzyskanego dla sieci MLP, z uwagi na to, że ponownie mamy tu iloczyn wejścia neuronu przez sygnał różnicowy ${}_{i=2}^{(2)} = (y_i - d_i)$. Ponieważ jednak pochodna względem danej wagi jest tym razem zależna od dwóch wyjść neuronu więc w powyższych wzorach uwzględniane są oba iloczyny. Druga różnica dotyczy pochodnej funkcji aktywacji: ponieważ neuron typu BOON jest liniowy, więc pochodna funkcji aktywacji jest stała, równa jedności. Z tego też względu możemy utożsamiać czynnik z sygnałem różnicowym⁴.

⁴por. wzór (3.24)

Podobnie dla neuronów pierwszej warstwy mamy:

$$\begin{aligned}
 \frac{\partial E}{\partial u_1^{(1)}} &= \left[(y_1 - d_1) u_1^{(2)} - (y_2 - d_2) w_1^{(2)} \right] x_1 + \\
 &\quad + \left[(y_3 - d_3) u_2^{(2)} - (y_4 - d_4) w_2^{(2)} \right] x_3 ; \\
 \frac{\partial E}{\partial w_1^{(1)}} &= \left[(y_1 - d_1) u_1^{(2)} - (y_2 - d_2) w_1^{(2)} \right] x_3 - \\
 &\quad - \left[(y_3 - d_3) u_2^{(2)} - (y_4 - d_4) w_2^{(2)} \right] x_1 ; \\
 \frac{\partial E}{\partial u_2^{(1)}} &= \left[(y_1 - d_1) w_1^{(2)} + (y_2 - d_2) u_1^{(2)} \right] x_2 + \\
 &\quad + \left[(y_3 - d_3) w_2^{(2)} + (y_4 - d_4) u_2^{(2)} \right] x_4 ; \\
 \frac{\partial E}{\partial w_2^{(1)}} &= \left[(y_1 - d_1) w_1^{(2)} + (y_2 - d_2) u_1^{(2)} \right] x_4 - \\
 &\quad - \left[(y_3 - d_3) w_2^{(2)} + (y_4 - d_4) u_2^{(2)} \right] x_2 ;
 \end{aligned} \tag{4.10}$$

Również tutaj zauważamy podobieństwo do wzoru (3.30). Zwróćmy jednak uwagę, że tym razem sumujemy tylko po dwa składniki, co pozwala nam zapisać sumy bezpośrednio. Symbolowi sumowania we wzorze (3.31) odpowiada tu dodawanie wewnętrz nawiasów kwadratowych (zauważmy, że ilość składników tej sumy określa ilość neuronów następnej warstwy połączonych z neuronem rozważanym, a ta wynosi w przypadku sieci FONN zawsze dwa). A zatem możemy powiedzieć, że wyrażenia w nawiasach kwadratowych definiują tu sygnał ⁽¹⁾. Podobnie jak w przypadku drugiej warstwy iloczyn sygnału delta i sygnału wejściowego występuje w każdym wzorze dwukrotnie, z uwagi na „dwuwyjściowość” neuronów.

Zauważmy tu jeden istotny fakt: otrzymane wzory mają uniwersalny charakter. Ponieważ liczba wejść i wyjść neuronu typu BOON jest zawsze stała i równa dwa, zatem różnorodność możliwych schematów połączeń wyraża się zasadniczo jedynie w określeniu kolejności wyjść poszczególnych operacji bazowych w warstwie⁵. Jednakże rozważając pojedynczy neuron nie musimy wiedzieć w jakiej warstwie się on znajduje, ani z czym są połączone jego wejścia i wyjścia. Istotne jest tylko jakie są w jego przypadku wartości sygnałów wejściowych i sygnałów .

Biorąc powyższe pod uwagę możemy uogólnić wyrażenia (4.9), (4.10) otrzymując wzory na składniki wektora gradientu i wektora sygnału błędu:

$$\begin{bmatrix} \frac{\partial E}{\partial u} \\ \frac{\partial E}{\partial w} \end{bmatrix} = \begin{bmatrix} v_1 & v_2 \\ v_2 & -v_1 \end{bmatrix} \cdot \begin{bmatrix} {}^{(L)}_1 \\ {}^{(L)}_2 \end{bmatrix} . \tag{4.11}$$

⁵Dwie zasadniczo różne pod tym względem architektury widzimy na przykładzie rys. 2.3 i 2.5.

$$\begin{bmatrix} (L-1) \\ 1 \\ (L-1) \\ 2 \end{bmatrix} = P_2^T \cdot \begin{bmatrix} (L) \\ 1 \\ (L) \\ 2 \end{bmatrix} ; \quad (4.12)$$

gdzie P_2^T oznacza transpozycję P_2 .

Parametry v_1 i v_2 we wzorach (4.11), (4.12) reprezentują wejścia operacji bazowej, wektor $\begin{bmatrix} (L) \\ 1 \\ (L) \\ 2 \end{bmatrix}^T$ określa wartości sygnału błędu propagowane wstecz z następnej warstwy do bieżącej, a wektor $\begin{bmatrix} (L-1) \\ 1 \\ (L-1) \\ 2 \end{bmatrix}^T$ definiuje nowe wartości sygnału błędu do propagacji wstecz z bieżącej warstwy do poprzedniej.

Ponieważ macierz P_1 (4.3) stanowi specjalny przypadek macierzy P_2 (4.2) dla $u = 1$, odpowiednie wzory służące do adaptacji sieci opartej na algorytmie z mnożnikami tangensowymi, mogą być wyprowadzone z zależności (4.11), (4.12) następująco:

$$\frac{\partial E}{\partial t} = \begin{bmatrix} v_2 & -v_1 \end{bmatrix} \cdot \begin{bmatrix} (L) \\ 1 \\ (L) \\ 2 \end{bmatrix} ; \quad (4.13)$$

$$\begin{bmatrix} (L-1) \\ 1 \\ (L-1) \\ 2 \end{bmatrix} = P_1^T \cdot \begin{bmatrix} (L) \\ 1 \\ (L) \\ 2 \end{bmatrix} ; \quad (4.14)$$

Wzory (4.11) – (4.14) mają znaczenie ogólne, tj. można je stosować do adaptacji dowolnego neuronu typu BOON niezależnie od jego położenia w strukturze sieci. Co więcej, nie zakłada się tu żadnej konkretnej architektury, jako że do trenowania sieci wystarcza informacja na temat indeksów połączonych ze sobą wejściem/wyjściem neuronów. Możliwe jest zatem rozdzielenie sposobu definiowania struktury sieci, potencjalnie bardzo złożonej, od samego algorytmu obliczania pochodnych i sygnału błędu. Mając zaś dane składniki wektora gradientu dowolna znana metoda gradientowa może być zastosowana do minimalizacji funkcji błędu sieci.

Powysze możliwości zostały wykorzystane w konstrukcji symulatora szybkich ortogonalnych sieci neuronowych, zawierającego dwa osobne moduły: generator struktury sieci i właściwy symulator odpowiedzialny za trening sieci i propagację w przód.

Obliczanie wszystkich składników wektora gradientu dla sieci opartej na dwuetapowym szybkim algorytmie transformatyjnej przebiega zatem zgodnie z poniższym algorytmem:

1. Przypisz $L :=$ numer ostatniej warstwy
2. Dla wszystkich wyjść sieci oblicz sygnał błędu $^{(L)} = y - d$
3. W oparciu o sygnał $^{(L)}$ dla wszystkich
 - a) składniki wektora gradientu (na podst. wzoru (4.11)/(4.13))

- b) sygnał błędu $^{(L-1)}$ (na podst. wzoru (4.12)/(4.14))
4. Przypisz $L := L - 1$
 5. O ile nie osiągnięto pierwszej warstwy wróć do punktu 3
 6. Koniec

Powyższy algorytm można łatwo modyfikować w celu np. uczenia tylko niektórych warstw. Dla warstwy, w której neurony mają mieć ustalone wagи (niezmienne w trakcie adaptacji) pomijany jest po prostu punkt 3.a). W większości przypadków z adaptacji wyłączony jest cały pierwszy etap, zawierający wyłącznie operacje dodawania/odejmowania. W takiej sytuacji dla warstw pierwszego etapu punkt 3.b) można również pominąć i powyższy algorytm kończy się po osiągnięciu pierwszej warstwy etapu drugiego.

Z uwagi na fakt, iż przedstawiony sposób obliczania gradientu dla sieci typu FONN jest w istocie przypadkiem szczególnym algorytmu propagacji wstecznej błędu stosowanego dla „zwykłych” sieci typu MLP, możliwe jest zatem połączenie obu typów architektury w jednej, hybrydowej sieci. Sieć taka może być złożona z dwóch zasadniczych bloków: sieci typu FONN, połączonej z jedną, lub kilkoma warstwami „gostymi” o liniowej bądź nieliniowej funkcji aktywacji. Zauważmy, że zarówno sygnał propagowany w przód z części typu FONN do warstwy wyjściowej (warstw wyjściowych), jak i sygnał błędu propagowany wstecz do bloku sieci FONN nie wymaga w tej sytuacji żadnych dodatkowych przekształceń. Ten rodzaj złożonej architektury sieci, zapewniający m.in. łatwą możliwość regulacji ilości wyjść (sieć typu FONN ma zawsze tyle samo wyjść ile wejść) został wykorzystany jako ogólny model klasyfikatora neuronowego, a jego możliwości zostały zweryfikowane na przykładzie kilku różnych zbiorów danych w rozdziale 6.

Prezentację zaproponowanej szybkiej ortogonalnej sieci neuronowej zakończymy oszacowaniem złożoności obliczeniowej algorytmu propagacji wstecznej. Liczba operacji arytmetycznych, tj. liczba mnożeń () i dodawań () dla poszczególnych macierzy jest następująca:

$$\begin{aligned} (P_4) &= 8 ; & (P_2) &= 8 ; & (P_1) &= 4 ; \\ (P_4) &= 2 ; & (P_2) &= 4 ; & (P_1) &= 3 ; \end{aligned} \quad (4.15)$$

Interpretując powyższe wartości musimy pamiętać, że odnoszą się one tylko do obliczania wektora gradientu i propagacji wstecznej błędu. Jako że jednym z podstawowych parametrów wpływających na wydajność gradientowych algorytmów minimalizacji błędu sieci jest liczba jej wag, zatem dwu- i czterokrotne zmniejszenie tej liczby, wynikające z zastosowania neuronów typu BOON-2 i BOON-1, odpowiednio, zapewnia znaczącą poprawę efektywności sieci FONN w stosunku

do nie-ortogonalnej szybkiej sieci opartej na tym samym rodzaju przekształcenia ortogonalnego. Teza ta zostanie zweryfikowana eksperymentalnie w następnym rozdziale na przykładzie sieci opartej na przekształceniu cosinusowym typu drugiego.

4.3 Podsumowanie

W rozdziale tym autor zaproponował i przedstawił szczegóły konstrukcji nowego rodzaju jednokierunkowych sieci neuronowych o strukturze połączeń zredukowanej w oparciu o schemat szybkich algorytmów przekształceń ortogonalnych. Podstawowym elementem innowacyjnym w stosunku do znanych sieci tego typu jest nowy rodzaj neuronu, neuron typu BOON (p. str. 61), wykorzystujący ortogonalność operacji bazowych do redukcji liczby adaptowanych wag z czterech do dwóch lub do jednej. Autor zaproponował metodę adaptacji neuronów typu BOON i przedstawił jej wyprowadzenie dla obu przypadków, tj. z redukcją dwukrotną i czterokrotną, podkreślając jednocześnie analogię do przedstawionego w rozdziale 3 wyprowadzenia wzorów na adaptację wstępową błędu w ogólnym przypadku sieci typu MLP.

Uzyskana metoda adaptacji jest lokalna, czyli może być stosowana do uczenia poszczególnych neuronów sieci, bez względu na szczegóły schematu ich połączeń wyznaczone konkretnym rodzajem transformaty ortogonalnej. Stanowi to istotne uproszczenie w stosunku do znanych z literatury sieci opartych na szybkich schematach obliczeniowych i umożliwia łatwą zmianę architektury sieci bez modyfikacji procedur obliczeniowych propagacji w przód i adaptacji. Podkreślona została ponadto możliwość połączenia zaprezentowanej sieci neuronowej z sieciami typu MLP, co stanowi podstawę architektury klasyfikatorów neuronowych wykorzystywanych w rozdziale 6.

5 Przykłady konstrukcji i rezultaty adaptacji sieci typu FONN

Implementacja szybkich sieci nastręcza pewnych problemów związanych z nietypowym schematem połączeń między neuronami. Duża część dostępnych pakietów oprogramowania zakłada domyślny schemat połączeń w sieciach *feed forward*, czyli „każdy z każdym”. Dodatkowo, algorytm propagacji wstecz w szybkich sieciach ortogonalnych wymaga specjalnego opracowania. W związku z tym wszystkie badania omówione w tym i następnym rozdziale zostały przeprowadzone w oparciu o stworzone przez autora oprogramowanie symulacyjne zaprojektowane specjalnie do badań nad sieciami typu FONN. Niewielka część testów została wykonana na wcześniejszej, bardziej uniwersalnej wersji symulatora, w której możliwe było specyfikowanie dowolnej ilości wejść i wyjść neuronów, jednak kosztem większej złożoności obliczeniowej adaptacji i propagacji w przód. Zdecydowana większość została natomiast przeprowadzona na aktualnej wersji, specjalizowanej do testowania sieci złożonej z neuronów typu BOON, o maksymalnej ilości wejść i wyjść równej dwa¹. Kod źródłowy, dokumentację i przykłady zastosowania aktualnej wersji symulatora zamieszczono na załączonej płycie CD.

W tym rozdziale, obok wykazania słuszności pierwszej tezy pracy poprzez porównanie efektywności adaptacji szybkiej oraz szybkiej ortogonalnej sieci neuronowej (podrozdział 5.1), przedstawionych zostało kilka różnych rodzajów architektury sieci typu FONN. W szczególności, zaimplementowano i przedstawiono wyniki symulacji sieci opartej na przekształceniu cosinusowym drugiego i czwartego rodzaju, oraz przedstawiono modyfikację algorytmu transformaty cosinusowej drugiego rodzaju umożliwiającą obliczanie przekształceń Hartley'a i Fouriera. Na tej podstawie została skonstruowana sieć oparta na przekształceniu Fouriera, oraz przedstawiono jej modyfikację umożliwiającą obliczanie widma amplitudowego transformaty Fouriera. Następnie przedstawiono własne wyprowadzenie

¹W obecnej wersji możliwe jest również definiowanie klasycznych warstw neuronowych typu „każdy z każdym”, tak więc w stosunku do wersji pierwotnej symulatora nie ma tylko możliwości użycia pośrednich schematów połączeń

dwuetapowego jednolitego szybkiego algorytmu *dwuwymiarowej* transformaty cosinusowej drugiego rodzaju oraz jego modyfikację umożliwiającą obliczanie dwuwymiarowego przekształcenia Fouriera i jego widma amplitudowego. Opracowane algorytmy zostały wykorzystane jako podstawa konstrukcji sieci neuronowej do przetwarzania danych dwuwymiarowych (obrazów).

Poprzez konstrukcję sieci neuronowej typu FONN w oparciu o powyższe rodzaje przekształceń pokazana została elastyczność zaproponowanego modelu sieci i jego niezależność od szczegółów architektury połączeń wynikającej z wyboru konkretnego szybkiego algorytmu transformaty ortogonalnej. Dodatkowo, pewne interesujące możliwości dotyczące rekurencyjnego podejścia do konstrukcji i adaptacji sieci typu FONN zostały przedstawione w oparciu o algorytm transformaty cosinusowej typu czwartego.

5.1 Sieci typu FONN vs szybkie sieci neuronowe

Trzy grupy testów zostały przeprowadzone w celu porównania możliwości nie-ortogonalnej szybkiej sieci opartej na przekształceniu cosinusowym rodzaju drugiego oraz dwóch szybkich sieci ortogonalnych opartych na algorytmie FCT2 oraz mFCT2, odpowiednio [120]. Struktura połączeń w przypadku sieci nie-ortogonalnej była wyznaczona przez graf przepływu z rys. 2.1 z tym, że zastosowano tu reprezentację każdej operacji bazowej drugiego etapu w postaci pary zwykłych neuronów (por. rys. 4.1). Sieci ortogonalne oparte były na grafach przepływu odpowiednio z rysunków 2.1 i 2.3 i wykorzystywały neurony typu BOON-2 i BOON-1, odpowiednio.

Wszystkie trzy sieci były testowane z wykorzystaniem kilku zbiorów danych zróżnicowanych pod względem długości i liczby wektorów wejściowych. Elementy wektorów wejściowych były losowane za pomocą generatora liczb losowych o rozkładzie jednostajnym, a wektory docelowe były obliczane jako wynik przekształcenia cosinusowego drugiego rodzaju wektorów wejściowych. Adaptacja sieci dotyczyła tylko jej drugiego etapu (por. grafy przepływu z rys. 2.1, 2.3) i dla każdego zbioru danych była powtarzana dziesięciokrotnie z losowego punktu startowego w przestrzeni wag, a uśrednione wyniki przedstawiono w tablicach 5.1, 5.2, 5.3.

Pierwsze dwie kolumny zawierają rozmiar (N) i liczbę (P) wektorów wejściowych, (sr) oznacza wartość średnią, a (std) odchylenie standardowe. Jako kryterium stopu przyjęto osiągnięcie wartości błędu poniżej 1e-9. Powyższe wyniki, w tym czasy adaptacji, uzyskano na maszynie z procesorem Intel Celeron M, 1.40 GHz. Z uwagi na dużą ilość wektorów w zbiorach uczących nie wykorzystano dodatkowego zbioru testowego.

Tablica 5.1: Wyniki adaptacji dla szybkiej sieci neuronowej

N	P	Epoki (śr)	Epoki (std)	Czas (śr) [s]	Czas (std)	Wagi
8	4	49	6.063	0.1329	0.0619442	20
16	8	119	23.7445	1.2891	0.245518	68
32	16	125	37.5847	8.2673	2.41184	196
64	32	172	81.5784	78.6579	37.2662	516

Tablica 5.2: Wyniki dla szybkiej sieci ortogonalnej (neurony typu BOON-2)

N	P	Epoki (śr)	Epoki (std)	Czas (śr) [s]	Czas (std)	Wagi
8	4	25	1.96214	0.0782	0.0474443	10
16	8	46	3.74299	0.525	0.0564659	34
32	16	58	1.84662	3.9814	0.126828	98
64	32	71	2.5865	33.3281	1.16706	258

Uczenie sieci ortogonalnych okazało się być stabilnym procesem w sensie odchylenia standardowego jego czasu trwania. Najciekawszą obserwację stanowi fakt niemal stałej liczby epok w przypadku sieci opartej na algorytmie mFCT2 (tablica 5.3). Bliższa analiza wykazała, iż dla danego zbioru uczącego uzyskiwano zawsze zbliżony stan końcowy sieci, odpowiadający stanowi otrzymywanej przez bezpośrednie obliczanie wartości wag zgodnie z algorytmem mFCT2.

Stosunkowo duże bezwzględne wartości średnie czasu adaptacji dla większych wartości N wynikają z zastosowania do wszystkich testów pierwszej wersji symulatora (p. opis na stronie 67), która dzięki większej ogólności i elastyczności umożliwia testowanie również szybkich, nie-ortogonalnych sieci (tab. 5.1). Jednakże porównanie pomiędzy tablicami wykazuje zdecydowaną wyższość zaproponowanych sieci typu FONN, co jest szczególnie widoczne na przykładzie sieci z neuronami typu BOON-1. Uśredniając uzyskane czasy adaptacji dla wszystkich wartości N

Tablica 5.3: Wyniki dla szybkiej sieci ortogonalnej (neurony typu BOON-1)

N	P	Epoki (śr)	Epoki (std)	Czas (śr) [s]	Czas (std)	Wagi
8	4	14	1.22066	0.0625	0.0394899	5
16	8	25	0.916515	0.297	0.0270222	17
32	16	24	0.538516	1.7124	0.04304	49
64	32	22	0	10.9764	0.1246	129

zauważamy, że zysk wynikający z zastosowania obu wariantów sieci ortogonalnej odpowiada w przybliżeniu (a nawet nieco przewyższa) stopniowi redukcji liczby wag.

Podsumowując przedstawione wyniki, wykorzystanie ortogonalności operacji bazowych szybkich algorytmów pozwoliło na dwu- i czterokrotne zredukowanie liczby adaptowanych wag, przy zachowaniu możliwości nauczenia sieci wybranego przekształcenia trygonometrycznego. Dzięki temu uzyskano również dwu i czterokrotne skrócenie czasu oraz znaczące zwiększenie wydajności i stabilności procesu adaptacji.

5.2 Rekurencyjne uczenie sieci typu FONN

W tym podrozdziale przedstawiono możliwość dalszego przyspieszenia procesu nauki sieci typu FONN wynikającą z rekurencyjnego charakteru procedury jej konstrukcji [121]. Wykazano, iż wykorzystanie rekurencji w nauczaniu szybkiej sieci ortogonalnej umożliwia zredukowanie liczby adaptowanych wag nawet do rzędu $O(N)$ dla N -elementowych wektorów wejściowych. Dwa różne warianty określenia schematu rekurencji zostały zaproponowane i zastosowane do sieci opartej na algorytmie mFCT4 (rys. 2.5). Z uwagi na ogólność zaproponowanych rozwiązań możliwe jest również rekurencyjne nauczanie sieci opartej na innych rodzajach szybkich algorytmów przekształceń ortogonalnych.

Podstawowe założenie konstrukcyjne rozważanych tu szybkich przekształceń, *divide et impera*, prowadzi do wysokiej modularności otrzymywanych algorytmów. Co więcej, rekurencyjny charakter procesu konstrukcji zapewnia symetrię poszczególnych bloków grafu transformaty. Reprezentacja macierzowa jego pojedynczej warstwy jest sumą prostą pewnej liczby podmacierzy, zależnej od aktualnego poziomu rekurencji.

$$\mathcal{A}_{N \times N}^l = \bigoplus_{i=1}^m B_{\frac{N}{m} \times \frac{N}{m}}^{i,l} ; \quad (5.1)$$

gdzie l jest poziomem rekurencji, natomiast $m = 2^l$ i N stanowią rozmiar wektorów wejściowych/wyjściowych.

Możliwe są dwa ogólne warianty uczenia sieci wykorzystujące powyższą właściwość. Zakładając równość wszystkich macierzy $B^{i,l}$ dla danego l (co możemy określić jako ścisłą symetrię rekurencji)

$$\forall_{i,j=1,\dots,m} B^{i,l} = B^{j,l} ; \quad (5.2)$$

możemy trenować osobno blok sieci odpowiadający $B^{1,l}$, a następnie „sklonować” go tak, aby otrzymać \mathcal{A}^l . Podobnie, w następnym kroku tylko neurony reprezen-

tujące $B^{1,l-1}$ są adaptowane, po czym wykorzystujemy je do utworzenia A^{l-1} .

Jednakże, w przypadku niektórych algorytmów transformat ortogonalnych powyższe założenie może nie być prawdziwe [143]. W takich wypadkach, jeśli $B^{2,l}$ jest różne od $B^{1,l}$, musi być adaptowane oddzielnie, co odbywa się razem z adaptacją $B^{1,l-1}$. Podobnie, dla dowolnego k będącego potęgą dwójki, taką, że $k \leq m=2$, jeśli:

$$\bigoplus_{i=1}^k B^{i,l} \neq \bigoplus_{j=k+1}^{2k} B^{j,l}; \quad (5.3)$$

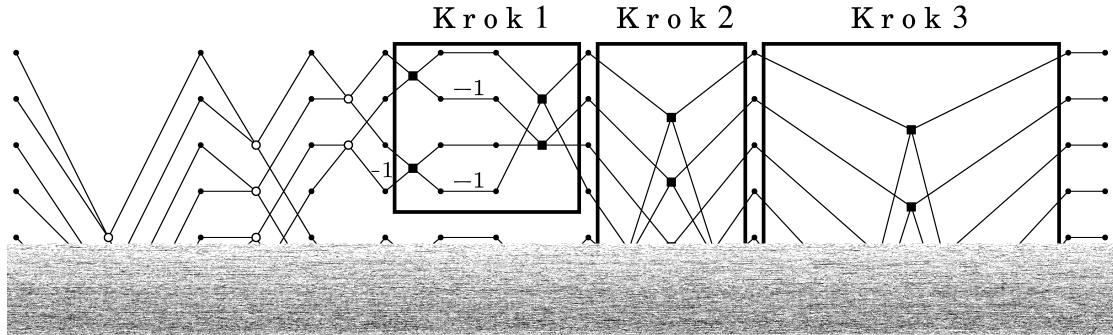
wówczas blok odpowiadający $\bigoplus_{j=k+1}^{2k} B^{j,l}$ będzie adaptowany razem z adaptacją $B^{1,l-\log_2(2k)}$. Naturalnie, ten wariant może być również użyty dla algorytmów spełniających warunek (5.2).

Oba warianty definiują rekurencyjny schemat uczenia sieci, w którym rozpoczynamy od najgłębszego poziomu rekurencji i uczymy kolejne bloki, „zamrażając” (czyli wyłączając z dalszej adaptacji) wagę bloków poprzednio nauczonych.

Należy zauważyć, że złożoność obliczeniowa wyrażona liczbą adaptowanych wag jest rzędu $O(N)$ dla pierwszego wariantu, podczas gdy w drugim jest równa złożoności standardowej, nie-rekurencyjnej metodzie nauczania, tj. $O(N \log N)$. Warto również zwrócić uwagę na fakt, że jakkolwiek zaprezentowane metody adaptacji ograniczają klasę przekształceń liniowych możliwych do zrealizowania przez sieć, jednak nadają się do zastosowania w przypadku wielu znanych szybkich transformat opartych na rekurencyjnym schemacie rekurencji.

Testy proponowanych metod adaptacji prowadzone były analogicznie jak w poprzednim podrozdziale. W szczególności nauce podlegała tylko część sieci odpowiadająca drugiemu etapowi algorytmu mFCT4. Na rys. 5.1 wyróżniono bloki odpowiadające pierwszemu wariantowi uczenia rekurencyjnego. Pozostałe wagi otrzymywaly swoje ostateczne wartości na drodze klonowania. Rys. 5.2 prezentuje etapy adaptacji według drugiego wariantu rekurencyjnego. Warto zauważyć, że chociaż proces adaptacji obejmuje w tym przypadku wszystkie wagi sieci, jednak jego specyficzny podział na etapy sugeruje możliwość uzyskania lepszych wyników adaptacji w stosunku do zwykłego podejścia nie-rekurencyjnego. Proces testowania obejmował trzy grupy testów weryfikujących pierwszy wariant uczenia rekurencyjnego, drugi wariant oraz zwykłe podejście nie-rekurencyjne. Spół konstrukcji zbiorów treningowych i wszystkie warunki przeprowadzania testów były takie same jak w podrozdziale 5.1. Uśrednione wyniki zaprezentowano w tablicach 5.4, 5.5, 5.6, odpowiednio. Dodatkowo, w przypadku adaptacji rekurencyjnej (tablice 5.4, 5.5) zbiory uczące zawierały również wejściowe i docelowe wektory o długości: $N=2, N=4, \dots, 4$ dla wszystkich pośrednich kroków rekurencji.

W przypadku nauczania rekurencyjnego uzyskiwana dokładność ulegała stopniowemu pogarszaniu w kolejnych krokach w efekcie wyłączania kolejnych bloków



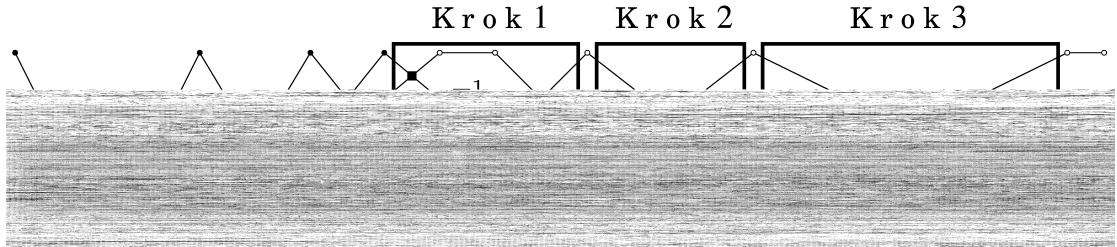
Rysunek 5.1: Etapy adaptacji zgodnie z pierwszym wariantem uczenia rekurencyjnego sieci opartej na algorytmie mFCT4 ($N=16$)

neuronów z dalszej adaptacji. Z tego względu początkowy próg błędu był ustalony poniżej dokładności docelowej $1e-9$ i był w każdym kroku mnożony przez stały współczynnik > 1 tak, aby osiągnąć ostatecznie docelową wartość $1e-9$.

Najistotniejszym faktem obserwowanym w uzyskanych wynikach jest zmniejszenie ilości czasu potrzebnego do zrealizowania procesu uczenia w obu przypadkach adaptacji rekurencyjnej (tablice 5.4, 5.5). Czas ten został zredukowany — w stosunku do sieci trenowanej bez wykorzystania rekurencji — blisko o połowę w przypadku drugiego wariantu adaptacji rekurencyjnej (tablica 5.5) i niemal trzykrotnie w przypadku wariantu pierwszego (tablica 5.4). Warte podkreślenia jest

Tablica 5.4: Rezultaty adaptacji rekurencyjnej (pierwszy wariant)

N	P	Epoki (śr)	Czas (śr) [s]	Wagi
8	4	42	0.1312	8
16	8	70	0.5436	16
32	16	116	4.278	32
64	32	191	38.3017	64



Rysunek 5.2: Etapy adaptacji zgodnie z drugim wariantem uczenia rekurencyjnego sieci opartej na algorytmie mFCT4 ($N=16$)

to, że wyniki przedstawione w tablicy 5.5 zostały uzyskane pomimo faktu, iż ilość wag biorących udział w procesie adaptacji była taka sama jak w podejściu nie-rekurencyjnym. Ta obserwacja podkreśla rolę organizacji procesu nauki i korzyści wynikające z podejścia rekurencyjnego.

Uzyskana liczba epok może być myląca. Jest ona większa o jedną-trzecią w przypadku drugiego wariantu i nieznacznie niższa (w odniesieniu do adaptacji nie-rekurencyjnej) dla wariantu pierwszego, pomimo znaczcej redukcji czasu obliczeń. Podstawowym powodem tej dysproporcji jest fakt, iż całkowita liczba epok zaprezentowana w tablicach 5.4, 5.5 zawiera również epoki składające się na

Tablica 5.5: Rezultaty adaptacji rekurencyjnej (drugi wariant)

N	P	Epoki (śr)	Czas (śr) [s]	Wagi
8	4	77	0.1982	12
16	8	128	0.9125	32
32	16	208	6.8644	80
64	32	311	61.5312	192

Tablica 5.6: Rezultaty adaptacji nie-rekurencyjnej

N	P	Epoki (śr)	Czas (śr) [s]	Wagi
8	4	74	0.2391	12
16	8	98	1.361	32
32	16	146	11.3735	80
64	32	202	109.525	192

pośrednie kroki rekurencji, w których adaptacji podlegały pod-sieci o rozmiarach $4; 8; \dots; N=2$. Średni czas trwania pojedynczej epoki jest zatem znacznie mniejszy w porównaniu z nauczaniem nie-rekurencyjnym.

Należy zauważyć, że czasy przedstawione w tablicach 5.4, 5.5 również obejmują cały proces adaptacji, tj. stanowią sumy czasów otrzymanych dla wszystkich kroków rekurencji. Naturalnie, każdy zbiór danych był użyty do adaptacji niezależnie od pozostałych.

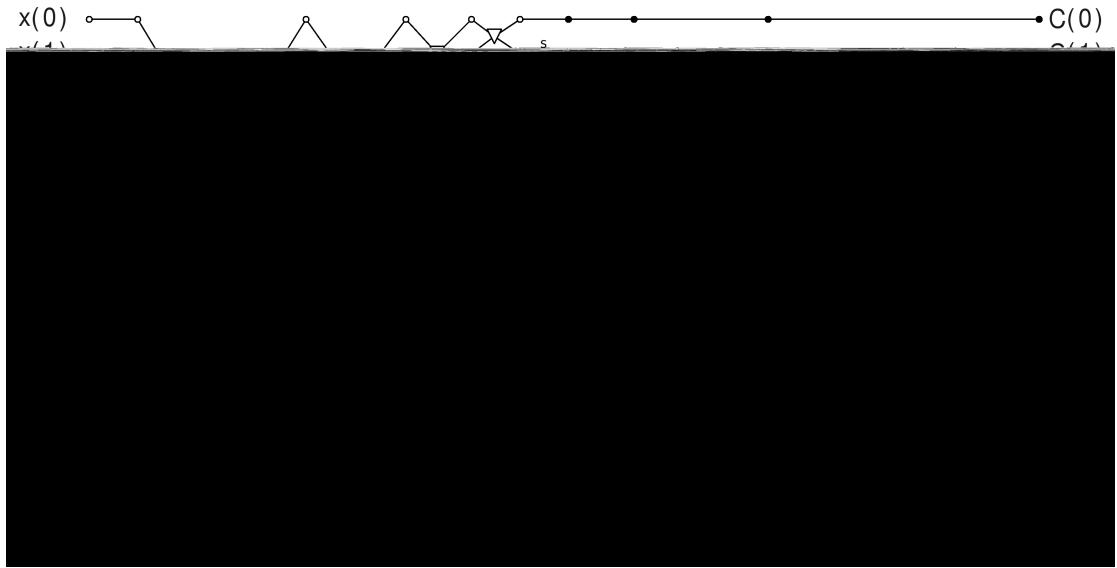
Podsumowując, oba zaprezentowane warianty adaptacji sieci typu FONN wykorzystujące rekurencyjny charakter jej struktury okazały się bardziej wydajne od zwykłego podejścia nie-rekurencyjnego, w sensie czasu trwania procesu nauki, umożliwiając jednocześnie skuteczne nauczenie się odpowiedniego przekształcenia ortogonalnego. Naturalnie, pewnym ograniczeniem adaptacji rekurencyjnej jest konieczność znajomości wektorów docelowych dla wszystkich etapów pośrednich², co w zastosowaniach praktycznych może wymagać specjalnego przygotowania danych. Z drugiej strony, ogólność przyjętych założeń wskazuje na możliwość wykorzystania zaprezentowanego podejścia do wielu innych architektur sieci opartych na rekurencyjnym sposobie konstrukcji.

5.3 Transformata Hartley'a, Fouriera oraz widmo amplitudowe Fouriera w realizacji neuronowej

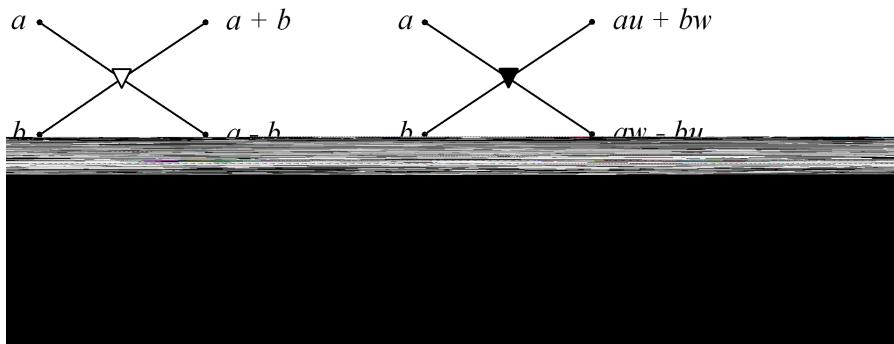
Punkt wyjścia do stworzenia struktury sieci realizującej wszystkie rozważane typy przekształceń stanowi sieć oparta na przekształceniu cosinusowym typu drugiego (FCT2):

W celu uproszczenia zapisu będziemy odtąd stosowali nieco inne oznaczenia operacji bazowych i odpowiadających im neuronów typu BOON, zgodnie z rysunkiem 5.4. Unikamy w ten sposób konieczności zmiany znaku niektórych połączeń

²przyjmując model treningu z nauczycielem

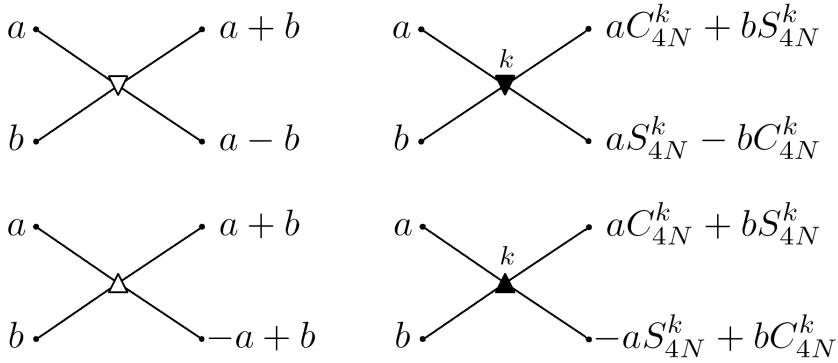
Rysunek 5.3: Struktura sieci realizującej przekształcenie FCT2 dla $N = 16$

(por. m.in. etap pierwszy grafu z rys. 2.1). Zamiast tego, zmiana znaku zostaje włączona do obliczeń wykonywanych przez same neurony operacji bazowych, co wymaga niewielkich, trywialnych zmian w sposobie ich adaptacji. Symbole u, w oznaczają wagi adaptowane w procesie nauki sieci.

Rysunek 5.4: Ogólne oznaczenia operacji bazowych (górny rzad – operacje typu q ; dolny rzad – operacje typu p)

Indeksy przy operacjach bazowych na rys. 5.3 mogą być użyte do wstępnego obliczenia wartości wag sieci zgodnie z opisem na rysunku 5.5, tak aby realizowała ona przekształcenie cosinusowe typu drugiego bez konieczności nauki. Wagi oznaczone na rysunku 5.3 jako s muszą być wówczas zainicjalizowane wartością $\sqrt{2}=2$.

Zwróćmy uwagę na fakt, że na rysunku 5.3 wprowadzono dodatkową, pierwszą warstwę dokonującą permutacji d_N (polegającej na uporządkowaniu elementów wektora wejściowego w kolejności bitowo-inwersyjnej). Równoważnie, możliwe jest



Rysunek 5.5: Operacje bazowe przekształcenia cosinusowego typu drugiego

użycie struktury pierwszego etapu dostosowanej do wektorów wejściowych z naturalną kolejnością elementów (struktura taka jednakże nie mogłaby wykonywać obliczeń *in-place*; por. rysunki 2.1, 2.3, 2.5).

5.3.1 Szybka transformata Hartley'a

Realizacja przekształcenia Hartley'a wymaga dwóch modyfikacji powyższej struktury (Rys. 5.6):

- dodania warstwy wykonującej specjalną permutację wejściową t_N , niezależnie od warstwy realizującej permutację d_N :

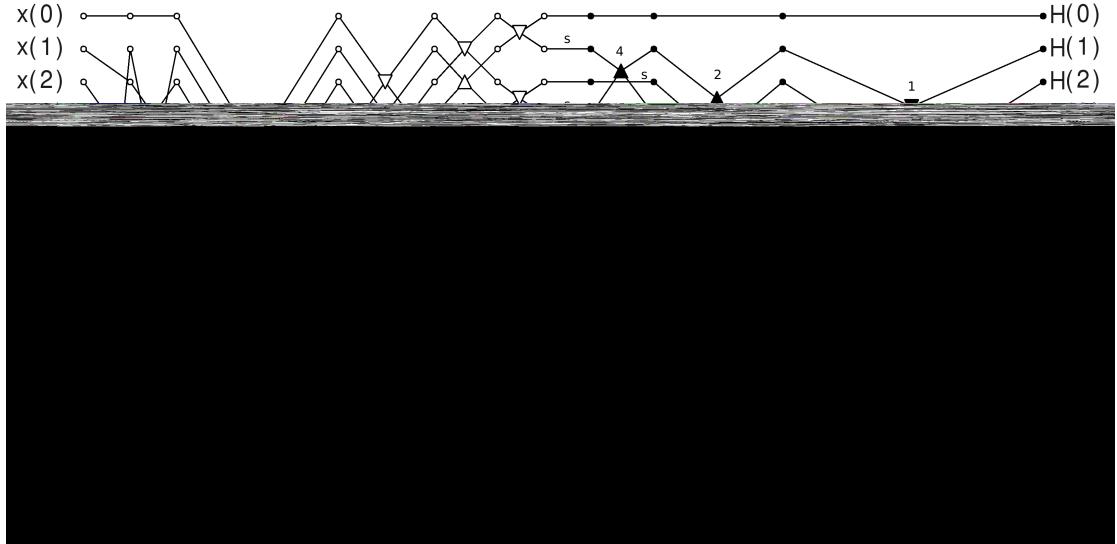
$$\begin{aligned} t_N(2n) &= n ; \\ t_N(2n+1) &= N-1-n ; \end{aligned} \tag{5.4}$$

gdzie $n = 0; 1; \dots; N-2-1$.

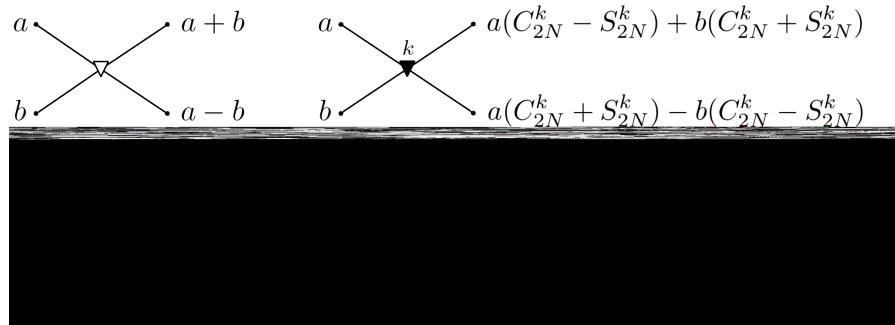
Oczywiście obie warstwy można połączyć w jedną, realizującą permutację $d_N \circ t_N$; tak czy inaczej, funkcja warstw permutujących sprowadza się do zwykłej zmiany kolejności elementów wektora wejściowego (w szczególności nie wymaga żadnych operacji arytmetycznych).

- Zmiany typu operacji bazowych z p na q w ostatniej warstwie.

Tu również możliwa jest wstępna inicjalizacja wszystkich wag sieci tak, aby realizowała ona przekształcenie Hartley'a bez konieczności nauki. W tym celu wagi w ostatniej warstwie muszą być zainicjalizowane zgodnie z rysunkiem 5.7. Zauważmy ponadto, że $N=2$ -te wyjście ostatniej warstwy nie jest tutaj mnożone przez współczynnik s (por. rys. 5.3).



Rysunek 5.6: Struktura sieci realizującej przekształcenie Hartley'a dla $N = 16$



Rysunek 5.7: Operacje bazowe przekształcenia Hartley'a

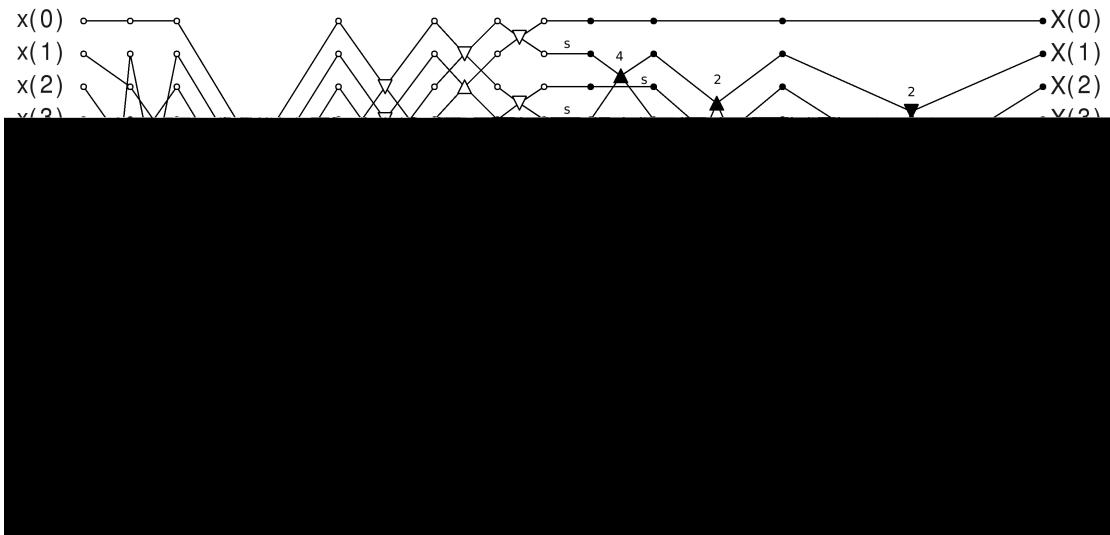
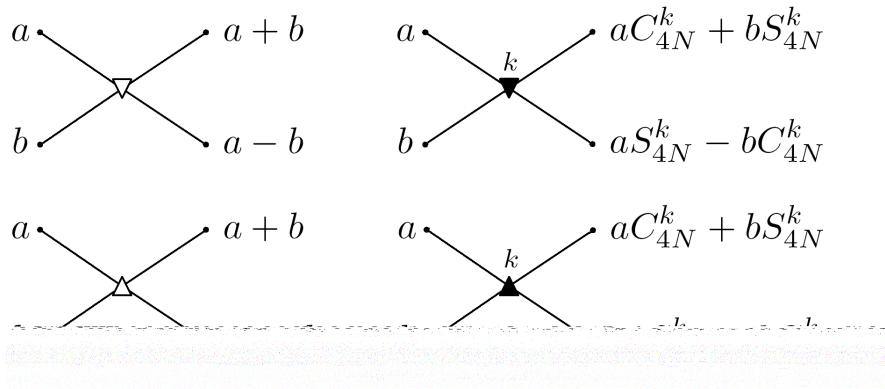
5.3.2 Szybka transformata Fouriera

Struktura i typy operacji w grafie sieci realizującej szybką transformatę Fouriera są takie same jak w przypadku transformaty Hartley'a, jednak dokonując wstępnej inicjalizacji wag należy zastosować inne wartości współczynników ostatniej warstwy (Rys. 5.8, 5.9).

Zależność między wyjściem sieci $X(n)$, a zespoloną wartością transformaty Fouriera $F_N(n) = DFT_N\{x(k)\}$ wyraża się następująco:

$$\begin{aligned}
 X(0) &= Re\{F_N(0)\} ; \\
 X(N=2) &= Re\{F_N(N=2)\} ; \\
 X(n) &= Re\{F_N(n)\} ; \\
 X(N=2+n) &= Im\{F_N(N=2-n)\} ;
 \end{aligned} \tag{5.5}$$

gdzie $n = 1; 2; \dots; N=2 - 1$.

Rysunek 5.8: Struktura sieci realizującej przekształcenie Fouriera dla $N = 16$ 

Rysunek 5.9: Operacje bazowe przekształcenia Fouriera

5.3.3 Szybka transformata Fouriera - sieć oparta na algorytmie z mnożnikami tangensowymi

Analogicznie jak w przypadku dwóch poprzednich sieci należy zastosować permutację t_N elementów wejściowych i zmienić typy i wartości wag operacji bazowych ostatniej warstwy. Tym razem jednak modyfikacjom tym podlega struktura grafu przekształcenia cosinusowego z mnożnikami tangensowymi (rys. 5.10).

Należy zauważyć, że w przypadku preinicjalizacji nie używamy współczynników s, natomiast stosujemy dodatkową warstwę wyjściową z wagami zdefiniowanymi przez wzory (2.66).

Na rysunkach 5.11, 5.12 przedstawiono zmodyfikowaną strukturę sieci realizującej szybkie przekształcenie Fouriera oraz operacje bazowe dla algorytmów z mnożnikami tangensowymi.



Rysunek 5.10: Struktura sieci realizującej przekształcenie cosinusowe typu drugiego, z mnożnikami tangensowymi, dla $N = 16$

Na specjalną uwagę zasługują współczynniki warstwy wyjściowej oznaczone na rys. 5.11 jako \hat{U}_N^k . Z uwagi na zmianę typu operacji w ostatniej warstwie motylkowej modyfikacji podlega również ostatni etap rekurencji we wzorach (2.66): współczynniki C_{4N}^k muszą być zastąpione przez współczynniki S_{2N}^k . Uwzględniając tożsamość:

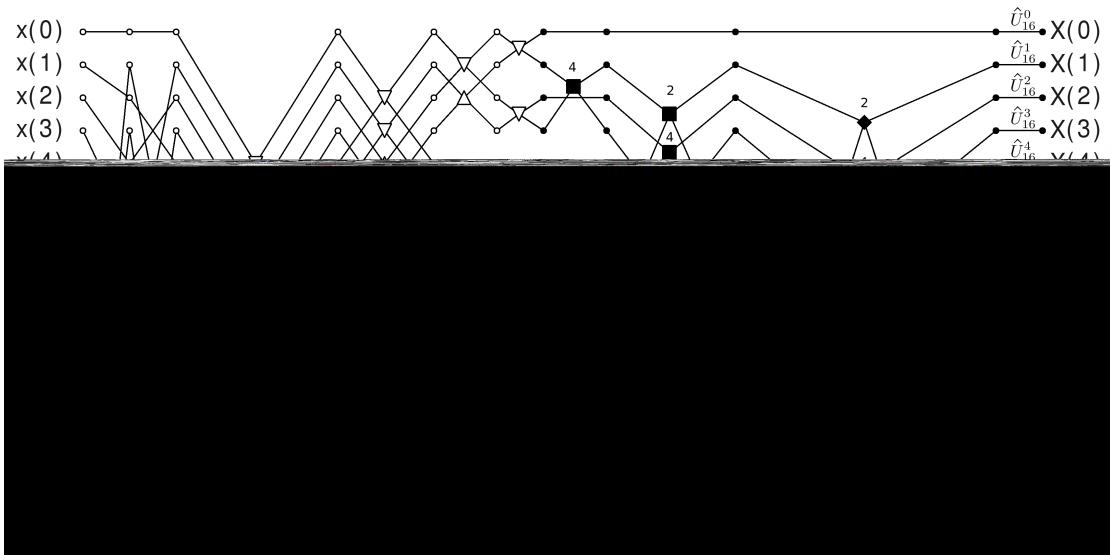
$$\sin\left(\frac{2\pi k}{2N}\right) = \cos\left(\frac{2\pi k}{4N}\right) = 2 \sin\left(\frac{2\pi k}{4N}\right); \quad (5.6)$$

współczynniki \hat{U}_N^k obliczamy jako:

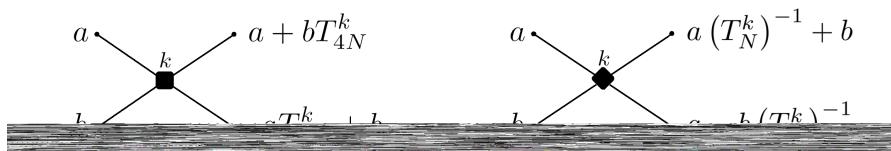
$$\hat{U}_N^k = U_N^k \cdot V_N^k; \quad (5.7)$$

gdzie U_N^k są współczynnikami warstwy wyjściowej algorytmu mFCT2 (2.66), zaś współczynniki V_N^k definiujemy następująco:

$$\begin{aligned} V_N^0 &= 1; \\ V_N^{N/2} &= \sqrt{2}; \\ V_N^k &= V_N^{N-k} = 2 \cdot \sin\left(\frac{2\pi k}{4N}\right); \\ k &= 1; 2; \dots; \frac{N}{2} - 1; \end{aligned} \quad (5.8)$$



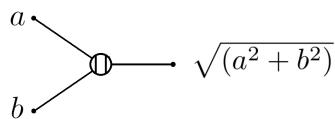
Rysunek 5.11: Struktura sieci realizującej przekształcenie Fouriera, z mnożnikami tangensowymi, dla $N = 16$



Rysunek 5.12: Operacje bazowe przekształceń z mnożnikami tangensowymi

5.3.4 Widmo amplitudowe transformaty Fouriera

W celu zapewnienia możliwości nauki widma amplitudowego należy do sieci realizującej transformację Fouriera dołączyć specjalną warstwę wyjściową złożoną z elementów obliczających moduły liczb zespolonych (rys. 5.13) Schematy kom-

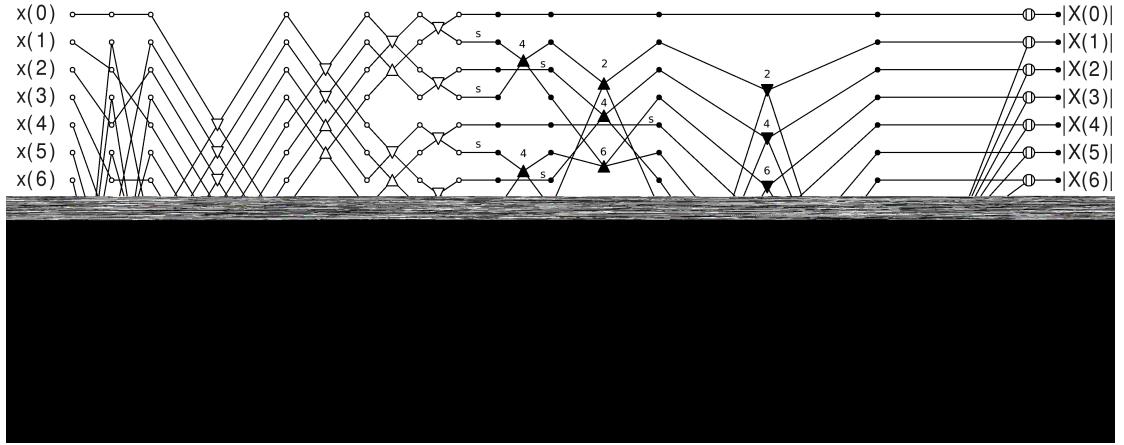


Rysunek 5.13: Element obliczający moduł liczby zespolonej

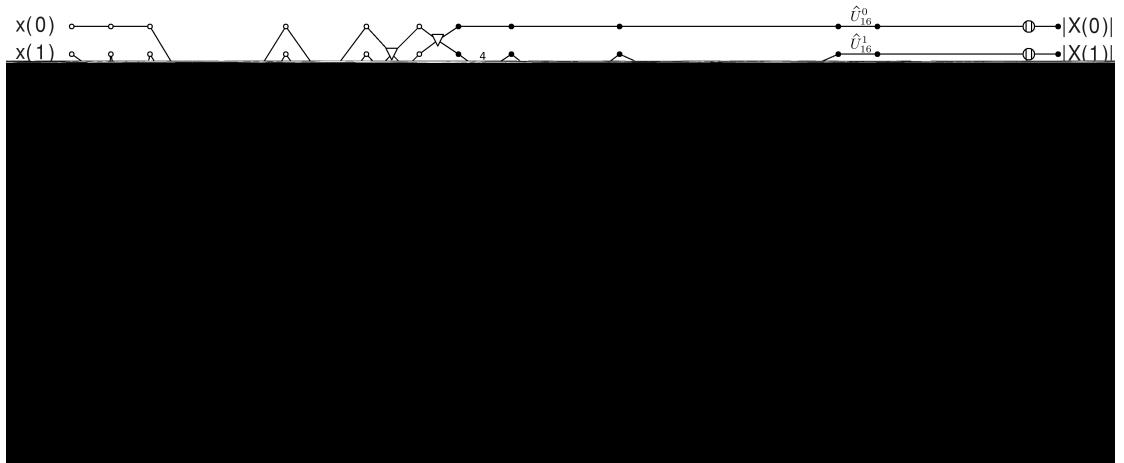
pletej sieci opartej na algorytmie bez mnożników tangensowych i z mnożnikami zaprezentowano na rysunkach 5.14 i 5.15, odpowiednio.

Nauczanie sieci zawierającej węzły z rys 5.13 opiera się na następujących obserwacjach:

1. Węzły tego typu nie posiadają wag, zatem rozmiar wektora gradientu sieci nie ulega zmianie.
2. Wartość sygnału błędu przechodzącego przez ten węzeł podczas propagacji



Rysunek 5.14: Sieć obliczająca widmo amplitudowe Fouriera dla N=16



Rysunek 5.15: Sieć obliczająca widmo amplitudowe Fouriera dla N=16 (algorytm z mnożnikami tangensowymi)

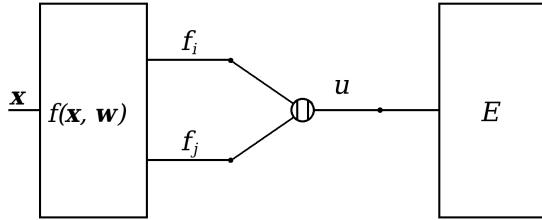
wstecznej musi zostać odpowiednio zmodyfikowana.

W celu obliczenia skorygowanej wartości sygnału błędu rozważmy schemat przedstawiony na rysunku 5.16. Schemat ten zawiera dwa podstawowe bloki obliczeniowe: blok odpowiadający naszej sieci neuronowej oraz blok reprezentujący funkcję błędu, przy czym oba bloki połączone są poprzez węzeł obliczający wartość modułu (rys. 5.16). Biorąc pod uwagę, że:

$$u(w) = \sqrt{(f_i^2(w) + f_j^2(w))} : \quad (5.9)$$

zależność wartości błędu od wektora wag przedstawia się następująco:

$$\begin{aligned} \frac{\partial E}{\partial w} &= \frac{\partial E}{\partial u} \frac{\partial u}{\partial w} = \frac{\partial E}{\partial u} \frac{1}{2u} \left(2f_i \frac{\partial f_i}{\partial w} + 2f_j \frac{\partial f_j}{\partial w} \right) = \\ &= \frac{\partial E}{\partial u} \frac{f_i \partial f_i}{u \partial w} + \frac{\partial E}{\partial u} \frac{f_j \partial f_j}{u \partial w} : \end{aligned} \quad (5.10)$$



Rysunek 5.16: Schemat obliczeń uwzględniający dodatkowy blok obliczający moduł

Stąd zatem widać, że poszukiwana modyfikacja sygnałów błędów przesyłanych podczas propagacji wstecz sprowadza się do wymnożenia ich przez iloraz wejścia/wyjścia rozważanych elementów obliczających moduły.

Pewien wyjątek stanowią elementy obliczające pierwsze oraz ostatnie wyjście sieci (rysunki 5.14 i 5.15). Zwróćmy uwagę na to, że posiadają one tylko jedno wejście odpowiadające części rzeczywistej odpowiednio: pierwszego i $N=2$ -tego wyjściowego współczynnika przekształcenia Fouriera. Ich działanie sprowadza się zatem do obliczania wartości bezwzględnej liczby rzeczywistej, przy czym informacja o faktycznym znaku ich wartości wejściowej jest używana do zanegowania wartości sygnału błędu podczas propagacji wstecz.

5.3.5 Adaptacja – rezultaty testów i wnioski

Wszystkie zaprezentowane sieci zostały zaimplementowane i poddane testowaniu zgodnie z procedurą analogiczną do przedstawionej w podrozdziale 5.1. W szczególności wektory wejściowe o różnych długościach były losowane za pomocą generatora liczb losowych o rozkładzie jednostajnym w przedziale $[-1; 1]$, adaptacja sieci dotyczyła tylko jej drugiego etapu i dla każdego zbioru danych była powtarzana dziesięciokrotnie z losowego punktu startowego w przestrzeni wag. Kryterium stopu stanowił warunek minimalizacji błędu treningowego poniżej wartości $1e-10$. Wyniki otrzymywane w przypadku poszczególnych rodzajów sieci dla P wejściowych/wyjściowych wektorów N -elementowych, uzyskane na maszynie z procesorem Intel Core 2 Duo (T7500), 2.20 GHz przedstawiono w tablicach 5.7, 5.8, 5.9.

Porównanie uzyskanych wyników wykazuje niewielką różnicę między siecią opartą na transformacie Hartley'a, a siecią opartą na transformacie Fouriera, z przewagą tej pierwszej (warto odnotować fakt, że taka sama tendencja występuje w wynikach testów szybkiej sieci neuronowej zaprezentowanych w pracy [131]). Znacznie większe różnice występują pomiędzy obiema tymi sieciami, a siecią zbudowaną w oparciu o algorytm obliczania widma amplitudowego Fouriera. W tym

Tablica 5.7: Wyniki adaptacji sieci opartej na algorytmie transformaty Hartley'a

N	P	Epoki (śr)	Epoki (std)	Czas (śr) [s]	Czas (std)
8	4	69.7	4.36	0.019	0.007
16	8	72.3	2.90	0.017	0.006
32	16	97.1	4.83	0.043	0.021
64	32	119.4	3.20	0.149	0.013
128	64	162.4	8.40	0.745	0.037
256	128	260.2	12.96	6.797	0.418
512	256	394.3	33.38	47.052	3.618
1024	512	643.8	39.98	6m 38s	23.770

Tablica 5.8: Wyniki adaptacji sieci opartej na algorytmie transformaty Fouriera

N	P	Epoki (śr)	Epoki (std)	Czas (śr) [s]	Czas (std)
8	4	77.5	5.46	0.009	0.005
16	8	101.7	11.53	0.017	0.011
32	16	125.7	7.40	0.065	0.021
64	32	147.3	16.55	0.174	0.015
128	64	196.8	13.67	0.871	0.057
256	128	307.5	41.49	8.103	0.899
512	256	482.6	52.11	57.411	5.211
1024	512	796.4	68.86	7m 35s	35.066

ostatnim przypadku potrzeba większej ilości epok i dłuższego czasu, aby uzyskać ten sam poziom błędu treningowego. Uzasadnienie tego faktu leży zapewne w tym, że zadaną wartość modułu liczby zespolonej można uzyskać dla dowolnego kąta przesunięcia fazowego, a tym samym dla wielu różnych wartości jej części rzeczywistej i urojonej. Niejednoznaczność ta w pewnym stopniu utrudnia sieci nauczenie się przekształcenia Fouriera, wymuszanego poprzez zbiór treningowy.

Analiza wpływu różnych parametrów na proces uczenia wskazała na istotność przyjętego sposobu inicjalizacji wag. Wagi początkowo losowane były z przedziału $[-1; 1]$, jednak znacznie korzystniejsze okazało się ograniczenie tego zakresu do przedziału $[0; 1]$. W przypadku inicjalizacji wag zarówno wartościami ujemnymi, jak i dodatnimi, proces adaptacji wykazywał tendencję do zatrzymywania się w minimum lokalnym, co było widoczne zwłaszcza dla większych wartości N (stosunek ilości niepowodzeń do ilości wszystkich prób adaptacji dochodzący, a niekiedy nawet przekraczający 50%). Ograniczenie się do dodatnich wartości wag podczas inicjalizacji pozwoliło na znaczące zredukowanie tego zjawiska.

Tablica 5.9: Wyniki adaptacji sieci opartej na algorytmie obliczania widma amplitudowego transformaty Fouriera

N	P	Epoki (śr)	Epoki (std)	Czas (śr) [s]	Czas (std)
8	4	160.2	24.78	0.015	0.011
16	8	173.1	22.47	0.031	0.013
32	16	196.8	20.90	0.062	0.012
64	32	203.5	15.41	0.241	0.021
128	64	266.5	19.72	1.259	0.078
256	128	397.6	47.94	10.279	0.916
512	256	739.5	98.11	1m 24s	7.852
1024	512	1371.8	389.20	11m 31s	103.833

Istota problemu wydaje się tutaj tkwić w definicji i sposobie działania neuronów typu BOON. Preinicjalizując wagi neuronów sieci zgodnie z właściwym jej szybkim algorytmem, czyli określając ich wartości docelowe w sposób deterministyczny, przekonujemy się, że większość z nich (lub nawet wszystkie, w zależności od rodzaju transformaty) jest dodatnia, z zakresu $[0; 1]$. Przykładowo, rozważając graf algorytmu przekształcenia Fouriera (rys. 5.8, 5.9), widzimy, że argumenty funkcji \sin , i \cos używanych do obliczenia wartości współczynników S_{4N}^k i C_{4N}^k w drugim etapie są ograniczone do przedziału $[0; \pi/2]$, w którym obie te funkcje przyjmują wartości dodatnie. Ponadto, zauważamy, że wagi neuronów typu BOON wyznaczają bezpośrednio wartości współczynników S_{4N}^k i C_{4N}^k dla pierwszego (górнего) wyjścia, gdzie występują one ze znakiem „+”. Zanegowane wartości współczynników pojawiające się we wzorze na obliczanie drugiego (dolnego) wyjścia nie występują jawnie jako samodzielne wagi, a znak „-” jest uwzględniony w samym algorytmie adaptacji.

Ostatecznie, po przeprowadzeniu wstępnej serii testów, przyjęto procedurę losowania wag neuronów z przedziału $[0.25; 0.75]$. To pozwoliło na ograniczenie liczby niepowodzeń w procesie adaptacji do pojedynczych przypadków. Należy zauważać, że wyniki zaprezentowane w tablicach 5.7, 5.8, 5.9 były za każdym razem obliczane jako średnia z dziesięciu *udanych* prób adaptacji, tj. zakończonych minimalizacją błędu poniżej wartości $1e-10$.

5.4 Dwuwymiarowe sieci typu FONN

Z racji istnienia dwuwymiarowych wariantów wielu szybkich przekształceń ortogonalnych (e.g. wzory (2.62), (2.63)) istnieje naturalna możliwość wykorzystania ich

do konstrukcji szybkiej ortogonalnej sieci neuronowej przeznaczonej do analizy danych dwuwymiarowych (obrazów). Należy zauważyć, że istnieją dwa podstawowe sposoby definicji szybkiego algorytmu dwuwymiarowego przekształcenia ortogonalnego: metoda „wiersze-kolumny”, wykorzystująca tzw. *własność rozłączności* danej transformaty, oraz podejście bezpośrednie. Własność rozłączności wyraża się możliwością uzyskania dwuwymiarowego przekształcenia obrazu wejściowego za pomocą przekształcenia jego wierszy algorytmem jednowymiarowym, a następnie kolumn tak uzyskanej reprezentacji – również algorytmem jednowymiarowym. Metoda bezpośrednią wymaga natomiast specjalnego opracowania algorytmu dwuwymiarowego.

W tym podrozdziale przedstawimy obie metody w odniesieniu do szybkiej transformaty cosinusowej rodzaju drugiego (2D-FCT2) [122][125]. Sieć skonstruowana według metody bezpośrednią zostanie następnie zmodyfikowana w celu umożliwienia obliczania dwuwymiarowego przekształcenia Fouriera i jego widma amplitudowego.

5.4.1 Transformata cosinusowa – metoda „wiersze-kolumny”

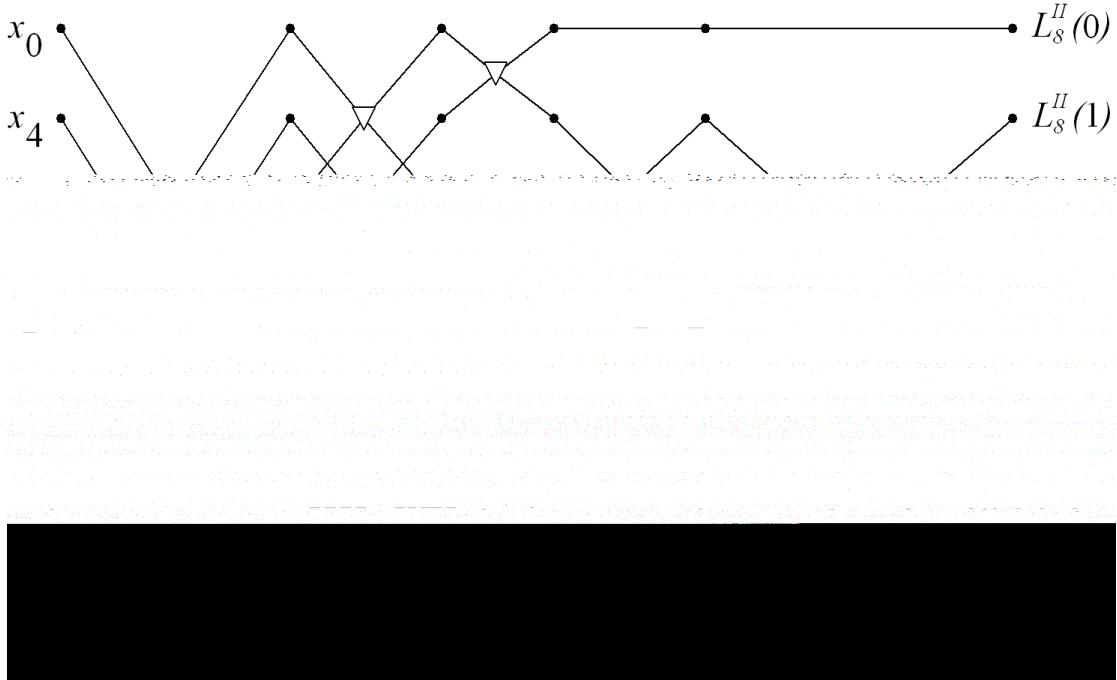
Przypomnijmy jeszcze raz schemat połączeń sieci zbudowanej na podstawie transformaty cosinusowej drugiego rodzaju (rys. 5.17). Zauważmy, że wejścia x_0, \dots, x_{N-1} muszą tu być podane w porządku bitowo-inwersyjnym, podczas gdy wyjścia otrzymywane są w kolejności naturalnej. Wyjście sieci z rys. 5.17 odpowiadające wartości $L_N^{II} (N=2)$ musi być wymnożone przez współczynnik:

$$s = \frac{\sqrt{2}}{2} : \quad (5.11)$$

W przypadku dwuwymiarowym potrzebujemy sieci zdolnej nauczyć się transformaty 2D-DCT^{II} zdefiniowanej wzorem (2.63). Dla uproszczenia założymy, że $N = M$, tj. że szerokość i wysokość obrazu wejściowego jest taka sama, aczkolwiek poniższe rozważania łatwo uogólnić na przypadek danych wejściowych reprezentowanych przez macierze prostokątne.

Projekt architektury sieci dwuwymiarowej opiera się na fakcie, iż przekształcenie 2D-DCT^{II} może być obliczone poprzez zastosowanie jednowymiarowego przekształcenia FCT2 dla każdego wiersza danych wejściowych, a następnie również jednowymiarowego przekształcenia FCT2 dla każdej kolumny. Pierwsza kolumna dwuwymiarowej transformaty cosinusowej może być zatem obliczona za pomocą sieci z rys. 5.18.

Każdy z bloków po lewej stronie oblicza jednowymiarowe przekształcenie FCT2



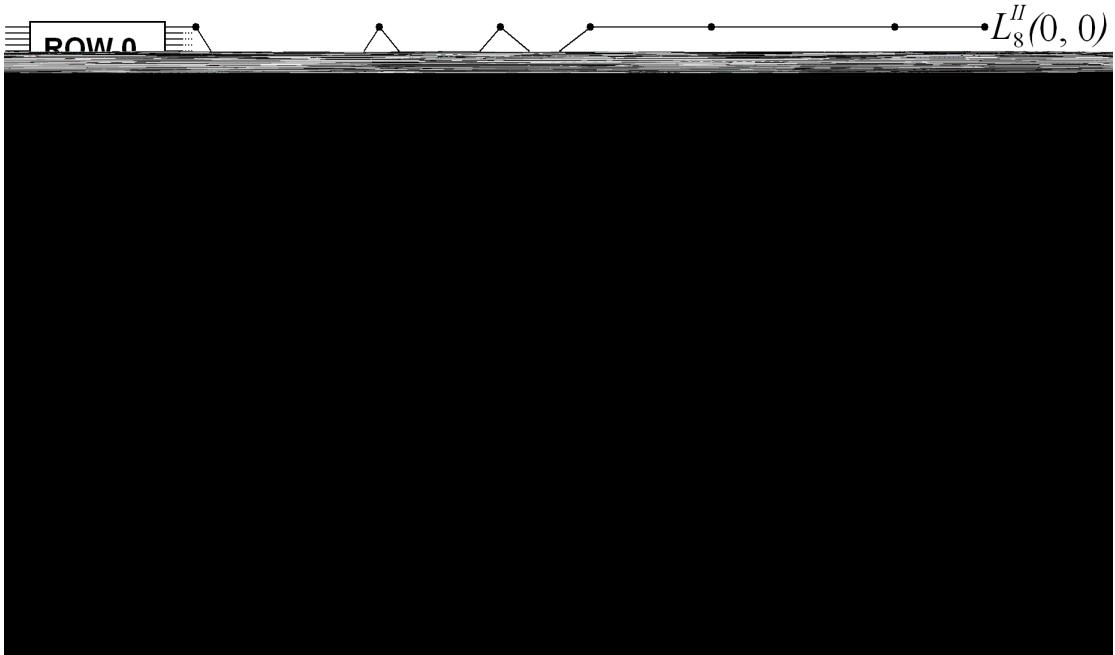
Rysunek 5.17: Struktura sieci opartej na algorytmie FCT2 dla $N = 8$ (przypadek jednowymiarowy)

pojedynczego wiersza wejścia i ma taką samą architekturę jak sieć dokonująca obliczenia pierwszej kolumny stanowiąca główną część rysunku. Zauważmy, że ta sieć ma nieco inną strukturę niż sieć zaprezentowana na rys. 5.17. Obie struktury są w gruncie rzeczy równoważne – „bardziej regularna” struktura na rys. 5.18 wymaga zmiany typów niektórych operacji bazowych drugiego etapu i wykonania dodatkowej permutacji wyjściowej P_N^i , $i = 0; 1; \dots; N - 1$. Permutacja P_N^i może być zdefiniowana następującym wzorem rekurencyjnym:

$$\begin{aligned}
 P_2^0 &= 0; \quad P_2^1 = 1; \\
 P_{2K}^k &= P_K^k; \quad P_{2K}^{K+k} = P_K^{K-k} + K; \\
 P_{2K}^0 &= 0; \quad P_{2K}^K = K; \\
 k &= 1; 2; \dots; K - 1; \\
 K &= 2; 4; 8; \dots; N=2;
 \end{aligned} \tag{5.12}$$

Należy zauważyć, że bloki obliczające wiersze są ustawione w kolejności bitowo-inwersywnej, analogicznie do pojedynczych wejść na rys. 5.17, zaś wyjścia są w kolejności naturalnej.

Pozostałe kolumny są obliczane w ten sam sposób, co można przedstawić duplikując część odpowiedzialną za obliczenie pierwszej kolumny z rys. 5.18 dla każdego



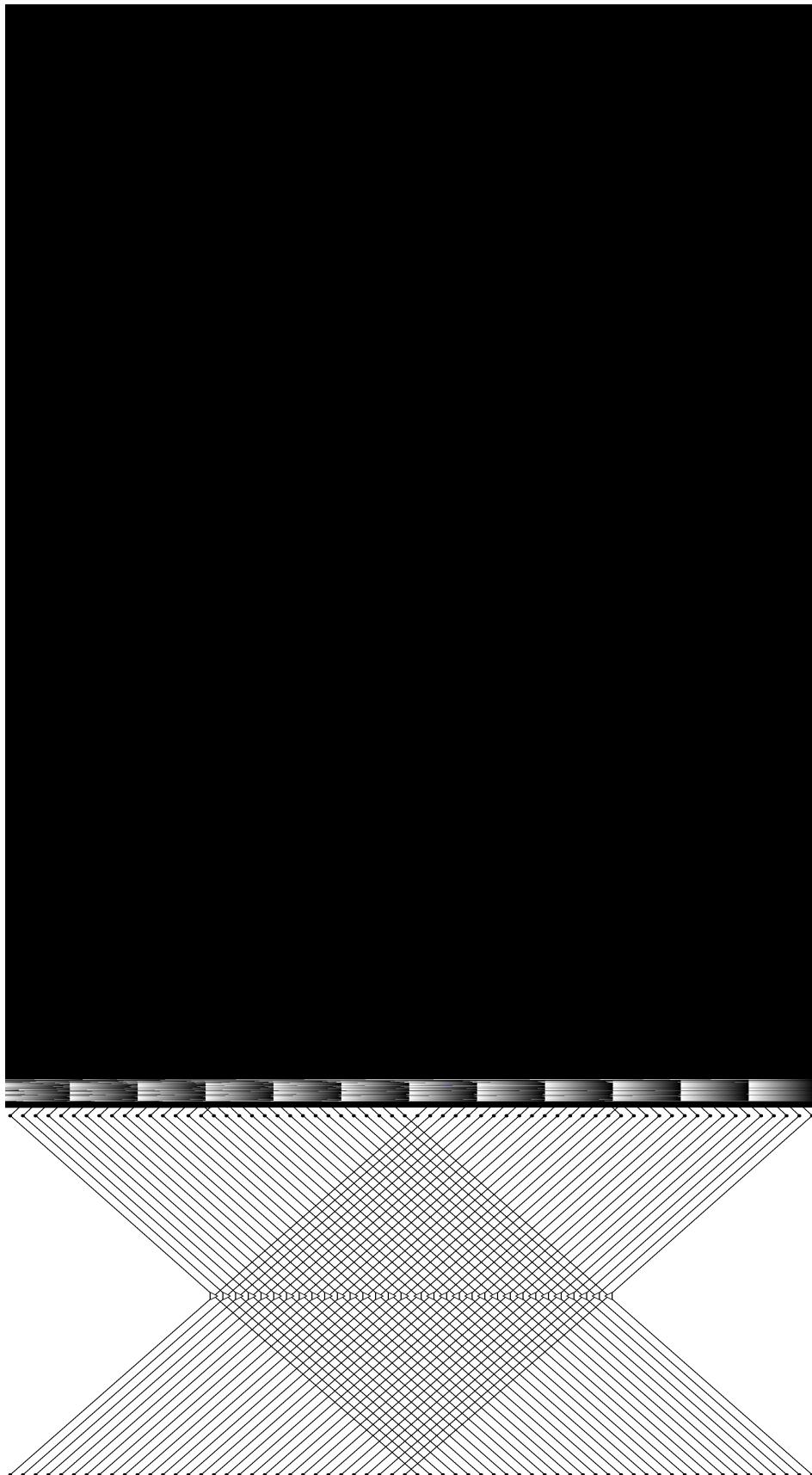
Rysunek 5.18: Dwuwymiarowa architektura sieci typu FONN (pierwsza kolumna)

wyjścia bloków obliczających wiersze. Warto również zauważyć, że liniowość bloków obliczających wiersze pozwala na przesunięcie ich w środek części obliczającej kolumny. W rezultacie tego zabiegu otrzymujemy ostateczną postać sieci dwuwymiarowej przedstawioną na rys. 5.19.

Bloki odpowiedzialne za obliczanie wierszy, zaznaczone prostokątnymi ramkami, znajdują się teraz w środkowej części sieci, co umożliwia jej ponowny podział na dwa etapy — złożonego z trywialnych operacji, zwykle nie adaptowanego oraz drugiego, podlegającego adaptacji — analogicznie do przypadku jednowymiarowego (rys. 5.17). Co więcej, struktura pierwszego etapu sieci dwuwymiarowej o rozmiarze $N \times N$ jest identyczna do pierwszego etapu sieci jednowymiarowej o rozmiarze $N \cdot N$. Bliższa analiza wykazuje, że zwykłe zduplikowanie części obliczającej kolumny na rys. 5.18 prowadzi do zakłócenia struktury pierwszego etapu w stosunku do schematu jednowymiarowego, co może być skorygowane poprzez odwrócenie porządku wyjść trywialnej części co drugiego bloku obliczającego wiersze, zgodnie z rys. 5.19.

Wejścia sieci z rys. 5.19 są ustawione w kolejności bitowo-inwersyjnej zarówno względem numerów wierszy, jak i kolumn. A zatem wymagana sekwencja wejściowa może być otrzymana jako konkatenacja wierszy tablicy $[x_{D_i, D_j}]$, gdzie $i; j = 0; 1; \dots; N - 1$ i gdzie D_i oznacza wektor indeksów w kolejności bitowo-inwersyjnej z zakresu $\{0; 1; \dots; N - 1\}$.

Wektor wyjść sieci jest konkatenacją wierszy następującej tablicy:



Rysunek 5.19: Kompletna architektura sieci typu FONN ($N \times M = 8 \times 8$)

$$\begin{bmatrix} L_{NN}^{II}(P_N^0; P_N^0) & \dots & L_{NN}^{II}(P_N^0; P_N^{N-1}) \\ L_{NN}^{II}(P_N^1; P_N^0) & \dots & L_{NN}^{II}(P_N^1; P_N^{N-1}) \\ \vdots & \ddots & \vdots \\ L_{NN}^{II}(P_N^{N-1}; P_N^0) & \dots & L_{NN}^{II}(P_N^{N-1}; P_N^{N-1}) \end{bmatrix} ;$$

gdzie P_N^i jest permutacją zdefiniowaną przez (5.12). Ponadto, wyjścia sieci odpowiadające elementom wyjściowym przekształcenia 2D-DCT^{II}: $L_{NN}^{II}(0; N=2)$, $L_{NN}^{II}(N=2; 0)$ oraz $L_{NN}^{II}(N=2; N=2)$ powinny być dodatkowo wymnożone przez S , s i s^2 (wzór (5.11)) odpowiednio.

W przypadku wykorzystania algorytmu z mnożnikami tangensowymi, współczynniki S nie są wykorzystywane, natomiast typy operacji bazowych ulegają modyfikacji. Wszystkie operacje bazowe drugiego etapu (neurony) reprezentowane przez symbol N zostają zamienione na operacje typu $\textcircled{+}$, zaś operacje reprezentowane symbolem H na $\textcircled{-}$. Dodatkowo, wartości wyjściowe $L_{NN}^{II}(p; q)$ są otrzymywane przez wymnożenie odpowiedniego wyjścia sieci przez współczynnik:

$$U_{NN}^{(p,q)} = U_N^p \cdot U_N^q ; \quad (5.13)$$

gdzie wartości U_N^p , U_N^q są zdefiniowane wzorami (2.66).

Warto zauważyć, że drugi etap sieci przedstawionej na rys. 5.19 może być rozszerzony do postaci odpowiadającej przypadkowi jednowymiarowemu, tj. sieci opartej na algorytmie jednowymiarowego przekształcenia FCT2 o rozmiarze $N \times N$, podobnie jak w przypadku pierwszego etapu. Taki zabieg implikowałby jednakże uwzględnienie kilku dodatkowych neuronów typu BOON w części drugiego odpowiedzialnej za obliczanie kolumn, co rozszerzyłoby zakres przekształceń liniowych możliwych do uzyskania kosztem pewnego spowolnienia procesu nauki. Liczba neuronów w zaprezentowanej architekturze dla wejścia o rozmiarze $N \times N$ jest równa:

$$B = N^2 (\log_2 N - 2) + 2N : \quad (5.14)$$

Zaprezentowana sieć została zaimplementowana, a jej własności zostały porównane do jednowarstwowej, „gęstej”, liniowej sieci neuronowej, o schemacie połączeń „każdy z każdym”, zawierającej neurony bez biasu [94, 107]. Warto zauważyć, że różnica pomiędzy obydwooma rodzajami sieci odpowiada różnicy pomiędzy szybką transformatą cosinusową rodzaju drugiego (FCT2), a algorytmem bezpośrednim (DCT2). Na wejścia testowanych sieci podawano macierze kwadratowe o rozmiarach 8×8 , 16×16 , 32×32 , 64×64 , uzyskane — podobnie jak w poprzednich testach — za pomocą generatora liczb losowych. Wektory docelowe obliczano za pomocą wzoru (2.63).

Tablica 5.10 prezentuje wyniki procesu adaptacji uzyskane na komputerze z procesorem Intel Celeron M, 1.40 GHz, dla gęstej liniowej sieci neuronowej (DCT2) oraz dla zaproponowanej szybkiej sieci w wariantie zwykłym (FCT2) i z mnożnikami tangensowymi(mFCT2). We wszystkich przypadkach adaptacja byłakończona po osiągnięciu wartości błędu poniżej 10^{-9} . Każdy test był powtarzany dziesięciokrotnie, a w tablicy przedstawiono wartości średnie. Wartość P oznacza, podobnie jak poprzednio, liczbę wejściowych/docelowych macierzy o rozmiarze $N \times N$.

Tablica 5.10: Wyniki adaptacji

NxN		8x8	16x16	32x32	64x64
P		32	128	512	512
Epoki (śr)	DCT2	96.2	163.7	>300	n/a
	FCT2	53	77.8	129.5	220.3
	mFCT2	26.37	26.7	30.2	35.8
Czas (śr) [s]	DCT2	0.71	90.2	>1 godz.	n/a
	FCT2	0.081	2.85	90.47	868.7
	mFCT2	0.05	1.13	30.94	188.3

Jak łatwo zauważyc, gęsta sieć (DCT2) jest mało efektywna dla większych rozmiarów danych wejściowych. Z drugiej strony, szybka ortogonalna sieć zapewnia względnie umiarkowany czas obliczeń, szczególnie w przypadku wariantu opartego na algorytmie z mnożnikami tangensowymi, co w dużej mierze wynika ze znaczającej redukcji liczby wag (tablica 5.11). Na szczególne podkreślenie zasługuje fakt, że czas zaledwie trzech minut jest wystarczający do adaptacji sieci w przypadku 512 obrazów o rozmiarze 64×64 , co może być rozdzielnością wystarczającą do zadań bezpośredniej analizy treści i rozpoznawania obrazów.

Tablica 5.11: Liczba adaptowanych wag sieci

NxN	8x8	16x16	32x32	64x64
P	32	128	512	512
DCT2	4096	65536	1048576	16777216
FCT2	160	1088	6272	33024
mFCT2	80	544	3136	16512

5.4.2 Transformata cosinusowa – metoda bezpośrednia

W tej części przedstawiona zostanie konstrukcja dwuetapowego, jednorodnego algorytmu dwuwymiarowej szybkiej transformaty cosinusowej drugiego rodzaju metodą bezpośrednią. Zaprezentujemy wszystkie etapy konstrukcji algorytmu, dla którego podstawę stanowi dekompozycja dwuwymiarowej transformaty cosinusowej II rodzaju 2D-DCT^{II}, zdefiniowanej wzorem (2.63). Skoncentrujemy się przy tym zasadniczo na samym algorytmie, pamiętając, że konwersja grafu przepływu algorytmu do postaci sieci typu FONN jest bezpośrednią i nie nastręcza większych trudności.

Wzór rozkładu

Rozdzielając sumy we wzorze (2.63) na składniki parzyste i nieparzyste względem obu indeksów $n; m$ mamy:

$$\begin{aligned}
 L_{N \times M}^{II}(p; q) = & \sum_{n=0}^{\frac{N}{2}-1} \sum_{m=0}^{\frac{M}{2}-1} x(2n; 2m) C_{4N}^{(4n+1)p} C_{4M}^{(4m+1)q} + \\
 & + \sum_{n=0}^{\frac{N}{2}-1} \sum_{m=0}^{\frac{M}{2}-1} x(2n+1; 2m) C_{4N}^{(4n+3)p} C_{4M}^{(4m+1)q} + \\
 & + \sum_{n=0}^{\frac{N}{2}-1} \sum_{m=0}^{\frac{M}{2}-1} x(2n; 2m+1) C_{4N}^{(4n+1)p} C_{4M}^{(4m+3)q} + \\
 & + \sum_{n=0}^{\frac{N}{2}-1} \sum_{m=0}^{\frac{M}{2}-1} x(2n+1; 2m+1) C_{4N}^{(4n+3)p} C_{4M}^{(4m+3)q} ;
 \end{aligned} \tag{5.15}$$

Zauważmy, że ze korzystając ze wzoru na cosinus różnicy i - odpowiednio - sumy kątów możemy w innej formie przedstawić współczynniki C :

$$\begin{aligned}
 C_{4N}^{(4n+1)p} &= C_{4N}^{2(2n+1)p-p} = C_{4(N/2)}^{(2n+1)p} C_{4N}^p + S_{4(N/2)}^{(2n+1)p} S_{4N}^p ; \\
 C_{4N}^{(4n+3)p} &= C_{4N}^{2(2n+1)p+p} = C_{4(N/2)}^{(2n+1)p} C_{4N}^p - S_{4(N/2)}^{(2n+1)p} S_{4N}^p ;
 \end{aligned} \tag{5.16}$$

i analogicznie:

$$\begin{aligned}
 C_{4M}^{(4m+1)q} &= C_{4(M/2)}^{(2m+1)q} C_{4M}^q + S_{4(M/2)}^{(2m+1)q} S_{4M}^q ; \\
 C_{4M}^{(4m+3)q} &= C_{4(M/2)}^{(2m+1)q} C_{4M}^q - S_{4(M/2)}^{(2m+1)q} S_{4M}^q ;
 \end{aligned} \tag{5.17}$$

Wzór (5.15) przyjmie teraz postać³:

$$\begin{aligned}
 L_{N \times M}^{II}(p,q) = & \sum_{n=0}^{\frac{N}{2}-1} \sum_{m=0}^{\frac{M}{2}-1} x(2n,2m) \left[C_{4(N/2)}^{(2n+1)p} C_{4N}^p + S_{4(N/2)}^{(2n+1)p} S_{4N}^p \right] \cdot \left[C_{4(M/2)}^{(2m+1)q} C_{4M}^q + S_{4(M/2)}^{(2m+1)q} S_{4M}^q \right] + \\
 & + \sum_{n=0}^{\frac{N}{2}-1} \sum_{m=0}^{\frac{M}{2}-1} x(2n+1,2m) \left[C_{4(N/2)}^{(2n+1)p} C_{4N}^p - S_{4(N/2)}^{(2n+1)p} S_{4N}^p \right] \cdot \left[C_{4(M/2)}^{(2m+1)q} C_{4M}^q + S_{4(M/2)}^{(2m+1)q} S_{4M}^q \right] + \\
 & + \sum_{n=0}^{\frac{N}{2}-1} \sum_{m=0}^{\frac{M}{2}-1} x(2n,2m+1) \left[C_{4(N/2)}^{(2n+1)p} C_{4N}^p + S_{4(N/2)}^{(2n+1)p} S_{4N}^p \right] \cdot \left[C_{4(M/2)}^{(2m+1)q} C_{4M}^q - S_{4(M/2)}^{(2m+1)q} S_{4M}^q \right] + \\
 & + \sum_{n=0}^{\frac{N}{2}-1} \sum_{m=0}^{\frac{M}{2}-1} x(2n+1,2m+1) \left[C_{4(N/2)}^{(2n+1)p} C_{4N}^p - S_{4(N/2)}^{(2n+1)p} S_{4N}^p \right] \cdot \left[C_{4(M/2)}^{(2m+1)q} C_{4M}^q - S_{4(M/2)}^{(2m+1)q} S_{4M}^q \right],
 \end{aligned} \tag{5.18}$$

skąd po wymnożeniu nawiasów i przegrupowaniu otrzymujemy:

$$\begin{aligned}
 L_{N \times M}^{II}(p,q) = & C_{4N}^p C_{4M}^q \sum_{n=0}^{\frac{N}{2}-1} \sum_{m=0}^{\frac{M}{2}-1} [x(2n,2m) + x(2n+1,2m) + x(2n,2m+1) + x(2n+1,2m+1)] C_{4(N/2)}^{(2n+1)p} C_{4(M/2)}^{(2m+1)q} + \\
 & + C_{4N}^p S_{4M}^q \sum_{n=0}^{\frac{N}{2}-1} \sum_{m=0}^{\frac{M}{2}-1} [x(2n,2m) + x(2n+1,2m) - x(2n,2m+1) - x(2n+1,2m+1)] C_{4(N/2)}^{(2n+1)p} S_{4(M/2)}^{(2m+1)q} + \\
 & + S_{4N}^p C_{4M}^q \sum_{n=0}^{\frac{N}{2}-1} \sum_{m=0}^{\frac{M}{2}-1} [x(2n,2m) - x(2n+1,2m) + x(2n,2m+1) - x(2n+1,2m+1)] S_{4(N/2)}^{(2n+1)p} C_{4(M/2)}^{(2m+1)q} + \\
 & + S_{4N}^p S_{4M}^q \sum_{n=0}^{\frac{N}{2}-1} \sum_{m=0}^{\frac{M}{2}-1} [x(2n,2m) - x(2n+1,2m) - x(2n,2m+1) + x(2n+1,2m+1)] S_{4(N/2)}^{(2n+1)p} S_{4(M/2)}^{(2m+1)q}.
 \end{aligned} \tag{5.19}$$

Współczynniki S można wyrazić za pomocą współczynników C korzystając ze wzoru na cosinus różnicy kątów w następującej zależności:

$$\begin{aligned}
 C_{4N}^{(2n+1)(N-p)} &= C_{4N}^{(2n+1)N-(2n+1)p} = \\
 &= C_{4N}^{(2n+1)N} \cdot C_{4N}^{(2n+1)p} + S_{4N}^{(2n+1)N} \cdot S_{4N}^{(2n+1)p} = \\
 &= \cos\left(\pi N + \frac{\pi}{2}\right) \cdot C_{4N}^{(2n+1)p} + \sin\left(\pi N + \frac{\pi}{2}\right) \cdot S_{4N}^{(2n+1)p} = \\
 &= 0 \cdot C_{4N}^{(2n+1)p} + (-1)^n \cdot S_{4N}^{(2n+1)p} = (-1)^n \cdot S_{4N}^{(2n+1)p}.
 \end{aligned} \tag{5.20}$$

Mnożąc obie strony przez $(-1)^n$ uzyskujemy:

$$S_{4N}^{(2n+1)p} = (-1)^n \cdot C_{4N}^{(2n+1)(N-p)}, \tag{5.21}$$

i analogicznie:

$$\begin{aligned}
 S_{4(N/2)}^{(2n+1)p} &= (-1)^n \cdot C_{4(N/2)}^{(2n+1)(N/2-p)}, \\
 S_{4(M/2)}^{(2m+1)q} &= (-1)^m \cdot C_{4(M/2)}^{(2m+1)(M/2-q)}.
 \end{aligned} \tag{5.22}$$

³ W celu ograniczenia konieczności łamania wierszy, wzory dotyczące transformat dwuwymiarowych będą od tego miejsca zapisywane mniejszą czcionką

Wzór (5.19) możemy zapisać teraz jako:

$$\begin{aligned}
 L_{N \times M}^{II}(p,q) = & C_{4N}^p C_{4M}^q \sum_{n=0}^{\frac{N}{2}-1} \sum_{m=0}^{\frac{M}{2}-1} [x(2n,2m) + x(2n+1,2m) + \\
 & + x(2n,2m+1) + x(2n+1,2m+1)] C_{4(N/2)}^{(2n+1)p} C_{4(M/2)}^{(2m+1)q} + \\
 & + C_{4N}^p S_{4M}^q \sum_{n=0}^{\frac{N}{2}-1} \sum_{m=0}^{\frac{M}{2}-1} [x(2n,2m) + x(2n+1,2m) - \\
 & - x(2n,2m+1) - x(2n+1,2m+1)] (-1)^m C_{4(N/2)}^{(2n+1)p} C_{4(M/2)}^{(2m+1)(M/2-q)} + \\
 & + S_{4N}^p C_{4M}^q \sum_{n=0}^{\frac{N}{2}-1} \sum_{m=0}^{\frac{M}{2}-1} [x(2n,2m) - x(2n+1,2m) + \\
 & + x(2n,2m+1) - x(2n+1,2m+1)] (-1)^n C_{4(N/2)}^{(2n+1)(N/2-p)} C_{4(M/2)}^{(2m+1)q} + \\
 & + S_{4N}^p S_{4M}^q \sum_{n=0}^{\frac{N}{2}-1} \sum_{m=0}^{\frac{M}{2}-1} [x(2n,2m) - x(2n+1,2m) - \\
 & - x(2n,2m+1) + x(2n+1,2m+1)] (-1)^{n+m} C_{4(N/2)}^{(2n+1)(N/2-p)} C_{4(M/2)}^{(2m+1)(M/2-q)}.
 \end{aligned} \tag{5.23}$$

W ten sposób uzyskaliśmy wzór rozkładu algorytmu 2D-FCT2. Istotnie, przyjmując oznaczenia:

$$\begin{aligned}
 L_1(p,q) = & \text{DCT}_{\frac{N}{2} \times \frac{M}{2}}^{II} \{x(2n,2m) + x(2n+1,2m) + x(2n,2m+1) + x(2n+1,2m+1)\}, \\
 L_2(p,q) = & \text{DCT}_{\frac{N}{2} \times \frac{M}{2}}^{II} \{(-1)^m (x(2n,2m) + x(2n+1,2m) - x(2n,2m+1) - x(2n+1,2m+1))\}, \\
 L_3(p,q) = & \text{DCT}_{\frac{N}{2} \times \frac{M}{2}}^{II} \{(-1)^n (x(2n,2m) - x(2n+1,2m) + x(2n,2m+1) - x(2n+1,2m+1))\}, \\
 L_4(p,q) = & \text{DCT}_{\frac{N}{2} \times \frac{M}{2}}^{II} \{(-1)^{m+n} (x(2n,2m) - x(2n+1,2m) - x(2n,2m+1) + x(2n+1,2m+1))\},
 \end{aligned} \tag{5.24}$$

$$\begin{aligned} C_{4M}^{M-q} &= S_{4M}^q, \\ S_{4M}^{M-q} &= C_{4M}^q. \end{aligned} \quad (5.29)$$

Stosujemy je do obliczenia na podstawie wzoru (5.19) transformaty dla:

$L_{N \times M}^{II}(p; M - q); L_{N \times M}^{II}(N - p; q)$ oraz $L_{N \times M}^{II}(N - p; M - q)$. Wykorzystując ponownie zależność (5.22) otrzymujemy ostatecznie procedurę podstawową dwuwymiarowej transformaty cosinusowej II rodzaju:

$$\begin{aligned} L_{N \times M}^{II}(p, q) &= C_{4N}^p C_{4M}^q L_1(p, q) + C_{4N}^p S_{4M}^q L_2(p, \frac{M}{2} - q) + S_{4N}^p C_{4M}^q L_3(\frac{N}{2} - p, q) + S_{4N}^p S_{4M}^q L_4(\frac{N}{2} - p, \frac{M}{2} - q), \\ L_{N \times M}^{II}(p, M - q) &= -C_{4N}^p S_{4M}^q L_1(p, q) + C_{4N}^p C_{4M}^q L_2(p, \frac{M}{2} - q) - S_{4N}^p S_{4M}^q L_3(\frac{N}{2} - p, q) + S_{4N}^p C_{4M}^q L_4(\frac{N}{2} - p, \frac{M}{2} - q), \\ L_{N \times M}^{II}(N - p, q) &= -S_{4N}^p C_{4M}^q L_1(p, q) - S_{4N}^p S_{4M}^q L_2(p, \frac{M}{2} - q) + C_{4N}^p C_{4M}^q L_3(\frac{N}{2} - p, q) + C_{4N}^p S_{4M}^q L_4(\frac{N}{2} - p, \frac{M}{2} - q), \\ L_{N \times M}^{II}(N - p, M - q) &= S_{4N}^p S_{4M}^q L_1(p, q) - S_{4N}^p C_{4M}^q L_2(p, \frac{M}{2} - q) - C_{4N}^p S_{4M}^q L_3(\frac{N}{2} - p, q) + C_{4N}^p C_{4M}^q L_4(\frac{N}{2} - p, \frac{M}{2} - q), \end{aligned} \quad (5.30)$$

gdzie

$$\begin{aligned} L_1(p, q) &= \text{DCT}_{\frac{N}{2} \times \frac{M}{2}}^{II} \{x(2n, 2m) + x(2n+1, 2m) + x(2n, 2m+1) + x(2n+1, 2m+1)\}, \\ L_2(p, q) &= \text{DCT}_{\frac{N}{2} \times \frac{M}{2}}^{II} \{(-1)^m (x(2n, 2m) + x(2n+1, 2m) - x(2n, 2m+1) - x(2n+1, 2m+1))\}, \\ L_3(p, q) &= \text{DCT}_{\frac{N}{2} \times \frac{M}{2}}^{II} \{(-1)^n (x(2n, 2m) - x(2n+1, 2m) + x(2n, 2m+1) - x(2n+1, 2m+1))\}, \\ L_4(p, q) &= \text{DCT}_{\frac{N}{2} \times \frac{M}{2}}^{II} \{(-1)^{m+n} (x(2n, 2m) - x(2n+1, 2m) - x(2n, 2m+1) + x(2n+1, 2m+1))\}, \end{aligned} \quad (5.31)$$

$n = 0, 1, \dots, N/2 - 1,$
 $m = 0, 1, \dots, M/2 - 1,$
 $p = 1, 2, \dots, N/2 - 1,$
 $q = 1, 2, \dots, M/2 - 1.$

Ponadto dla $p = 0, q = 0, p = N=2$ lub $q = M=2$ możliwe jest zredukowanie ilości obliczeń:

$$\begin{aligned} L_{N \times M}^{II}(0, 0) &= L_1(0, 0), \\ L_{N \times M}^{II}(N/2, M/2) &= 1/2 \cdot L_4(0, 0), \\ L_{N \times M}^{II}(0, M/2) &= \sqrt{2}/2 \cdot L_2(0, 0), \\ L_{N \times M}^{II}(N/2, 0) &= \sqrt{2}/2 \cdot L_3(0, 0), \\ L_{N \times M}^{II}(0, q) &= C_{4M}^q L_1(0, q) + S_{4M}^q L_2(0, \frac{M}{2} - q), \\ L_{N \times M}^{II}(0, M - q) &= -S_{4M}^q L_1(0, q) + C_{4M}^q L_2(0, \frac{M}{2} - q), \\ L_{N \times M}^{II}(p, 0) &= C_{4N}^p L_1(p, 0) + S_{4N}^p L_3(\frac{N}{2} - p, 0), \\ L_{N \times M}^{II}(N - p, 0) &= -S_{4N}^p L_1(p, 0) + C_{4N}^p L_3(\frac{N}{2} - p, 0), \end{aligned} \quad (5.32)$$

$$\begin{aligned}
L_{N \times M}^{II}(N/2, q) &= \sqrt{2}/2 \cdot C_{4M}^q L_3(0, q) + \sqrt{2}/2 \cdot S_{4M}^q L_4(0, \frac{M}{2} - q) , \\
L_{N \times M}^{II}(N/2, M-q) &= -\sqrt{2}/2 \cdot S_{4M}^q L_3(0, q) + \sqrt{2}/2 \cdot C_{4M}^q L_4(0, \frac{M}{2} - q) , \\
L_{N \times M}^{II}(p, M/2) &= \sqrt{2}/2 \cdot C_{4N}^p L_2(p, 0) + \sqrt{2}/2 \cdot S_{4N}^p L_4(\frac{N}{2} - p, 0) , \\
L_{N \times M}^{II}(N-p, M/2) &= -\sqrt{2}/2 \cdot S_{4N}^p L_2(p, 0) + \sqrt{2}/2 \cdot C_{4N}^p L_4(\frac{N}{2} - p, 0) .
\end{aligned}$$

Graf algorytmu

W dalszej części podamy sposób implementacji zaprezentowanej procedury podstawowej dla obrazów kwadratowych o wymiarach $N \times N$. Uzyskana procedura podstawowa może być wykorzystana do konstrukcji grafu algorytmu 2D-FCT2 o “płaskiej” strukturze, analogicznej do grafów przekształceń jednowymiarowych. W tym celu należy ustawić elementy dwuwymiarowej tablicy wejściowej (obrazu) i wyjściowej (widma) w jednowymiarowy ciąg. Kolejność w jakiej szeregujemy elementy widma definiujemy za pomocą funkcji $f_{N \times N}(p; q)$ zwracającej indeks elementu o współrzędnych p, q w wyjściowym ciągu. W niniejszym opracowaniu przyjęto funkcję $f_{N \times N}$ o charakterze rekurencyjnym:

$$\begin{aligned}
f_{1 \times 1}(0, 0) &= 0 , \\
f_{2K \times 2K}(p, q) &= f_{K \times K}(p, q) , \\
f_{2K \times 2K}(p, q+K) &= f_{K \times K}(p, q) + K^2 , \\
f_{2K \times 2K}(p+K, q) &= f_{K \times K}(p, q) + 2K^2 , \\
f_{2K \times 2K}(p+K, q+K) &= f_{K \times K}(p, q) + 3K^2 , \\
p, q &= 0, 1, \dots, K-1 , \\
K &= 1, 2, 4, \dots, \frac{N}{2} .
\end{aligned} \tag{5.33}$$

Wartości funkcji $f_{2 \times 2}, f_{4 \times 4}, f_{8 \times 8}$ zaprezentowano w tablicy 5.12.

Tablica 5.12: Kolejność elementów widma 2D-FCT2

<table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>0</td><td>1</td><td>4</td><td>5</td></tr> <tr><td>2</td><td>3</td><td>6</td><td>7</td></tr> <tr><td>8</td><td>9</td><td>12</td><td>13</td></tr> <tr><td>10</td><td>11</td><td>14</td><td>15</td></tr> </table> 2×2	0	1	4	5	2	3	6	7	8	9	12	13	10	11	14	15	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>0</td><td>1</td><td>4</td><td>5</td></tr> <tr><td>2</td><td>3</td><td>6</td><td>7</td></tr> <tr><td>8</td><td>9</td><td>12</td><td>13</td></tr> <tr><td>10</td><td>11</td><td>14</td><td>15</td></tr> </table> 4×4	0	1	4	5	2	3	6	7	8	9	12	13	10	11	14	15	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>0</td><td>1</td><td>4</td><td>5</td><td>16</td><td>17</td><td>20</td><td>21</td></tr> <tr><td>2</td><td>3</td><td>6</td><td>7</td><td>18</td><td>19</td><td>22</td><td>23</td></tr> <tr><td>8</td><td>9</td><td>12</td><td>13</td><td>24</td><td>25</td><td>28</td><td>29</td></tr> <tr><td>10</td><td>11</td><td>14</td><td>15</td><td>26</td><td>27</td><td>30</td><td>31</td></tr> <tr><td>32</td><td>33</td><td>36</td><td>37</td><td>48</td><td>49</td><td>52</td><td>53</td></tr> <tr><td>34</td><td>35</td><td>38</td><td>39</td><td>50</td><td>51</td><td>54</td><td>55</td></tr> <tr><td>40</td><td>41</td><td>44</td><td>45</td><td>56</td><td>57</td><td>60</td><td>61</td></tr> <tr><td>42</td><td>43</td><td>46</td><td>47</td><td>58</td><td>59</td><td>62</td><td>63</td></tr> </table> 8×8	0	1	4	5	16	17	20	21	2	3	6	7	18	19	22	23	8	9	12	13	24	25	28	29	10	11	14	15	26	27	30	31	32	33	36	37	48	49	52	53	34	35	38	39	50	51	54	55	40	41	44	45	56	57	60	61	42	43	46	47	58	59	62	63
0	1	4	5																																																																																															
2	3	6	7																																																																																															
8	9	12	13																																																																																															
10	11	14	15																																																																																															
0	1	4	5																																																																																															
2	3	6	7																																																																																															
8	9	12	13																																																																																															
10	11	14	15																																																																																															
0	1	4	5	16	17	20	21																																																																																											
2	3	6	7	18	19	22	23																																																																																											
8	9	12	13	24	25	28	29																																																																																											
10	11	14	15	26	27	30	31																																																																																											
32	33	36	37	48	49	52	53																																																																																											
34	35	38	39	50	51	54	55																																																																																											
40	41	44	45	56	57	60	61																																																																																											
42	43	46	47	58	59	62	63																																																																																											

Elementy obrazu wejściowego podawane są w porządku bitowo-inwersyjnym, definiowanym przez funkcję $g_{N \times N}(n; m)$:

$$\begin{aligned}
 g_{1 \times 1}(0,0) &= 0, \\
 g_{2K \times 2K}(n,m) &= g_{K \times K}(n,m) \cdot 4, \\
 g_{2K \times 2K}(n,m+K) &= g_{K \times K}(n,m) \cdot 4 + 1, \\
 g_{2K \times 2K}(n+K,m) &= g_{K \times K}(n,m) \cdot 4 + 2, \\
 g_{2K \times 2K}(n+K,m+K) &= g_{K \times K}(n,m) \cdot 4 + 3, \\
 n,m &= 0,1,\dots,K-1, \\
 K &= 1,2,4,\dots,\frac{N}{2}.
 \end{aligned} \tag{5.34}$$

Wartości funkcji $g_{2 \times 2}$, $g_{4 \times 4}$, $g_{8 \times 8}$ zaprezentowano w tablicy 5.13.

Tablica 5.13: Kolejność elementów obrazu wejściowego 2D-FCT2

2×2	4×4	8×8																																																																																				
<table border="1"> <tbody> <tr><td>0</td><td>1</td></tr> <tr><td>2</td><td>3</td></tr> </tbody> </table>	0	1	2	3	<table border="1"> <tbody> <tr><td>0</td><td>4</td><td>1</td><td>5</td></tr> <tr><td>8</td><td>12</td><td>9</td><td>13</td></tr> <tr><td>2</td><td>6</td><td>3</td><td>7</td></tr> <tr><td>10</td><td>14</td><td>11</td><td>15</td></tr> </tbody> </table>	0	4	1	5	8	12	9	13	2	6	3	7	10	14	11	15	<table border="1"> <tbody> <tr><td>0</td><td>16</td><td>4</td><td>20</td><td>1</td><td>17</td><td>5</td><td>21</td></tr> <tr><td>32</td><td>48</td><td>36</td><td>52</td><td>33</td><td>49</td><td>37</td><td>53</td></tr> <tr><td>8</td><td>24</td><td>12</td><td>28</td><td>9</td><td>25</td><td>13</td><td>29</td></tr> <tr><td>40</td><td>56</td><td>44</td><td>60</td><td>41</td><td>57</td><td>45</td><td>61</td></tr> <tr><td>2</td><td>18</td><td>6</td><td>22</td><td>3</td><td>19</td><td>7</td><td>23</td></tr> <tr><td>34</td><td>50</td><td>38</td><td>54</td><td>35</td><td>51</td><td>39</td><td>55</td></tr> <tr><td>10</td><td>26</td><td>14</td><td>30</td><td>11</td><td>27</td><td>15</td><td>31</td></tr> <tr><td>42</td><td>58</td><td>46</td><td>62</td><td>43</td><td>59</td><td>47</td><td>63</td></tr> </tbody> </table>	0	16	4	20	1	17	5	21	32	48	36	52	33	49	37	53	8	24	12	28	9	25	13	29	40	56	44	60	41	57	45	61	2	18	6	22	3	19	7	23	34	50	38	54	35	51	39	55	10	26	14	30	11	27	15	31	42	58	46	62	43	59	47	63
0	1																																																																																					
2	3																																																																																					
0	4	1	5																																																																																			
8	12	9	13																																																																																			
2	6	3	7																																																																																			
10	14	11	15																																																																																			
0	16	4	20	1	17	5	21																																																																															
32	48	36	52	33	49	37	53																																																																															
8	24	12	28	9	25	13	29																																																																															
40	56	44	60	41	57	45	61																																																																															
2	18	6	22	3	19	7	23																																																																															
34	50	38	54	35	51	39	55																																																																															
10	26	14	30	11	27	15	31																																																																															
42	58	46	62	43	59	47	63																																																																															

Procedura podstawowa szybkich algorytmów jednowymiarowych definiuje metodę obliczenia N -punktowego przekształcenia na podstawie dwóch przekształceń $N=2$ -punktowych. W dwuetapowych, jednorodnych algorytmach wymaga to typowo wprowadzenia po jednej warstwie operacji dwupunktowych na wejściu i na wyjściu bloków przekształceń $N=2$ -punktowych.

W przypadku dwuwymiarowym wzory (5.30), (5.31), definiujące metodę obliczania przekształcenia $N \times N$ -punktowego na podstawie czterech przekształceń $N=2 \times N=2$ -punktowych, wskazują na konieczność zastosowania warstw operacji czteropunktowych. Możemy jednak zapisać je w postaci (5.35), (5.36) umożliwiającej użycie operacji dwupunktowych.

$$\begin{aligned}
L_{N \times M}^{II}(p, q) &= C_{4N}^p \left(C_{4M}^q L_1(p, q) + S_{4M}^q L_2(p, \frac{M}{2} - q) \right) + S_{4N}^p \left(C_{4M}^q L_3(\frac{N}{2} - p, q) + S_{4M}^q L_4(\frac{N}{2} - p, \frac{M}{2} - q) \right), \\
L_{N \times M}^{II}(p, M-q) &= C_{4N}^p \left(-S_{4M}^q L_1(p, q) + C_{4M}^q L_2(p, \frac{M}{2} - q) \right) + S_{4N}^p \left(-S_{4M}^q L_3(\frac{N}{2} - p, q) + C_{4M}^q L_4(\frac{N}{2} - p, \frac{M}{2} - q) \right), \\
L_{N \times M}^{II}(N-p, q) &= -S_{4N}^p \left(C_{4M}^q L_1(p, q) + S_{4M}^q L_2(p, \frac{M}{2} - q) \right) + C_{4N}^p \left(C_{4M}^q L_3(\frac{N}{2} - p, q) + S_{4M}^q L_4(\frac{N}{2} - p, \frac{M}{2} - q) \right), \\
L_{N \times M}^{II}(N-p, M-q) &= -S_{4N}^p \left(-S_{4M}^q L_1(p, q) + C_{4M}^q L_2(p, \frac{M}{2} - q) \right) + C_{4N}^p \left(-S_{4M}^q L_3(\frac{N}{2} - p, q) + C_{4M}^q L_4(\frac{N}{2} - p, \frac{M}{2} - q) \right),
\end{aligned} \tag{5.35}$$

gdzie

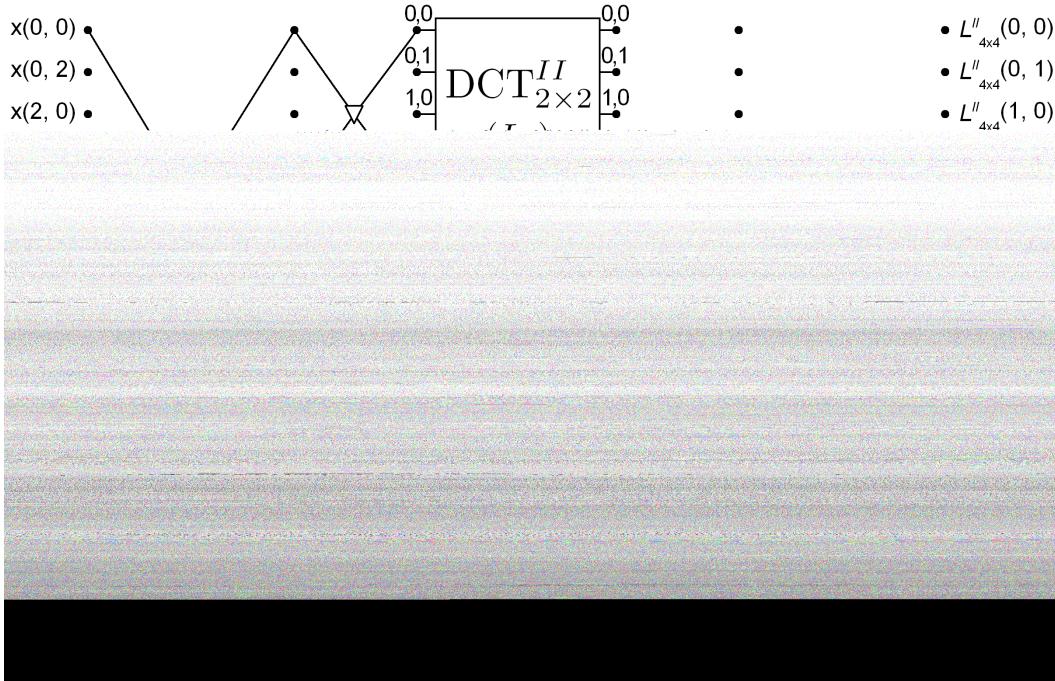
$$\begin{aligned}
L_1(p, q) &= \text{DCT}_{\frac{N}{2} \times \frac{M}{2}}^{II} \{ [x(2n, 2m) + x(2n+1, 2m)] + [x(2n, 2m+1) + x(2n+1, 2m+1)] \}, \\
L_2(p, q) &= \text{DCT}_{\frac{N}{2} \times \frac{M}{2}}^{II} \{ (-1)^m ([x(2n, 2m) + x(2n+1, 2m)] - [x(2n, 2m+1) + x(2n+1, 2m+1)]) \}, \\
L_3(p, q) &= \text{DCT}_{\frac{N}{2} \times \frac{M}{2}}^{II} \{ (-1)^n ([x(2n, 2m) - x(2n+1, 2m)] + [x(2n, 2m+1) - x(2n+1, 2m+1)]) \}, \\
L_4(p, q) &= \text{DCT}_{\frac{N}{2} \times \frac{M}{2}}^{II} \{ (-1)^{m+n} ([x(2n, 2m) - x(2n+1, 2m)] - [x(2n, 2m+1) - x(2n+1, 2m+1)]) \}, \\
n &= 0, 1, \dots, N/2-1, \\
m &= 0, 1, \dots, M/2-1, \\
p &= 1, 2, \dots, N/2-1, \\
q &= 1, 2, \dots, M/2-1.
\end{aligned} \tag{5.36}$$

Tak przedstawiona procedura podstawowa wymaga wprowadzenia *dwoch* warstw operacji dwupunktowych na wejściu i również dwóch na wyjściu bloków przekształceń $N=2 \times N=2$ -punktowych.

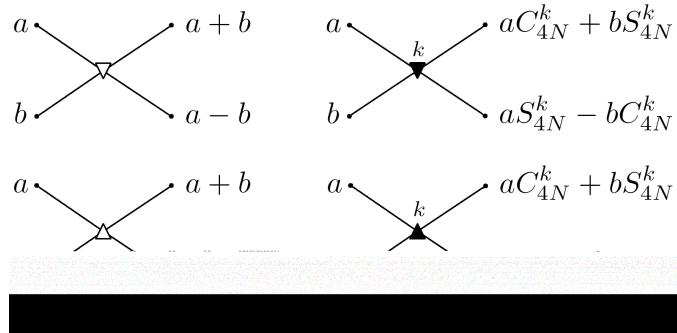
Rozważmy przykład zastosowania wzorów (5.35), (5.36) na przykładzie obliczenia transformaty 4×4 -punktowej. Na rysunku (5.20) przedstawiono obliczenia wartości wejściowych bloków transformaty 2×2 -punktowej dla $n = 0; m = 0$ oraz wartości wyjściowych dla $p = 1; q = 1$. Znaczenie operacji podstawowych wyjaśniono na rysunku (5.21).

Łatwo zauważyc, że obliczenia pozostałych wartości wejściowych i wyjściowych przebiegają analogicznie, co zaprezentowano na rysunku 5.22. Na rysunku tym bloki transformaty 2×2 -punktowej zostały przedstawione w jawnej postaci; uzyskaliśmy tym samym kompletny graf przepływu 4×4 -punktowej transformaty cosinusowej II rodzaju.

Kilka rzeczy zasługuje tu na uwagę. Po pierwsze, ze względem na uproszczone obliczenia dla $p = 0$ lub $q = 0$

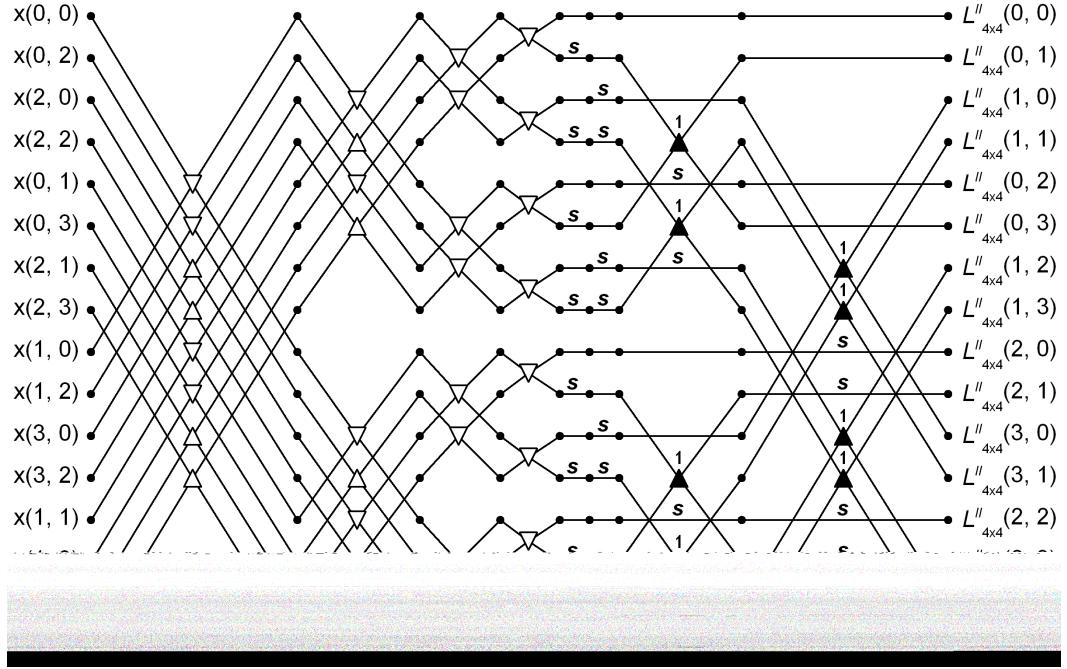


Rysunek 5.20: Procedura podstawowa 4×4 -punktowej transformaty cosinusowej dla $n = 0; m = 0; p = 1; q = 1$



Rysunek 5.21: Operacje podstawowe $N \times N$ -punktowej transformaty cosinusowej

nej realizacji można naturalnie uniknąć wielokrotnego mnożenia stosując współczynnik s w odpowiedniej potędze. Po drugie, ze względu na obecność czynników $(-1)^n; (-1)^m; (-1)^{n+m}$ we wzorze (5.36), niektóre operacje bazowe w warstwach wejściowych uległy odwróceniu, tworząc schemat odbiegający od pierwszego etapu znanych jednorodnych przekształceń jednowymiarowych. Okazuje się, że jest możliwe zastosowanie w warstwach wejściowych schematu identycznego jak w przypadku transformat jednowymiarowych (pierwszy etap dwuetapowego algorytmu jest wówczas dokładnie taki sam dla dwuwymiarowej transformaty $N \times N$ -punktowej, jak i dla jednowymiarowego przekształcenia $N \cdot N$ -punktowego). Wymaga to zastosowania dodatkowej permutacji h elementów wejściowych. Per-

Rysunek 5.22: Graf przepływu 4×4 -punktowej transformaty cosinusowej

mutacji h definiujemy następująco:

$$h = h_J \circ \dots \circ h_1 \circ h_0 , \quad (5.37)$$

$$J = 2 \log_2(N/2) ,$$

gdzie permutacje składowe h_j są zdefiniowane jako:

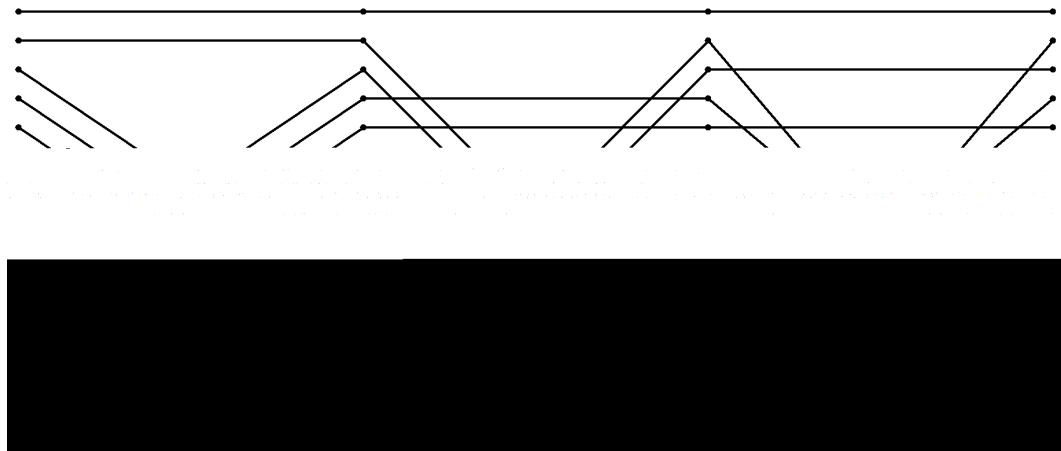
$$h_j(i) = \begin{cases} i & , \text{ dla } (2i + d/2) \bmod (2d) < d ; \\ d(2^{j+1} - \lfloor i/d \rfloor - 1) + (i \bmod d) & , \text{ dla } (2i + d/2) \bmod (2d) \geq d ; \end{cases} \quad (5.38)$$

$$d = \frac{N^2}{2^{j+1}} ,$$

$$i = 0, 1, \dots, N^2 - 1 .$$

Przykład permutacji h dla transformaty 4×4 -punktowej zaprezentowano na rysunku 5.23. Jakkolwiek permutacja h jest złożeniem pewnej liczby permutacji h_j , oczywiście możliwe jest wstępne, jednorazowe wykonanie wszystkich permutacji składowych, tak aby we właściwych obliczeniach używać już bezpośredniego odwzorowania wejście-wyjście.

Zauważmy, że ze wzoru (5.35) wynika, iż dla każdej operacji bazowej jej wejściami są wyjścia bloków L_1, L_2, L_3, L_4 o różnych indeksach (dla przykładu, wyjście $(p; q)$ bloku L_1 jest wejściem tej samej operacji bazowej, co wyjście $(p; \frac{M}{2} - q)$ bloku L_2). Z tego względu, dla większych rozmiarów transformaty (od 8×8 począwszy), schemat połączeń operacji bazowych drugiego etapu rozważanego al-



Rysunek 5.23: Permutacja elementów wejściowych transformaty 4×4 -punktowej

gorytmu będzie odbiegał znacząco od struktury pierwszego etapu. Istnieje jednak możliwość nadania mu bardziej regularnej struktury, za pomocą zmiany typu niektórych operacji bazowych oraz wprowadzenia dodatkowej permutacji wyjściowej P . W ten sposób schemat połączeń operacji bazowych drugiego etapu staje się lustrzanym odbiciem struktury etapu pierwszego, z dokładnością do typu i współczynników poszczególnych operacji (z pominięciem zredukowanych operacji bazowych dla $p = 0$ lub $q = 0$, które oczywiście można również pozostawić dla zachowania pełnej symetrii).

Definicja permutacji P została podana w podrozdziale 5.4.1 w postaci wzoru (5.12). Powtórzymy ją tutaj z uwagi na zmianę sposobu notacji:

$$\begin{aligned}
 P_2(0) &= 0 ; \\
 P_2(1) &= 1 ; \\
 P_{2K}(k) &= P_K(k) ; \\
 P_{2K}(K+k) &= P_K(K-k) + K ; \\
 P_{2K}(0) &= 0 ; \\
 P_{2K}(K) &= K ; \\
 k &= 1;2;\dots;K-1 ; \\
 K &= 2;4;8;\dots;N=2 ;
 \end{aligned} \tag{5.39}$$

Permutację P stosujemy do obu współrzędnych p i q wynikowego elementu widma niezależnie. A zatem mając obliczoną i -tą wartość wyjściową grafu przepływu dla algorytmu $N \times N$ -punktowego, otrzymujemy jej ostateczne położenie

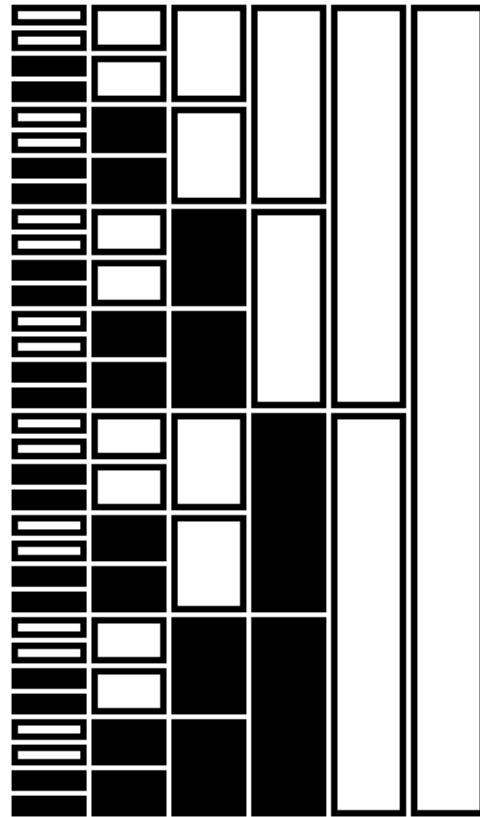
$(p; q)$ w wynikowej macierzy widma 2D DCT następująco:

$$\begin{aligned} (p'; q') &= f_{N \times N}^{-1}(i) : \\ (p; q) &= (P_N(p'); P_N(q')) : \end{aligned} \quad (5.40)$$

Przekształcenie schematu połączeń drugiego etapu do postaci regularnej wymaga również, aby niektóre operacje bazowe zostały zamienione na operacje oznaczone odwróconymi trójkątami (pokazane w prawym górnym rogu na rysunku 5.21).

Przyjmijmy numerację warstw etapu drugiego od prawej do lewej oraz numerację operacji bazowych w każdej warstwie od góry do dołu, począwszy od zera. Wówczas i -ta operacja bazowa w l -tej warstwie $N \times N$ -punktowego przekształcenia zostanie zamieniona, jeśli:

$$\left\lfloor i \cdot \frac{2^l}{N^2} \right\rfloor \bmod 2 \neq 0 \quad \wedge \quad l \geq \log_2 \left(\frac{N^2}{8} \right) : \quad (5.41)$$



Rysunek 5.24: Operacje zmodyfikowane (zaznaczone na czarno) dla transformaty 16×16 -punktowej; fragmenty warstw oznaczono pionowymi prostokątami, przy czym dwie skrajnie lewe warstwy, niepodlegające modyfikacji, pominięto

Dla przykładu, na rysunku 5.24 pokazano, które operacje zostaną zamienione w grafie transformaty 16×16 -punktowej.

Ostatnia kwestia dotyczy wartości współczynników operacji bazowych drugiego etapu. Podczas ich obliczania należy uwzględnić następujące obserwacje:

1. Ze wzorów (5.35) wynika, że dla przekształcenia $N \times M$ -punktowego każda warstwa o numerze parzystym wykonuje dokładnie takie same obliczenia, jak analogiczna warstwa jednowymiarowego przekształcenia N -punktowego, przy czym tutaj są one powielane (w obrębie tej samej warstwy) M -krotnie dla wszystkich wartości $q = 0; 1; \dots; M - 1$. Innymi słowy, każda warstwa o numerze parzystym zawiera M identycznych zestawów operacji bazowych (z których każdy odpowiada analogicznej warstwie N -punktowego przekształcenia jednowymiarowego), różniących się tylko wartościami podawanymi na ich wejścia.

2. Analogicznie, każda warstwa o numerze nieparzystym wykonuje dokładnie takie same obliczenia, jak analogiczna warstwa jednowymiarowego przekształcenia M -punktowego, przy czym tutaj są one powielane N -krotnie dla wszystkich wartości $p = 0; 1; \dots; N - 1$.

A zatem, w celu wyznaczenia wartości współczynników danej operacji w algorytmie $N \times M$ -punktowym należy jedynie określić, jakiej operacji w jednowymiarowym algorytmie N -, bądź M -punktowym ona odpowiada. Konieczne jest przy tym uwzględnienie zarówno permutacji $f_{N \times N}$, jak i permutacji P , wpływających na kolejność operacji bazowych w każdej warstwie.

Przyjmijmy ponownie numerację warstw etapu drugiego od prawej do lewej i numerację operacji bazowych od góry do dołu. Wówczas współczynnik k i -tej operacji w l -tej warstwie algorytmu $N \times N$ -punktowego obliczamy następująco:

$$\begin{aligned} l' &= 2 \cdot \lfloor l/2 \rfloor ; \\ b &= \sqrt{(2^{l'})} ; \\ n &= \frac{N}{b} ; \\ (p'; q') &= f_{n \times n}^{-1} (i \bmod (n^2)) ; \\ k &= \begin{cases} \frac{P_n(p')}{b} , & \text{dla } l' = l ; \\ \frac{P_n(q')}{b} , & \text{dla } l' \neq l : \end{cases} \end{aligned} \tag{5.42}$$

Ponadto, jeśli operacja spełnia warunki (5.41) i jej typ został zmieniony, wówczas do współczynnika obliczonego w (5.42) należy dodać N .

Reasumując, pokazaliśmy sposób konstrukcji dwuetapowego jednorodnego algorytmu dwuwymiarowego przekształcenia cosinusowego drugiego rodzaju. Zaproponowany algorytm dla przekształcenia $N \times N$ -punktowego zawiera dwa etapy złożone z $\log_2 N$ warstw, przy czym każda warstwa składa się z $N=2$ dwupunktowych operacji bazowych. Przyjmując numerację warstw $0; 1; \dots; \log_2 N - 1$ od lewej

do prawej dla pierwszego i od prawej do lewej dla drugiego etapu, schematy połączeń obu etapów są identyczne. Z uwagi na to, że przedstawiony algorytm jest typu *in-place*, numery obu wejść każdej operacji bazowej są takie same jak numery jej wyjścia, zgodnie ze wzorami:

$$\begin{aligned} \text{in}_{i,l}^1 &= \text{out}_{i,l}^1 = \lfloor i/d \rfloor \cdot 2d + (i \bmod d); \\ \text{in}_{i,l}^2 &= \text{out}_{i,l}^2 = \lfloor i/d \rfloor \cdot 2d + (i \bmod d) + d; \\ d &= \frac{N}{2^{l+1}}; \end{aligned} \quad (5.43)$$

gdzie l jest numerem warstwy, zaś $\text{in}_{i,l}^1, \text{in}_{i,l}^2, (\text{out}_{i,l}^1, \text{out}_{i,l}^2)$ oznaczają odpowiednio pierwsze i drugie wejście (wyjście) i -tej operacji bazowej w l -tej warstwie.

Algorytm z mnożnikami tangensowymi

Algorytm z mnożnikami tangensowymi otrzymujemy przekształcając równania (5.35) do postaci:

$$\begin{aligned} L_{N \times M}^{II}(p,q) &= C_{4N}^p \left[C_{4M}^q \left(L_1(p,q) + T_{4M}^q L_2(p, \frac{M}{2}-q) \right) + T_{4N}^p C_{4M}^q \left(L_3(\frac{N}{2}-p, q) + T_{4M}^q L_4(\frac{N}{2}-p, \frac{M}{2}-q) \right) \right], \\ L_{N \times M}^{II}(p, M-q) &= C_{4N}^p \left[C_{4M}^q \left(-T_{4M}^q L_1(p,q) + L_2(p, \frac{M}{2}-q) \right) + T_{4N}^p C_{4M}^q \left(-T_{4M}^q L_3(\frac{N}{2}-p, q) + L_4(\frac{N}{2}-p, \frac{M}{2}-q) \right) \right], \\ L_{N \times M}^{II}(N-p, q) &= C_{4N}^p \left[-T_{4N}^p C_{4M}^q \left(L_1(p,q) + T_{4M}^q L_2(p, \frac{M}{2}-q) \right) + C_{4M}^q \left(L_3(\frac{N}{2}-p, q) + T_{4M}^q L_4(\frac{N}{2}-p, \frac{M}{2}-q) \right) \right], \\ L_{N \times M}^{II}(N-p, M-q) &= C_{4N}^p \left[-T_{4N}^p C_{4M}^q \left(-T_{4M}^q L_1(p,q) + L_2(p, \frac{M}{2}-q) \right) + C_{4M}^q \left(-T_{4M}^q L_3(\frac{N}{2}-p, q) + L_4(\frac{N}{2}-p, \frac{M}{2}-q) \right) \right]. \end{aligned} \quad (5.44)$$

Zauważmy, że wyrażenia w nawiasach mogą być zaimplementowane z wykorzystaniem operacji bazowych przedstawionych na rysunku 5.12. Natomiast współczynniki C_{4N}^p i C_{4M}^q występujące w powyższych wzorach nie są uwzględniane podczas obliczeń dla poszczególnych warstw (podobnie współczynniki s z drugiego etapu grafu przedstawionego na rys. 5.22 są pomijane). Dopiero po obliczeniu wartości wyjściowych dla ostatniej warstwy wprowadza się poprawkę w postaci mnożenia tych wartości przez współczynniki zdefiniowane wzorem (5.13).

W praktyce, graf algorytmu z mnożnikami tangensowymi otrzymujemy zmieniając wszystkie operacje drugiego etapu reprezentowane przez symbol \mathbb{N} na operacje typu \mathbb{C} , zaś operacje reprezentowane symbolem \mathbb{H} na \mathbb{C} oraz dodając warstwę wyjściową dokonującą mnożenia każdego wyjścia (p, q) (por. wzory (5.40)) przez współczynnik $U_{NN}^{(p,q)}$ zdefiniowany wzorem (5.13).

5.4.3 Transformata Fouriera

W niniejszym podrozdziale przedstawiono sposób konstrukcji szybkiej ortogonalnej sieci neuronowej opartej na dwuwymiarowym przekształceniu Fouriera. Punkt wyjścia do stworzenia zaprezentowanej sieci stanowi wyprowadzony wyżej dwuetapowy, jednorodny algorytm dwuwymiarowego szybkiego przekształcenia cosinusowego drugiego rodzaju (2D-FCT2). Przedstawiony tu sposób modyfikacji tego algorytmu umożliwia określenie wartości wag umożliwiające wstępna realizację przekształcenia Fouriera. Z tego względu posługiwać się będziemy w dalszym ciągu określeniem "współczynniki operacji bazowych" rozumiejąc przez to wartości wag jakie mogą zostać użyte do preinicjalizacji neuronów sieci. Oprócz sieci wykorzystującej neurony z dwoma wagami przedstawiono również, w kolejnym podrozdziale, wariant z mnożnikami tangensowymi, wykorzystujący neurony z jedną wagą. Oba warianty mogą być wykorzystane do obliczenia widma amplitudowego Fouriera po dodaniu specjalnej warstwy wyjściowej.

Realizacja neuronowa dwuwymiarowego przekształcenia Fouriera

Naszym celem jest obliczenie dwuwymiarowego widma przekształcenia Fouriera zdefiniowanego wzorem (2.62)

Każda wartość wyjściowa definiowana równaniem (2.62) jest liczbą zespoloną o części rzeczywistej i urojonej równej odpowiednio:

$$\begin{aligned} \operatorname{Re}\{X_{N \times M}(p; q)\} &= \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} x(n; m) (C_N^{pn} C_M^{qm} - S_N^{pn} S_M^{qm}) ; \\ \operatorname{Im}\{X_{N \times M}(p; q)\} &= \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} x(n; m) (-C_N^{pn} S_M^{qm} - S_N^{pn} C_M^{qm}) ; \end{aligned} \quad (5.45)$$

gdzie:

$$\begin{aligned} C_K^k &= \cos\left(2 \frac{k}{K}\right) ; \\ S_K^k &= \sin\left(2 \frac{k}{K}\right) ; \end{aligned} \quad (5.46)$$

Do obliczenia wartości zdefiniowanych przez równania (5.45) wykorzystamy dwuwymiarowe przekształcenie cosinusowe dane jako:

$$L_{N \times M}^{II}(p; q) = \operatorname{DCT}_{N \times M}^{II}\{x(n; m)\} = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} x(n; m) C_{4N}^{(2n+1)p} C_{4M}^{(2m+1)q} ; \quad (5.47)$$

W tym celu wartości wejściowe $x(n; m)$ należy poddać następującej permutacji:

$$\begin{aligned} x_T(2n; 2m) &= x(n; m) ; \\ x_T(2n+1; 2m) &= x(N-1-n; m) ; \\ x_T(2n; 2m+1) &= x(n; M-1-m) ; \\ x_T(2n+1; 2m+1) &= x(N-1-n; M-1-m) ; \end{aligned} \quad (5.48)$$

Rozważmy następującą dekompozycję przekształcenia cosinusowego sygnału $x_T(n; m)$:

$$\begin{aligned} L_T^{II}(p; q) &= \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} x_T(n; m) C_{4N}^{(2n+1)p} C_{4M}^{(2m+1)q} = \\ &= \sum_{n=0}^{N/2-1} \sum_{m=0}^{M/2-1} x_T(2n; 2m) C_{4N}^{(4n+1)p} C_{4M}^{(4m+1)q} + \\ &+ \sum_{n=0}^{N/2-1} \sum_{m=0}^{M/2-1} x_T(2n+1; 2m) C_{4N}^{(4n+3)p} C_{4M}^{(4m+1)q} + \\ &+ \sum_{n=0}^{N/2-1} \sum_{m=0}^{M/2-1} x_T(2n; 2m+1) C_{4N}^{(4n+1)p} C_{4M}^{(4m+3)q} + \\ &+ \sum_{n=0}^{N/2-1} \sum_{m=0}^{M/2-1} x_T(2n+1; 2m+1) C_{4N}^{(4n+3)p} C_{4M}^{(4m+3)q} : \end{aligned} \quad (5.49)$$

Uwzględniając permutację (5.48) mamy:

$$\begin{aligned} L_T^{II}(p; q) &= \sum_{n=0}^{N/2-1} \sum_{m=0}^{M/2-1} x(n; m) C_{4N}^{(4n+1)p} C_{4M}^{(4m+1)q} + \\ &+ \sum_{n=0}^{N/2-1} \sum_{m=0}^{M/2-1} x(N-1-n; m) C_{4N}^{(4n+3)p} C_{4M}^{(4m+1)q} + \\ &+ \sum_{n=0}^{N/2-1} \sum_{m=0}^{M/2-1} x(n; M-1-m) C_{4N}^{(4n+1)p} C_{4M}^{(4m+3)q} + \\ &+ \sum_{n=0}^{N/2-1} \sum_{m=0}^{M/2-1} x(N-1-n; M-1-m) C_{4N}^{(4n+3)p} C_{4M}^{(4m+3)q} , \end{aligned} \quad (5.50)$$

co umożliwia zmianę granic sumowania:

$$\begin{aligned}
L_T^{II}(p; q) = & \sum_{n=0}^{N/2-1} \sum_{m=0}^{M/2-1} x(n; m) C_{4N}^{(4n+1)p} C_{4M}^{(4m+1)q} + \\
& + \sum_{n=N/2}^{N-1} \sum_{m=0}^{M/2-1} x(n; m) C_{4N}^{(4(N-n)-1)p} C_{4M}^{(4m+1)q} + \\
& + \sum_{n=0}^{N/2-1} \sum_{m=M/2}^{M-1} x(n; m) C_{4N}^{(4n+1)p} C_{4M}^{(4(M-m)-1)q} + \\
& + \sum_{n=N/2}^{N-1} \sum_{m=M/2}^{M-1} x(n; m) C_{4N}^{(4(N-n)-1)p} C_{4M}^{(4(M-m)-1)q} ;
\end{aligned} \tag{5.51}$$

Uwzględniając tożsamość:

$$C_{4N}^{(4(N-n)-1)p} = \cos \left(2 \frac{(4n+1)p}{4N} - 2p \right) = C_{4N}^{(4n+1)p} ; \tag{5.52}$$

i analogicznie:

$$C_{4M}^{(4(M-m)-1)q} = \cos \left(2 \frac{(4m+1)q}{4M} - 2q \right) = C_{4M}^{(4m+1)q} ; \tag{5.53}$$

możemy zsumować wszystkie składniki w (5.51) bezpośrednio, otrzymując ostatecznie:

$$L_T^{II}(p; q) = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} x(n; m) C_{4N}^{(4n+1)p} C_{4M}^{(4m+1)q} ; \tag{5.54}$$

Rozważmy teraz sposób obliczania wyrażenia (5.54) w czterech przypadkach. Uwzględniając tożsamości:

$$\begin{aligned}
C_{4N}^{(4n+1)p} &= C_N^{np} C_{4N}^p - S_N^{np} S_{4N}^p ; \\
C_{4M}^{(4m+1)q} &= C_M^{mq} C_{4M}^q - S_M^{mq} S_{4M}^q ; \\
C_{4N}^{(4n+1)(N-p)} &= S_N^{np} C_{4N}^p + C_N^{np} S_{4N}^p ; \\
C_{4M}^{(4m+1)(M-q)} &= S_M^{mq} C_{4M}^q + C_M^{mq} S_{4M}^q ;
\end{aligned} \tag{5.55}$$

możemy zapisać:

$$\begin{aligned}
 L_T^{II}(p; q) &= \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} x(n; m) (C_N^{pn} C_{4N}^p - S_N^{pn} S_{4N}^p) (C_M^{qm} C_{4M}^q - S_M^{qm} S_{4M}^q) ; \\
 L_T^{II}(p; M-q) &= \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} x(n; m) (C_N^{pn} C_{4N}^p - S_N^{pn} S_{4N}^p) (S_M^{qm} C_{4M}^q + C_M^{qm} S_{4M}^q) ; \\
 L_T^{II}(N-p; q) &= \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} x(n; m) (S_N^{pn} C_{4N}^p + C_N^{pn} S_{4N}^p) (C_M^{qm} C_{4M}^q - S_M^{qm} S_{4M}^q) ; \\
 L_T^{II}(N-p; M-q) &= \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} x(n; m) (S_N^{pn} C_{4N}^p + C_N^{pn} S_{4N}^p) (S_M^{qm} C_{4M}^q + C_M^{qm} S_{4M}^q) ;
 \end{aligned} \tag{5.56}$$

gdzie $p = 1; \dots; N-2 - 1; q = 1; \dots; M-2 - 1$.

Na podstawie powyższych zależności, uwzględniając równania (5.45) otrzymujemy⁴:

$$\begin{aligned}
 Re\{X_{N \times M}(p, q)\} &= L_T^{II}(p, q) (C_{4N}^p C_{4M}^q - S_{4N}^p S_{4M}^q) + L_T^{II}(p, M-q) (S_{4N}^p C_{4M}^q + C_{4N}^p S_{4M}^q) + \\
 &\quad + L_T^{II}(N-p, q) (S_{4N}^p C_{4M}^q + C_{4N}^p S_{4M}^q) - L_T^{II}(N-p, M-q) (C_{4N}^p C_{4M}^q - S_{4N}^p S_{4M}^q) , \\
 Im\{X_{N \times M}(p, q)\} &= L_T^{II}(p, q) (S_{4N}^p C_{4M}^q + C_{4N}^p S_{4M}^q) - L_T^{II}(p, M-q) (C_{4N}^p C_{4M}^q - S_{4N}^p S_{4M}^q) - \\
 &\quad - L_T^{II}(N-p, q) (C_{4N}^p C_{4M}^q - S_{4N}^p S_{4M}^q) - L_T^{II}(N-p, M-q) (S_{4N}^p C_{4M}^q + C_{4N}^p S_{4M}^q) , \\
 Re\{X_{N \times M}(N-p, q)\} &= L_T^{II}(p, q) (C_{4N}^p C_{4M}^q + S_{4N}^p S_{4M}^q) - L_T^{II}(p, M-q) (S_{4N}^p C_{4M}^q - C_{4N}^p S_{4M}^q) + \\
 &\quad + L_T^{II}(N-p, q) (S_{4N}^p C_{4M}^q - C_{4N}^p S_{4M}^q) + L_T^{II}(N-p, M-q) (C_{4N}^p C_{4M}^q + S_{4N}^p S_{4M}^q) , \\
 Im\{X_{N \times M}(N-p, q)\} &= -L_T^{II}(p, q) (S_{4N}^p C_{4M}^q - C_{4N}^p S_{4M}^q) - L_T^{II}(p, M-q) (C_{4N}^p C_{4M}^q + S_{4N}^p S_{4M}^q) + \\
 &\quad + L_T^{II}(N-p, q) (C_{4N}^p C_{4M}^q + S_{4N}^p S_{4M}^q) - L_T^{II}(N-p, M-q) (S_{4N}^p C_{4M}^q - C_{4N}^p S_{4M}^q) ,
 \end{aligned} \tag{5.57}$$

lub, równoważnie:

$$\begin{aligned}
 Re\{X_{N \times M}(p, q)\} &= L_T^{II}(p, q) C_{4NM}^{pM+qN} + L_T^{II}(p, M-q) S_{4NM}^{pM+qN} + L_T^{II}(N-p, q) S_{4NM}^{pM+qN} - L_T^{II}(N-p, M-q) C_{4NM}^{pM+qN} , \\
 Im\{X_{N \times M}(p, q)\} &= L_T^{II}(p, q) S_{4NM}^{pM+qN} - L_T^{II}(p, M-q) C_{4NM}^{pM+qN} - L_T^{II}(N-p, q) C_{4NM}^{pM+qN} - L_T^{II}(N-p, M-q) S_{4NM}^{pM+qN} , \\
 Re\{X_{N \times M}(N-p, q)\} &= L_T^{II}(p, q) C_{4NM}^{pM-qN} - L_T^{II}(p, M-q) S_{4NM}^{pM-qN} + L_T^{II}(N-p, q) S_{4NM}^{pM-qN} + L_T^{II}(N-p, M-q) C_{4NM}^{pM-qN} , \\
 Im\{X_{N \times M}(N-p, q)\} &= -L_T^{II}(p, q) S_{4NM}^{pM-qN} - L_T^{II}(p, M-q) C_{4NM}^{pM-qN} + L_T^{II}(N-p, q) C_{4NM}^{pM-qN} - L_T^{II}(N-p, M-q) S_{4NM}^{pM-qN} .
 \end{aligned} \tag{5.58}$$

Zauważmy, że z uwagi na własność symetrii widma Fouriera dla sygnałów rzeczywistych nie ma konieczności wykonywania osobnych obliczeń dla $X_{N \times M}(p; M-q)$ oraz dla $X_{N \times M}(N-p; M-q)$. Natomiast w przypadkach gdy $p = 0, q = 0$,

⁴Ponownie stosujemy tu mniejszą czcionkę aby uniknąć częstego łamania wierszy

$p = N=2$ lub $q = M=2$ obliczenia upraszczają się do postaci:

$$\begin{aligned}
 Re\{X_{N \times M}(0,0)\} &= L_T^{II}(0,0) , \\
 Re\{X_{N \times M}(N/2,M/2)\} &= 2 \cdot L_T^{II}(N/2,M/2) , \\
 Re\{X_{N \times M}(N/2,0)\} &= \frac{2}{\sqrt{2}} \cdot L_T^{II}(N/2,0) , \\
 Re\{X_{N \times M}(0,M/2)\} &= \frac{2}{\sqrt{2}} \cdot L_T^{II}(0,M/2) , \\
 Re\{X_{N \times M}(0,q)\} &= L_T^{II}(0,q) C_{4M}^q + L_T^{II}(0,M-q) S_{4M}^q , \\
 Re\{X_{N \times M}(N/2,q)\} &= \frac{2}{\sqrt{2}} \cdot L_T^{II}(N/2,q) C_{4M}^q + \frac{2}{\sqrt{2}} \cdot L_T^{II}(N/2,M-q) S_{4M}^q , \\
 Re\{X_{N \times M}(p,0)\} &= L_T^{II}(p,0) C_{4N}^p + L_T^{II}(N-p,0) S_{4N}^p , \\
 Re\{X_{N \times M}(p,M/2)\} &= \frac{2}{\sqrt{2}} \cdot L_T^{II}(p,M/2) C_{4N}^p + \frac{2}{\sqrt{2}} \cdot L_T^{II}(N-p,M/2) S_{4N}^p , \\
 Im\{X_{N \times M}(0,q)\} &= L_T^{II}(0,q) S_{4M}^q - L_T^{II}(0,M-q) C_{4M}^q , \\
 Im\{X_{N \times M}(N/2,q)\} &= \frac{2}{\sqrt{2}} \cdot L_T^{II}(N/2,q) S_{4M}^q - \frac{2}{\sqrt{2}} \cdot L_T^{II}(N/2,M-q) C_{4M}^q , \\
 Im\{X_{N \times M}(p,0)\} &= L_T^{II}(p,0) S_{4N}^p - L_T^{II}(N-p,0) C_{4N}^p , \\
 Im\{X_{N \times M}(p,M/2)\} &= \frac{2}{\sqrt{2}} \cdot L_T^{II}(p,M/2) S_{4N}^p - \frac{2}{\sqrt{2}} \cdot L_T^{II}(N-p,M/2) C_{4N}^p .
 \end{aligned} \tag{5.59}$$

Wzory (5.58) można zaimplementować w postaci warstwy operacji czteropunktowych lub dwóch warstw operacji dwupunktowych. Biorąc jednak pod uwagę graf dwuwymiarowego, jednorodnego, dwuetapowego przekształcenia cosinusowego drugiego rodzaju (2D-FCT2) okazuje się, że wzory (5.58) można połączyć z obliczeniami wykonywanymi w dwóch ostatnich warstwach tego grafu. W ten sposób nie ma konieczności zwiększenia ilości warstw, a jedynie współczynniki dotychczasowych operacji ulegają zmianie. Po wymnożeniu macierzowej reprezentacji przekształcenia liniowego wykonywanego przez dwie ostatnie warstwy grafu 2D-FCT2 przez macierz przekształcenia definiowanego przez wzory (5.58) otrzymujemy przekształcenie, które powinny realizować dwie ostatnie warstwy po modyfikacji:

$$\begin{bmatrix} Re\{X_{N \times M}(p;q)\} \\ Im\{X_{N \times M}(p;q)\} \\ Re\{X_{N \times M}(N-p;q)\} \\ Im\{X_{N \times M}(N-p;q)\} \end{bmatrix} = A \cdot \begin{bmatrix} L_{T1}(p;q) \\ L_{T2}(p;M=2-q) \\ L_{T3}(N=2-p;q) \\ L_{T4}(N=2-p;M=2-q) \end{bmatrix}; \tag{5.60}$$

gdzie macierz A zdefiniowana jest następująco:

$$A = \begin{bmatrix} C_{4NM}^{2(pM+qN)} & S_{4NM}^{2(pM+qN)} & S_{4NM}^{2(pM+qN)} & -C_{4NM}^{2(pM+qN)} \\ S_{4NM}^{2(pM+qN)} & -C_{4NM}^{2(pM+qN)} & -C_{4NM}^{2(pM+qN)} & -S_{4NM}^{2(pM+qN)} \\ C_{4NM}^{2(pM-qN)} & -S_{4NM}^{2(pM-qN)} & S_{4NM}^{2(pM-qN)} & C_{4NM}^{2(pM-qN)} \\ -S_{4NM}^{2(pM-qN)} & -C_{4NM}^{2(pM-qN)} & C_{4NM}^{2(pM-qN)} & -S_{4NM}^{2(pM-qN)} \end{bmatrix}; \tag{5.61}$$

zaś $L_{T1}; \dots; L_{T4}$ oznaczają bloki $L_1; \dots; L_4$ (p. wzory (5.36)) realizujące przekształcenie $DCT_{\frac{N}{2} \times \frac{M}{2}}^{II}$ sygnału poddanego permutacji (5.48).

Zauważmy, że otrzymane współczynniki różnią się od współczynników we wzorach (5.58) tylko obecnością dwójką w indeksie górnym. Zauważmy przy tym, iż przyjmując założenie, że sygnały wejściowe i wyjściowe będą reprezentowane przez macierze kwadratowe, tj., że $N = M$, współczynniki te możemy zapisać prościej jako:

$$C_{4NM}^{2(pM+qN)} = C_{4N^2}^{2(pN+qN)} = \cos\left(2 \frac{2(pN+qN)}{4N^2}\right) = C_{4N}^{2(p+q)} ; \quad (5.62)$$

i analogicznie dla pozostałych współczynników.

W przypadkach gdy $p = 0, q = 0, p = N=2$ lub $q = M=2$ mamy:

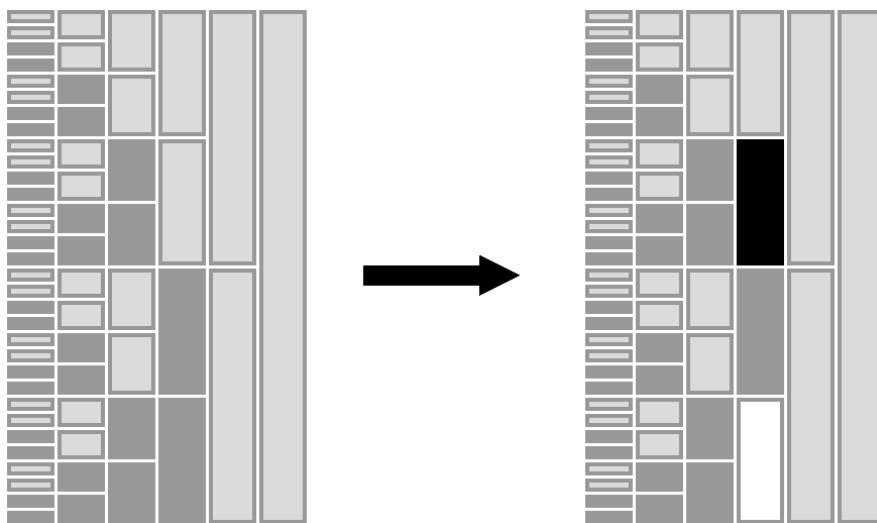
$$\begin{aligned} Re\{X_{N \times M}(0;0)\} &= L_{T1}(0;0) ; \\ Re\{X_{N \times M}(N=2;M=2)\} &= L_{T4}(0;0) ; \\ Re\{X_{N \times M}(N=2;0)\} &= L_{T3}(0;0) ; \\ Re\{X_{N \times M}(0;M=2)\} &= L_{T2}(0;0) ; \\ Re\{X_{N \times M}(0;q)\} &= L_{T1}(0;q)C_{4M}^{2q} + L_{T2}(0;M=2-q)S_{4M}^{2q} ; \\ Re\{X_{N \times M}(N=2;q)\} &= L_{T3}(0;q)C_{4M}^{2q} + L_{T4}(0;M=2-q)S_{4M}^{2q} ; \\ Re\{X_{N \times M}(p;0)\} &= L_{T1}(p;0)C_{4N}^{2p} + L_{T3}(N=2-p;0)S_{4N}^{2p} ; \\ Re\{X_{N \times M}(p;M=2)\} &= L_{T2}(p;0)C_{4N}^{2p} + L_{T4}(N=2-p;0)S_{4N}^{2p} ; \\ Im\{X_{N \times M}(0;q)\} &= L_{T1}(0;q)S_{4M}^{2q} - L_{T2}(0;M=2-q)C_{4M}^{2q} ; \\ Im\{X_{N \times M}(N=2;q)\} &= L_{T3}(0;q)S_{4M}^{2q} - L_{T4}(0;M=2-q)C_{4M}^{2q} ; \\ Im\{X_{N \times M}(p;0)\} &= L_{T1}(p;0)S_{4N}^{2p} - L_{T3}(N=2-p;0)C_{4N}^{2p} ; \\ Im\{X_{N \times M}(p;M=2)\} &= L_{T2}(p;0)S_{4N}^{2p} - L_{T4}(N=2-p;0)C_{4N}^{2p} ; \end{aligned} \quad (5.63)$$

Porównanie wzorów (5.63) z grafem 2D-FCT2 wskazuje, że dla tych przypadków zasadniczo wystarczy podwoić wartości w indeksie górnym współczynników C, S oraz pominąć mnożenia przez współczynniki $\sqrt{2}=2$ w ostatnich dwóch warstwach oryginalnego grafu przekształcenia cosinusowego i na wyjściu otrzymamy odpowiednie elementy widma Fouriera (w szczególności, nie jest konieczna zmiana struktury połączeń operacji bazowych).

Nieco inaczej jest w pozostałych przypadkach, dla których stosuje się wzór (5.60). Wzór ten należy zaimplementować na dwóch ostatnich warstwach grafu przekształcenia tak aby zachować jego jednolitą strukturę. Postać macierzy A (5.61), a w szczególności fakt, że każdy jej wiersz zawiera tylko dwa różne współczynniki (ew. z przeciwnym znakiem), sugeruje faktoryzację z użyciem w przedostatniej

warstwie wyłącznie trywialnych operacji typu dodawanie/odejmowanie. Podstawowy problem stanowi to, że operacje te będą dotyczyć wyjść pary bloków L_{T1} , L_{T4} oraz pary L_{T2} , L_{T3} , co narusza jednolity schemat przedostatniej warstwy, zgodnie z którym operacjom poddawane są pary L_{T1} , L_{T2} oraz L_{T3} , L_{T4} .

Tę trudność można rozwiązać zamieniając ze sobą miejscami bloki L_{T2} i L_{T4} . Zauważmy, że bloki te są identyczne co do struktury i współczynników operacji bazowych, (z wyjątkiem ich ostatniej warstwy, w której należy zmienić typy operacji bazowych zgodnie z rys. 5.25. Wystarczy zatem zmienić ze sobą ich sygnały



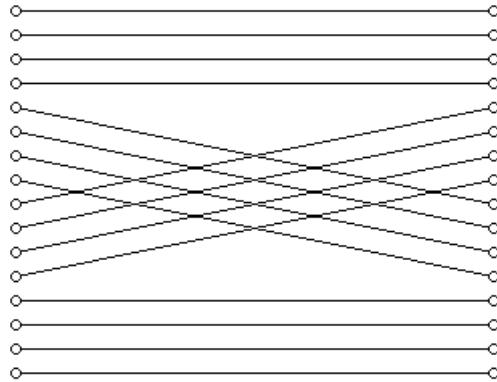
Rysunek 5.25: Modyfikacja typów operacji bazowych

wejściowe, tj. przekierować tę część wyjść drugiej warstwy całego grafu, która trafia do bloku L_{T2} do bloku L_{T4} , a tę część, która trafia do bloku L_{T4} przekierować do L_{T2} . Przekierowanie takie z kolei można zrealizować bez jakichkolwiek modyfikacji dwóch pierwszych warstw, stosując dodatkową permutację wejścia (rys. 5.26), zdefiniowaną następująco:

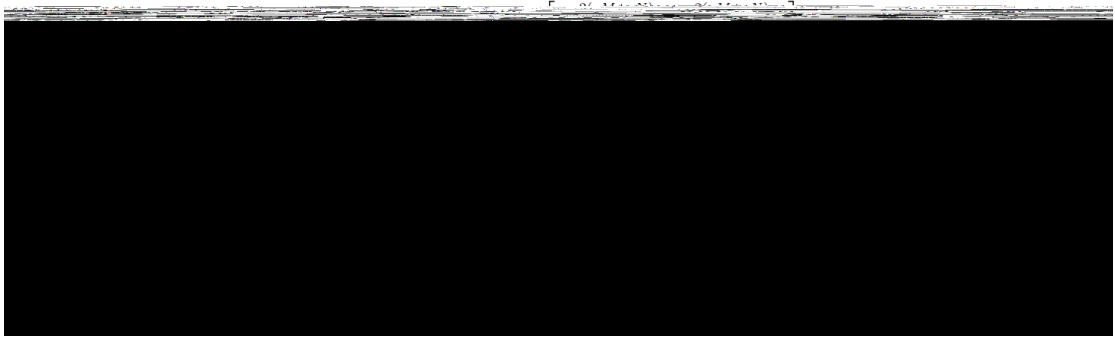
$$R(n) = \begin{cases} x(n) & , \text{ dla } n < N=4 ; \\ x(n + N=4) & , \text{ dla } n = N=4; N=4 + 1; \dots; N=2 - 1 ; \\ x(n - N=4) & , \text{ dla } n = N=2; N=2 + 1; \dots; 3 \cdot N=4 - 1 ; \\ x(n) & , \text{ dla } n \geq 3 \cdot N=4 : \end{cases} \quad (5.64)$$

Permutację R stosujemy bezpośrednio przed pierwszą warstwą operacji bazowych. Warto zauważyć, że ze względu na strukturę pierwszego etapu rozważanego grafu skutkuje ona w istocie poszukiwaną przez nas zamianą wejścia dla drugiego i czwartego bloku (nie – drugiego i trzeciego, jak można by przypuszczać na podstawie rys. 5.26).

Tak zatem wzór (5.60) możemy ostatecznie zaimplementować jak na rys. 5.27.

Rysunek 5.26: Permutacja R .

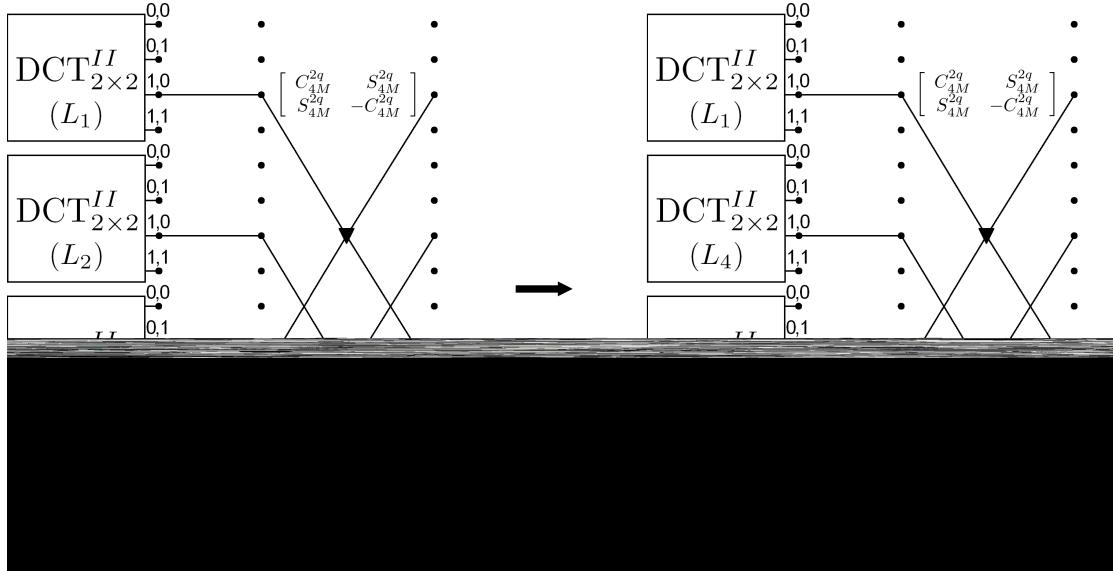
Jak widać, zakładamy tu istotnie zamianę drugiego i czwartego wejścia. Zwróćmy

Rysunek 5.27: Przykład obliczeń transformaty Fouriera wykonywanych w dwóch ostatnich warstwach dla $p; q$ różnych od $0; N=2; M=2$.

także uwagę na to, że zamieniono tu wyjścia (drugie z trzecim). Dzięki temu pierwsza połowa wyjść całej sieci oblicza części rzeczywiste, a druga urojone, odpowiednich elementów widma.

Zamiana miejscami bloków L_{T2} i L_{T4} komplikuje nieco sytuację w przypadkach gdy $p = 0, q = 0, p = N=2$ lub $q = M=2$. Aby zachować jednorodną strukturę grafu przekształcenia wykonujemy zatem następujące operacje:

1. Dla $p = 0 \vee p = N=2$ obliczenia zdefiniowane wzorami (5.63) wykonywane są w ostatniej warstwie. Wystarczy zatem zmienić typ i współczynniki operacji bazowych przyjmujących wejścia z bloków L_{T2} i L_{T4} (rys. 5.28).
2. Dla $q = 0 \vee q = M=2$ wzory (5.63) implementowane są w warstwie przedostatniej. Pozostawienie obliczeń w tej warstwie naruszyłoby jednorodną strukturę grafu w sytuacji zamiany bloków L_{T2} i L_{T4} . Przyjęto zatem rozwiązanie polegające na przesunięciu obliczeń do warstwy ostatniej. Jedyną operacją konieczną do wykonania w warstwie przedostatniej jest zamiana

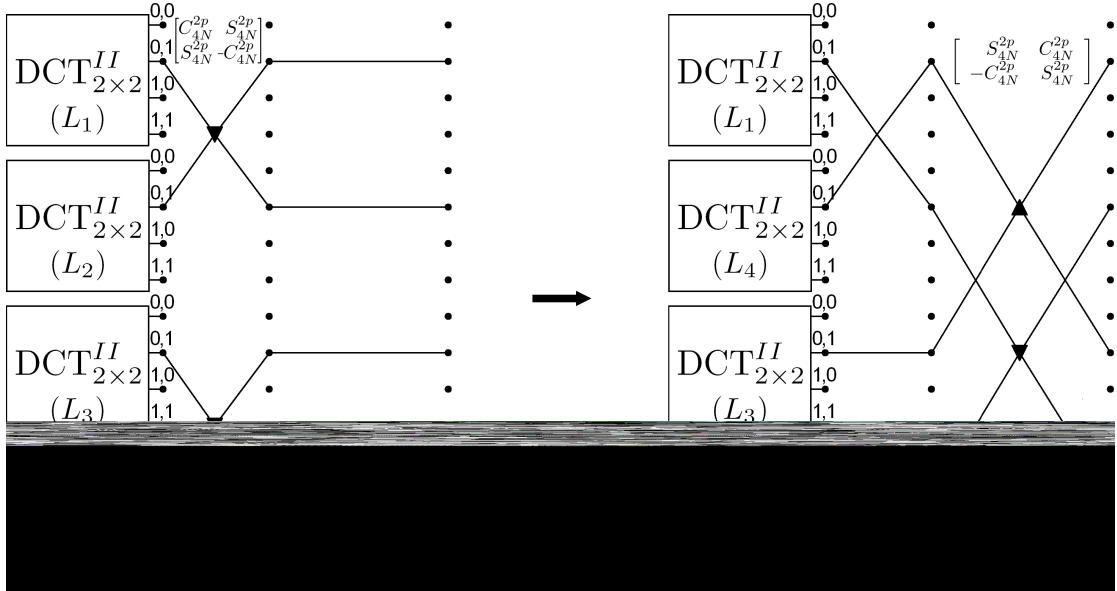


Rysunek 5.28: Przykład modyfikacji obliczeń transformaty Fouriera wykonywanych w dwóch ostatnich warstwach dla $p = 0 \vee p = N=2$.

miejscami wyjść pierwszych dwóch bloków (czyli L_{T1} i L_{T4}). Ten zabieg można zrealizować za pomocą trywialnej operacji bazowej ze współczynnikami $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$, a dla pozostawienia pełnej symetrii grafu również do wyjść dwóch ostatnich bloków można zastosować trywialną, tożsamościową operację bazową (rys. 5.29) ze współczynnikami $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.

A zatem, dla wszystkich wartości p i q przedostatnia warstwa zawiera wyłącznie operacje trywialne (dodawanie/odejmowanie, zamiana, identyczność). Naturalnie, w realizacji neuronowej operacje te mogą być realizowane w postaci zwykłych neuronów typu BOON z wagami wstępnie zainicjalizowanymi wartościami $-1; 0; 1$.

Opisane powyżej modyfikacje struktury grafu powodują zamianę kolejności niektórych elementów widma (dla $p = 0, q = 0, p = N=2$ lub $q = M=2$). Biorąc jednak pod uwagę specyfikę dwuwymiarowego widma Fouriera, a w szczególności występujące w nim symetrie, korekcja tych zmian za pomocą dodatkowej permutacji wyjść nie wydaje się celowa. W zależności od pożąданej postaci wyniku przetwarzania i tak konieczne będzie wprowadzenie odpowiedniego uporządkowania wyjść, (na przykład poprzez zduplikowanie niektórych wartości w celu uzyskania pełnego, symetrycznego widma). Zastosowanie wszystkich powyższych modyfikacji i pozostawienie tej samej permutacji jaka została zaproponowana dla dwuwymiarowego przekształcenia cosinusowego (wzór (5.33)) prowadzi do uzyskania kolejności wyjść jak na rys. 5.30. Strzałki reprezentują podciągi elementów widma (krótkie: $N=2-1$ elementowych; długie: N elementowych), a liczby informują w jakiej kolejności



Rysunek 5.29: Przykład modyfikacji obliczeń transformaty Fouriera wykonywanych w dwóch ostatnich warstwach dla $q = 0 \vee q = M=2$.

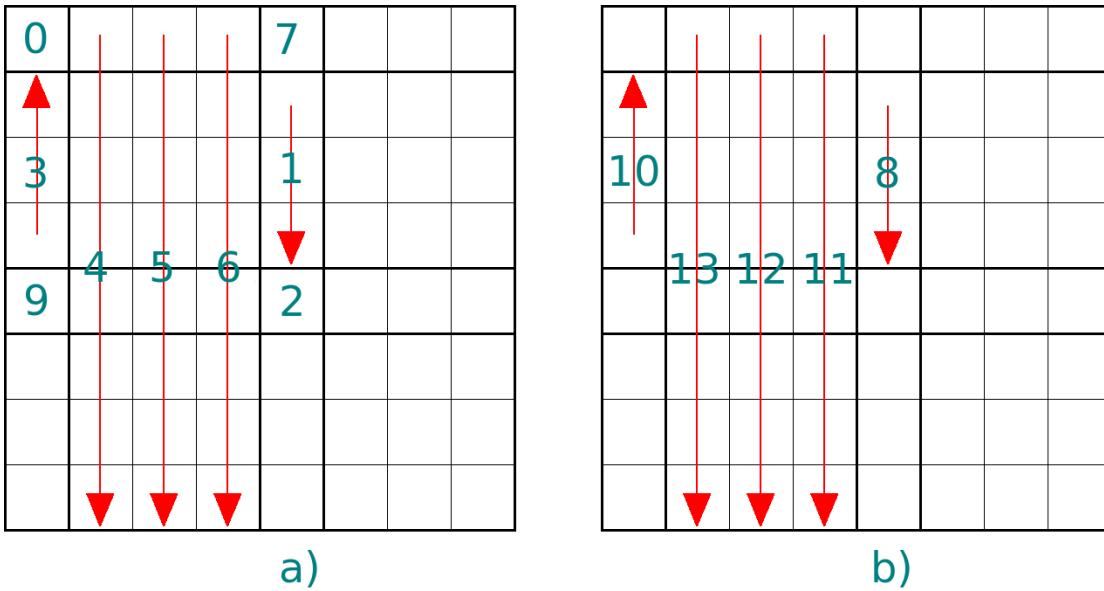
podciągi te są uszeregowane w ciągu wartości wyjściowych transformaty. Wartości elementów nie objętych strzałkami wynikają z własności symetrii transformaty Fouriera dla sygnałów rzeczywistych. Zatem przykładowo, początkowe wyjścia grafu przekształcenia reprezentować będą kolejno wartości:

$$\begin{aligned}
 & Re\{X_{N \times N}(0; 0)\} ; \\
 & Re\{X_{N \times N}(1; N=2)\} ; \\
 & Re\{X_{N \times N}(2; N=2)\} ; \\
 & \quad \vdots \\
 & Re\{X_{N \times N}(N=2; N=2)\} ; \\
 & Re\{X_{N \times N}(N=2 - 1; 0)\} ; \\
 & Re\{X_{N \times N}(N=2 - 2; 0)\} ; \\
 & \quad \vdots
 \end{aligned} \tag{5.65}$$

Kompletny graf sieci realizującej przekształcenie Fouriera o rozmiarze 4×4 wraz z permutacją wejściową i wyjściową zaprezentowano na rys. 5.31.

5.4.4 Sieć oparta na algorytmie z mnożnikami tangensowymi

Wykorzystanie algorytmu z mnożnikami tangensowymi zasadniczo nie wpływa na modyfikacje połączeń operacji bazowych, powoduje natomiast zmianę ich typu



Rysunek 5.30: Przykład sekwencji elementów dla części rzeczywistej a) i urojonej b) widma transformaty Fouriera o rozmiarze $N \times N = 8 \times 8$.

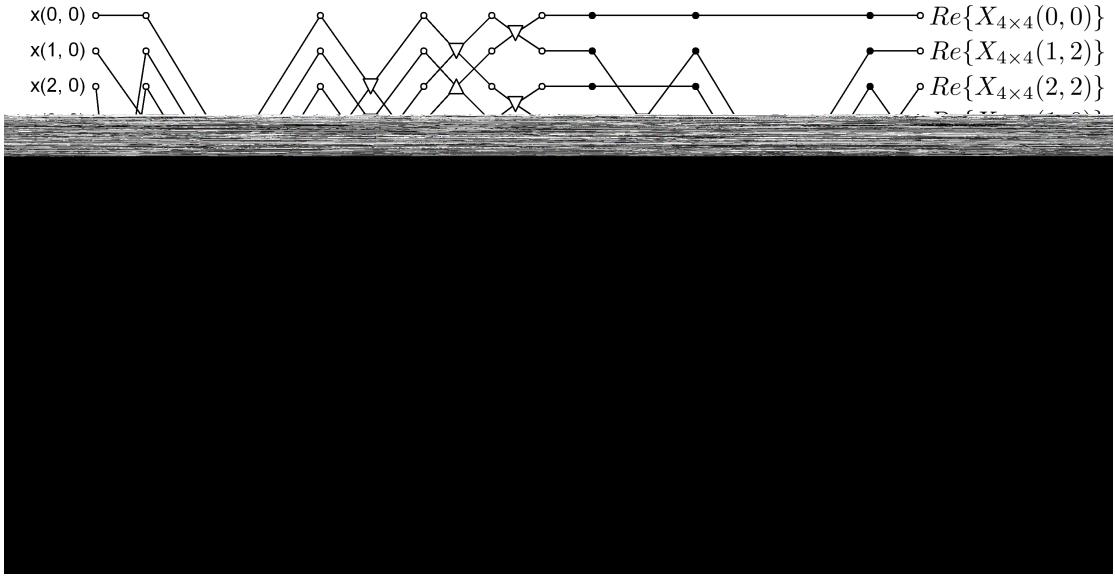
z neuronów o dwóch wagach na neurony z jedną wagą (rys. 5.32). Szczególnej uwagi wymagają wartości współczynników operacji bazowych ostatnich dwóch warstw (współczynniki we wszystkich wcześniejszych warstwach są identyczne jak w algorytmie 2D-FCT2, z uwzględnieniem rys. 5.25).

Ponieważ wszystkie operacje warstwy przedostatniej posiadają współczynniki $-1; 0; 1$, czyli de facto nie wymagają mnożeń, ich zamiana na operacje z jedną wagą jest trywialna. Modyfikacja operacji w ostatniej warstwie oparta jest natomiast na prostych przekształceniach:

$$\begin{aligned}
 out_1 &= in_1 \cdot C_{4NM}^{2(pM+qN)} + in_2 \cdot S_{4NM}^{2(pM+qN)} = S_{4NM}^{2(pM+qN)} \left(in_1 \cdot (T^{-1})_{4NM}^{2(pM+qN)} + in_2 \right) ; \\
 out_2 &= in_1 \cdot S_{4NM}^{2(pM+qN)} - in_2 \cdot C_{4NM}^{2(pM+qN)} = S_{4NM}^{2(pM+qN)} \left(in_1 - in_2 \cdot (T^{-1})_{4NM}^{2(pM+qN)} \right) ;
 \end{aligned} \tag{5.66}$$

gdzie $in_1; in_2; out_1; out_2$ oznaczają odpowiednio pierwsze i drugie wejście oraz pierwsze i drugie wyjście pojedynczej operacji bazowej, zaś

$$(T^{-1})_K^k = \operatorname{ctg}(2\pi k/K) : \tag{5.67}$$

Rysunek 5.31: Graf sieci realizującej przekształcenie Fouriera o rozmiarze 4×4 .

Analogicznie:

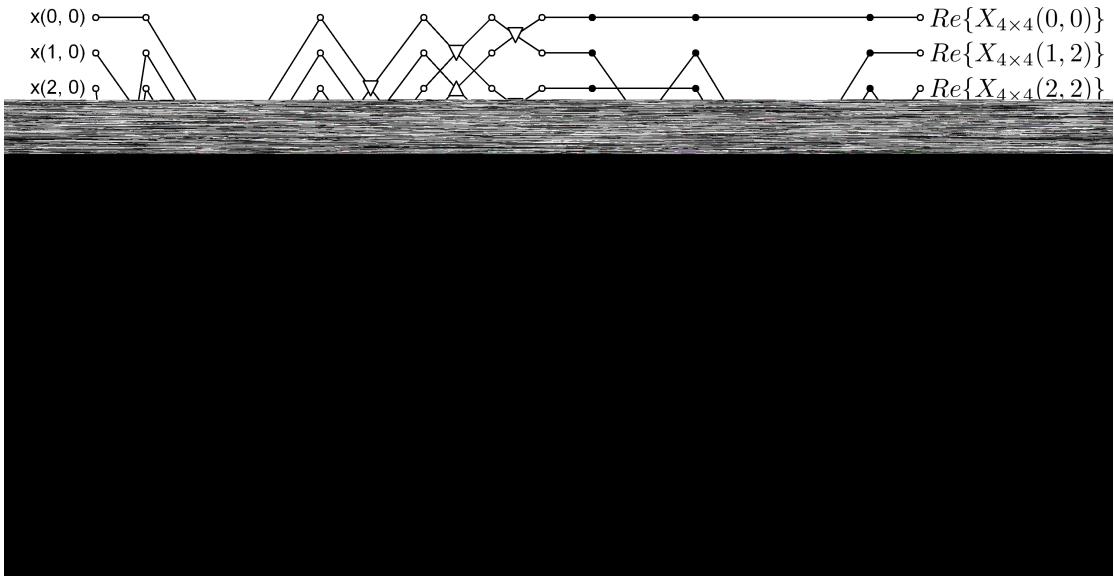
$$\begin{aligned} out_1 &= in_1 \cdot C_{4NM}^{2(pM-qN)} + in_2 \cdot S_{4NM}^{2(pM-qN)} = S_{4NM}^{2(pM-qN)} \left(in_1 \cdot (T^{-1})_{4NM}^{2(pM-qN)} + in_2 \right) ; \\ out_2 &= in_1 \cdot S_{4NM}^{2(pM-qN)} - in_2 \cdot C_{4NM}^{2(pM-qN)} = S_{4NM}^{2(pM-qN)} \left(in_1 - in_2 \cdot (T^{-1})_{4NM}^{2(pM-qN)} \right) ; \end{aligned} \quad (5.68)$$

Podobnie dla przypadków gdy $q = 0$ lub $q = M=2$ mamy:

$$\begin{aligned} out_1 &= in_1 \cdot C_{4M}^{2q} + in_2 \cdot S_{4M}^{2q} = S_{4M}^{2q} \left(in_1 \cdot (T^{-1})_{4M}^{2q} + in_2 \right) ; \\ out_2 &= in_1 \cdot S_{4M}^{2q} - in_2 \cdot C_{4M}^{2q} = S_{4M}^{2q} \left(in_1 - in_2 \cdot (T^{-1})_{4M}^{2q} \right) ; \end{aligned} \quad (5.69)$$

i analogicznie dla $p = 0$ lub $p = N=2$. We wzorach (5.66), (5.68), (5.69) operacje w nawiasach realizowane są bezpośrednio w postaci neuronów z jedną wagą, inicjalizowaną współczynnikami cotangensowymi (5.67), natomiast wyciągnięte przed nawias współczynniki S są uwzględniane w ramach obliczeń ostatniej warstwy.

Algorytm z mnożnikami tangensowymi zakłada, że każde wyjście grafu transformaty jest mnożone przez pojedynczy współczynnik kompensujący zredukowane wagi operacji bazowych, przy czym mnożenie to odbywa się w ostatniej warstwie grafu (na rys. 5.32 realizowane jest przez ostatnią warstwę dokonującą równocześnie permutacji wyjść). Podstawę obliczeń tych współczynników dla transformaty Fouriera stanowi rekurencyjny algorytm podany dla przekształcenia 2D-FCT2.



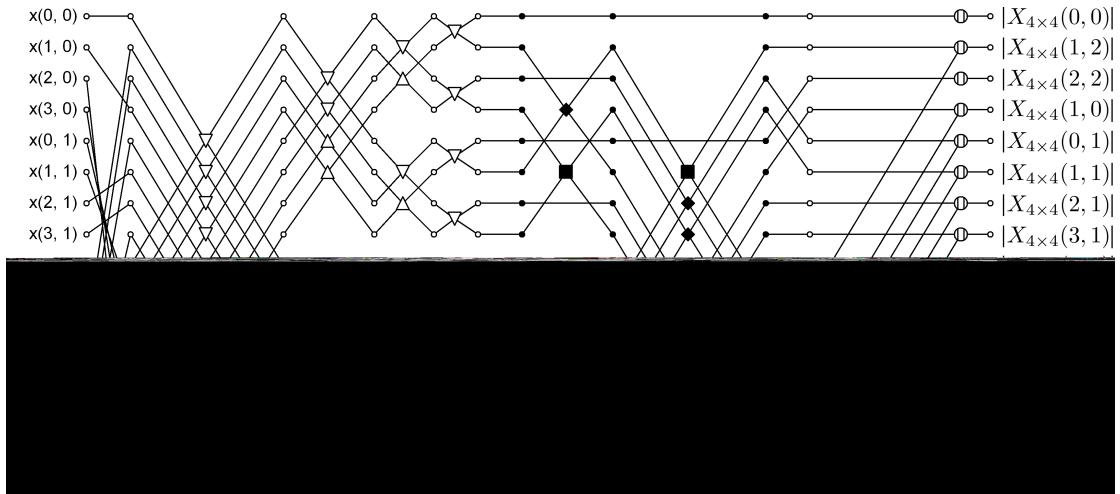
Rysunek 5.32: Graf sieci realizującej przekształcenie Fouriera oparte na algorytmie z mnożnikami tangensowymi o rozmiarze 4×4 .

Ponieważ istotnym modyfikacjom poddano tu tylko dwie ostatnie warstwy grafu, tylko ostatni etap rekurencji wymaga zmiany. Mając zatem obliczone współczynniki ostatniej warstwy $U_{NN}^{(p,q)}$ dla przekształcenia 2D-FCT2 należy “cofnąć” obliczenia z ostatniego etapu rekurencji, dzieląc $U_{NN}^{(p,q)}$ przez $C_{4N}^p \cdot C_{4M}^q$, a następnie mnożąc przez odpowiedni współczynnik S zdefiniowany jednym ze wzorów (5.66), (5.68), (5.69), w zależności od tego której operacji bazowej dotyczy.

5.4.5 Widmo amplitudowe dwuwymiarowej transformaty Fouriera

Podobnie jak w przypadku jednowymiarowym (por. podrozdział 5.3.4), w celu umożliwienia sieci obliczenia widma amplitudowego konieczne jest dodanie specjalnej warstwy wyjściowej (rys. 5.33). Jej schemat połączeń jest dość prosty, z uwagi na to, że każdy z jej elementów otrzymuje jako wartości wejściowe odpowiednio część rzeczywistą i urojoną tego samego elementu widma. Z tego też względu otrzymywana kolejność elementów widma amplitudowego jest taka sama jak w przypadku widma rzeczywistego, z wyjątkiem ostatniego elementu, oznaczonego na rys. 5.30 numerem 9, który teraz występuje bezpośrednio za elementem oznaczonym jako siódmy.

Pewien wyjątek stanowią elementy obliczające wartości $|X_{N \times M}(0;0)|$, $|X_{N \times M}(N=2;M=2)|$, $|X_{N \times M}(0;M=2)|$ i $|X_{N \times M}(N=2;0)|$, posiadające tylko jedno wejście. Sytuacja ta odpowiada obliczeniom odpowiednio: pierwszego i $N=2$ -tego wyjściowego elementu jednowymiarowego przekształcenia Fouriera o rozmiarze N ,



Rysunek 5.33: Graf sieci obliczającej widmo amplitudowe przekształcenia Fouriera opartej na algorytmie z mnożnikami tangensowymi o rozmiarze 4×4 .

co jest realizowane w identyczny sposób, poprzez zwykłe obliczenie wartości bezwzględnej liczby rzeczywistej.

5.5 Podsumowanie

Możliwości praktycznej realizacji przedstawionego w rozdziale 4 nowego rodzaju sieci neuronowej (FONN) zostały tu zweryfikowane poprzez jej implementację w oparciu o kilka różnych schematów szybkich dwuetapowych algorytmów transformat ortogonalnych. W szczególności były to:

1. Algorytm szybkiej transformaty cosinusowej drugiego rodzaju z mnożnikami tangensowymi i bez mnożników (mFCT2, FCT2) – podrozdział 5.1.
2. Algorytm szybkiej transformaty cosinusowej czwartego rodzaju (mFCT4) z mnożnikami tangensowymi (podrozdział 5.2).
3. Algorytm transformaty Hartley'a oparty na zmodyfikowanym algorytmie FCT2 (podrozdział 5.3.1).
4. Algorytm transformaty Fouriera z mnożnikami tangensowymi i bez mnożników, oparty na zmodyfikowanym algorytmie mFCT2 oraz FCT2, odpowiednio (podrozdziały 5.3.2, 5.3.3).
5. Algorytm obliczania widma amplitudowego transformaty Fouriera oparty na algorytmach z poprzedniego punktu (z mnożnikami tangensowymi i bez mnożników) – podrozdział 5.3.4.

6. Algorytm dwuwymiarowej transformaty cosinusowej drugiego rodzaju (2D-FCT2) skonstruowany metodą wiersze-kolumny (podrozdział 5.4.1).
7. Algorytm dwuwymiarowej transformaty cosinusowej drugiego rodzaju (2D-FCT2) skonstruowany metodą bezpośrednią (podrozdział 5.4.2).
8. Algorytm dwuwymiarowej transformaty Fouriera z mnożnikami tangensowymi i bez mnożników (podrozdziały 5.4.3, 5.4.4), oparty na zmodyfikowanym algorytmie z poprzedniego punktu.
9. Algorytm obliczania widma amplitudowego dwuwymiarowej transformaty Fouriera oparty na algorytmach z poprzedniego punktu (z mnożnikami tangensowymi i bez mnożników) – podrozdział 5.4.5.

Wszystkie przedstawione sieci zostały zaimplementowane i poddane testom weryfikującym poprawność implementacji wyrażającą się możliwością nauczenia ich transformaty wybranej jako podstawa ich konstrukcji (z dowolną dokładnością, rozpoczynając adaptację z losowego punktu w przestrzeni wag). W każdym przypadku wynik weryfikacji był pozytywny, a najbardziej interesujące rezultaty zaprezentowano w poszczególnych podrozdziałach:

1. W podrozdziale 5.1 przedstawiono porównanie między znaną z literatury siecią opartą na szybkim algorytmie, w którym każda operacja bazowa została zastąpiona dwoma zwykłymi neuronami liniowymi, a analogczną siecią typu FONN wykorzystującą neurony typu BOON-2 i BOON-1. Uzyskane wyniki dokumentują bardzo znaczną poprawę efektywności nauczania w stosunku do sieci nie-ortogonalnej, zarówno w sensie redukcji liczby epok, jak i skrócenia czasu adaptacji, uzasadniając tym samym prawdziwość pierwszej tezy pracy. Warto tu zauważyć, że w zdecydowanej większości wszystkich przeprowadzanych przez autora testów, łącznie z testami obejmującymi klasyfikację i rozpoznawanie wzorców (p. rozdział 6), redukcja liczby wag właściwa sieci FONN, zwłaszcza w wariancie z mnożnikami tangensowymi, przynosiła znaczące skrócenie czasu nauki i ograniczenie liczby epok.
2. W podrozdziale 5.2 zaprezentowano wyniki analizy przypadku dynamicznego rozszerzania i douczania struktury sieci FONN w oparciu o rekurencyjny charakter konstrukcji szybkiego algorytmu wyznaczającego jej schemat połączeń. Przy założeniu znanej postaci wektorów docelowych na każdym etapie rekurencji, podejście to przyniosło znaczące skrócenie czasu adaptacji.
3. Omówienie metod konstrukcji sieci opartej o transformaty Hartley'a i Fouriera zakończono prezentacją wyników adaptacji obu sieci, a także sieci

opartej na algorytmie obliczania widma amplitudowego Fouriera (podrozdział 5.3.5).

4. Dla sieci opartej na dwuwymiarowym przekształceniu cosinusowym przedstawiono porównanie ze zwykłą jednowarstwową siecią liniową, wykazując zdecydowaną przewagę zredukowanej architektury sieci FONN, zwłaszcza w wariancie z mnożnikami tangensowymi.

Należy podkreślić, że metody konstrukcji powyższych algorytmów, przedstawione w niniejszym rozdziale stanowią własne wyprowadzenia autora, przy czym niektóre z nich nie były dotąd znane w literaturze (w szczególności dotyczy to szybkich, dwuetapowych jednolitych algorytmów transformaty Fouriera z mnożnikami tangensowymi oraz dwuwymiarowej transformaty Fouriera z mnożnikami tangensowymi i bez). Nowatorskim pomysłem jest również opracowanie i wprowadzenie do architektury sieci elementów umożliwiających obliczanie widma amplitudowego transformaty Fouriera (podrozdział 5.3.4). Tym samym obok pełnej realizacji pierwszego celu pracy, uzyskano dodatkowe rezultaty pod postacią opracowanych szybkich algorytmów umożliwiających m.in. konstrukcję sieci typu FONN opartą na schemacie obliczeniowym jedno- i dwuwymiarowej transformaty Fouriera i jej widma amplitudowego. Różnorodność zaprezentowanych algorytmów, definiujących strukturę połączeń sieci, jest oczywistym dowodem na elastyczność zaprezentowanej w rozdziale 4 metody konstrukcji i adaptacji neuronów typu BOON i złożonych z nich szybkich ortogonalnych sieci neuronowych.

6 Zastosowanie sieci typu FONN w problemach klasyfikacji i rozpoznawania wzorców

Uwzględniając własności szybkich ortogonalnych sieci neuronowych, m.in. stosunkowo niewielką liczbę wag¹ oraz związki z liniowymi przekształceniami ortogonalnymi zauważamy, że mogą one być szczególnie przydatne na początkowych etapach procesu klasyfikacji. Interesującą możliwość stanowi zwłaszcza zastąpienie ekstraktora cech wykorzystującego przekształcenie cosinusowe, Fouriera, czy inne przekształcenie ortogonalne – siecią typu FONN zbudowaną w oparciu o ten sam rodzaj transformaty. W takim wypadku wykorzystanie w charakterze klasyfikatora również sieci neuronowej, np. typu nieliniowego perceptronu wielowarstwowego (MLP), umożliwia łączną adaptację obu sieci (por. opis na stronie 65), tworzących w ten sposób razem hybrydową sieć neuronową, operującą na surowych, nieprzetworzonych wcześniej danych wejściowych.

W kolejnych podrozdziałach przedstawiono analizę i weryfikację eksperymentalną systemu rozpoznawania wzorców opartego na takim podejściu. Przeprowadzone testy zostały podzielone na część obejmującą sygnały jednowymiarowe (podrozdział 6.1) oraz klasyfikację obrazów (podrozdział 6.2).

6.1 Analiza sygnałów jednowymiarowych

Na wstępie należy zaznaczyć, że wszystkie zaprezentowane w rozdziale 5 wyniki dotyczyły bardzo podstawowego przypadku weryfikacji możliwości nauczenia się przez sieć typu FONN tej transformaty ortogonalnej, która stanowiła podstawę jej konstrukcji (w ten sposób uzyskujemy zatem zasadniczo tylko informację o poprawności konstrukcji sieci). Znacznie bardziej interesujące jest nauczenie sieci innego przekształcenia, o postaci nieznanej z góry, zapewniającego lepsze możli-

¹w porównaniu do sieci opartych na połączeniach „każdy z każdym”

wości klasyfikacji zadanego zbioru wzorców. W pierwszym przykładzie (podrozdział 6.1.1) porównano pod tym względem transformatę cosinusową i jej neuronowy odpowiednik w postaci sieci FONN [123]. Klasyfikator stanowiła w obu przypadkach sieć typu MLP. Z uwagi na potencjalnie nadmiarową ilość warstw (sieć FONN + nieliniowa warstwa ukryta sieci MLP + warstwa wyjściowa), w kolejnych przykładach uproszczono nieco architekturę łącząc szybką sieć ortogonalną z warstwą wyjściową za pośrednictwem warstwy nielinowej o schemacie połączeń jeden-do-jednego. W drugim przykładzie (podrozdział 6.1.2) na podstawie specjalnie skonstruowanego zbioru danych poddano analizie niezależność od przesunięcia danych wejściowych, właściwą dla widma amplitudowego transformaty Fouriera [126], a w trzecim (podrozdział 6.1.3) przedstawiono wyniki klasyfikacji rzeczywistego zbioru danych zawierającego fragmenty muzyki klasycznej [124].

6.1.1 Ekstrakcja cech

Ogólny schemat systemu klasyfikacji zawiera trzy główne bloki odpowiedzialne za: normalizację², ekstrakcję cech i klasyfikację. Rozważamy tutaj dwie różne możliwości konstrukcji bloku ekstrakcji cech, a zatem do wykonania wszystkich testów zaprojektowano dwa warianty systemu klasyfikacji (rys. 6.1).



Rysunek 6.1: Dwa warianty przetwarzania: ekstrakcja cech oparta na DCT (na górze) i adaptowalny ekstraktor cech oparty na sieci typu FONN (na dole)

Pierwszy wariant, wykorzystujący zwykłą transformatę cosinusową, stanowi punkt odniesienia dla wariantu drugiego, w którym zastosowano „adaptowalną transformatę cosinusową”, czyli sieć neuronową typu FONN opartą na algorytmie transformaty cosinusowej tego samego rodzaju. Obydwa warianty zostały użyte dwukrotnie: za pierwszym razem wykorzystano przekształcenie mFCT2, a za drugim razem mFCT4.

W praktyce, z uwagi na możliwość wstępnej inicjalizacji wag umożliwiającej uzyskanie przekształcenia cosinusowego bez konieczności nauki (p. strona 57), oba warianty były w istocie implementowane za pomocą sieci FONN. Różnica polegała

²Ścisłe: ten blok obejmował odjęcie składowej stałej i normalizację wektorów wejściowych

na tym, że w pierwszym wariantie wagi były wyznaczone zgodnie z odpowiednim szybkim algorytmem i nie podlegały adaptacji.

W przypadku obu wariantów jako cechy przekazywane do klasyfikacji wybierano F pierwszych wyjść ekstraktora cech (z pominięciem składowej stałej). Ta strategia odpowiadała zatem — dla pierwszego wariantu — wyborowi współczynników wyjściowych DCT związanych ze składowymi sygnałów wejściowych o najniższych częstotliwościach. Blok klasyfikatora składał się z perceptronu typu MLP z jedną nielinową warstwą ukrytą złożoną z neuronów o sigmoidalnej funkcji aktywacji zdefiniowanej wzorem (3.6). Liczba neuronów ukrytych była dobrana eksperymentalnie, natomiast liczba wejść i wyjść perceptronu odpowiadała ilości cech F oraz ilości klas w zbiorze danych, odpowiednio. Każde wyjście reprezentowało dokładnie jedną z klas, a decyzja odnośnie ostatecznej klasy była podejmowana zgodnie ze wzorem (3.16).

W drugim wariantie konstrukcji systemu wartość sygnału błędu była propagowana wstecz również z pierwszej warstwy sieci MLP do ostatniej warstwy sieci FONN, stanowiąc podstawę adaptacji tej drugiej. W ten sposób obie sieci stanowiły w istocie jedną sieć hybrydową o dwóch różnych architekturach (p. też opis na stronie 65). Integrację obu bloków podkreślał dodatkowo fakt, iż metoda minimizacji funkcji celu, wykorzystująca algorytm gradientów sprzężonych, oparta była na wektorze gradientu wyznaczonym dla wszystkich wag występujących w obu częściach systemu.

Materiał i procedura testowania

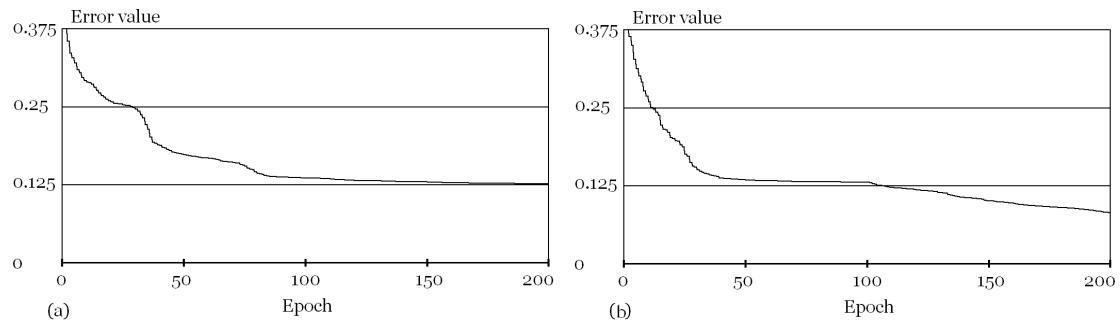
Do testowania zaproponowanego schematu klasyfikacji został wykorzystany zbiór *Synthetic Control Chart Time Series* [52]. Zbiór ten zawiera 600 ciągów 60-elementowych, podzielonych na sześć klas zróżnicowanych pod względem parametrów procesu losowego użytego do wygenerowania danego ciągu (*normal*, *cyclic*, *increasing trend*, *decreasing trend*, *upward shift*, *downward shift*). Cały zbiór został podzielony na dwa podzbiory: A_{train} zawierający 420 ciągów i A_{test} zawierający 180 ciągów. Każdy ciąg był rozszerzony do 64 elementów przez dodanie czterech elementów o wartości zero na końcu.

Wszystkie testy zostały przeprowadzone z użyciem tych samych parametrów adaptacji. Liczba epok została ustalona na 200, a liczba neuronów ukrytych w bloku MLP wynosiła 8. Jedynym zmiennym parametrem była liczba wejść klasyfikatora.

Z uwagi na to, że głównym celem testów było określenie efektywności sieci typu FONN w ekstrakcji cech, zastosowano specjalną procedurę nauczania złożoną z dwóch faz. W pierwszej fazie (epoki 1-100) tylko część MLP była adaptowana.

wartości współczynnika rozpoznania dla sieci ortogonalnej w przypadku niewystarczającej ilości cech ($F = 1; 2; 3$).

Na tej podstawie łatwo wnosićmy, że adaptacja wag sieci ortogonalnej poprawia potencjał klasyfikacyjny jej wyjścia w stosunku do współczynników wyjściowych DCT. Efekt ten jest również widoczny podczas drugiej fazy uczenia, kiedy po początkowym nauczeniu klasyfikatora włączana jest również adaptacja bloku FONN. Błąd, który zwykle osiąga plateau pod koniec pierwszej fazy zaczyna tu się ponownie zmniejszać. Typowe krzywe błędu dla zbioru treningowego zaprezentowano na rys. 6.2.



Rysunek 6.2: Krzywe błędu dla przypadku DCT+MLP (a) i dla FONN+MLP (b)

Rysunki 6.3, 6.4 ilustrują dodatkowo efekt uczenia sieci FONN. Wszystkie wektory danych ze zbioru A_{train} są zaprezentowane w przestrzeni wyznaczonej przez dwa pierwsze współczynniki DCT2. Można zauważyć (rys. 6.3), że chociaż w tym wypadku DCT2 łatwo rozróżnia trzy podzbiory zawierające po dwie klasy każdy, jednak skuteczne rozdzielenie klas w obrębie tych podzbiorów nie jest możliwe. Adaptacja transformaty, implementowanej jako sieć typu FONN (rys. 6.4) prowadzi do uzyskania lepiej odseparowanych rozkładów punktów danych w przestrzeni cech kosztem pewnego zmniejszenia odległości pomiędzy trzema podzbiorami.

Rozważając zastosowania praktyczne, są dwie zasadnicze kwestie, które należy wziąć pod uwagę. Po pierwsze, sieć ortogonalna może mieć tendencję do zwiększenia błędu generalizacji, jako że jej wartości wyjściowe nie ograniczają się ścisłe do niskoczęstotliwościowych składowych analizowanych sygnałów. Jest to szczególnie widoczne w wyniku dla zbioru A_{test} dla transformaty DCT4, dla $F = 4$, który jest nawet gorszy w przypadku adaptowanym, choć dla zbioru A_{train} jest wciąż lepszy w stosunku do wariantu bez adaptacji. Ta obserwacja podkreśla rolę zastosowania odpowiednich mechanizmów walidacji w trakcie procesu nauki.

Druga kwestia dotyczy konieczności przetwarzania danych przez blok FONN podczas procesu nauki. W pierwszym wariantie współczynniki wyjściowe DCT są obliczane jednorazowo, a następnie użyte do trenowania klasyfikatora. Zastosowanie neuronowego ekstraktora cech oznacza, że dane muszą być przetwarzane



Rysunek 6.3: Reprezentacja danych do klasyfikacji w dwuwymiarowej przestrzeni cech dla wariantu DCT2+MLP

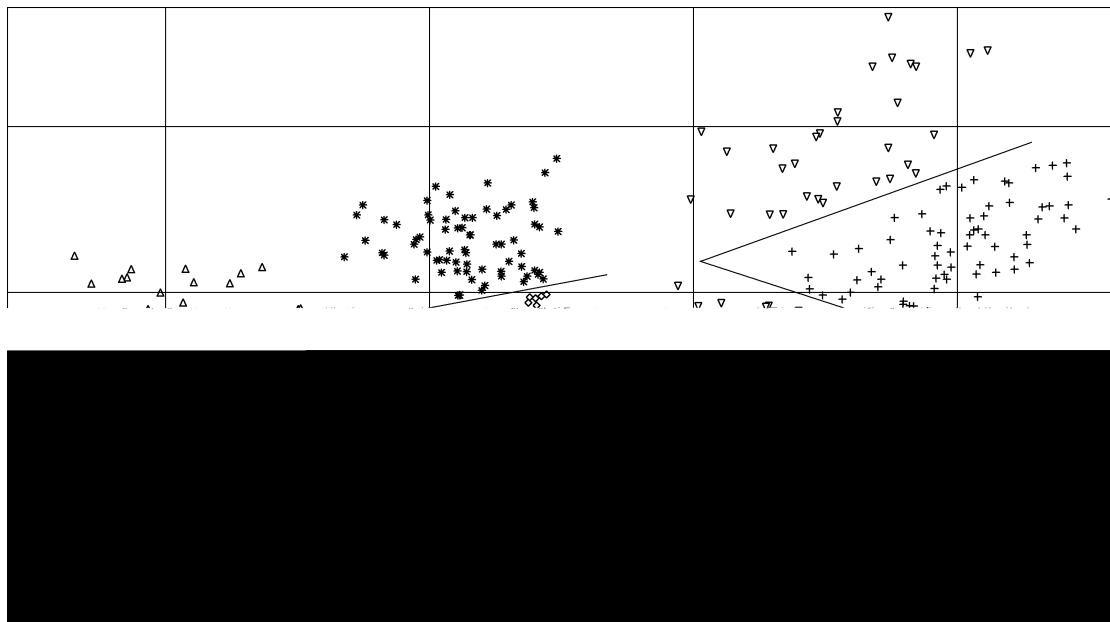
podczas każdej epoki, aby zapewnić adaptację jego wag.

Istnieje wiele czynników wpływających potencjalnie na ogólną efektywność klasyfikacji, takich jak typ i sposób implementacji przekształcenia ortogonalnego, czy charakterystyka zbioru danych. Zastosowany tu zbiór danych stanowi dość proste zadanie klasyfikacyjne, jednak w przypadku „wielowymiarowych” problemów z dziedziny np. rozpoznawania obrazów, gdzie ilość i lokalizacja istotnych współczynników widmowych mogą być trudne do jednoznacznego określenia, możliwość „dostrojenia” ekstraktora cech, czy nawet zredukowania przestrzeni cech mogą być wysoce pożądane.

Zaprezentowane rezultaty wykazują wyższość szybkiej ortogonalnej sieci neuronowej opartej na przekształceniu cosinusowym rodzaju drugiego i czwartego w stosunku do klasycznej wersji tych przekształceń w zadaniu ekstrakcji cech. Sieć typu FONN jest w stanie skoncentrować więcej informacji istotnej w procesie klasyfikacji na takiej samej liczbie współczynników wyjściowych. Wydaje się uzasadnione stwierdzenie, iż zaproponowane rozwiązanie może pomóc w redukcji wymiaru przestrzeni cech w przypadku złożonych problemów klasyfikacyjnych.

6.1.2 Klasyfikacja w oparciu o widmo amplitudowe

Widmo amplitudowe Fouriera jest często wykorzystywane jako naturalny ekstraktor cech w wielu różnych zadaniach klasyfikacyjnych, m.in. z uwagi na własność niezależności od przesunięcia sygnału wejściowego w dziedzinie czasu [32][89].



Rysunek 6.4: Reprezentacja danych do klasyfikacji w dwuwymiarowej przestrzeni cech dla wariantu FONN+MLP. Hiperpłaszczyzny separujące zostały zaznaczone ręcznie.

Przykładowo, w analizie danych audio postrzeganie barwy dźwięku jest w pierwszym rzędzie zależne od względnych amplitud, a nie od przesunięć fazowych składowych harmonicznych. W rozpoznawaniu obrazu zmiana położenia obserwatora/obiektu obserwacji, zwykle nieistotna z punktu widzenia klasyfikacji, jest odzwierciedlana przede wszystkim przez przesunięcie fazy w dziedzinie częstotliwości. Z drugiej strony spektrum fazowe zawiera również istotną informację o obrazie⁴, która może mieć znaczenie podczas klasyfikacji [93]. Z tego względu korzystna

Materiał i procedura testowania

Zbiór danych składa się z wektorów o rozmiarze $N = 256$, podzielonych na $K = 8$ klas. Wektory zostały uzyskane za pomocą generatora liczb losowych, przy czym ich amplitudowe i fazowe widma były losowane niezależnie, tak aby zagwarantować, że:

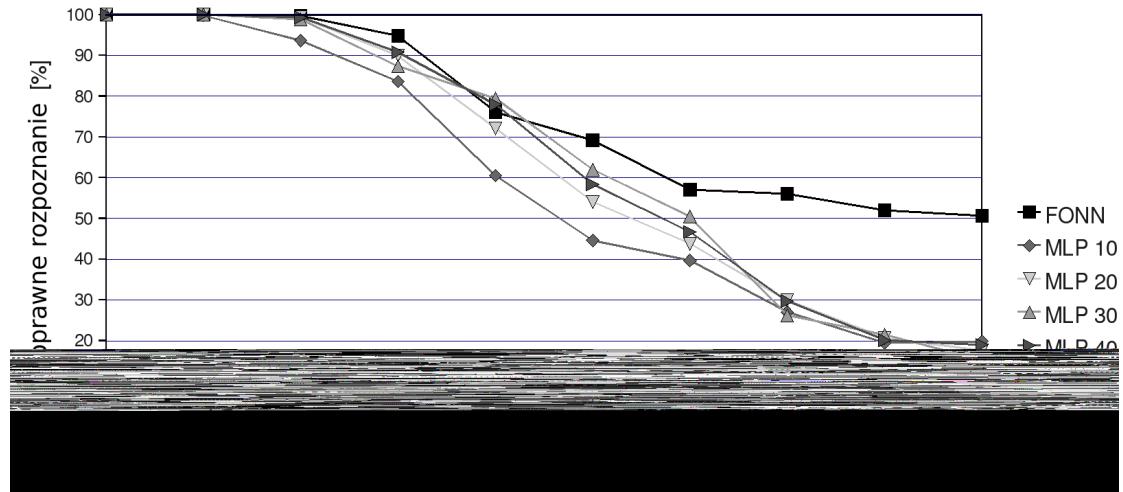
1. wszystkie wektory z pojedynczej klasy mają to samo widmo amplitudowe;
2. wszystkie wektory mające to samo widmo amplitudowe należą do jednej z dwóch klas.

A zatem są tylko cztery różne widma amplitudowe w całym zbiorze, każde wykorzystane do konstrukcji dwóch klas. W ten sposób żadna metoda klasyfikacji oparta na analizie samego tylko widma amplitudowego nie jest w stanie zapewnić poprawnego rozróżnienia klas z tej samej pary, dając maksymalny możliwy wynik ok 50% poprawnej klasyfikacji. Rozważenie dodatkowo widma fazowego, które jest różne w każdej klasie, jest zatem niezbędne do uzyskania wyższych współczynników rozpoznania.

Każda klasa w zbiorze treningowym zawiera 10 wektorów różniących się widmami fazowymi i składnikiem szumowym dodanym do ich (oprócz tego identycznych) widm amplitudowych. Każda klasa w zbiorze testowym zawiera 40 wektorów, w których konstrukcji uwzględniono składnik szumowy dodany zarówno do widma fazowego jak i amplitudowego. Zbiór testowy generowany był kilkakrotnie z wykorzystaniem różnych wartości SNR (ang. *signal to noise ratio*). Dla każdej wartości SNR adaptacja zarówno sieci FONN, jak i MLP była powtarzana dwudziestokrotnie, a uśrednione wyniki zaprezentowano na rys. 6.5. Jako kryterium stopu przyjęto wynik klasyfikacji na osobnym zbiorze walidacyjnym. W przypadku klasyfikatora FONN losowaniu podlegały tylko wagi warstwy wyjściowej; wszystkie pozostałe warstwy (czyli właściwa sieć FONN) były preinicjalizowane (choć później również podlegały adaptacji) zgodnie z algorytmem przekształcenia Fouriera. Należy podkreślić, iż dla każdej wartości SNR cały zbiór danych był identyczny w przypadku wszystkich testowanych sieci.

Uzyskane rezultaty i wnioski

Jak łatwo zauważać, klasyfikator oparty na sieci FONN zapewnia lepsze wyniki w obecności szumu, wykazując tym samym dobre zdolności generalizacyjne. Stosunkowo słabe wyniki uzyskane przez sieć MLP mogą być konsekwencją zaszumionego spektrum fazowego w zbiorze testowym. Sieć FONN bazująca na widmie



Rysunek 6.5: Wyniki klasyfikacji dla sieci FONN i MLP z różną liczbą neuronów ukrytych $H = 10; \dots; 40$

amplitudowym wydaje się bardziej odporna na zakłócenia fazowe, chociaż z drugiej strony jest również w stanie wydobyć z widma fazowego wystarczającą ilość informacji, aby rozróżnić pomiędzy klasami o tym samym widmie amplitudowym. Warto tu również zauważać, że zaproponowany klasyfikator oparty na sieci typu FONN posiada 2319 adaptowanych wag (część FONN: 1279, warstwa wyjściowa: 1040), podczas gdy sieci typu MLP zawierają od 2658 (MLP 10) do 10608 (MLP 40) wag. Jest przy tym oczywiste, że 10 neuronów ukrytych wydaje się suboptymalne dla rozważanego zbioru danych.

6.1.3 Rozpoznawanie rodzaju muzyki

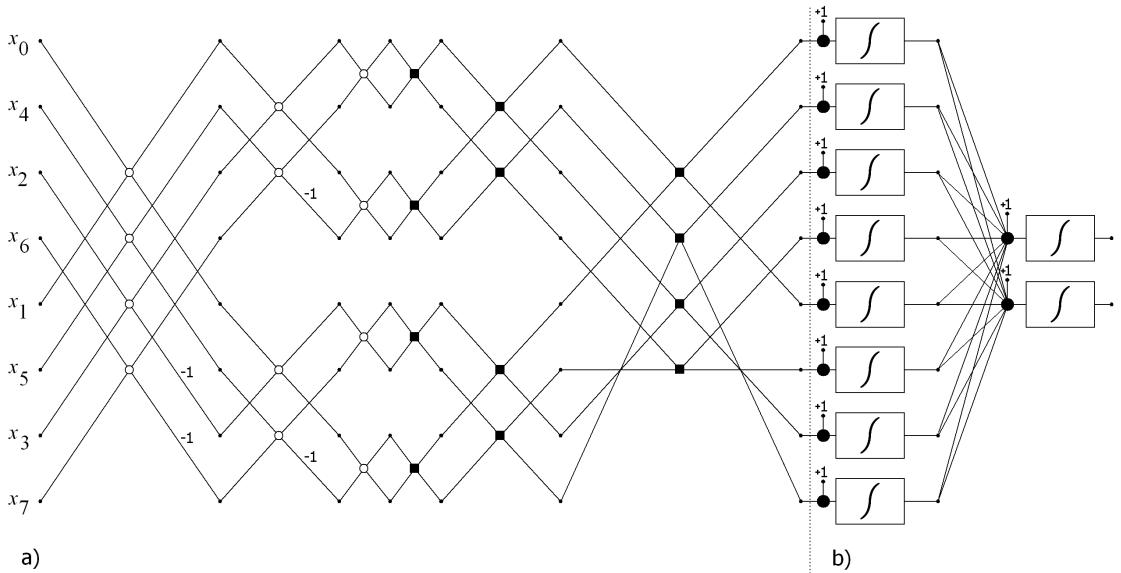
Problem automatycznej klasyfikacji materiału audio był analizowany przez dziesięciolecia, jeśli uznamy tu rozpoznawanie mowy za jeden z pierwszych i najistotniejszych przykładów. Bardziej ogólne podejście dotyczy określania typu analizowanych danych audio w oparciu o kilka dość szerokich kategorii, takich jak mowa, cisza, śmiech, muzyka, inne odgłosy wydawane przez ludzi, zwierzęta, urządzenia mechaniczne itp. [138]. Na potrzeby zastosowań tego typu zaproponowano szereg różnych rozwiązań i wieloetapowych systemów klasyfikacji [83]. Określanie rodzaju muzyki jest następnym krokiem w kierunku szczegółowej analizy danych audio, który z racji gwałtownie rosnącej dostępności dużych kolekcji multimedialnych rodzi coraz większe zainteresowanie praktyczne [138][78][83].

Do zadań klasyfikacji dźwięku stosowane są różnorodne sposoby ekstrakcji cech, oparte zazwyczaj na metodach widmowych, a jednym z najczęściej wykorzystywanych, podstawowych narzędzi jest krótkoczasowa transformata Fouriera (STFT). Analizie podlegają różne charakterystyki widma, m.in. jego środek ciężkości,

kształt, szybkość zmian w czasie i zawartość wyższych harmonicznych. Zawierając zbiór rozpoznawanych klas dźwięku do muzyki, można ponadto analizować takie cechy, jak struktura rytmiczna, czy tonalność.

Z uwagi na dużą przydatność metod widmowych w analizie dźwięku, wykorzystanie w zadaniach klasyfikacji sieci typu FONN wydaje się celowe. Proponowany klasyfikator (rys. 6.6) jest siecią hybrydową, złożoną z części typu FONN opartej na przekształceniu mFCT2 (nazywanej w analogii do przykładu z podrozdziału 6.1.1 „ekstraktorem cech”) oraz z warstwy wyjściowej odpowiedzialnej za ostateczną klasyfikację. Podział pomiędzy „ekstraktorem cech” a) i klasyfikatorem b) jest na rys. 6.6 zaznaczony linią przerywaną.

Przyjęto, że adaptacji podlega



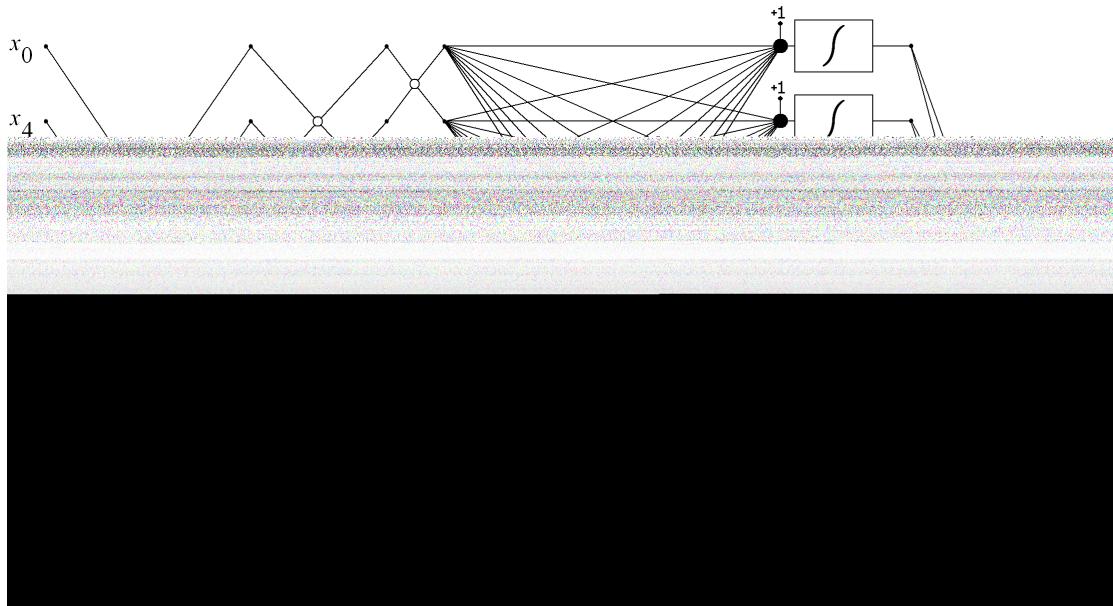
Rysunek 6.6: Architektura proponowanej sieci dla $N = 8$ i $K = 2$

tylko część odpowiadająca drugiemu etapowi transformaty mFCT2. Analizie poddawane są fragmenty surowych danych audio (po normalizacji), a rozmiar N części typu FONN odpowiada ilości próbek zawartych w przetwarzanym fragmencie, przy czym kolejność próbek powinna być podawana na wejście zaprezentowanej sieci w porządku bitowo-inwersyjnym.

Pierwsza warstwa klasyfikatora zawiera N węzłów sumujących wyjścia „ekstraktora” z biasem (połączonym przez wagę, adaptowaną w czasie uczenia). Wartości wyjściowe tych węzłów są przekształcane unipolarną sigmoidalną funkcją aktywacji. Następna (ostatnia) warstwa zawiera K neuronów z biasem i sigmoidalną funkcją aktywacji, z których każdy połączony jest ze wszystkimi wyjściami warstwy poprzedniej. Każde wyjście ostatniej warstwy odpowiada jednej z K docelowych klas.

Obie części tworzą razem sieć neuronową zbliżoną do sieci typu MLP, z istotną

różnicą jaką stanowi rzadki schemat połączeń międzyneuronalnych „warstwy ukrytej”, implementowanej w postaci sieci typu FONN w tym przypadku. To podobieństwo można łatwo zauważyć w porównaniu ze schematem połączeń sieci typu MLP, zaprezentowanym na rys. 6.7



Rysunek 6.7: Architektura sieci referencyjnej z ośmioma neuronami ukrytymi

Sieć neuronowa zaprezentowana na rys. 6.7 została również zaimplementowana i użyta w charakterze punktu odniesienia dla zaproponowanego rozwiązania opartego na sieci FONN. Należy zauważyć, że w klasycznym perceptronie wielowarstwowym wektory wejściowe są przekazywane bezpośrednio do warstwy o gęstym schemacie połączeń. W tym wypadku zastosowano wstępnie dodatkowe warstwy o rzadziej strukturze połączeń z dwóch powodów. Po pierwsze, nie są one poddawane adaptacji, więc mogą być traktowane jako ustalony blok wstępnego przetwarzania danych, identyczny w obydwu porównywanych sieciach neuronowych. W ten sposób można powiedzieć, że adaptacyjne części obu systemów pracują w tych samych warunkach. Po drugie, wyniki wstępnych testów wykazały, że takie rozwiązanie znaczco poprawia wyniki klasyfikacji.

Najistotniejszą różnicą pomiędzy oboma systemami, tzn. pomiędzy klasyfikatorami opartymi na sieci typu FONN oraz na sieci typu MLP, odpowiednio, jest liczba wag poddawanych adaptacji, dana jako:

$$N_w = \frac{N}{2} (\log_2 (16N) + 2K) + K ; \quad (6.1)$$

w przypadku sieci FONN oraz:

$$N_w = (N + 1) \cdot H + (H + 1) \cdot K ; \quad (6.2)$$

w przypadku sieci MLP, gdzie H jest liczbą neuronów ukrytych.

Przykładowo, przyjmując $N = 256$ i $K = 5$, liczba wag sieci MLP w zależności od liczby neuronów ukrytych jest zaprezentowana w tablicy 6.3. Dla porównania, liczba wag w przypadku sieci FONN jest ustalona i równa 2821 dla tych samych wartości N i K , co stanowi mniej niż jest wymagane przez sieć MLP z jedenastoma ukrytymi neuronami.

Tablica 6.3: Ilość wag sieci MLP w zależności od ilości neuronów ukrytych

H	Wagi warstwy ukrytej	Wagi warstwy wyjściowej	Łącznie
2	514	15	529
4	1028	25	1053
8	2056	45	2101
12	3084	65	3149
16	4112	85	4197
24	6168	125	6293
32	8224	165	8389
48	12336	245	12581
64	16448	325	16773
100	25700	505	26205

Materiał i procedura testowania

Materiał audio wykorzystany we wszystkich testach był złożony z 500 krótkich (60s) przykładów muzyki klasycznej podzielonych równo na pięć klas zawierających odpowiednio: muzykę orkiestrową, fortepianową, wokalną, kameralną (kwartet smyczkowy) i jazz. Każdy przykład pochodził z innego utworu, bądź części utworu. Z tej liczby 400 przykładów (po 80 z każdej klasy) stanowiło zbiór treningowy, a pozostałe 100 było użyte jako zbiór testowy. Wszystkie przykłady pochodziły z serwisu internetowego oferującego możliwość nieodpłatnego pobierania fragmentów utworów klasycznych on-line [19]. Dźwięk monofoniczny był nagrywany z częstotliwością próbkowania 22050 Hz (16 bitów na próbce). Każdy przykład ze zbioru treningowego był reprezentowany przez dwanaście odcinków,

z których każdy zawierał po 256 próbek w dziedzinie czasu (ok. 12 ms), a każdy przykład ze zbioru testowego był reprezentowany przez 99 odcinków o tej samej długości. W ten sposób wypadkowy rozmiar zbioru wejściowych wektorów treningowych wynosił 4800×256 próbek, a w przypadku wektorów testowych: 9900×256 . Wszystkie odcinki były wycinane z losowych miejsc w plikach audio z wyłączeniem pięciosekundowych marginesów na początku i na końcu plików. Dwa dodatkowe odcinki z każdego przykładu w zbiorze treningowym były wykorzystane do utworzenia zbioru walidacyjnego o rozmiarze 800×256 , służącego do wizualizacji błędu generalizacji podczas nauki. Wszystkie odcinki ze wszystkich zbiorów były poddawane usunięciu składowej stałej i normalizowane.

Wszystkie odcinki z pojedynczej klasy zbioru treningowego były traktowane identycznie podczas procesu adaptacji, niezależnie od tego, czy pochodziły z tego samego, czy z różnych przykładów. Klasyfikacja zbioru testowego polegała natomiast na analizie wszystkich 99 odcinków reprezentujących pojedynczy przykład. Dla każdego przykładu obliczany był uśredniony wektor wynikowy ze wszystkich 99 wyjść sieci i pozycja elementu o najwyższej wartości określała przypisaną etykietę klasy.

Dwie grupy testów klasyfikacyjnych zostały przeprowadzone w celu porównania możliwości klasyfikatora opartego na sieci FONN oraz sieci typu MLP. Druga grupa, wykorzystująca sieć MLP obejmowała kilka testów dla różnej liczby neuronów ukrytych.

Uzyskane rezultaty i wnioski

Jednym z istotnych problemów pojawiających się podczas testowania, nierozerwalnie związanych z własnościami gradientowych metod optymalizacji, były przypadki utknięcia procesu adaptacji w minimum lokalnym. Z tego względu z całkowitej liczby 20 testów przeprowadzonych dla każdego klasyfikatora i dla każdej liczby neuronów ukrytych wybrano tylko 30% najlepszych wyników. W większości pozostałych testów błąd treningowy dość szybko osiągał fazę *plateau*, a wynik klasyfikacji był bliski losowemu (20%). Tablica 6.4 zawiera wartości średnie z tych wybranych rezultatów, a najlepszy wynik uzyskany dla każdego klasyfikatora jest zaprezentowany w tablicy 6.5.

Podstawowym wnioskiem wynikającym z przeprowadzonych testów jest obserwacja, że klasyfikator oparty na sieci typu FONN był w stanie nauczyć się danych treningowych z dokładnością przekraczającą 75%. Zgodnie z oczekiwaniami wyniki na zbiorze treningowym dla perceptronu (MLP) w dużym stopniu zależały od ilości H zastosowanych neuronów ukrytych. Dla $H < 24$ niemożliwe było uzyskanie nawet pięćdziesięcioprocentowego rozpoznania. Rezultaty dla

Tablica 6.4: Rezultat klasyfikacji (uśredniony)

Klasyfikator	Błąd (tren.)	Liczba epok	Długość epoki [s]	Rozpoznanie (tren.)	Rozpoznanie (test)
MLP (H = 24)	0.364	209.17	1.01.417	48.31%	32.33%
MLP (H = 32)	0.341	275.67	1.02.2009	58.11%	32.17%
MLP (H = 48)	0.319	300.17	1.03.1989	60.93%	27.50%
MLP (H = 64)	0.258	405.67	1.04.1975	72.84%	27.67%
FONN	0.275	1589.17	1.02.138	76.22%	76.17%

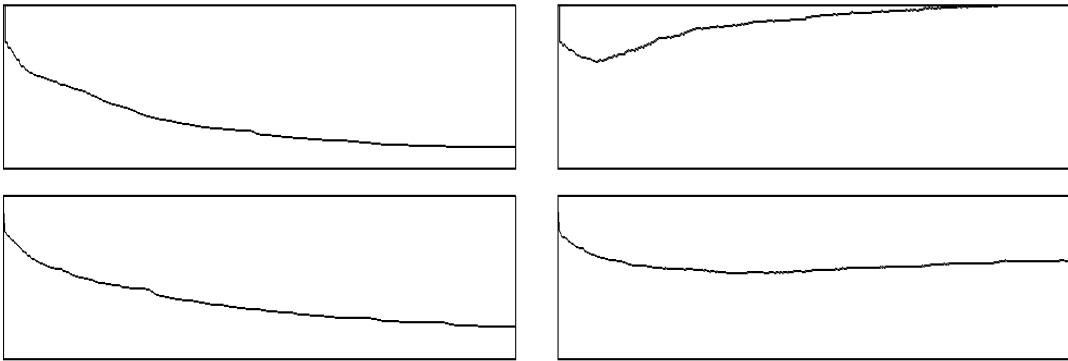
Tablica 6.5: Rezultat klasyfikacji (najlepszy przypadek)

Klasyfikator	Błąd (tren.)	Liczba epok	Długość epoki [s]	Rozpoznanie (tren.)	Rozpoznanie (test)
MLP (H = 24)	0.364	199	1.43	49%	37%
MLP (H = 32)	0.337	339	2.11	60%	36%
MLP (H = 48)	0.362	80	3.71	50%	32%
MLP (H = 64)	0.238	499	4.02	78%	34%
FONN	0.277	1879	2.51	76%	78%

zbioru treningowego porównywalne z wynikami uzyskanymi za pomocą sieci typu FONN wymagały przynajmniej 64 neuronów ukrytych. Jednakże w tym wypadku błąd walidacji wzrastał znacząco po osiągnięciu swojego minimum. Wiązało się to z niskim współczynnikiem rozpoznania (spadającym do niemal losowego poziomu 27.67%) dla zbioru testowego. Pewnym zaskoczeniem była obserwacja, że zatrzymanie procesu adaptacji w minimum błędu walidacji nie przynosiło lepszych wyników dla zbioru testowego. Wręcz przeciwnie, najczęściej najlepsze wyniki na zbiorze testowym uzyskiwane były dla najniższych możliwych wartości błędu treningowego, czyli pod koniec adaptacji. Fakt ten związany jest zapewne ze zwiększoną polaryzacją wyjść neuronów pod koniec nauki, powodującą większy błąd, w przypadku stanów niezgodnych z wartościami zdefiniowanymi wektorami docelowymi, co nie musi przekładać się na błędy klasyfikacyjne (por. przykład na stronie 53). Z tego względu zdecydowano się na uczenie sieci aż do stanu wysycenia dla wszystkich rodzajów klasyfikatorów, zarówno opartych na sieci FONN jak i na sieci MLP.

Najistotniejsza obserwacja związana jest z faktem, iż efekt rosnącego błędu generalizacji praktycznie nie występował w przypadku sieci FONN (rys. 6.8)

W większości przypadków błąd walidacji zmniejszał się początkowo i pozostawał niemal stały przez resztę procesu adaptacji. Ponownie, pomimo że tylko



Rysunek 6.8: Typowe krzywe błędu na zbiorze treningowym (po lewej) i walidacyjnym (po prawej) dla sieci MLP z 64 neuronami ukrytymi (na górze) i sieci FONN (na dole)

błąd treningowy ulegał zmniejszeniu w dalszym etapie procesu adaptacji, osiągając zwykle znacznie mniejsze wartości końcowe od błędu walidacji, skuteczność klasyfikacji na zbiorze testowym również ulegała poprawie, osiągając poziom porównywalny do rezultatów uzyskanych na zbiorze treningowym (ponad 75%).

Ten wynik jest porównywalny do rezultatów zaprezentowanych w pracach Tzanetakisa i Cooka [138], którzy uzyskali wynik klasyfikacji 88% na zbiorze danych zawierającym tylko cztery klasy (chór, orkiestra, fortepian, kwartet smyczkowy) za pomocą dużego zbioru złożonych cech obliczanych zarówno w dziedzinie czasu, jak i częstotliwości. Należy podkreślić, że zbiór danych użyty w naszych badaniach zawierał dodatkowo dość heterogeniczną klasę muzyki jazzowej obejmującą również fragmenty wokalne i fortepianowe.

Macierz pomyłek dla najlepszego klasyfikatora FONN została zaprezentowana w tablicy 6.6. Każdy wiersz tablicy odpowiada faktycznej klasie, każda kolumna prezentuje wyniki klasyfikacji, a przekątna zawiera liczbę prawidłowo rozpoznanych przykładów. Jak można stwierdzić, uzyskane rezultaty z grubsza odpowiadają różnorodności generatorów dźwięku obecnych w ramach poszczególnych klas. Istotnie, najbardziej homogeniczne przykłady z klasy muzyki fortepianowej są rozpoznawane bezbłędnie, podczas gdy fragmenty jazzowe i orkiestrowe, zawierające wielką różnorodność źródeł dźwięku — od perkusji do instrumentów smyczkowych i dętych — następują pewnych trudności we właściwej klasyfikacji.

Należy podkreślić, że zaproponowany system klasyfikacji przeprowadza „ślepą” analizę, nie zakładając żadnej apriorycznej wiedzy na temat własności analizowanych danych. Co więcej, nie brane są pod uwagę żadne długo-okresowe charakterystyki dźwięku, z uwagi na względnie krótki czas trwania fragmentów wycinanych losowo z różnych miejsc plików.

Główną wadą zaproponowanego rozwiązania są długie czasy treningu i ryzyko

Tablica 6.6: Macierz pomyłek dla klasyfikatora FONN

Rodzaj muzyki	kameralna	jazz	orkiestrowa	fortepianowa	wokalna
kameralna	15	3	1	1	0
jazz	3	13	2	2	0
orkiestrowa	1	2	12	4	1
fortepianowa	0	0	0	20	0
wokalna	1	0	0	1	18

utknięcia w minimum lokalnym. Warto jednak wspomnieć, że w większości przypadków łatwo jest wykryć, z dużym prawdopodobieństwem, zatrzymanie adaptacji w minimum lokalnym na samym początku nauki i zrestartować cały proces z nowego, losowego punktu w przestrzeni wag. Jeżeli chodzi o długi czas adaptacji, może on być w wielu zastosowaniach zrównoważony bardzo krótkim czasem propagacji w przód w fazie testowania uprzednio wytrenowanej sieci, wynikającym z braku konieczności obliczania wektorów cech, występującej typowo w innych rozwiązaniach. Wartości uzyskane podczas testów przeprowadzanych na komputerze z procesorem Intel Celeron M, 1.40 GHz wynosiły zwykle ok. 19ms dla całego przykładu, co daje ok. 0.19ms dla pojedynczego odcinka (256 próbek).

Podsumowując uzyskane wyniki warto podkreślić niskie wartości błędu generalizacji, sugerujące iż zaproponowany klasyfikator oparty na sieci typu FONN posiada — w przypadku zadanego problemu klasyfikacji rodzaju muzyki — architekturę bliską optymalnej, nie wymagającą np. dodatkowego obcinania wag, czy innych modyfikacji. Zaprezentowany system zapewnia możliwość uzyskania wyników zbliżonych do rozwiązań innych autorów, pomimo istotnych różnic w zasadzie funkcjonowania, a w szczególności pomimo radykalnego uproszczenia fazy przygotowania danych do właściwej klasyfikacji, wyrażonego faktycznym brakiem tradycyjnie rozumianej ekstrakcji cech.

6.2 Klasyfikacja i rozpoznawanie obrazów

Analogicznie do zaprezentowanych powyżej przykładów klasyfikacji danych reprezentowanych w postaci ciągów, sieć typu FONN oparta na dwuwymiarowym przekształceniu Fouriera (podrozdział 5.4.5) została wykorzystana do konstrukcji systemu rozpoznawania obrazów. Zastosowano tę samą metodę połączenia szybkiej ortogonalnej sieci neuronowej z dodatkową warstwą wyjściową o ilości neuronów równej ilości klas. Przeprowadzono trzy grupy testów w oparciu zarówno o opraco-

wane samodzielnie zbiory danych, jak i z wykorzystaniem klasycznych baz obrazów do rozpoznawania (COIL, ETH). W podrozdziale 6.2.1 zaprezentowano wyniki kategoryzacji obiektów na podstawie ich zdjęć wykonanych z różnych punktów przestrzeni 3D [117]. W kolejnych dwóch podrozdziałach poddano analizie bardziej złożony przypadek, w którym pewne własności transformaty Fouriera (stanowiącej podstawę architektury sieci) zostały wykorzystane do uzyskania inwariancji względem obrotu, skalowania i translacji obiektu na obrazie.

6.2.1 Klasyfikacja i kategoryzacja obrazów

Rozpoznawanie obrazu (ang. *image recognition*, IR) jest jednym z kluczowych, a przy tym wciąż otwartych, obszarów badań nad sztuczną inteligencją [29][85][133]. Z jednej strony wysoki wymiar i redundancja danych wizualnych zmusza do poszukiwań efektywnych algorytmów ekstrakcji cech. Z drugiej strony, praktycznie nieograniczona zmiennaść możliwych układów obiektów i tła w ramach pojedynczej sceny sprawia, że wybór odpowiednich cech jest rzeczą bardzo trudną w ogólnym przypadku. Zadowalający poziom odporności na zniekształcenia i zakłócenia określonych rodzajów często wymaga ręcznego, starannego doboru cech, co znacząco ogranicza autonomię systemów IR.

Radykalnym rozwiązaniem może być pominięcie etapu ekstrakcji cech i klasyfikacja surowych lub prawie surowych danych przez odpowiedni klasyfikator. Takie podejście może przynieść bardzo dobre efekty, jeśli rozmiary obrazu są odpowiednio ograniczone, a zbiór danych dostatecznie duży [75]. Należy jednak pamiętać, że zwiększanie wymiarowości przestrzeni wejściowej łatwo prowadzi do „przeuczenia” klasyfikatora, co skutkuje wzrostem błędu generalizacji.

Rozwiązaniem tego problemu może być kontrolowanie złożoności klasyfikatora, co w przypadku wielowarstwowego perceptronu (MLP) zwykle oznacza redukcję rozmiaru warstw ukrytych. W tym podrozdziale zbadane zostanie inne podejście, oparte na sieci typu FONN zbudowanej na bazie algorytmu dwuwymiarowego przekształcenia Fouriera z mnożnikami tangensowymi, uzupełnionego o dodatkową warstwę umożliwiającą obliczanie elementów widma amplitudowego (p. podrozdział 5.4.5).

Materiał i procedura testowania

Zaproponowana sieć została wykorzystana do klasyfikacji obrazów z bazy ETH-80 [77]. Ścisłe mówiąc, sposób konstrukcji tego zbioru obrazów wyznacza zadanie określone przez jego twórców jako „kategoryzacja” obrazów, czyli proces przypisywania właściwej *kategorii* nieznanemu obiekowi reprezentowanemu przez jego

zdjęcia. Baza ETH-80 zawiera zdjęcia 80 obiektów należących do ośmiu kategorii (rys. 6.9). Każdy obiekt reprezentowany jest przez 41 zdjęć w odcieniach szarości



Rysunek 6.9: Wszystkie obiekty z kategorii „samochód” w tym samym położeniu i rozdzielczości 256×256 , wykonanych z punktów równomiernie rozmieszczonych na otaczającej go półkuli.

Test weryfikujący poprawność kategoryzacji nieznanego obiektu (*leave-one-out crossvalidation*) zakłada użycie w charakterze zbioru uczącego całej bazy z wyjątkiem wszystkich 41 zdjęć przedstawiających ten obiekt. Następnie sprawdzana jest poprawność przypisania każdego z tych zdjęć do właściwej kategorii obiektu. Łączny wynik otrzymywany jest przez powtórzenie tej procedury dla wszystkich 80 obiektów w bazie.

Testy przeprowadzono z użyciem dwóch różnych klasyfikatorów: opartego na sieci typu FONN (rys. 5.33) uzupełnionej o liniową warstwę wyjściową oraz opartego na sieci typu MLP. W obu przypadkach przetwarzanie wstępne było ograniczone do zmniejszenia rozmiarów obrazu do wymiarów $N \times M = 32 \times 32$, normalizacji i usunięcia składowej stałej. W przypadku sieci MLP wykorzystano kilka różnych ilości neuronów ukrytych H (ozn. jako $MLPH$). Wyniki rozpoznawania zostały zaprezentowane w tablicy 6.7.

Uzyskane rezultaty i wnioski

Wyższość zaproponowanej architektury neuronowej nad klasyczną, „gęstą” siecią jest widoczna nie tylko w znacząco lepszych wynikach rozpoznawania, ale również w zredukowanej liczbie wag podlegających adaptacji. Zastosowanie sieci FONN umożliwia uniknięcie problemu poszukiwania optymalnej liczby neuronów ukrytych. Ostatnia kolumna tablicy 6.7 pokazuje, że bardziej korzystne może być nawet zwiększenie wymiaru przestrzeni wejściowej i zastosowanie sieci FONN, zamiast zwiększania liczby neuronów ukrytych w sieci MLP, co w przypadku analizowanego problemu klasyfikacyjnego może łatwo prowadzić do spowalniania procesu adaptacji bez poprawy wyników klasyfikacji. Ponieważ uzyskany poziom rozpo-

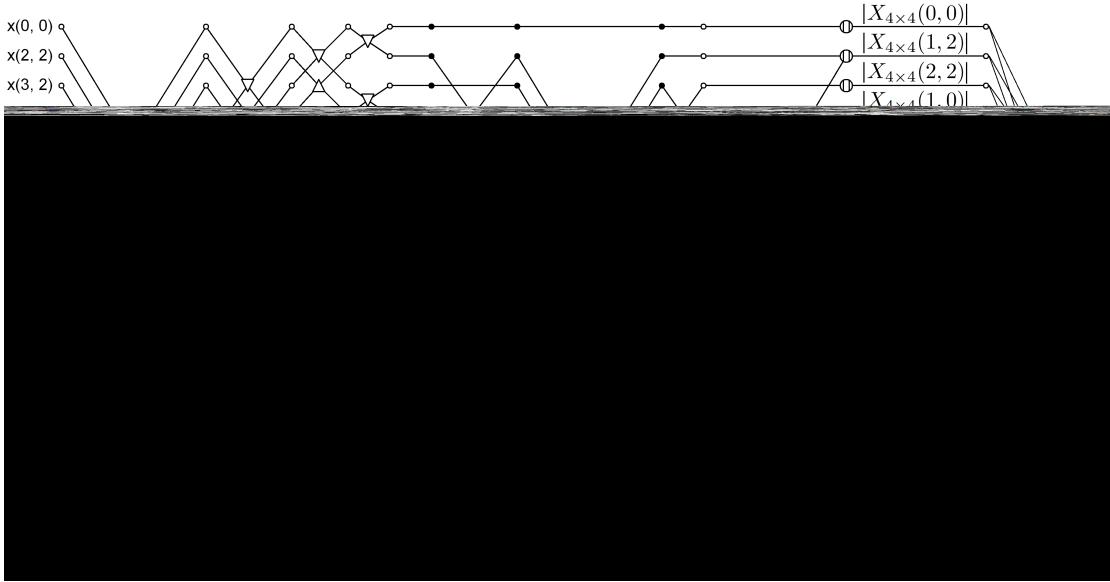
cech wyróżnić wstępna fazę tzw. *rejestracji* obrazu (ang. *image registration*), związanej z określeniem właściwego układu współrzędnych w jakich rozpatrywana ma być treść obrazu⁵. Faza ta ma kluczowe znaczenie dla wyników klasyfikacji, jako że większość systemów IR jest z natury wysoce wrażliwa na błędy rejestracji. W związku z tym rezultat rozpoznawania zależy od właściwego określenia parametrów możliwych transformacji aficznich rozpoznawanego obiektu i — co najistotniejsze — znalezienia samego obiektu, co może być problematyczne w przypadku obrazów rzeczywistych zawierających inne obiekty i zaszumione (przypadkowe) tło. Biorąc pod uwagę fakt, że skuteczna segmentacja i analiza złożonych scen stanowi wciąż trudny problem, warto jest rozważyć kwestię: czy sam klasyfikator mógłby być w pewnym stopniu odporny na transformacje aficznne surowych (niepoddanych rejestracji) obrazów?

Ponownie zakładamy tu maksymalne zredukowanie wstępnego przetwarzania obrazów poprzedzającego właściwą klasyfikację, tym razem w kontekście odporności na wybrane przekształcenia aficznne. Warto zauważyć, że LeCun *et al* w badaniach nad rozpoznawaniem obrazów z bazy MNIST [75] również podkreślali problem niezależności od translacji w kontekście adaptacji sieci neuronowej, przy czym do jego rozwiązania stosowali konwolucyjną sieć neuronową (ang. *convolutional neural network*) opartą na koncepcji replikowania konfiguracji wag neuronowych w przestrzeni.

W tym podrozdziale przedstawione zostanie inne rozwiązanie: analogia do własności niezależności od przesunięcia właściwej dla widma amplitudowego Fouriera [89][32][40] pozwoli nam otrzymać sieć neuronową, opartą na strukturze typu FONN, invariantną względem przesunięcia oraz rotacji i skalowania obrazu wejściowego [116]. W odróżnieniu od typowych rozwiązań opartych na widmie amplitudowym Fouriera, zaproponowany klasyfikator jest w stanie wykorzystać również informację zawartą w widmie fazowym do poprawy rezultatów klasyfikacji, co zostało zweryfikowane eksperymentalnie w przykładowych zadaniach klasyfikacyjnych.

Zaproponowany klasyfikator składa się z dwóch głównych części: sieci typu FONN zbudowanej w oparciu o algorytm dwuwymiarowego przekształcenia Fouriera, uzupełnionej o warstwę obliczającą widmo amplitudowe oraz z warstwy wyjściowej, w której liczba neuronów odpowiada liczbie klas (rys. 6.10). Połączenia oznaczone symbolem (*) wymagają dodatkowego wyjaśnienia. Są to zwykłe połączenia jeden-do-jednego, bez żadnych adaptowalnych współczynników, wymagające trywialnej modyfikacji procedury propagacji wstecz. Nie są one konieczne

⁵Jest to inne znaczenie słowa „rejestracja”, niż w przypadku rejestracji (akwizycji) obrazu za pomocą kamer, czujników, etc.

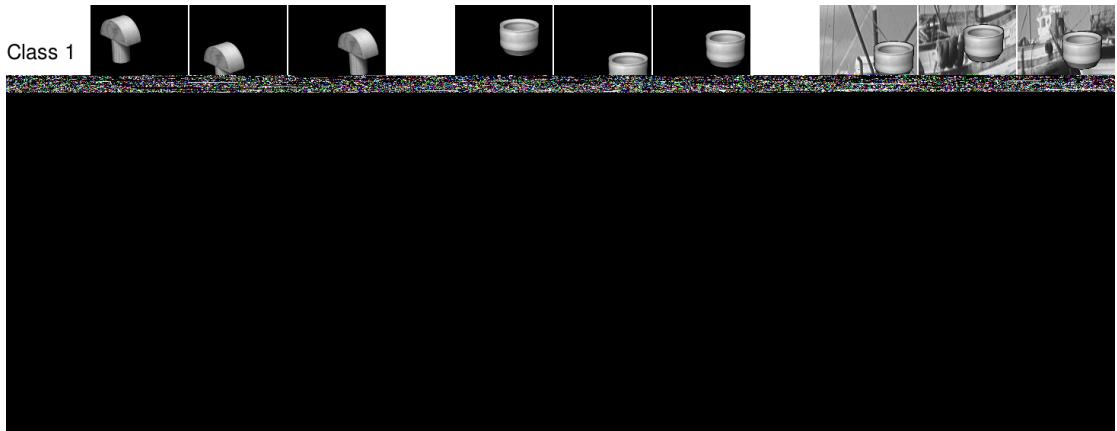


Rysunek 6.10: Kompletna architektura neuronowa dla obrazów wejściowych o rozmiarze $N \times N = 4 \times 4$ i dwóch klas

do obliczenia widma amplitudowego, które jest uzyskiwane z pierwszych $\frac{N^2}{2} + 2$ wyjść tej warstwy (warto zauważyć, że z uwagi na symetrię widma dla obrazów rzeczywistych [134] jest to dokładnie liczba wszystkich znaczących składników widmowych). Niemniej jednak, podczas testów weryfikujących możliwość wykorzystania informacji fazowej okazało się, że połączenia te znacząco poprawiają klasyfikację. Chociaż sieć była w stanie wykorzystać informację fazową w rozpoznawaniu obrazów również bez tych połączeń, co stanowi istotną i interesującą obserwację, jednak uwzględnienie ich umożliwiło osiągnięcie jeszcze lepszych rezultatów.

Materiał i procedura testowania

Niezależność od translacji Naszym podstawowym celem jest wykazanie, że zaproponowana sieć może klasyfikować obiekty niezależnie od ich pozycji na obrazie, w przeciwieństwie do typowych zastosowań sieci neuronowych, w których sygnał wejściowy powinien być dokładnie „wyrównany” w czasie/w przestrzeni [42]. Sieć typu MLP została początkowo wybrana jako podstawa porównania, podobnie jak we wcześniejszych podrozdziałach. Jednak z uwagi na problemy praktyczne, m.in. długi czas adaptacji i trudności z określeniem odpowiedniej liczby neuronów ukrytych, zdecydowano się na inne rozwiązanie. Ponieważ porównanie ze standardowym klasyfikatorem typu KNN (ang. „ K ” nearest neighbours) [133] po wielu przeprowadzonych eksperymentach wykazało, iż ten ostatni zapewnia zbliżone (w większości przypadków lepsze) wyniki, zatem przeprowadzono tylko nie-



Rysunek 6.11: Przedstawiciele poszczególnych klas (trzy obrazy z każdej klasy) w bazach **T1**, **T2**, **T3**

wielką część badań z użyciem sieci perceptronowej i przedstawiono ich wyniki dla różnej ilości neuronów ukrytych. We wszystkich pozostałych przypadkach testów zastosowano klasyfikator KNN z liczbą sąsiadów $K = 1$.

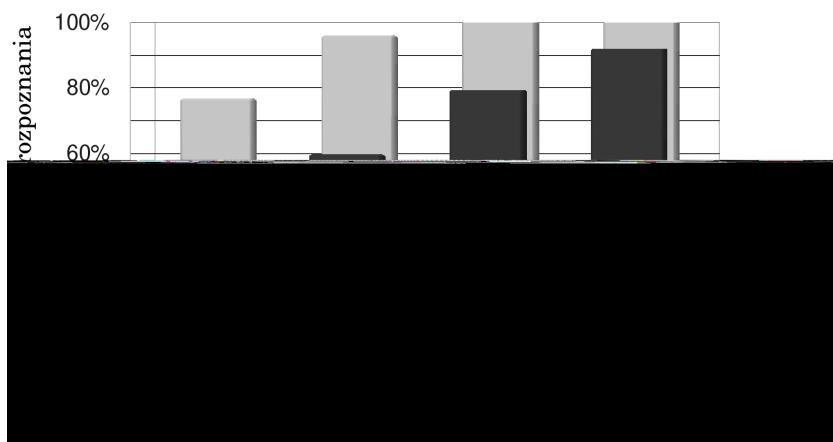
W celu weryfikacji możliwości uzyskania niezależności od translacji obrazu za pomocą zaproponowanej sieci stworzono trzy bazy danych: **T1**, **T2** i **T3**. Do ich konstrukcji wykorzystano obiekty z bazy COIL [92] przedstawione w wybranym położeniu (rys. 6.11).

- **T1:** Ta baza zawiera 4 obiekty (klasy) o tym samym rozmiarze i położeniu przedstawione na czarnym tle w losowych pozycjach.
- **T2:** Ta baza zawiera dwa obiekty o tym samym rozmiarze i położeniu przedstawione na czarnym tle. Każdy obiekt jest użyty do konstrukcji dwóch klas: w jednej klasie występuje w losowych pozycjach w prawej połowie obrazu, a w drugiej klasie występuje w połowie lewej.
- **T3:** Ta baza jest skonstruowana tak samo jak **T2**, ale obiekty są zaprezentowane na tle uzyskanym przez wycięcie losowego kwadratowego fragmentu z innego obrazu (wykorzystano klasyczny obraz *Fishing boat*).

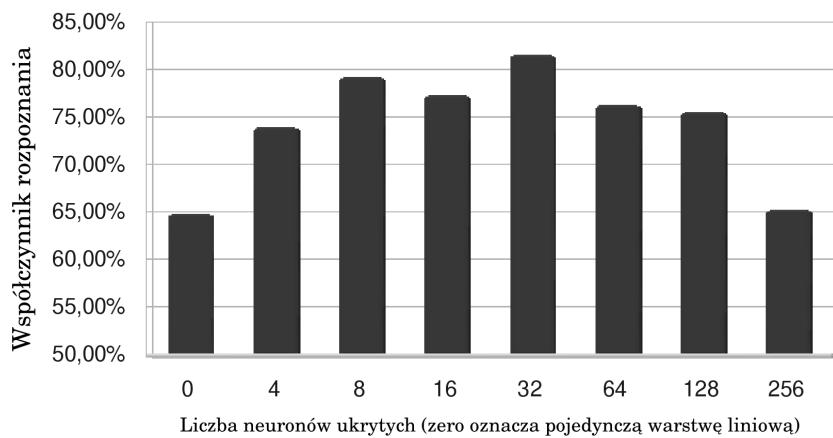
Każda baza danych była generowana wielokrotnie z różną liczbą obrazów treningowych w celu zapewnienia możliwości oceny własności generalizacyjnych klasyfikatora. Sto obrazów testowych było wykorzystanych we wszystkich przypadkach, a zbiór walidacyjny miał rozmiar równy jednej-czwartej rozmiaru zbioru treningowego (z zaokrągleniem w góre do najbliższej liczby całkowitej

ich właściwej klasyfikacji. Tak uzyskane 1024-elementowe wektory wejściowe były poddane normalizacji i usunięciu składowej stałej.

Porównanie pomiędzy wynikami uzyskanymi przez sieć FONN i klasyfikator KNN dla bazy **T1** zaprezentowano na rys. 6.12. Rysunek 6.13 prezentuje wyniki otrzymane za pomocą klasyfikatora MLP dla 50 obrazów treningowych z każdej klasy (łącznie 200 obrazów) z wykorzystaniem różnej liczby neuronów ukrytych. Wszystkie zaprezentowane wartości (z wyjątkiem wyników klasyfikatora KNN) są wynikami uśrednionymi z dziesięciokrotnie powtarzanej fazy adaptacji sieci z losowego punktu w przestrzeni wag.



Rysunek 6.12: Rezultaty klasyfikacji(baza **T1**)



Rysunek 6.13: Rezultaty klasyfikacji bazy **T1** z 50 obrazami na każdą klasę dla sieci MLP

Poza samymi wynikami sieci MLP, które są znaczco niższe od uzyskanych za pomocą klasyfikatora KNN (81.4% dla 32 neuronów ukrytych vs 91.8%, odpowiednio), również proces adaptacji był długotrwały i często zatrzymywał się przy wysokich wartościach błędu treningowego. Minimum krzywej błędu walida-

cji było zwykle osiągane po kilkuset epokach. Dla kontrastu, klasyfikator oparty na sieci typu FONN potrzebował średnio 61 epok, w celu osiągnięcia stuprocentowego rozpoznania na zbiorze walidacyjnym. Należy podkreślić, że obie sieci neuronowe były testowane w tych samych warunkach, łącznie z tymi samymi danymi wejściowymi/docelowymi, metodą adaptacji i innymi parametrami. Warto również zauważyć, że wynik klasyfikacji na zbiorze *treningowym* był dla obu sieci równy 100% we wszystkich przypadkach, łącznie z najmniejszym zbiorem treningowym złożonym z 5 obrazów na klasę. A zatem, otrzymane rezultaty mogą być traktowane w kategoriach błędu generalizacji.

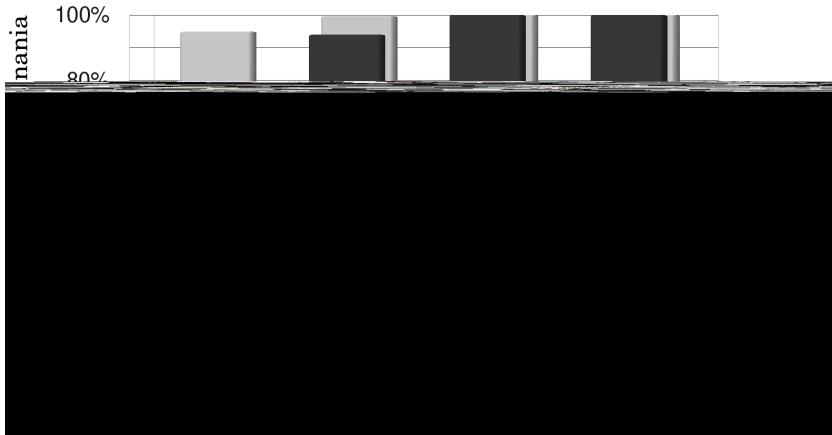
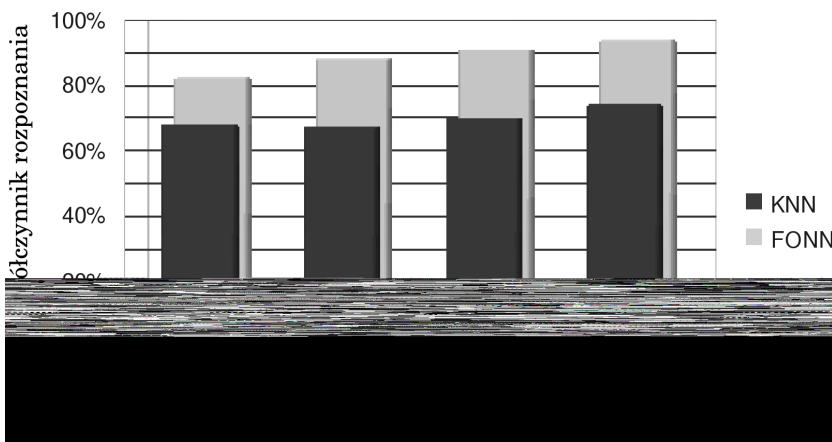
Prostym rozwiązaniem problemu niezależności od translacji jest poddanie analizie amplitudowego widma Fouriera zamiast surowego obrazu. Istotnie, klasyfikator KNN łatwo uzyskiwał stuprocentowy wynik klasyfikacji, nawet w oparciu o 5 obrazów na klasę, w sytuacji gdy użyto widma amplitudowego w charakterze danych wejściowych. Ten sam rezultat uzyskano w ciągu 2-5 epok w przypadku klasyfikatora FONN, analizującego surowe obrazy, ale z wagami neuronów preinicjalizowanymi zgodnie z algorytmem transformaty Fouriera.

Wyższość zaproponowanego „adaptacyjnego przekształcenia Fouriera” można zademonstrować w sytuacji, gdy informacja fazowa musi być również uwzględniona w procesie klasyfikacji. Uzyskanie wyniku ponad 50% w przypadku bazy **T2** w oparciu o samo widmo amplitudowe jest niemożliwe, ponieważ obrazy z klas 1, 3 oraz z klas 2, 4 mają takie same widma amplitudowe. Klasyfikator oparty na sieci typu FONN może skorzystać zarówno ze swojej własności niezależności od przesunięcia, jak i z potencjału adaptacyjnego. Wyniki zaprezentowane na rysunkach 6.14, 6.15 potwierdzają wysoką przydatność zaproponowanego klasyfikatora, najlepiej widoczną w przypadku, gdy obiekty prezentowane są na zaszumionym tle (rozpoznanie surowych obrazów z bazy **T3** dla 300 obiektów treningowych na klasę: 93.6% dla sieci FONN vs 73.8% dla KNN).

Niezależność od rotacji, translacji i skalowania Jedną z podstawowych metod uzyskania reprezentacji obrazu niezależnej od translacji i skalowania jest zastosowanie transformaty Fouriera-Mellina zdefiniowanej jako [33][119][127]:

$$M_f(\theta; k) = \frac{1}{2} \int_0^\infty \int_0^{2\pi} f(r; \theta) r^{-iv} e^{-ik\theta} d\theta \frac{dr}{r}; \quad (6.3)$$

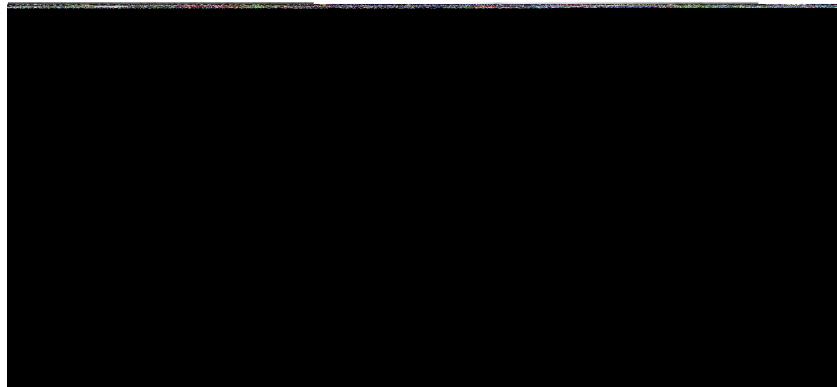
gdzie $(\theta; k) \in \mathbb{R} \times \mathbb{Z}$, $\theta \in [0; 2\pi]$, $r > 0$, zaś f jest ciągłą funkcją reprezentującą obraz w biegunowym układzie współrzędnych. W praktyce, dysponując obrazem cyfrowym reprezentację tę obliczamy w dwóch etapach – zamieniając rotację i skalowanie obrazu na translację, od której następnie uniezależniamy się obliczając

Rysunek 6.14: Rezultaty klasyfikacji(baza **T2**)Rysunek 6.15: Rezultaty klasyfikacji(baza **T3**)

amplitudowe widmo transformaty Fouriera [89].

Rotacja i skalowanie obrazu może zostać zredukowana do translacji jeśli dokonać zmiany układu współrzędnych z kartezjańskiego na logarytmiczno biegunowy [32] [89] [33] [118]. Trudnością, której nie można tu zaniedbać jest konieczność uprzedniego wyeliminowania translacji obrazu, ponieważ nawet niewielkie przesunięcia mogą wprowadzić ogromne różnice w reprezentacji logarytmiczno biegunowej. A zatem metoda uzyskiwania niezależności od przesunięcia musi być zastosowana najpierw. Metody oparte na reprezentacji w dziedzinie czasu (przestrzennej), takie jak środek ciężkości, deskryptory kształtu, czy konturów [32] [42] zależą w istotny sposób od jakości segmentacji, która może nie dać zadowalających rezultatów dla rzeczywistych obrazów. Alternatywą jest wykorzystanie widma amplitudowego obrazu, zapewniającego niezależność od translacji, a przy tym zachowującego obrót i skalowanie (z odwrotnością współczynnika skali). Przedstawienie amplitudowego widma Fouriera w reprezentacji logarytmiczno biegunowej (FFT-LPT) zapewnia zatem niezależność od translacji i jednocześnie zamienia ro-

tację i skalowanie w translacje (rys. 6.16) [89]. Uzyskanie inwariancji względem

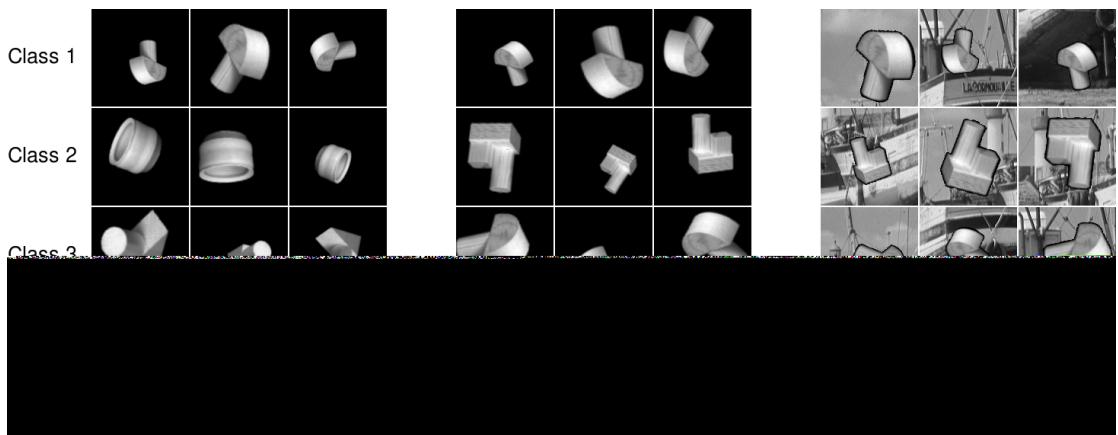


Rysunek 6.16: a) Widmo amplitudowe obrazu; b) Reprezentacja logarytmiczno-biegunkowa

rotacji, translacji i skalowania (RTS) przy użyciu logarytmiczno-biegunkowej reprezentacji widma amplitudowego jest możliwe w podobny sposób, jak to miało miejsce w przypadku przedstawionej wyżej metody uzyskiwania niezależności od translacji dla surowych obrazów. Możemy zatem obliczyć widmo amplitudowe Fouriera ponownie (FFT-LPT-FFT) i dokonać klasyfikacji, bądź zastosować klasyfikator oparty na sieci FONN w tym samym celu. Dokonamy teraz porównania tych dwóch metod demonstrując zalety wykorzystania podejścia opartego na sieci FONN w stosunku do klasyfikatora KNN.

Trzy bazy obrazów testowych: **RTS1**, **RTS2** i **RTS3** zostały skonstruowane w sposób analogiczny do zbiorów **T1**, **T2** i **T3** (rys. 6.17).

- **RTS1:** Ta baza zawiera 4 różne obiekty (klasy) przedstawione na czarnym tle. Obrazy w obrębie każdej klasy różnią się kątem obrotu z zakresu (0° , 360°), współczynnikiem skali z zakresu (0.9, 1.5) i pozycją.



Rysunek 6.17: Przedstawiciele poszczególnych klas (trzy obrazy z każdej klasy) w bazach **RTS1**, **RTS2**, **RTS3**

- **RTS2:** Ta baza zawiera dwa obiekty. Każdy obiekt jest użyty do konstrukcji dwóch klas: w jednej klasie jest obrócony o losowy kąt z zakresu (-30° , $+30^\circ$) lub (150° , 210°), co oznacza w praktyce pozycję „zblizioną do pionowej”; w drugiej klasie kąt jest losowany z zakresu (60° , 120°) lub (240° , 300°), zapewniającego pozycję „zblizioną do poziomej”. Oprócz tego współczynnik skali i pozycja są losowane dokładnie tak samo jak w bazie RTS1.
- **RTS3:** Ta baza jest skonstruowana tak samo jak **RTS2**, ale tło jest otrzymywane tak jak w przypadku bazy **T3**.

Procedura testowania była taka sama jak w przypadku zbiorów **T1 – T3** z tym wyjątkiem, że dla każdego obrazu wejściowego o rozmiarach ($w_{or} \times h_{or}$) = (128×128) obliczane było widmo amplitudowe Fouriera o tym samym rozmiarze, które następnie przekształcano do reprezentacji logarytmiczno-biegunkowej o rozmiarze ($w; h$) = (32×32). Ta reprezentacja była następnie normalizowana i poddawana klasyfikacji, dokładnie tak samo jak w przypadku surowych obrazów w przypadku baz **T1 – T3**.

Samo przekształcenie logarytmiczno-biegunkowe zrealizowane zostało metodą *inverse mapping* [42], w której dla każdego piksela obrazu wynikowego o współrzędnych ($: :$) obliczane były współrzędne piksela obrazu oryginalnego ($x; y$) zgodnie z zależnością odwrotną:

$$\begin{aligned} r &= e^{\frac{\rho-c}{a}} - b ; \\ x &= -r \cos \left(\frac{2}{h-1} \cdot ' \right) + \frac{w_{or}}{2} ; \\ y &= -r \sin \left(\frac{2}{h-1} \cdot ' \right) + \frac{h_{or}}{2} ; \end{aligned} \quad (6.4)$$

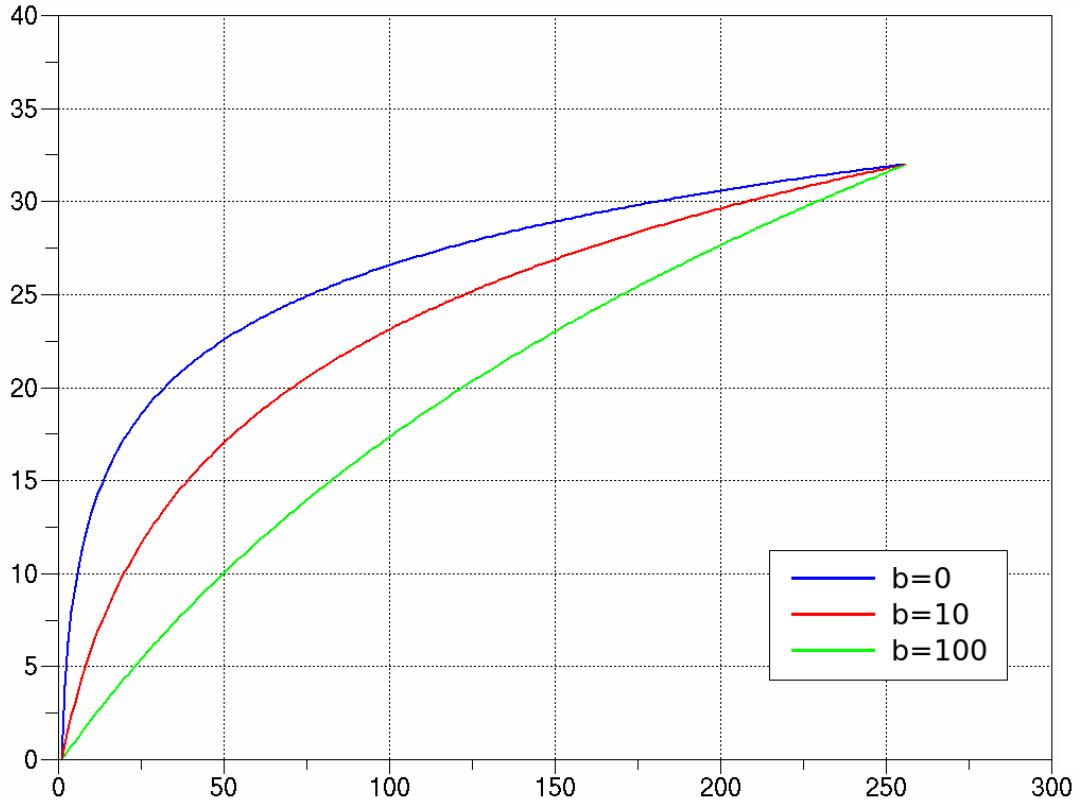
gdzie $' = 0; 1; \dots; h-1$, $= 0; 1; \dots; w-1$, a parametry a i c są definiowane jako:

$$\begin{aligned} a &= \frac{w-1}{r_{max} - r_{min}} ; \\ c &= \frac{(1-w) \cdot (r_{min} + b)}{r_{max} - r_{min}} ; \end{aligned} \quad (6.5)$$

Parametry r_{min} i r_{max} decydują o obszarze obrazu oryginalnego podlegającym mapowaniu logarytmiczno-biegunkowemu. Ich wartości po wstępnych testach zostały ustalone na: $r_{min} = 2$; $r_{max} = w_{or} = 2-1 = h_{or} = 2-1 = 63$. Ostateczną wartość piksela w reprezentacji logarytmiczno-biegunkowej uzyskiwano na podstawie wartości ($x; y$) obliczanych wg wzoru (6.4) w oparciu o interpolację dwuliniową [42].

Parametr b wpływa na kształt uzyskiwanej krzywej logarytmicznej, a zatem pośrednio na rozmiar reprezentacji logarytmiczno-biegunkowej poszczególnych ob-

szarów oryginalnego obrazu (rys. (6.18)) Należy jednak zauważać, że podstawa-



Rysunek 6.18: Wpływ parametru b na kształt krzywej logarytmicznej

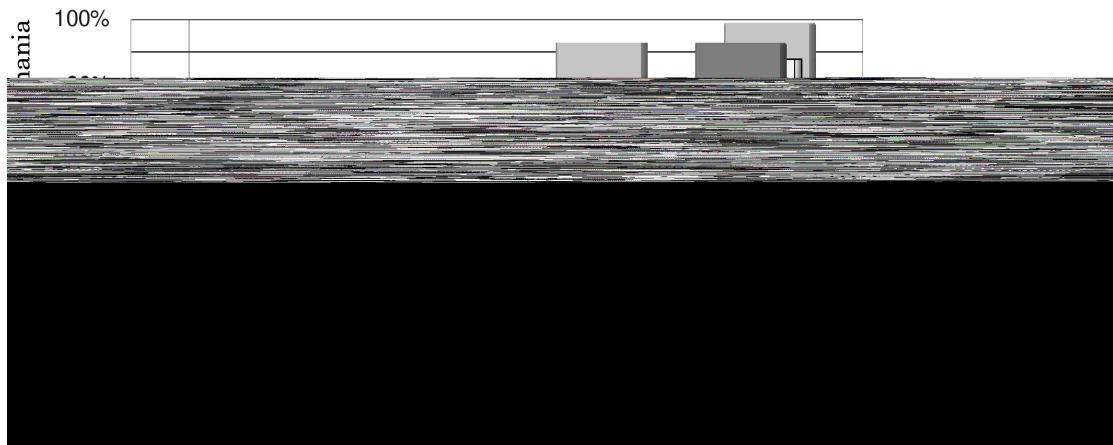
wym celem stosowania przekształcenia logarytmicznego jest tu zamiana skalowania na translację. Skalując obraz oryginalny ze współczynnikiem skali s , na podstawie (6.4) mamy:

$$s \cdot r = s \cdot e^{\frac{\rho-c}{a}} - s \cdot b = e^{\frac{\rho+a \ln(s)-c}{a}} - s \cdot b : \quad (6.6)$$

Widzimy zatem, że konieczne jest tutaj przyjęcie wartości $b = 0$. Tylko wówczas skalowanie ze współczynnikiem s jest dokładnie równoważne przesunięciu reprezentacji logarytmiczno-biegunkowej wzduł wzdłuż współrzędnej r o wartość $a \ln(s)$.

Dwa dodatkowe szczegóły są tu warte zauważenia. Po pierwsze, z uwagi na własność symetrii widma Fouriera dla sygnałów rzeczywistych [134] tylko połowa widma amplitudowego jest istotna. A zatem, parametry próbkowania logarytmiczno-biegunkowego były dobrane tak, aby otrzymywać reprezentację w rozmiarze 64×32 , która była następnie obcinana do ostatecznego rozmiaru 32×32 . Po drugie, jeszcze przed normalizacją intensywność (jasność) otrzymywanej reprezentacji była transformowana funkcją logarytmiczną w celu zwiększenia zakresu dynamicznego wysokoczęstotliwościowych składników widma [42].

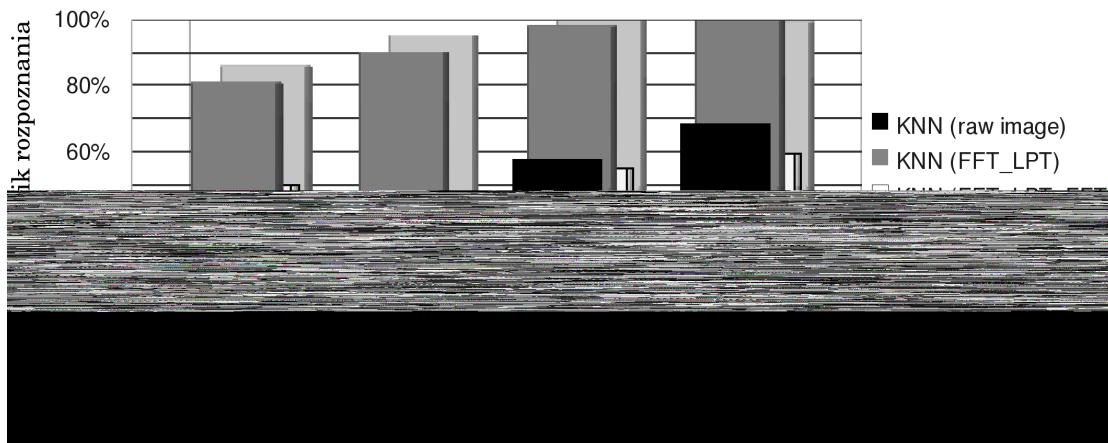
Porównanie pomiędzy wynikami sieci FONN i klasyfikatora KNN dla bazy **RTS1** zostało zaprezentowane na rys. 6.19. Zaprezentowano tu dwa dodatkowe



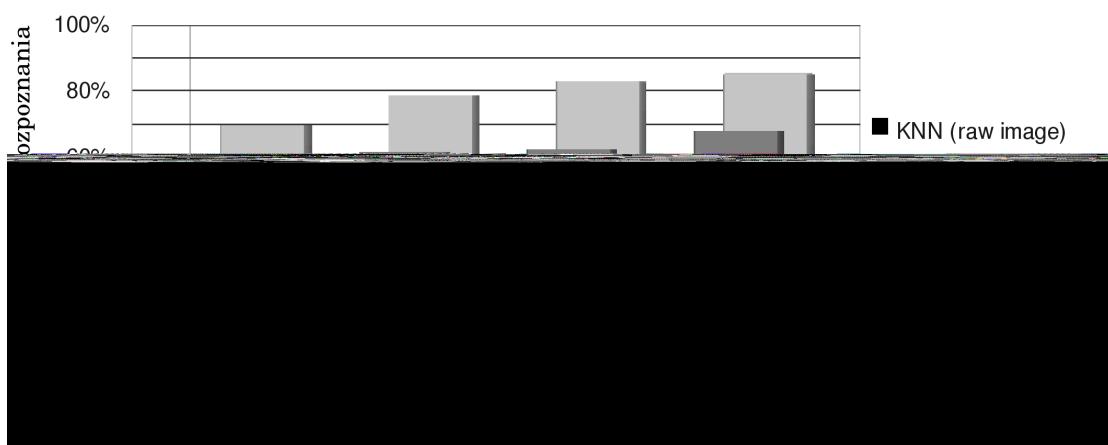
Rysunek 6.19: Wyniki klasyfikacji (baza **RTS1**)

wyniki, oprócz tych uzyskanych w oparciu o rozważaną reprezentację (FFT-LPT) za pomocą sieci FONN (kolor jasnoszary) i klasyfikatora KNN (ciemnoszary). Wyniki te uzyskano za pomocą: klasyfikatora KNN dokonującego klasyfikacji na tej samej reprezentacji jak w przypadku zbiorów **T1 – T3**, tj. na podstawie surowych obrazów (kolor czarny) oraz klasyfikatora KNN analizującego widmo amplitudowe rozważanej reprezentacji (FFT-LPT-FFT, kolor biały). Ten ostatni wynik pokazuje, iż dla obiektów poddanych odpowiedniej segmentacji, podejście oparte na modelu nie-adaptowanym, w pełni invariantnym względem przekształceń RTS daje dobre rezultaty. Jednak w bardziej złożonym przypadku, w którym informacja fazowa musi być również uwzględniona, ta metoda zdecydowanie zawodzi. Bazy **T2** i **T3** są skonstruowane w taki sposób, że przesunięcie reprezentacji FFT-LPT wzdłuż osi pionowej, odpowiadające obrotowi widma i — tym samym — obrotowi obrazu o ten sam kąt, nie może być zignorowane. W tym przypadku tylko operowanie na reprezentacji FFT-LPT umożliwia dokonanie poprawnego rozpoznawania, co zostało zademonstrowane przez klasyfikatory FONN i KNN (FFT-LPT). Istotną różnicą pomiędzy nimi jest to, że sieć FONN w istocie zapewnia niezależność od przesunięcia, z uwagi na swoją strukturalną analogię do algorytmu obliczania amplitudowego widma Fouriera, nie wykluczając jednocześnie wykorzystania informacji fazowej. W przeciwnieństwie do tego, klasyfikator KNN zapewnia podejście statystyczne, które może „symulować” niezależność od translacji w przypadku zapewnienia dostatecznie dużej ilości wzorców treningowych, jak zostało to pokazane dla zbiorów **T1 – T3**. Niemniej jednak, nie dysponuje on zasadniczo „wbudowaną” wiedzą na temat faktycznego modelu przekształceń afinicznych, które są nieistotne z punktu widzenia klasyfikacji. Wyniki uzyskane

dla bazy **RTS2**, a zwłaszcza dla bazy **RTS3** (rysunki 6.20 i 6.21) wydają się potwierdzać to wnioskowanie.



Rysunek 6.20: Wyniki klasyfikacji (baza **RTS2**)



Rysunek 6.21: Wyniki klasyfikacji (baza **RTS3**)

Wynik rozpoznawania za pomocą sieci FONN przy 50 obrazach treningowych z zaszumionym tłem w każdej klasie (69.56%) jest porównywalny z rezultatem jaki może zapewnić metoda KNN dla 300 obrazów treningowych (67.8%). Użycie 300 obrazów w przypadku klasyfikatora opartego na sieci typu FONN umożliwia uzyskanie współczynnika rozpoznania 85.06% co można uznać za dobry wynik dla tego szczególnego zbioru danych.

Podsumowanie i wnioski

Analizując otrzymane rezultaty warto przeanalizować zaproponowaną metodę klasyfikacji w nieco szerszym kontekście. Bazy obrazów w rodzaju **T1** i **T2** stanowiłyby łatwe, niemal trywialne zadanie dla większości istniejących współcześnie

systemów rozpoznawania obrazów, szczególnie, gdyby niezależność od przesunięcia i istotność pozycji obiektów w bazie **T2** była zdefiniowana z góry i uwzględniona w algorytmie klasyfikacji. Należy jednak zwrócić uwagę, że ogólne metody klasyfikacji nie dostosowują się łatwo i automatycznie do zmian lokalizacji obiektów, jak można to było zaobserwować na przykładzie klasyfikatorów KNN i MLP.

Skuteczna segmentacja obiektów z bazy **T3** byłaby problematyczna z uwagi na mnogość i różnorodność szczegółów obecnych w tle. Potencjalne metody rozwiązania tego problemu mogłyby opierać się na technikach korelacyjnych [42], które zapewne okazałyby się skuteczne, dodatkowo zapewniając dokładną informację o pozycji obiektu na obrazie. Wadą podejścia korelacyjnego jest konieczność dopasowywania każdego obiektu do analizowanego obrazu, co może trwać dłużej od propagacji w przód obrazu wejściowego przez strukturę sieci FONN (co wymaga liczby operacji arytmetycznych porównywalnej do algorytmu FFT). Przykładowo, obliczenie korelacji w dziedzinie częstotliwości, które może być bardziej efektywne dla obrazów o większych rozmiarach, i tak wymaga obliczenia odwrotnej transformaty Fouriera dla każdego wzorca (tj. dla każdej klasy) i znalezienia globalnego maksimum funkcji korelacji.

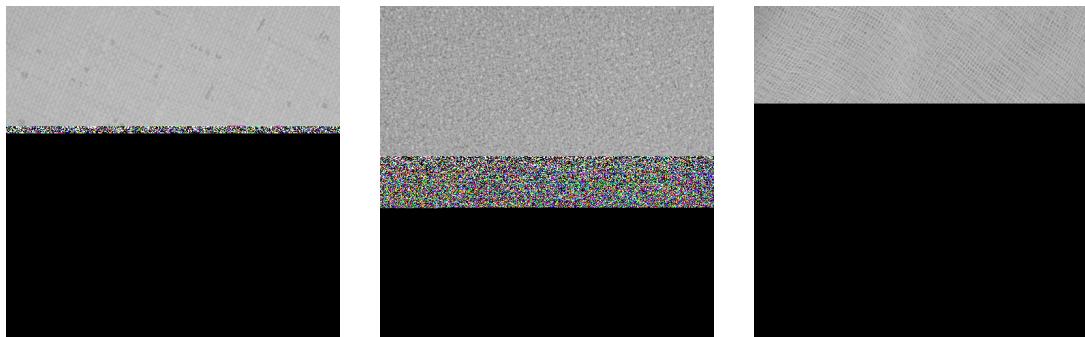
Z drugiej strony, FONN jest *siecią neuronową*, a zatem musi być trenowana, co może trwać dość długo. Niemniej jednak, to jest cena, jaką płacimy za zwartą reprezentację (wektor wag) w stosunku do np. metody KNN wymagającej zapamiętania wszystkich wzorców treningowych, szybkość samego rozpoznawania (z użyciem już nauczonej sieci) i możliwość adaptacji, której w metodach korelacyjnych może brakować. Warto również podkreślić, że zaprezentowany klasyfikator operuje bezpośrednio na obrazach treningowych, nie potrzebując dodatkowych informacji, takich jak np. zdjęcia prawidłowo posegmentowanych obiektów.

Bazy danych **RTS1** i **RTS2** mogłyby być skutecznie klasyfikowane metodami

zów, bez konieczności poddawania ich rejestracji, segmentacji, wykrywaniu krawędzi i innym typowym metodom przetwarzania wstępnego. Szczególną uwagę zwrócono na zademonstrowanie możliwości użycia informacji fazowej w procesie klasyfikacji obrazów, odróżniającej zaproponowaną metodę od większości typowych rozwiązań opartych na widmie amplitudowym. W istocie, pomimo oparcia konstrukcji klasyfikatora o model implikujący niezależność od przekształceń typu RTS, wykazano iż skuteczna klasyfikacja jest możliwa również w sytuacji celowego odejścia od tego modelu poprzez nadanie translacji znaczenia dyskryminacyjnego.

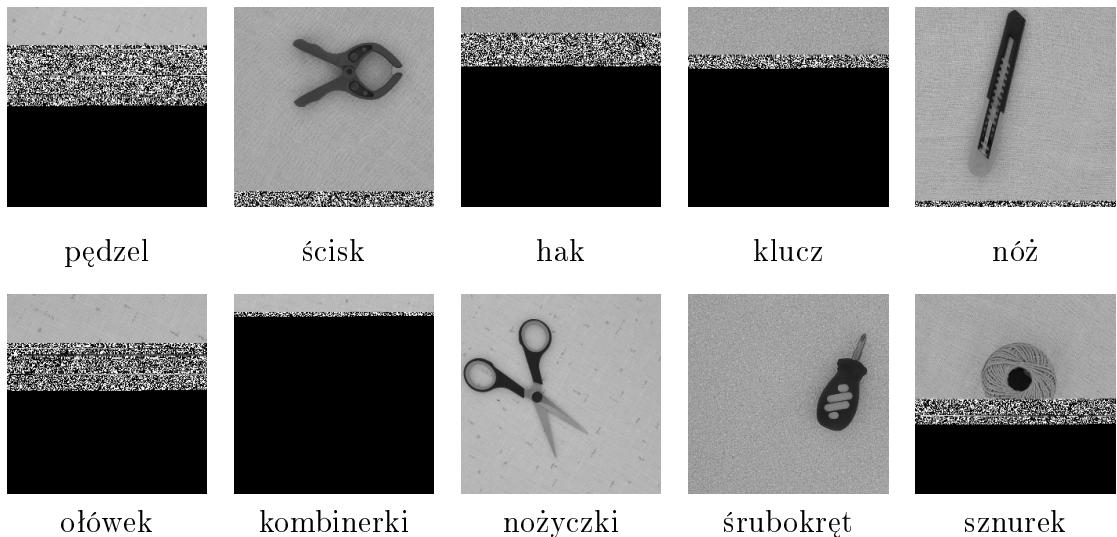
6.2.3 Rozpoznawanie obiektów na zdjęciach

W celu dodatkowej weryfikacji przydatności zaproponowanej sieci w zagadnienniach praktycznych, skonstruowana została baza **STaR** (Skalowanie, Translacja, Rotacja) zawierająca zdjęcia obiektów zróżnicowane pod względem tych samych przekształceń afinicznych, jak w przypadku zaprezentowanej w podrozdziale 6.2.2 bazy RTS3. Zastosowano nieco „łatwiejsze”, tzn. bardziej jednorodne tła w stosunku do bazy RTS3 (rys. 6.22), ale zwiększyły się liczby obiektów do dziesięciu (rys. 6.23). Większość obiektów stanowiły niewielkie narzędzia ręczne o zblisko-



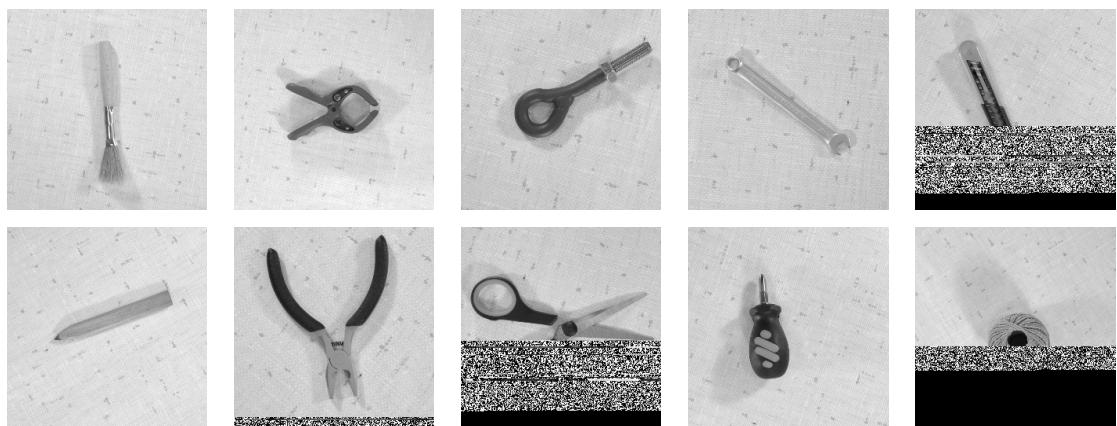
Rysunek 6.22: Różne rodzaje tła wykorzystane w bazie STaR

nych rozmiarach, w tym niektóre dosyć podobne do siebie (np. pędzel i ołówek kreślarski). Każdy obiekt był fotografowany dziesięciokrotnie na każdym z trzech tel z rysunku 6.22. Cała baza danych zawierała zatem $10 \times 10 \text{ obiektów} \times 3 \text{ tła} = 300 \text{ zdjęć}$, przy czym pozycja obiektu, jego orientacja (kąt obrotu na płaszczyźnie z zakresu $0^\circ - 360^\circ$) i przybliżenie (w zakresie 100% – 150%) były dobierane losowo dla każdego zdjęcia. Z tej liczby 50 zdjęć (po 5 zdjęć z każdej klasy) zostało wylosowanych w celu utworzenia zbioru testowego, pozostałe 250 stanowiły zbiór treningowy. Ponieważ wstępne testy wykazały, że efekt „przeuczenia”, tj. pogarszania wyniku klasyfikacji dla zbyt dużej ilości epok, występuje w bardzo niewielkim stopniu, zatem zrezygnowano z odrębnego zbioru walidacyjnego, za kryterium stopu przyjmując osiągnięcie ustalonej z góry liczby epok równej 100.



Rysunek 6.23: Przykładowe zdjęcia z bazy STaR (wszystkie obiekty)

Dodatkowo, w celu określenia odporności sieci na większe zróżnicowanie warunków akwizycji obrazów skonstruowano dwa inne zbiory testowe, liczące również po 50 obiektów. W pierwszym z nich (zbiór STaR-light, rys. 6.24) zastosowano zmienne warunki oświetleniowe (zaprezentowana wyżej podstawowa baza STaR zawiera zdjęcia wykonane w środowisku bezcieniowym), a w drugim (zbiór STaR-light30, rys. 6.25) dodatkowo zmieniono kąt między płaszczyzną obiektywu, a płaszczyzną tła z 0° na 30° . Ograniczono się tutaj tylko do pierwszego tła z rysunku 6.22, natomiast pozostałe parametry, tj. wartość translacji, rotacji i skalowania były również losowe. Warto zauważyć, że podstawowe rodzaje zniekształceń w stosunku do zdjęć ze zbioru treningowego obejmują cienie, odbicia światła od metalowych elementów i uwypuklenie niejednorodności tła (zwłaszcza w przypadku bazy STaR-light30).



Rysunek 6.24: Przykłady prawidłowo rozpoznanych zdjęć z bazy STaR-light



Rysunek 6.25: Przykłady prawidłowo rozpoznanych zdjęć z bazy STaR-light30

Zastosowano klasyfikator neuronowy zaprezentowany na rysunku 6.10, bez połączeń oznaczonych symbolem (*). Jako dane wejściowe dla klasyfikatora wykorzystano widmo amplitudowe transformaty Fouriera w reprezentacji logarytmiczno-biegunowej o rozmiarze 64×64 (FFT-LPT), zapewniającej niezależność od translacji oraz zamianę rotacji i skalowania na odpowiednie przesunięcia, zgodnie z rysunkiem 6.16.

Uzyskane rezultaty i wnioski

Przeprowadzono trzy serie testów, wykorzystujące w charakterze zbioru testowego odpowiednio: 50 zdjęć⁶ ze zbioru STaR (seria 1), 50 zdjęć ze zbioru STaR-light (seria 2) oraz 100 zdjęć z połączonych zbiorów STaR-light i STaR-light30 (seria 3). Każda seria obejmowała dziesięć powtórzeń procesu adaptacji z losowym punktu w przestrzeni wag, przy czym losowaniu podlegały tylko wagi warstwy wyjściowej, natomiast wagi części FONN były inicjalizowane zgodnie z algorytmem dwuwykazanego przekształcenia Fouriera. Uśrednione wyniki zaprezentowano na rysunku 6.26, przy czym dla porównania przedstawiono tu również wyniki uzyskane dla klasyfikatora KNN. Macierze pomyłek dla zbioru testowego STaR-light oraz dla połączonych zbiorów STaR-light i STaR-light30 zaprezentowano w tablicach 6.8 i 6.9, odpowiednio.

Porównując uzyskane wyniki z wykresem na rys. 6.21 zauważamy znaczące zwiększenie różnicy w skuteczności rozpoznawania obiektów pomiędzy siecią typu FONN, a metodą KNN. W szczególności, zaproponowany klasyfikator uzyskał stuprocentową poprawność klasyfikacji w testach z pierwszej serii, podczas gdy współczynnik rozpoznania dla metody KNN wyniósł poniżej 60%. Fakt ten może wynikać ze stosunkowo niewielkiej liczebności zbioru treningowego (25 obiektów

⁶Pozostałe 250 zdjęć stanowiło zbiór treningowy we wszystkich trzech seriach



Rysunek 6.26: Wyniki klasyfikacji

Tablica 6.8: Macierz pomyłek dla zbioru STaR-light

	pędzel	ścisk	hak	nóż	okówk	kombinerki	nożyczki	śrubokręt	klucz	sznurek
pędzel	28%	-	-	72%	-	-	-	-	-	-
ścisk	-	100%	-	-	-	-	-	-	-	-
hak	8%	-	82%	-	10%	-	-	-	-	-
nóż	-	-	-	100%	-	-	-	-	-	-
ołówek	36%	-	-	-	64%	-	-	-	-	-
kombinerki	-	-	-	-	-	100%	-	-	-	-
nożyczki	-	-	-	-	-	-	100%	-	-	-
śrubokręt	-	-	-	-	-	-	20%	80%	-	-
klucz	-	-	18%	40%	-	-	-	-	42%	-
sznurek	-	-	-	-	-	-	-	-	-	100%

na klasę), zwłaszcza wobec dużych rozmiarów danych wejściowych (64×64), podkreślając tym samym dobre właściwości generalizacyjne sieci FONN. Uwzględniając różnice pomiędzy zbiorami obrazów RTS3 i STaR, znaczco lepszy wynik klasyfikacji na tym drugim można prawdopodobnie przypisać dużo mniej złożonym rodzajom tła użytym do jego konstrukcji. Na szczególne podkreślenie zasługuje przy tym fakt, że wyższy współczynnik rozpoznania uzyskany został pomimo zwiększenia ilości klas do dziesięciu.

Zwiększenie zakresu znieksztalceń wynikających z oświetlenia i zmiany perspektywy (w przypadku zbioru STaR-light30) powoduje dość znaczący spadek współczynnika rozpoznania. Analiza macierzy pomyłek pokazuje, że największe problemy występują w przypadku trzech obiektów o wydłużonym kształcie i niewielkim kontraście w stosunku do podłoża (pędzel, klucz, ołówek), faktycznie wy-

Tablica 6.9: Macierz pomyłek dla zbioru STaR-light + STaR-light30

	pędzel	ścisk	hak	nóż	ołówek	kombinerki	nożyczki	śrubokręt	klucz	sznurek
pędzel	15%	-	-	66%	9%	-	-	-	10%	-
ścisk		70%	-	-	-	10%	20%	-	-	-
hak	5%	-	60%	9%	-	18%	-	8%	-	-
nóż	-	-	-	100%	-	-	-	-	-	-
ołówek	18%	-	10%	30%	32%	1%	-	-	9%	-
kombinerki		9%	-	-	-	91%	-	-	-	-
nożyczki		-	7%	-	-	2%	88%	-	3%	-
śrubokręt		-	-	-	-	-	19%	80%	1%	-
klucz	-	-	26%	19%	3%	-	-	2%	50%	-
sznurek	2%	-	9%	-	1%	-	18%	21%	-	49%

kazujących — subiektywnie — spore podobieństwo. W szczególności, pojawienie się wyraźnych cieni może np. powodować pozorne rozszerzenie jednego końca obiektu, charakterystyczne dla kształtu pędzla, a zmiana perspektywy może modyfikować stosunek szerokości do długości obiektów, będący również potencjalnym czynnikiem dyskryminacyjnym.

Analizując powyższe wyniki należy jednak mieć na uwadze, że wszystkie te dodatkowe rodzaje zniekształceń nie są uwzględnione w konstrukcji zbioru treningowego. Włączając zdjęcia typowe dla zbiorów STaR-light i STaR-light30 do zbioru treningowego można również w tym przypadku spodziewać się znacznej poprawy rezultatów rozpoznawania. Na koniec, warto podkreślić, że chociaż baza STaR wydaje się stanowić nieco prostszy problem klasyfikacyjny, niż przedstawiona wcześniej baza RTS3, z uwagi na potencjalnie większe szanse powodzenia różnych metod segmentacji oraz na możliwość skonstruowania zbioru cech różnicujących konkretne rodzaje obiektów między sobą, jednak prostota i uniwersalność zaproponowanego podejścia stanowi jego zasadniczą zaletę. Klasyfikator oparty na szybkiej ortogonalnej sieci neuronowej może być z powodzeniem zastosowany w praktyce do rozpoznawania obiektów bez konieczności zapewnienia ich właściwej lokalizacji, orientacji i wielkości oraz bez dodatkowych etapów przetwarzania takich jak segmentacja.

6.3 Możliwości przyspieszenia adaptacji

Pewne własności szybkich ortogonalnych sieci neuronowych wskazują na możliwość dodatkowego skrócenia czasu potrzebnego na adaptację/przetwarzanie w przód.

W szczególności, architektura połączeń neuronalnych sieci umożliwia zrównoleglenie obliczeń, co może przynieść wymierne korzyści w przypadku implementacji na maszynie wyposażonej w procesor wielordzeniowy, bądź wieloprocesorowej. Oprócz tego, możliwe jest zastosowanie dynamicznej rozbudowy struktury sieci, w sposób zbliżony do metody zaprezentowanej w rozdziale 5.2, również w kontekście klasyfikacji. Obie te możliwości zostały poddane analizie, implementacji i weryfikacji eksperymentalnej (w kolejnych podrozdziałach).

6.3.1 Dynamiczna rozbudowa struktury sieci

Przykładowe podejście oparte na dodawaniu i douczaniu kolejnych bloków sieci zostało przedstawione w rozdziale 5.2 z wykorzystaniem współczynników transformaty cosinusowej w charakterze wektorów docelowych. Wykorzystanie tej samej metodyki w odniesieniu do problemów klasyfikacyjnych jest o tyle kłopotliwe, że wymagana jest tutaj przynajmniej jedna dodatkowa warstwa wyjściowa o ilości neuronów uzależnionej od ilości klas. Z drugiej strony, istnieją pewne przesłanki sugerujące potencjalną przydatność takiego podejścia: w szczególności, struktura sieci typu FONN jest ściśle związana z przekształceniami ortogonalnymi takimi jak FCT, czy FFT, mającymi własność koncentracji zasadniczej części mocy sygnału na stosunkowo niewielkiej liczbie współczynników wyjściowych. Istnieje zatem możliwość, że poprawny wynik klasyfikacji będzie można uzyskać nie wykorzystując wszystkich wartości wyjściowych sieci typu FONN oraz że będzie w ten sposób możliwe zmniejszenie liczby epok w procesie adaptacji sieci.

W celu weryfikacji tej hipotezy zastosowano podejście oparte na dynamicznej rozbudowie ostatniej (wyjściowej) warstwy klasyfikatora. Założono przy tym, że liczba tworzących ją neuronów jest stała (z uwagi na stałą ilość klas w danym problemie klasyfikacyjnym), natomiast zmianom musi podlegać liczba ich wejść, odpowiadająca liczbie wyjść poprzedzającego i połączonego z nią bloku sieci FONN. Struktura sieci FONN została oparta na dwuwymiarowym algorytmie transformaty Fouriera z warstwą obliczającą amplitudę (rys. 6.10), przy czym zastosowano tu dodatkową warstwę pomiędzy częścią FONN, a warstwą wyjściową, dokonującą permutacji r_N :

$$\begin{aligned} r_N(2n) &= n ; \\ r_N(2n+1) &= n + N=2 ; \end{aligned} \tag{6.7}$$

gdzie $n = 0; 1; \dots; N=2 - 1$.

Rozważając graf sieci z rysunku 6.10 łatwo zauważyc, że permutacja ta odpowiada naprzemienemu uszeregowaniu wartości amplitudy i fazy (połączenia

oznaczone symbolem (*) kolejnych współczynników widma.

Początkowy rozmiar wyjścia części FONN został ustalony na 8 wyjść. Ścisłe rzeczą biorąc, rozmiar sieci był dla uproszczenia implementacji ustalony z góry i równy rozmiarowi docelowemu, jednak wszystkie wyjścia części FONN, poza pierwszą ósemką, otrzymywały w trakcie propagacji wstecznej zerowy sygnał błędu, dzięki czemu nie miały wpływu na modyfikację wag. Przyjmując problem klasyfikacyjny określony dla K klas, początkowa postać warstwy wyjściowej jest wyznaczona macierzą o wymiarach $K \times (1 + 8)$ (por. wzór (3.4)):

$$A_{out}^{(0)} = \begin{bmatrix} w_{10} & w_{11} & w_{12} & \dots & w_{18} \\ w_{20} & w_{21} & w_{22} & \dots & w_{28} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{K0} & w_{K1} & w_{K2} & \dots & w_{K8} \end{bmatrix}; \quad (6.8)$$

gdzie pierwsza kolumna odpowiada wartości biasu. Po zakończeniu procesu adaptacji, macierz ta jest rozszerzana do postaci macierzy o wymiarach $K \times (1 + 16)$:

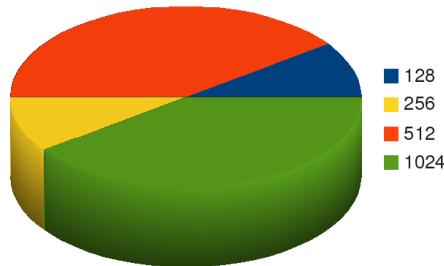
$$A_{out}^{(1)} = \begin{bmatrix} w_{10} & w_{11} & w_{12} & \dots & w_{18} & 0 & \dots & 0 \\ w_{20} & w_{21} & w_{22} & \dots & w_{28} & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ w_{K0} & w_{K1} & w_{K2} & \dots & w_{K8} & 0 & \dots & 0 \end{bmatrix}; \quad (6.9)$$

i rozpoczyna się drugi cykl adaptacji z udziałem 16 wyjść części FONN. Z uwagi na to, że ostatnie 8 wyjść części FONN (z pierwszych szesnastu) jest teraz połączonych z wagami warstwy wyjściowej o wartości zero, zatem wartości wyjściowe klasyfikatora w pierwszej epoce drugiego cyklu są takie same jak w ostatniej epoce cyklu pierwszego. Jednak w trakcie trwania drugiego cyklu adaptacji wagi te są modyfikowane i pozwalają uwzględniać w obliczeniach wszystkie 16 wyjść części FONN. Po zakończeniu drugiego cyklu ilość wejść ostatniej warstwy jest rozszerzana do ilości $1 + 32$ (liczba neuronów pozostaje wciąż równa K), z których ostatnie 16 otrzymuje wagi inicjalizowane wartością zero, po czym proces się powtarza aż do uzyskania warstwy wyjściowej o ilości wejść równej całkowitej ilości wyjść części FONN.

Przeprowadzone testy wskazują na możliwość wykorzystania powyższej procedury do dokładniejszego określenia warunków stopu algorytmu i skrócenia procesu adaptacji. Dla przykładu, w wyniku zastosowania dynamicznej rozbudowy sieci w problemie klasyfikacji bazy RTS3 (rys. 6.17) z wykorzystaniem 50 obiektów na klasę w zbiorze treningowym uzyskano redukcję średniej liczby epok⁷ ze 115

⁷wynik uśredniony z 10 powtórzeń

do 81. Po wstępnych eksperymentach przyjęto maksymalną liczbę epok pojedynczego cyklu równą 20, przy czym w każdym cyklu warunkiem stopu było uzyskanie minimum błędu walidacji. Zarówno w podejściu oryginalnym, jak i w wariantie z dynamiczną rozbudową sieci uzyskano zbliżone wyniki rozpoznania na zbiorze testowym (rzędu 70%) i na zbiorze treningowym (100%). W kolejnych powtórzeniach testu globalne maksimum poprawnej klasyfikacji uzyskiwano w różnej liczbie cykli, co przedstawiono na rys. 6.27. Warto zauważyć, że w ponad po-



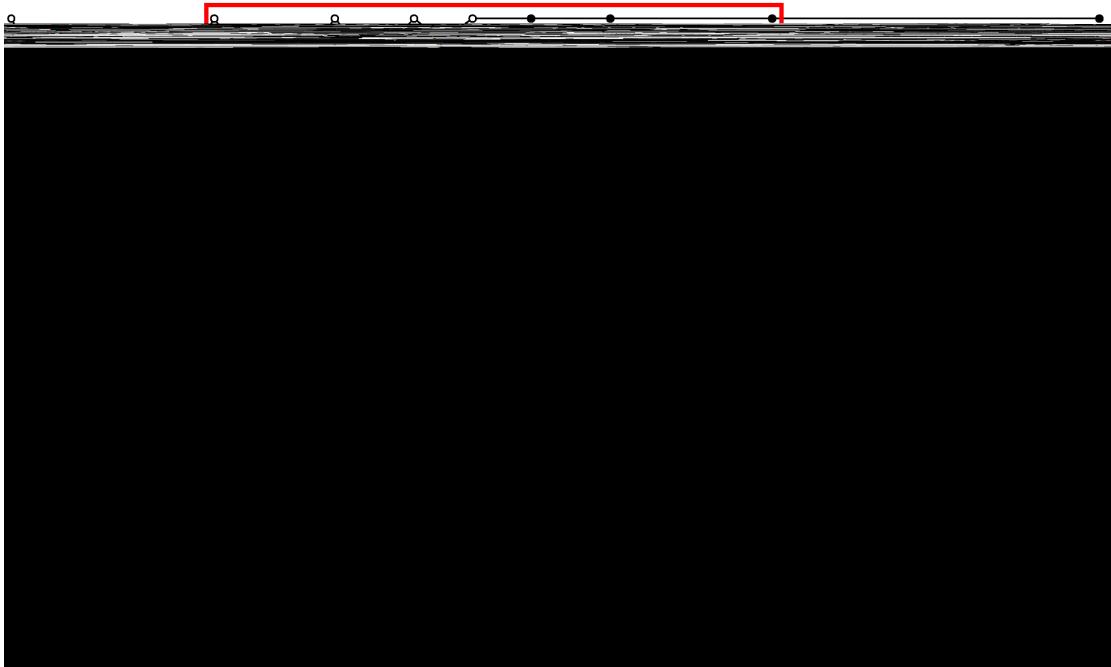
Rysunek 6.27: Rozmiar warstwy wyjściowej, dla którego uzyskano najlepsze rozpoznanie

wie przypadków nie było konieczne kontynuowanie nauki z udziałem maksymalnej liczby wyjść sieci typu FONN, a w jednym przypadku wystarczające okazało się uwzględnienie zaledwie 1=8 wszystkich wyjść. Uzyskane rezultaty wskazują zatem na możliwość istotnej redukcji liczby epok i — potencjalnie — również czasu nauki w procesie adaptacji sieci typu FONN w oparciu o jej dynamiczną rozbudowę i sukcesywne douczanie.

6.3.2 Równoległa realizacja obliczeń

Rozważając strukturę grafu szybkiej ortogonalnej sieci neuronowej łatwo zauważamy, że obliczenia wykonywane przez wszystkie warstwy z wyjątkiem pierwszej i ostatniej (oraz ew. warstw permutujących wyjścia lub wyjścia) mogą być rozdzielone na dwa niezależne procesy (wątki) obliczeniowe. Podobnie, jeżeli z tego zbioru usuniemy jego pierwszą i ostatnią warstwę, wówczas przetwarzanie może być rozdzielone na cztery procesy i tak dalej. W gruncie rzeczy, każda pojedyncza operacja bazowa w ramach jednej warstwy może być rozpatrywana jako odrębne zadanie obliczeniowe, co prowadzi do przyjęcia modelu równoległości drobnoziarnistej (ang. *fine-grained*). My jednak ograniczymy się tu do sytuacji przedstawionej na rysunku 6.28, w której obliczenia rozdzielane są na dwa niezależne wątki dokonujące obliczeń dla wszystkich warstw od drugiej do przedostatniej.

Warto zwrócić uwagę na to, iż sama struktura jednolitego, dwuetapowego algorytmu umożliwia łatwą implementację wielowątkową. W przypadku wyko-



Rysunek 6.28: Podział obliczeń dla wewnętrznych warstw sieci FONN na dwa wątki

rzystywanego symulatora, przetwarzającego dane warstwa po warstwie, funkcja wykonująca obliczenia dla pojedynczej warstwy otrzymuje dodatkowy parametr wskazujący, czy obliczenia mają dotyczyć górnej, dolnej, czy obu części operacji podstawowych tej warstwy. Funkcja ta jest wywoływana raz dla obu części pierwszej warstwy, a następnie tworzone są dwa wątki, które wywołują ją niezależnie dla obliczeń na, odpowiednio, górnej i dolnej części kolejnych warstw. Ostatnia warstwa jest obliczana ponownie w pojedynczym wątku. Ponieważ dla wszystkich środkowych warstw obliczenia prowadzone są na odrębnych obszarach danych, zatem praktycznie jedynymi punktami synchronizacji są momenty utworzenia i zakończenia wątków obliczeniowych. Brak konieczności wprowadzania dodatkowych mechanizmów synchronizacyjnych, blokowania, czy kopiowania danych, umożliwił przyjęcie wielowątkowego modelu przetwarzania równoległego ze współdzieloną pamięcią. Alternatywę stanowiła tu możliwość tworzenia nowych procesów, w miejsce wątków obliczeniowych, jednak wariant taki byłby mniej korzystny z uwagi chociażby na konieczność implementacji dodatkowych mechanizmów komunikacji międzyprocesowej (ang. *inter-process communication*, IPC), zwiększającej potencjalnie narzut czasowy związany z obsługą przetwarzania równoległego.

Powyższy sposób implementacji można łatwo rozszerzyć wprowadzając rekurencyjne rozdzielenie obliczeń przez kolejne wątki (pierwsza warstwa obliczana jest w jednym wątku, druga w dwóch, trzecia w czterech itd.). W praktyce jednak nie

ma większego sensu tworzenie większej liczby wątków niż dostępna liczba jednostek wykonawczych. W opisywanym przypadku ograniczenie się do dwóch wątków zostało podyktowane wykorzystaniem w procesie testowania maszyny z dwurdzeniowym procesorem Intel Core 2 Duo (T7500).

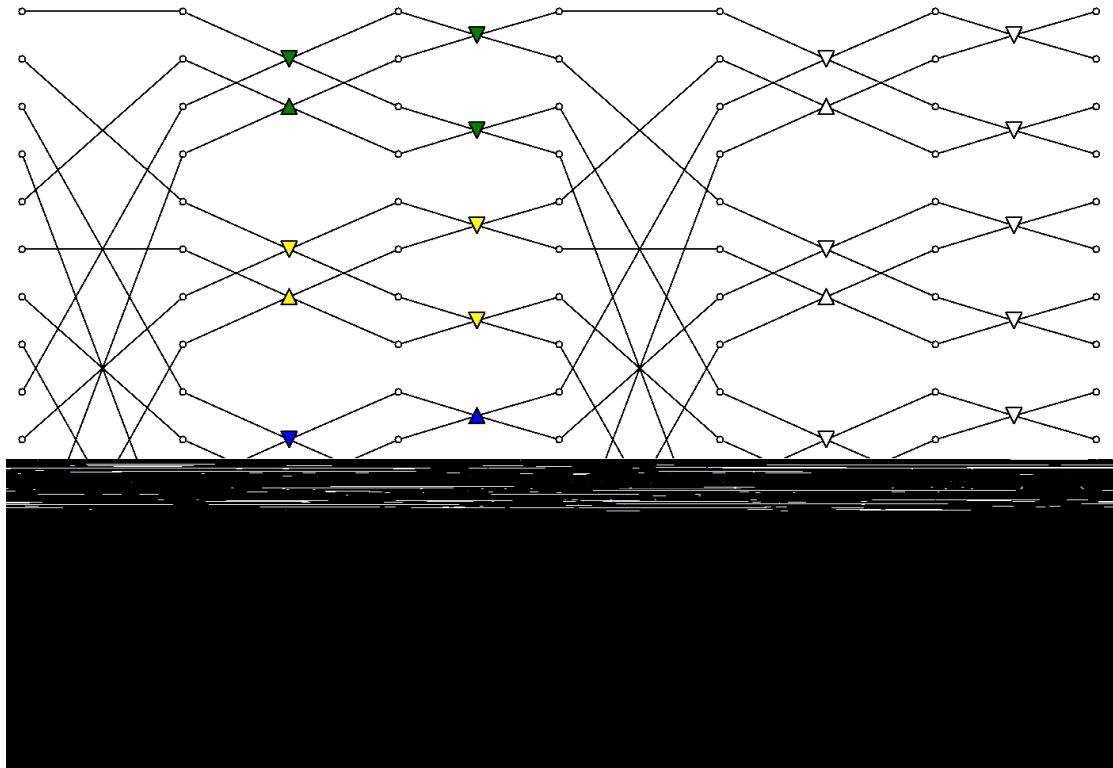
Należy także zauważać, że możliwe jest zasadniczo równoległe wykonanie obliczeń dla wszystkich warstw, włącznie z warstwami początkowymi i końcowymi. W tym celu konieczne jest wprowadzenie odpowiednich modyfikacji struktury sieci [152], stosownie do ilości dostępnych jednostek obliczeniowych. Na rysunku 6.29 przedstawiono pierwszy etap grafu przepływu, z zaznaczeniem czterech grup operacji bazowych, z których każda przydzielana jest do jednego z czterech wątków. Na tej podstawie kolejność operacji bazowych oraz kolejność ich wejść/wyjść



Rysunek 6.29: Przydział operacji bazowych pierwszych warstw do poszczególnych wątków

może zostać zmodyfikowana tak, aby uzyskać czytelny podział struktury dwóch pierwszych warstw na cztery wątki, co przedstawiono na rysunku 6.30. Modyfikacja ta wymaga wprowadzenia dodatkowych warstw permutujących, umieszczonych przed i po rozważanych warstwach.

Zauważmy, że podobną zasadę postępowania można zastosować dla innej liczby wątków lub innego rozmiaru sieci, przy czym w ogólnym przypadku odpowiednim modyfikacjom podlegać musiałaby inna liczba warstw. Również w analogiczny

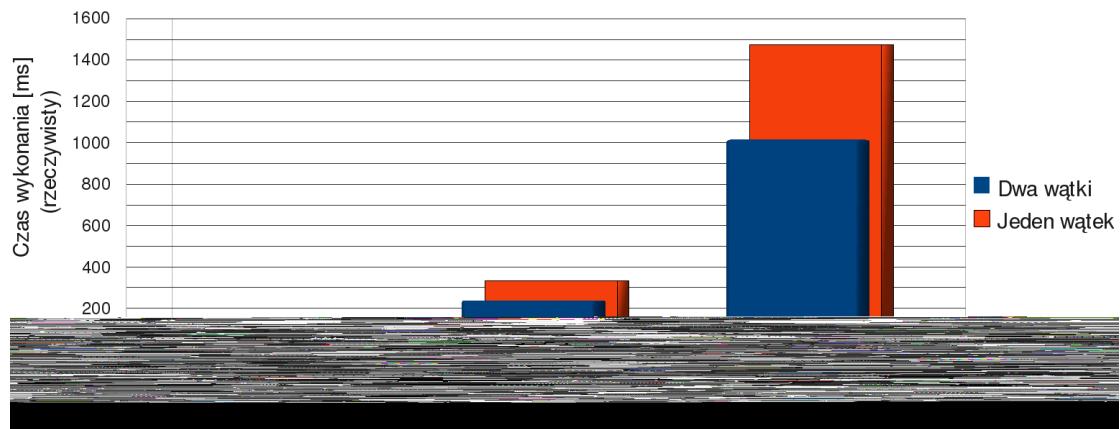


Rysunek 6.30: Zmodyfikowana struktura pierwszego etapu

sposób możemy zmodyfikować wyjściowe warstwy etapu drugiego, uwzględniając jego strukturę wyznaczoną przez przekształcenie ortogonalne, na którym oparto konstrukcję sieci.

Możliwości równoległej realizacji obliczeń w ramach symulatora szybkich ortogonalnych sieci neuronowych dotyczą zasadniczo samej propagacji sygnału w grafie przepływu definiującym jej strukturę. Inne operacje, takie jak optymalizacja gradientowa, czy obliczanie wartości funkcji błędu wykonywane są w tradycyjny, tj. jednowątkowy sposób. W związku z tym do testowania zastosowano procedurę opartą na obliczaniu czasu wykonania propagacji sygnału w przód, w realizacji dwuwątkowej zgodnie ze schematem z rysunku 6.28. Wykorzystano sieć opartą na algorytmie przekształcenia Fouriera z różnymi długościami wektora wejściowego, przy czym każdy test był powtórzony dziesięciokrotnie. W celu dodatkowego zmniejszenia narzutów czasowych związanych z ładowaniem programu, wczytywaniem danych, itd. pojedyncze powtórzenie testu obejmowało 100 cykli propagacji w przód (z poziomu działającej aplikacji). Uśrednione wyniki zaprezentowano na wykresie 6.31 oraz w tablicy 6.10.

Analizując otrzymane wyniki warto zauważyć, że maksymalną teoretyczną granicą wzrostu wydajności w przypadku rozbiecia procesu obliczeniowego na dwa wątki jest wzrost o 100%. Naturalnie granica ta jest w rozważanym przypadku



Rysunek 6.31: Porównanie czasu obliczeń (przetwarzanie w przód: jeden cykl)

Tablica 6.10: Porównanie czasu obliczeń (przetwarzanie w przód: jeden cykl)

N	P	Wątki	Czas [ms] (real)	Czas [ms] (user+sys)
256	256	1	74	72
256	256	2	55	76
512	512	1	332	323
512	512	2	235	348
1024	1024	1	1473	1435
1024	1024	2	1012	1558

niemożliwa do osiągnięcia, chociażby z uwagi na to, że część obliczeń nie jest wykonywana współbieżnie (pierwsza/ostatnia warstwa). Zaprezentowane rezultaty wskazują na faktyczną możliwość skrócenia czasu obliczeń odpowiadającego wzrostowi wydajności o blisko 50%, przy czym zysk ten jest tym większy im większa jest ilość i rozmiar wektorów wejściowych i — co za tym idzie — również rozmiar sieci.

Ciekawe jest porównanie przedstawionego w tablicy 6.10 rzeczywistego czasu wykonania z czasem zużytym przez CPU (ostatnia kolumna tablicy). Zaprezentowane wyniki zostały uzyskane za pomocą standardowego narzędzia UNIX-owego: polecenia `time` informującego o rzeczywistym czasie działania programu (*real*), czasie zużytym w trybie użytkownika (*user mode*) i czasie zużytym w trybie jądra⁸ (*kernel mode* a.k.a. *system mode*). Czas rzeczywisty określa czas jaki faktycznie upłynął i może być — w systemie wielozadaniowym — powiększony o czas zużyty przez inne aplikacje działające w tym samym momencie w tle. Suma pozostałych dwóch wartości mówi o czasie zużytym przez procesor na wykonanie

⁸Ten ostatni miał znikomy wpływ w analizowanym przypadku

wyłącznie analizowanej aplikacji. W przypadku aplikacji wielowątkowej wykonywanej na maszynie wieloprocesorowej (lub z procesorem wielordzeniowym) czas zużyty przez wszystkie procesory/rdzenie jest sumowany, więc w szczególności może on być większy niż rzeczywisty czas jaki upłynął podczas pracy programu. Zjawisko to możemy zaobserwować w drugim, czwartym i szóstym wierszu tablicy 6.10. Warto zwrócić tutaj uwagę na dwa fakty. Po pierwsze, dla każdego z trzech rozmiarów problemu czas zużyty przez CPU (user + sys) jest nieznacznie większy dla aplikacji wielowątkowej w stosunku do jednowątkowej, co odzwierciedla pewien narzut wynikający z konieczności stworzenia wątków i zarządzania wykonaniem wspólnieżnym. Z drugiej strony, czas rzeczywisty w przypadku implementacji współbieżnej jest znacznie niższy, co przekłada się bezpośrednio na faktyczny wzrost wydajności w stosunku do oryginalnej realizacji jednowątkowej.

Na koniec trzeba wspomnieć o ważnym czynniku wpływającym na wydajność działania programu, jakim jest sposób zarządzania pamięcią operacyjną i dostępem do danych. Element ten ma istotne znaczenie w przypadku rozważanego symulatora, z uwagi na potencjalnie duże rozmiary danych wejściowych i wielowarstwową strukturę sieci, w której każda warstwa musi przechowywać dane potrzebne do adaptacji metodą propagacji wstecznej. Przeprowadzone testy wskazały na istotny wpływ operacji dostępu do pamięci na sumaryczny czas wykonania programu, co stanowi dodatkową możliwość poprawy efektywności w przypadku optymalizacji tych operacji ukierunkowanej na równoległą realizację obliczeń. Warto tu zauważyć, że włączenie opcji optymalizacyjnych podczas komilacji programu (w kompilatorze GNU GCC 4.2.4) umożliwiło uzyskanie dodatkowego skrócenia czasu obliczeń, nieraz w znacznym stopniu, w zależności od wyboru konkretnych opcji i rozmiaru problemu. Znamienny przy tym był fakt bardzo istotnego zmniejszenia przewagi implementacji współbieżnej nad oryginalnym podejściem jednowątkowym (wyniki zaprezentowane powyżej uzyskane zostały bez użycia dodatkowych optymalizacji). Obserwacje te wskazują na możliwość dodatkowego przyspieszenia procesu symulacji sieci, wymagającą pogłębionej analizy problemów związanych z dostępem pamięci operacyjnej i pamięci typu *cache* przez aplikację wielowątkową oraz technik optymalizacyjnych wykorzystywanych przez kompilator.

6.4 Podsumowanie

W niniejszym rozdziale zaprezentowano zastosowania sieci typu FONN do klasyfikacji i rozpoznawania wzorców. Zgodnie z założeniami poczynionymi w podrozdziałach 1.1 i 3.3 proces rozpoznawania nie obejmował fazy ekstrakcji cech, predefiniowanych specjalnie dla konkretnego zbioru danych testowych, i był za-

sadniczo realizowany w całości przez hybrydową sieć neuronową złożoną z sieci typu FONN oraz dodatkowej warstwy wyjściowej. Szczególny nacisk położono na różnorodność problemów klasyfikacyjnych obejmujących zarówno dane o charakterze przebiegów czasowych (podrozdział 6.1), jak i obrazy (podrozdział 6.2). Do najciekawszych rezultatów można zaliczyć m.in. wyniki uzyskane podczas klasyfikacji danych audio, z uwagi na skuteczność zaproponowanej metody oraz na możliwości zastosowań praktycznych. Warte uwagi są również wyniki testów związanych ze specyficzną cechą widma amplitudowego transformaty Fouriera, jaką jest niezależność od przesunięcia danych wejściowych w dziedzinie czasu. Wykorzystanie szybkiej sieci ortogonalnej opartej na algorytmie obliczania widma amplitudowego i rozszerzenie zakresu przekształceń afinicznych dzięki transformacji logarytmiczno-biegunowej wyznacza interesujący kierunek potencjalnego rozwoju systemów automatycznej klasyfikacji obrazów, na co wskazują wyniki zaprezentowane w podrozdziałach 6.2.2 i 6.2.3. Na podkreślenie zasługuje fakt skonstruowania bazy obrazów przeznaczonej specjalnie do testowania niezależności od przekształceń typu rotacja-translacja-skalowanie (podrozdział 6.2.3), jak również przedstawione w podrozdziale 6.3 metody przyspieszenia adaptacji sieci możliwe do wykorzystania w rozwiązańach praktycznych.

Reasumując, uzyskane rezultaty — szczególnie w zestawieniu z uniwersalnością zaproponowanego modelu klasyfikacji i z radykalnym ograniczeniem konieczności wstępного przetwarzania danych — stanowią dobrą demonstrację skuteczności sieci typu FONN w problemach klasyfikacji wzorców. Tym samym drugą tezę pracy można uznać za słuszną, a drugi cel pracy – za zrealizowany.

7 Wnioski i podsumowanie pracy

Podsumowując całość wykonanych prac i przedstawionych wyników eksperymentalnych należy stwierdzić, że oba zasadnicze cele niniejszej rozprawy zostały zrealizowane. Do najistotniejszych rezultatów i nowatorskich rozwiązań zaproponowanych w ramach realizacji pierwszego celu należy zaliczyć:

- wprowadzenie nowego rodzaju neuronu (BOON) wykorzystującego ortogonalność operacji bazowych szybkiego algorytmu stanowiącego podstawę konstrukcji sieci,
- przedstawienie metody adaptacji neuronów typu BOON umożliwiającej elastyczne definiowanie struktury połączeń sieci, łącznie z tworzeniem połączeń hybrydowych z warstwami innych typów,
- wprowadzenie elementów sieci neuronowej umożliwiających obliczanie widma amplitudowego transformaty Fouriera,
- dwu- i czterokrotną redukcję liczby wag sieci typu FONN w stosunku do znanych szybkich sieci neuronowych przy zachowaniu możliwości realizacji danej transformaty ortogonalnej,
- dwu- i czterokrotne skrócenie średniego czasu adaptacji i redukcję liczby epok, a także zmniejszenie odpowiednich wartości odchylenia standardowego w stosunku do znanych szybkich sieci neuronowych.

Drugi z postawionych celów został zrealizowany z wykorzystaniem różnych rodzajów architektury szybkiej ortogonalnej sieci neuronowej i różnorodnych zbiorów danych do klasyfikacji, obejmujących zarówno sygnały jedno- jak i dwuwymiarowe. W ten sposób podkreślona została uniwersalność zaproponowanego rozwiązania i jego przydatność w zastosowaniu do szerokiego spektrum problemów klasyfikacyjnych.

Należy stwierdzić, że podstawową korzystną właściwością szybkich ortogonalnych sieci neuronowych, widoczną w przypadku zastosowania ich do klasyfikacji i rozpoznawania wzorców, jest zdolność do uzyskiwania relatywnie niskich błędów

generalizacji. Z wielu przeprowadzonych testów wynika, że problem „przeuczenia” sieci występuje tu w niewielkim stopniu w porównaniu do typowych sieci jedno-kierunkowych typu MLP.

Analizując wyniki rozpoznawania wzorców w poszczególnych przypadkach, warto podkreślić pewne specyficzne cechy strukturalne szybkich ortogonalnych sieci neuronowych, a w szczególności fakt, że sieci te w ogólnym przypadku posiadają taką samą liczbę wyjść jak i wejść. Ponieważ jednym z istotnych zadań realizowanych w procesie ekstrakcji cech jest redukcja wymiaru danych, określenie szybkiej ortogonalnej sieci neuronowej mianem „ekstraktora cech” może być dyskusyjne. Wydaje się ono uzasadnione w przypadku opisanym w podrozdziale 6.1.1, gdzie tylko niewielka część wyjść sieci typu FONN jest uwzględniana w procesie klasyfikacji. Warto jednak zauważyć, że w praktyce, w podobnie prostych przypadkach, tj. wobec możliwości dokonania efektywnej klasyfikacji w oparciu o niewielką liczbę współczynników, użycie szybkiej ortogonalnej sieci neuronowej może nie być uzasadnione. Zastosowanie w takiej sytuacji zwykłej, pojedynczej, „gęstej” warstwy liniowej o niewielkiej liczbie neuronów może zapewnić mniejszą liczbę wag do adaptacji, przy jednoczesnym rozszerzeniu zbioru możliwych do realizacji przekształceń liniowych.

Najbardziej interesujące wydają się zatem takie problemy klasyfikacyjne, w których trudno jest określić niewielki zbiór współczynników wyjściowych wykorzystanej transformaty, wystarczający do skutecznej klasyfikacji. Rozpoznawanie danych audio (podrozdział 6.1.3) wydaje się tu być dobrym przykładem, w którym użycie szybkiej ortogonalnej sieci neuronowej zapewnia wysoki współczynnik rozpoznania i niski błąd generalizacji w stosunku do podejścia bezpośredniego opartego na sieci typu MLP. W tym przypadku, z uwagi na wykorzystanie wszystkich wyjść sieci typu FONN, można stwierdzić, że umożliwia ona nie tyle zastąpienie ekstraktora cech, co jego eliminację. Tym sposobem zaproponowany model analizy danych i rozpoznawania wzorców istotnie zakłada adaptację sieci neuronowej w oparciu o surowe dane wejściowe w ich oryginalnej (z dokładnością do normalizacji) postaci.

Zastosowanie rekurencji w procesie adaptacji (podrozdział 5.2) oraz wykorzystanie własności widma amplitudowego transformaty Fouriera (podrozdziały 6.1.2, 6.2.2) stanowią interesujące przykłady wykorzystania pewnych szczególnych cech szybkich sieci ortogonalnych, które są warte dalszej analizy i badań. Warto tu podkreślić, iż to właśnie specyfika i odrębność sieci typu FONN w stosunku do innych znanych modeli sieci neuronowych stanowią w dużej mierze o oryginalności i nowatorstwie niniejszej pracy. Potencjalne kierunki dalszych badań obejmują konstrukcję sieci złożonych z neuronów ortogonalnych typu BOON o innym sche-

macie połączeń, łącznie z możliwością dynamicznego określania struktury połączeń w oparciu o algorytmy genetyczne i inne metody, oraz konstrukcję sieci hybrydowych z wykorzystaniem sieci typu FONN w połączeniu z innymi rodzajami sieci neuronowych.

Bibliografia

- [1] A. Abadpour, S. Kasaei. PCA eigenimages and their application to compression and watermarking. *Image Vision Comput.*, 26(7):878–890, 2008.
- [2] E. Acocella, A. Alcaim. Mathematical formulation of shape-adaptive 2-D transforms. *Signal Processing Letters, IEEE*, 8(10):273–275, Oct 2001.
- [3] R. Adamczak. *Zastosowanie sieci neuronowych do klasyfikacji danych doświadczalnych*. Praca doktorska, Katedra Metod Komputerowych, Uniwersytet Mikołaja Kopernika, 2001.
- [4] N. Ahmed, T. Natarajan, K. R. Rao. Discrete Cosine Transfom. *IEEE Trans. Comput.*, 23(1):90–93, 1974.
- [5] E. Alexandre, L. Cuadra, L. Álvarez, M. Rosa-Zurera, F. López-Ferreras. Two-layer automatic sound classification system for conversation enhancement in hearing aids. *Integr. Comput.-Aided Eng.*, 15(1):85–94, 2008.
- [6] F. Aminian, M. Aminian. Fault Diagnosis of Nonlinear Analog Circuits Using Neural Networks with Wavelet and Fourier Transforms as Preprocessors. *J. Electron. Test.*, 17(6):471–481, 2001.
- [7] A. D. Arbatli, H. L. Akin. Rule extraction from trained neural networks using genetic algorithms. *Proceedings of the second world congress on Non-linear analysts: part 3*, strony 1639–1648, Elmsford, NY, USA, 1997. Pergamon Press, Inc.
- [8] K. Arbter, W. E. Snyder, H. Burhardt, G. Hirzinger. Application of Affine-Invariant Fourier Descriptors to Recognition of 3-D Objects. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(7):640–647, 1990.
- [9] B. Ayrulu, B. Barshan. Neural networks for improved target differentiation and localization with sonar. *Neural Netw.*, 14(3):355–373, 2001.

Bibliografia

- [10] B. Barshan, B. Ayrulu. Fractional Fourier transform pre-processing for neural networks and its application to object recognition. *Neural Networks*, 15(1):131–40, 2002.
- [11] I. Bartolini, M. Patella. WARP: Accurate Retrieval of Shapes Using Phase of Fourier Descriptors and Time Warping Distance. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(1):142–147, 2005.
- [12] P. N. Belhumeur, J. P. Hespanha, D. J. Kriegman. Fisherfaces: Recognition Using Class Specific Linear Projection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(7):711–720, 1997.
- [13] J. Ben-Arie, Z. Wang. Pictorial Recognition of Objects Employing Affine Invariance in the Frequency Domain. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(6):604–618, 1998.
- [14] S. Ben-Yacoub, B. Fasel, J. Lüttin. FFT. *Proceedings of Second International Conference on Audio and Video-based Biometric Person Authentication AVBPA'99*, strony 31–36, 1999.
- [15] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1996.
- [16] R. N. Bracewell. *The Hartley Transform (Oxford Engineering Science Series)*. Oxford University Press, USA, 1986.
- [17] K. Brandenburg, M. Bosi. Overview of MPEG audio: current and future standards for low-bit-rate audio coding. *J Audio Eng Soc*, 45:4–21, 1997.
- [18] O. M. Bruno, L. G. Nonato, M. A. Pazoti, J. B. Neto. Topological multi-contour decomposition for image analysis and image retrieval. *Pattern Recognition Letters*, 29(11):1675 – 1683, 2008.
- [19] <http://www.classical.com>.
- [20] D. Carevic, T. Caelli. Loeve transform. *Graph. Models Image Process.*, 59(1):27–38, 1997.
- [21] W. Chen, M. J. Er, S. Wu. PCA and LDA in DCT domain. *Pattern Recognition Letters*, 26(15):2474 – 2482, 2005.
- [22] W. Chen, C. Smith, S. Fralick. A Fast Computational Algorithm for the Discrete Cosine Transform. *IEEE Trans. on Communications*, (25):1004–1009, 1977.

- [23] S. Chitroub, A. Houacine, B. Sansal. SAR imagery. *Intell. Data Anal.*, 6(2):187–207, 2002.
- [24] J. W. Cooley, J. W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of Computation*, 19(90):297–301, 1965.
- [25] M. Cowling. *Non-Speech Environmental Sound Classification System for Autonomous Surveillance*. Praca doktorska, Faculty of Engineering and Information Technology, Griffith University, Gold Coast Campus, 2004.
- [26] Y. L. Cun, J. Denker. Optimal brain damage. *Advances in Neural Information Processing Systems*, wolumen 2, strony 598 – 605. Morgan Kaufmann, 1989.
- [27] Z. Cvetković, M. V. Popović. New fast recursive algorithms for the computation of discrete cosine and sine transforms. *IEEE Trans. Signal Processing*, (40):2083–2086, 1992.
- [28] A. Czyżewski. *Dźwięk cyfrowy*. Akademicka Oficyna Wydawnicza EXIT, 2001.
- [29] E. R. Davies. *Machine Vision : Theory, Algorithms, Practicalities*. Morgan Kaufmann, 2004.
- [30] P. Dayan, L. F. Abbott. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. The MIT Press, 2005.
- [31] F. M. de S. Matos, L. V. Batista, J. v. d. Poel. Face recognition using DCT coefficients selection. *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, strony 1753–1757, 2008.
- [32] S. Derrode, F. Ghorbel. Mellin transform approximations for gray-level image reconstruction and complete invariant description. *Comput. Vis. Image Underst.*, 83(1):57–78, 2001.
- [33] S. Derrode, F. Ghorbel. Robust and efficient Fourier-Mellin transform approximations for gray-level image reconstruction and complete invariant description. *Computer Vision and Image Understanding*, (83):57–78, 2001.
- [34] C. Ding, X. He. K-means clustering via principal component analysis. *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, strona 29, New York, NY, USA, 2004. ACM.

Bibliografia

- [35] E. Feig, S. Winograd. Fast Algorithms for the Discrete Cosine Transform. *IEEE Trans. Signal Processing*, (40):2174–2193, 1992.
- [36] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals Eugen.*, 7:179–188, 1936.
- [37] R. Fletcher, C. M. Reeves. Function Minimization by Conjugate Gradients. *Computer Journal*, 7:149–154, 1964.
- [38] J. Fourier. *Théorie analytique de la chaleur*. Chez Firmin Didot, Père et Fils, 1822.
- [39] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980.
- [40] F. Ghorbel. A complete invariant description for gray-level images by the harmonic analysis approach. *Pattern Recognition Letters*, 15(10):1043–1051, 1994.
- [41] J. C. Gilbert, J. Nocedal. Global Convergence Properties of Conjugate Gradient Methods for Optimization. *SIAM Journal on Optimization*, 2(1):21–42, 1992.
- [42] R. C. Gonzalez, R. E. Woods. *Digital Image Processing (3rd edition)*. Prentice Hall, 2007.
- [43] I. J. Good. The Interaction Algorithm and Practical Fourier Series. *J. Roy. Statist. Soc. Ser. B*, 20:361–372, 1958.
- [44] S. Grossberg. A neural theory of punishment and avoidance. *II. Quantitative theory*. *Mathematical Biosciences*, (15), 1972.
- [45] M. Hamidi, J. Pearl. Comparison of the cosine and Fourier transforms of Markov-1 signals. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 24:428–429, 1976.
- [46] R. M. Haralick. A Storage Efficient Way to Implement the Discrete Cosine Transform. *IEEE Trans. Comput.*, 25(7):764–765, 1976.
- [47] R. V. L. Hartley. A More Symmetrical Fourier Analysis Applied to Transmission Problems. *Proceedings of the IRE*, wolumen 30, strony 144–150, 1942.

- [48] B. Hassibi. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in Neural Information Processing Systems*, wolumen 5, strony 164 – 171. Morgan Kaufmann, 1992.
- [49] D. O. Hebb. *The organization of behaviour: A neurophysiological theory*. John Wiley and Sons, 1949.
- [50] M. Heideman, D. Johnson, C. Burrus. Fourier transform. *ASSP Magazine, IEEE*, 1:14–21, 1984.
- [51] M. R. Hestenes, E. Stiefel. Methods of Conjugate Gradients for Solving Linear Systems. *Journal of Research of the National Bureau of Standards*, 49:409–436, 1952.
- [52] S. Hettich, S. D. Bay. The UCI KDD Archive [<http://kdd.ics.uci.edu>], 1999.
- [53] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.
- [54] H. Hotelling. Analysis of a complex of statistical variables with principal components. *Journal of Educational Psychology*, 24:417—441, 1933.
- [55] H. S. Hou. A fast recursive algorithm for computing the discrete cosine transform. *IEEE Trans. Acoustics, Speech, and Signal Processing*, (35), 1987.
- [56] M. Hubert, S. Engelen. PCA and classification in biosciences. *Bioinformatics*, 20(11):1728–1736, 2004.
- [57] JPEG Coding of Still Pictures, ISO/IEC JTC 1/SC 29/WG 1.
- [58] M. Jacymirski. Szybkie algorytmy jednolite kosinusowych transformat drugiego i trzeciego rodzaju o mnożnikach tangensowych. *Automatyka*, strony 727–741, 2003.
- [59] M. Jacymirski, P. Szczepaniak. Neural realization of fast linear filters. *Proc. of the 4th EURASIP-IEEE Region 8 International Symposium on Video/Image Processing and Multimedia Communications VIPromCom*, strony 153–157, 2002.
- [60] M. Jacymirski, T. Wiechno. A novel method of building Fast Fourier Transform algorithms. *Proceedings of the International Conference on Signals and Electronic Systems (ICSES'2001)*, strony 415–422, 2001.

Bibliografia

- [61] A. K. Jain. A fast Karhunen-Loeve transform for a class of random processes. *IEEE Trans. Commun.*, (24):1023–1029, 1976.
- [62] A. K. Jain. A sinusoidal family of unitary transforms. *IEEE Trans. Pattern Anal. Machine Intell.*, (1):356–365, 1979.
- [63] A. K. Jain, P. Flynn, A. A. Ross. *Handbook of Biometrics*. Springer, 2007.
- [64] I. T. Jolliffe. *Principal Component Analysis*. Springer, wydanie 2, 2002.
- [65] B. Kapralos, N. Mekuz. HRTFS for interactive virtual environments. *ACE'07: Proceedings of the international conference on Advances in computer entertainment technology*, strony 256–257. ACM, 2007.
- [66] S. Kara. Classification of mitral stenosis from Doppler signals using short time Fourier transform and artificial neural networks. *Expert Syst. Appl.*, 33(2):468–475, 2007.
- [67] K. Karhunen. Zur Spektraltheorie Stochastischer Prozesse. *Ann. Acad. Sci. Fennicae*, 34:3–7, 1946.
- [68] H. Kauppinen, T. Seppänen, M. Pietikäinen. An Experimental Comparison of Autoregressive and Fourier-Based Descriptors in 2D Shape Classification. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17(2):201–207, 1995.
- [69] G. U. Kharat, S. V. Dudul. Neural Network Classifier for Human Emotion Recognition from Facial Expressions Using Discrete Cosine Transform. *ICETET '08: Proceedings of the 2008 First International Conference on Emerging Trends in Engineering and Technology*, strony 653–658, Washington, DC, USA, 2008. IEEE Computer Society.
- [70] H.-C. Kim, D. Kim, S. Y. Bang. Face recognition using the mixture-of-eigenfaces method. *Pattern Recogn. Lett.*, 23(13):1549–1558, 2002.
- [71] D. King, W. B. Lyons, C. Flanagan, E. Lewis. Signal processing technique utilising Fourier transform methods and Artificial Neural Network pattern recognition for interpreting complex data from a multipoint optical fibre sensor system. *WISICT '04: Proceedings of the winter international symposium on Information and communication technologies*, strony 1–6. Trinity College Dublin, 2004.
- [72] T. Kohonen. *Self-Organization and Associative Memory*, wolumen 8 serii *Springer Series in Information Sciences*. Springer, Berlin, Heidelberg, 1984.

- [73] B. Kostek. Musical instrument classification and duet analysis employing music information retrieval techniques. *Proceedings of the IEEE*, 92:712–729, 2004.
- [74] J. Lampinen, J. Laaksonen, E. Oja. Pattern recognition. *Neural Network Systems. Techniques and Applications*, 5, 1998.
- [75] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.
- [76] P. Lee, F.-Y. Huang. Restructured Recursive DCT and DST Algorithms. *IEEE Trans. Signal Processing*, 42:1600–1609, 1994.
- [77] B. Leibe, B. Schiele. Analyzing appearance and contour based methods for object categorization. *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR'03)*, 2003.
- [78] T. Li, M. Ogihara, Q. Li. A comparative study on content-based music genre classification. *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in information retrieval*, strony 282–289. ACM, 2003.
- [79] Z. Liu, Y. Wang, T. Chen. Audio Feature Extraction and Analysis for Scene Segmentation and Classification. *J. VLSI Signal Process. Syst.*, 20(1/2):61–79, 1998.
- [80] M. Loeve. *Probability Theory*. VanNostrand, Princeton, NJ, 1955.
- [81] M. Loog, R. P. W. Duin, R. Haeb-Umbach. Multiclass Linear Dimension Reduction by Weighted Pairwise Fisher Criteria. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(7):762–766, 2001.
- [82] J. Lu, K. N. Plataniotis, A. N. Venetsanopoulos. Regularization studies of linear discriminant analysis in small sample size scenarios with application to face recognition. *Pattern Recogn. Lett.*, 26(2):181–191, 2005.
- [83] L. Lu, H. J. Zhang. Content analysis for audio classification and segmentation. *IEEE Transactions on Speech and Audio Processing*, (10):504–516, 2002.
- [84] R. G. Lyons. *Wprowadzenie do cyfrowego przetwarzania sygnałów*. Wydawnictwa Komunikacji i Łączności, 2000.

Bibliografia

- [85] W. Malina. *Podstawy automatycznej klasyfikacji obrazów*. Wydawnictwo Politechniki Gdańskiej, 2002.
- [86] H. Malvar. Fast computation of discrete cosine transform through fast Hartley transform. *Electronics Letters*, 22:352–353, 1986.
- [87] Z. Markov, D. T. Larose. *Data Mining the Web: Uncovering Patterns in Web Content, Structure, and Usage*. Wiley-Interscience, 2007.
- [88] W. McCulloch, W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, 5(4):115–133, December 1943.
- [89] R. Milanese, M. Cherbuliez. A Rotation, Translation, and Scale-Invariant Approach to Content-Based Image Retrieval. *Journal of Visual Communication and Image Representation*, 10(2):186–196, 1999.
- [90] M. L. Minsky, S. A. Papert. *Perceptrons*. MIT Press, Cambridge, MA, 1969.
- [91] M. Narasimha, A. Peterson. On the computation of discrete cosine transform. *IEEE Trans. Comm.*, 26:934–936, 1978.
- [92] S. A. Nene, S. K. Nayar, H. Murase. Columbia object image library (COIL-20). Raport instytutowy, 1996.
- [93] A. V. Oppenheim, M. H. Hayes, J. S. Lim. Iterative procedures for signal reconstruction from phase. W. T. Rhodes, redaktor, *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, wolumen 231, strony 121–129, 1980.
- [94] S. Osowski. *Sieci neuronowe do przetwarzania informacji, wyd. 2*. Oficyna Wydawnicza Politechniki Warszawskiej, 2006.
- [95] S. Osowski, D. Nghia. Neural networks for classification of 2-D patterns. *Signal Processing Proceedings, 2000. WCCCC-ICSP 2000. 5th International Conference on*, wolumen 3, strony 1568–1571, 2000.
- [96] C. H. Park, H. Park. Fingerprint classification using fast Fourier transform and nonlinear discriminant analysis. *Pattern Recognition*, 38(4):495 – 503, 2005.
- [97] K. Pearson. On Lines and Planes of Closest Fit to Systems of Points in Space. *Philosophical Magazine*, 2:559–572, 1901.

-
- [98] T. C. Poh, N. F. M. Lani, L. W. Kin. Multi-dimensional features reduction of PCA on SVM classifier for imaging surveillance application. *ISPRA'08: Proceedings of the 7th WSEAS International Conference on Signal Processing, Robotics and Automation*, strony 192–197, 2008.
 - [99] D. Puchala, M. Yatsymirskyy. Neural Network in Fast Adaptive Fourier Descriptor Based Leaves Classification. *ICAISC '08: Proceedings of the 9th international conference on Artificial Intelligence and Soft Computing*, strony 135–145, Berlin, Heidelberg, 2008. Springer-Verlag.
 - [100] D. Puchała. Fast Orthogonal Networks in Signal Compression. *Proceedings of ISDMCI Conference*, strony 177–180, 2009.
 - [101] L. Rabiner, B.-H. Juang. *Fundamentals of speech recognition*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.
 - [102] E. Rahtu, M. Salo, J. Heikkila. Affine Invariant Pattern Recognition Using Multiscale Autoconvolution. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(6):908–918, 2005.
 - [103] K. R. Rao, P. Yip. *Discrete cosine transform: algorithms, advantages, applications*. Academic Press Professional, Inc., San Diego, CA, USA, 1990.
 - [104] F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, 1962.
 - [105] H. A. Rowley, S. Baluja, T. Kanade. Human Face Detection in Visual Scenes. Raport instytutowy, Carnegie Mellon University. Computer Science Technical Report CMU-CS-95-158, 1995.
 - [106] D. Rumelhart, G. Hinton, R. Williams. Learning internal representation by error propagation. *Parallel Distributed Processing*. MIT Press, 1986.
 - [107] L. Rutkowski. *Metody i techniki sztucznej inteligencji*. Wydawnictwo Naukowe PWN, 2005.
 - [108] K. Sayood. *Kompresja danych*. Read Me, 2002.
 - [109] S. Serhatlioglu, F. Hardalaç, I. Güler. Classification of Transcranial Doppler Signals Using Artificial Neural Network. *J. Med. Syst.*, 27(2):205–214, 2003.
 - [110] Y. Sheng, H. H. Arsenault. Mellin descriptors. *J. Opt. Soc. Am. A*, (3):771, 1986.

Bibliografia

- [111] T. Sikora, B. Makai. Shape-Adaptive DCT for Generic Coding of Video. *IEEE Trans. Circuits Syst. Video Technol.*, (5):59–62, 1995.
- [112] P. K. Singh. Unsupervised segmentation of medical images using DCT coefficients. *VIP '05: Proceedings of the Pan-Sydney area workshop on Visual information processing*, strony 75–81, Darlinghurst, Australia, 2004. Australian Computer Society, Inc.
- [113] I. N. Sneddon. *Fourier Transforms (Dover Books on Mathematics)*. Dover Publications, 1995.
- [114] K. Sohn, S. H. Lim. PCA for Molecular Classification. *FSKD '07: Proceedings of the Fourth International Conference on Fuzzy Systems and Knowledge Discovery*, strony 275–279, Washington, DC, USA, 2007. IEEE Computer Society.
- [115] B. Stasiak. Sieć neuronowa do rozpoznawania i klasyfikacji obrazów; praca magisterska (Instytut Informatyki, Politechnika Łódzka), 2004.
- [116] B. Stasiak. Fast Orthogonal Neural Network for Rotation-Translation- and Scale-Invariant Image Recognition. *Proc. of 12th IASTED International Conference on Intelligent Systems and Control*, 2009.
- [117] B. Stasiak. Two-dimensional Fast Orthogonal Neural Network for Image Recognition. E. Bayro-Corrochano, J. O. Eklundh, redaktorzy, *Proceedings of Iberoamerican Congress on Pattern Recognition, CIARP 2009, LNCS 5856*. Springer Verlag, 2009.
- [118] B. Stasiak, M. Yatsymirskyy. Analiza porównawcza deskryptorów obrazu opartych na transformacie Fouriera-Mellina. D. Rutkowska, J. Kacprzyk, W. Kosiński, K. Przybyszewski, J. Starczewski, redaktorzy, *Selected Problems of Computer Science*, strony 569–577. Akademicka Oficyna Wydawnicza EXIT, Warszawa, 2005.
- [119] B. Stasiak, M. Yatsymirskyy. Zastosowanie transformaty Fouriera-Mellina do kategoryzacji obiektów 3D. *Zeszyty Naukowe Wydziału ETI PG, Technologie Informacyjne*, 6:185–192, 2005.
- [120] B. Stasiak, M. Yatsymirskyy. Fast Orthogonal Neural Networks. L. Rutkowski, R. Tadeusiewicz, L. A. Zadeh, J. M. Żurada, redaktorzy, *Proceedings of 8th International Conference on Artificial Intelligence and Soft Computing (ICAISC)*, wolumen 4029 serii *Lecture Notes in Computer Science*, strony 142–149. Springer, 2006.

- [121] B. Stasiak, M. Yatsymirskyy. Recursive Learning of Fast Orthogonal Neural Networks. *Proceedings of International Conference on Signals and Electronic Systems ICSES06*, Łódź, strony 653–656, 2006.
- [122] B. Stasiak, M. Yatsymirskyy. Fast orthogonal neural networks for 2D signal processing. *Electrical Review, Conferences, No 2*, strony 167–170, 2007.
- [123] B. Stasiak, M. Yatsymirskyy. On Feature Extraction Capabilities of Fast Orthogonal Neural Networks. B. Beliczyński, A. Dzieliński, M. Iwanowski, B. Ribeiro, redaktorzy, *Proc. of International Conference on Adaptive and Natural Computing Algorithms (ICANNGA)*, wolumen 4432 serii *Lecture Notes in Computer Science*, strony 27–36. Springer, 2007.
- [124] B. Stasiak, M. Yatsymirskyy. Application of fast orthogonal neural network in content-based music genre classification. *Polish Journal of Environmental Studies*, 17(2A):81–85, 2008.
- [125] B. Stasiak, M. Yatsymirskyy. Fast homogeneous algorithm of two-dimensional cosine transform, type II with tangent multipliers. *Electrical Review, No 12*, strony 290–292, 2008.
- [126] B. Stasiak, M. Yatsymirskyy. Fast Orthogonal Neural Network for Adaptive Fourier Amplitude Spectrum Computation in Classification Problems. *Proc. International Conference on Man-Machine Interactions (ICMMI'09)*, strony 327–334, 2009.
- [127] B. Stasiak, M. Yatsymirskyy. Frequency Domain Methods for Content-Based Image Retrieval in Multimedia Databases. D. Zakrzewska, E. Menasalvas, L. Byczkowska-Lipińska, redaktorzy, *Methods and Supporting Technologies for Data Analysis*, strony 137–166. Springer Verlag Berlin Heidelberg, 2009.
- [128] K. Stokfiszewski, P. S. Szczepaniak. Image Compression with a Fast Transform Based Architecture Neural Networks. *Proceedings of the 7th Conference on Computer Methods and Systems*. Oprogramowanie Naukowo-Techniczne, Kraków, 2009.
- [129] K. K. Sung, T. Poggio. Example Based Learning for View-Based Human Face Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20:39–51, 1995.
- [130] J. Szabatin. *Podstawy Teorii Sygnałów*. Wydawnictwa Komunikacji i Łączności, 2003.

Bibliografia

- [131] P. S. Szczepaniak. *Obliczenia inteligentne, szybkie przekształcenia i klasifikatory*. Akademicka Oficyna Wydawnicza EXIT, 2004.
- [132] R. Tadeusiewicz. *Sieci neuronowe*. Akademicka Oficyna Wydawnicza, Warszawa, 1993.
- [133] R. Tadeusiewicz, M. Flasiński. *Rozpoznawanie obrazów*. PWN, 1991.
- [134] R. Tadeusiewicz, P. Korohoda. *Komputerowa analiza i przetwarzanie obrazów*. Wydawnictwo Fundacji Postępu Telekomunikacji, 1997.
- [135] S. Theodoridis, K. Koutroumbas. *Pattern Recognition, Fourth Edition*. Academic Press, 2009.
- [136] B. D. Tseng, W. C. Miller. On Computing the Discrete Cosine Transform. *IEEE Trans. Comput.*, 27(10):966–968, 1978.
- [137] M. Turk, A. Pentland. Eigenfaces for recognition. *J. Cognitive Neuroscience*, 3(1):71–86, 1991.
- [138] G. Tzanetakis, P. Cook. Musical genre classification of audio signals. *Speech and Audio Processing, IEEE Transactions on*, 10(5):293–302, 2002.
- [139] http://xiph.org/vorbis/doc/Vorbis_I_spec.html.
- [140] V. N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag, 1995.
- [141] V. N. Vapnik, A. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of probability and its applications*, 16:264–280, 1971.
- [142] G. K. Wallace. JPEG still picture compression standard. *Commun. ACM*, 34(4):30–44, 1991.
- [143] Z. Wang. Fourier transform. *IEEE Trans. Acoust. Speech Signal Process*, (32):816, 1984.
- [144] Z. Wang. On computing the discrete Fourier and cosine transforms. *IEEE Trans. ASSP*, (33):1341–1344, 1985.
- [145] J. Watkinson. *MPEG Handbook*. Butterworth-Heinemann, Newton, MA, USA, 2001.

- [146] T. Westra, R. R. De Wulf. Monitoring Sahelian floodplains using Fourier analysis of MODIS time-series data and artificial neural networks. *Int. J. Remote Sens.*, 28(7):1595–1610, 2007.
- [147] B. Widrow. Generalization and information storage in networks of Adaline neurons. *Self-Organizing Systems*, strony 435–461. Spartan Books, 1962.
- [148] B. Widrow, M. E. Hoff. Adaptive switching circuits. *IRE WESCON Convention Record*, 4:96–104, 1960.
- [149] B. Widrow, R. G. Winter, R. A. Baxter. Layered Neural Nets for Pattern Recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, (36):1109–1118, 1988.
- [150] J.-L. Wu, W.-J. Duh, S.-H. Hsu. A novel two-stage algorithm for DCT and IDCT. *IEEE Transactions on Signal Processing*, 40:1610–1612, 1992.
- [151] J.-L. Wu, W.-J. Duh, S.-H. Hsu. Basis-vector-decomposition based two-stage computational algorithms for DFT and DHT. *IEEE Transactions on Signal Processing*, 41:1562–1575, 1993.
- [152] M. Yatsymirskyy. *The Fast Orthogonal Trigonometric Transform Algorithms (in Ukrainian)*. Academic Express LTD, Lviv, 1997.
- [153] M. Yatsymirskyy. Shifted in the time and frequency domains cosine and sine transforms fast algorithms with homogeneous structure. *Izvestiya Vysshikh Uchebnykh Zavedenii, Radioelektronika*, 43:66–75, 2000.
- [154] M. Yatsymirskyy, R. Liskevych. Lattice structures for Fourier, Hartley, cosine and sine transformations (in Ukrainian). *Modelling and Information Technologies*, (2):173–181, 1999.
- [155] H. Zha, X. He, C. Ding, H. Simon, M. Gu. Spectral Relaxation for K-means Clustering. *Advances in neural information processing systems*, strony 1057–1064. MIT Press, 2001.
- [156] C. Zhang, X. Chen, W.-b. Chen. PCA-Based Vehicle Classification Framework. *ICDEW '06: Proceedings of the 22nd International Conference on Data Engineering Workshops*, strona 17. IEEE Computer Society, 2006.
- [157] D. Zhang, G. Lu. A comparative study of curvature scale space and Fourier descriptors for shape-based image retrieval. *Journal of Visual Communication and Image Representation*, 14(1):39–57, 2003.

Bibliografia

- [158] G. Zhang, Z. M. Ma, Q. Tong, Y. He, T. Zhao. Shape Feature Extraction Using Fourier Descriptors with Brightness in Content-Based Medical Image Retrieval. *Intelligent Information Hiding and Multimedia Signal Processing, International Conference on*, strony 71–74, 2008.
- [159] M. Zhu. Where Are Linear Feature Extraction Methods Applicable? *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(12):1934–1944, 2005.