



POLITECHNIKA
LUBELSKA
WYDZIAŁ PODSTAW
TECHNIKI

KIERUNEK: MATEMATYKA



Praca magisterska

Przegląd autoenkoderów stosowanych w nienadzorowanym uczeniu
maszynowym

An overview of autoencoders used in unsupervised machine learning

Praca wykonana pod kierunkiem:
dra Dariusza Majerka

Autor:
Alicja Hołowiecka
nr albumu: 89892

Lublin 2022

Spis treści

Wstęp	5
Rozdział 1. Część teoretyczna	7
1.1. Sztuczne sieci neuronowe	7
1.2. Sieci splotowe	11
1.2.1. Czym jest sieć splotowa	11
1.2.2. Elementy sieci splotowej	12
1.3. Autoenkodery	21
1.3.1. Czym jest autoenkoder	21
1.3.2. Rodzaje autoenkoderów	22
1.3.3. Zastosowania autoenkoderów	33
Rozdział 2. Część praktyczna	39
2.1. Prosty autoenkoder	39
2.2. Autoenkoder splotowy	41
2.3. Autoenkoder odszumiający	43
2.4. Wyszukiwanie obrazu	45
2.5. Wykrywanie anomalii przy użyciu autoenkodera	47
2.6. Generowanie obrazów (autoenkoder wariancyjny)	47
Podsumowanie i wnioski	53
Bibliografia	55
Spis rysunków	57
Spis tabel	61
Załączniki	63
Streszczenie (Summary)	65

Wstęp

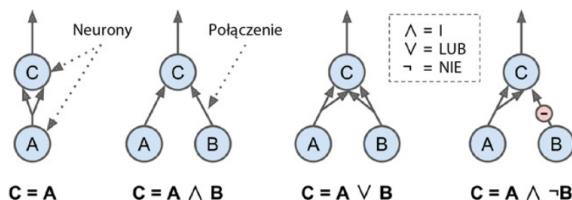
tu będzie jakiś wstęp

Rozdział 1

Część teoretyczna

1.1. Sztuczne sieci neuronowe

Sztuczny neuron jest systemem składającym się z co najmniej jednego binarnego wejścia i dokładnie jednego binarnego wyjścia. Wyjście uaktywnia się, jeżeli jest aktywna określona liczba wejść. Sztuczne neurony są podstawowym elementem sztucznych sieci neuronowych. Do pewnego stopnia są inspirowane budową biologicznego neuronu. Na rysunku 1.1 przedstawione są przykładowe sztuczne sieci neuronowe (SSN lub z angielskiego ANN) wykonujące różne operacje logiczne. W tych przykładach przyjęto, że neuron uaktywni się, gdy przynajmniej dwa wejścia będą aktywne [6].

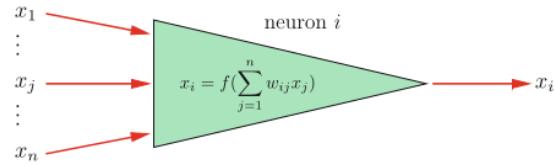


Rysunek 1.1: Przykładowe sztuczne sieci neuronowe rozwiązuje proste zadania logiczne

Źródło: [6]

Jedną z najprostszych architektur SSN jest **perceptron**, którego podstawą jest sztuczny neuron zwany **progową jednostką logiczną** (ang. *Threshold Logic Unit* - TLU) lub **liniową jednostką progową** (ang. *Linear Threshold Unit* - LTU). Wartościami wejść i wyjść są liczby, a każde połączenie ma przyporządkowaną wagę. Jednostka TLU oblicza ważoną sumę sygnałów wejściowych, a następnie zostaje użyta funkcja skokowa na tej sumie. Schemat takiej jednostki został przedstawiony na rysunku 1.2. Często używaną funkcją skokową jest **funkcja Heaviside'a**, określona równaniem

$$H(z) = \begin{cases} 0, & \text{jeśli } z < 0 \\ 1, & \text{jeśli } z \geq 0 \end{cases}$$



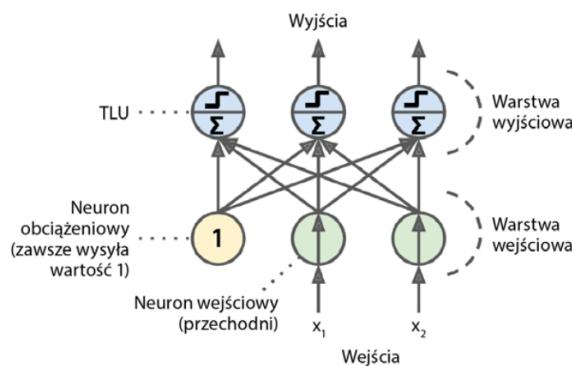
Rysunek 1.2: Struktura sztucznego neuronu, który stosuje funkcję skokową f na ważonej sumie sygnałów wejściowych

Źródło: [5]

Czasami zamiast niej stosuje się również **funkcję signum**:

$$sgn(z) = \begin{cases} -1 & \text{jeśli } z < 0 \\ 0, & \text{jeśli } z = 0 \\ 1, & \text{jeśli } z > 0 \end{cases}$$

Perceptron jest złożony z jednej warstwy jednostek TLU, w której każdy neuron jest połączony ze wszystkimi wejściami. Warstwa tego typu nazywana jest **warstwą gęstą**. Warstwa, do której są dostarczane dane wejściowe, jest nazywana **warstwą wejściową** (ang. *input layer*). Najczęściej do tej warstwy jest wstawiany również **neuron obciążeniowy** (ang. *bias neuron*) $x_0 = 1$, który zawsze wysyła wartość 1. Na rysunku 1.3 znajduje się perceptron z dwoma neuronami wejściowymi i jednym obciążeniowym, a także z trzema neuronami w warstwie wyjściowej.



Rysunek 1.3: Perceptron z trzema neuronami wejściowymi i trzema wyjściami

Źródło: [6]

Obliczanie sygnałów wyjściowych w warstwie gęstej przedstawia się wzorem

$$h_{\mathbf{W}, \mathbf{b}}(\mathbf{X}) = \phi(\mathbf{X}\mathbf{W} + \mathbf{b})$$

gdzie \mathbf{X} - macierz cech wejściowych, \mathbf{W} - macierz wag połączeń (oprócz neuronu obciążeniowego), \mathbf{b} - wektor obciążzeń zawierający wagi połączeń neuronu obciążeniowego ze

wszystkimi innymi neuronami, ϕ - tzw. **funkcja aktywacji**, w przypadku TLU jest to funkcja skokowa.

Algorytm uczący, który służy do trenowania perceptronu, jest silnie inspirowany działaniem neuronu biologicznego. Gdy biologiczny neuron często pobudza inną komórkę nerwową, to połączenia między nimi stają się silniejsze. Reguła ta jest nazywana **regułą Hebb'a**. Perceptrony są uczone za pomocą odmiany tej reguły, w której połączenia są wzmacniane, jeśli pomagają zmniejszyć wartość błędu. Dokładniej, w danym momencie perceptron przetwarza jeden przykład uczący i wylicza dla niego predykcję. Na każdy neuron wyjściowy odpowiadający za nieprawidłową prognozę następuje zwiększenie wag połączeń ze wszystkimi wejściami przyczyniającymi się do właściwej prognozy. Aktualizowanie wag przedstawia się następującym wzorem

$$\Delta w_{ij} = \eta(y_j - \hat{y}_j)x_i$$

gdzie w_{ij} - waga połączenia między i -tym neuronem wejściowym a j -tym neuronem wyjściowym, x_i - i -ta wartość wejściowa bieżącego przykładu uczącego, \hat{y}_j - wynik j -tego neuronu wyjściowego dla bieżącego przykładu uczącego, y_j - docelowy wynik j -tego neuronu, η - współczynnik uczenia.

Perceptron ma wiele wad związanych z niemożnością rozwiązania pewnych trywialnych problemów (np. zadanie klasyfikacji rozłącznej czyli XOR). Część tych ograniczeń można wyeliminować, stosując architekturę SSN złożoną z wielu warstw perceptronów, czyli **perceptron wielowarstwowy** (ang. *Multi-Layer Perceptron*). Składa się on z jednej warstwy wejściowej (przechodniej), co najmniej jednej warstwy jednostek TLU - tzw. **warstwy ukryte** (ang. *latent layers*) i ostatniej warstwy jednostek TLU - warstwy wyjściowej. Oprócz warstwy wejściowej każda warstwa zawiera neuron obciążający i jest w pełni połączona z następną warstwą. Sieć zawierająca wiele warstw ukrytych nazywamy **głęboką siecią neuronową** (ang. *Deep Neural Network* - DNN).

Do uczenia perceptronów wielowarstwowych wykorzystywany jest algorytm **propagacji wstecznej** (ang. *backpropagation*). Propagacja wsteczna jest właściwie algorytmem gradientu prostego [15]. Można go zapisać jako

$$w_{updated} = w_{old} - \eta \nabla E$$

gdzie E jest funkcją kosztu (funkcją straty) [15]. Proces jest powtarzany do momentu uzyskania zbieżności z rozwiązaniem, a każdy przebieg jest nazywany **epoką** (ang. *epoch*).

Uwaga 1.1. Wagi połączeń wszystkich warstw ukrytych należy koniecznie zainicjować losowo. W przeciwnym przypadku proces uczenia zakończy się niepowodzeniem. Na przykład jeśli wszystkie wagi i obciążenia zostaną zainicjowane wartością 0, to model będzie działał tak, jak gdyby składał się tylko z jednego neuronu. Przy zainicjowaniu wag losowo, sym-

tria zostanie złamana i algorytm propagacji wstecznej będzie w stanie wytrenować zespół zróżnicowanych neuronów [6].

Aby algorytm propagacji wstecznej działał prawidłowo, kluczową zmianą jest zastąpienie funkcji skokowej przez inne **funkcje aktywacji**. Zmiana ta jest konieczna, ponieważ funkcja skokowa zawiera jedynie płaskie segmenty i przez to nie pozwala korzystać z gradientu.

Najczęściej używana jest **funkcja logistyczna (sigmoidalna)**

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

Ma ona w każdym punkcie zdefiniowaną pochodną niezerową, dzięki czemu algorytm gradientu prostego może na każdym etapie uzyskać lepsze wyniki. Zbiór wartości tej funkcji wynosi od 0 do 1.

Inną popularną funkcją aktywacji jest **tangens hiperboliczny**

$$\tanh(z) = 2\sigma(2z) - 1$$

Funkcja ta jest ciągła i różniczkowalna, a jej zakres wartości wynosi -1 do 1 . Dzięki temu zakresowi wartości wynik każdej warstwy jest wyśrodkowany wobec zera na początku uczenia, co często pomaga w szybszym uzyskaniu zbieżności.

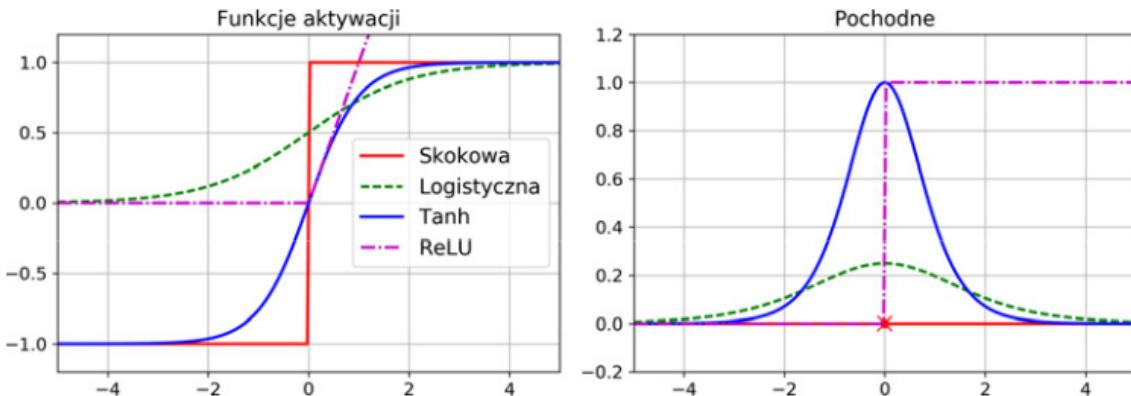
Wśród popularnych funkcji aktywacji należy także wyróżnić **funkcję ReLU** (ang. *Rectified Linear Unit* - prostowana jednostka liniowa) o wzorze

$$ReLU(z) = \max(0, z).$$

Jest ona ciągła, ale nieróżniczkowalna w punkcie 0 . Jej pochodna dla $z < 0$ wynosi zero. Jej atutem jest szybkość przetwarzania. Nie ma ona maksymalnej wartości wyjściowej. W zadaniach regresji bywa wykorzystywany „wygładzony” wariant funkcji ReLU, czyli funkcja **softplus**:

$$softplus(z) = \log(1 + \exp(z))$$

Na rysunku 1.4 przedstawiono popularne funkcje aktywacji wraz z ich pochodnymi.



Rysunek 1.4: Przykładowe funkcje aktywacji wraz z pochodnymi

Źródło: [6]

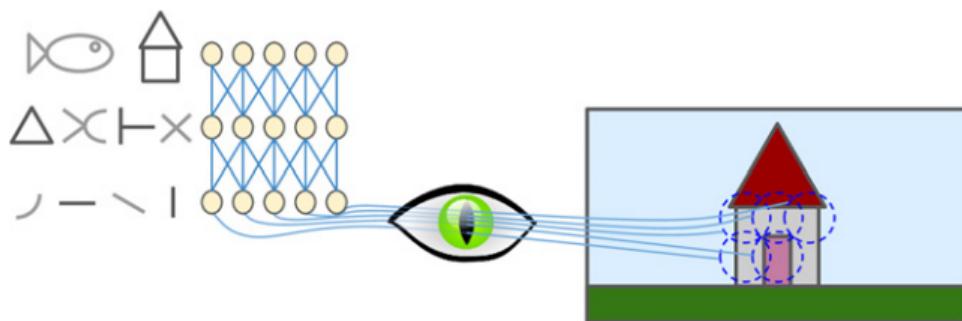
1.2. Sieci splotowe

1.2.1. Czym jest sieć splotowa

Splotowe sieci neuronowe (ang. *convolutional neural networks*, CNN) są rodzajem sieci neuronowych służących do przetwarzania danych o znanej topologii siatki. Przykładem takich danych są szeregi czasowe, które można uznać za jednowymiarową siatkę z próbami w regularnych odstępach czasu, oraz dane graficzne, które można interpretować jako dwuwymiarową siatkę pikseli. Nazwa sieci splotowych pochodzi od wykorzystywanego przez te sieci działania matematycznego nazywanego **splotem** (konwolucją). Można powiedzieć, że sieci splotowe to po prostu sieci neuronowe, które w przynajmniej jednej z warstw zamiast ogólnego mnożenia macierzy wykorzystują splot [7]. Splotowe sieci neuronowe stanowią wynik badań nad korą wzrokową. Od lat 80-tych XX wieku są używane głównie rozpoznawania obrazów, w zagadnieniach takich jak klasyfikacja, detekcja obrazów, transfer stylu. Stanowią podstawę usług takich jak wyszukiwanie obrazu, inteligentne samochody, automatyczne systemy klasyfikowania filmów. Są skuteczne również w innych dziedzinach niż jedynie komputerowe widzenie - takie dziedziny to na przykład rozpoznawanie mowy (*voice recognition*) oraz przetwarzanie języka naturalnego (*Natural Language Processing, NLP*) [6].

Ponieważ splotowe sieci neuronowe są silnie inspirowane działaniem biologicznej kory wzrokowej, to ten podrozdział zostanie poświęcony wyjaśnieniu jej działania. Na podstawie badań prowadzonych pod koniec lat 50-tych XX wieku [8] [9] wykazano, że neurony biologiczne w korze wzrokowej reagują na określone wzorce w niewielkich obszarach pola wzrokowego, zwanych **polami receptivejnymi**, a w miarę przepływu sygnału wzrokowego przez kolejne moduły w mózgu neurony rozpoznają coraz bardziej skomplikowane wzorce wykrywane w coraz większych polach receptivejnych. Wiele neuronów stanowiących korę

wzrokową tworzy **lokalne pola recepcyjne**, reagujące jedynie na bodźce wzrokowe mieszczące się w określonym rejonie pola wzrokowego, przy czym lokalne pola recepcyjne poszczególnych neuronów mogą się na siebie nakładać. Takie pola recepcyjne łącznie tworzą całe pole wzrokowe. Dodatkowo badacze zauważali, iż pewne neurony reagują wyłącznie na obrazy składające się z linii poziomych, lub innych linii ułożonych w konkretny sposób (dwa neurony mogą nawet mieć to samo pole recepcyjne, ale reagować na różne ułożenie linii). Badacze stwierdzili, że niektóre komórki nerwowe mają większe pola recepcyjne i wykrywają bardziej skomplikowane kształty. Takie neurony, odpowiedzialne za rozpoznanie bardziej skomplikowanych kształtów, znajdują się na wyjściu neuronów reagujących na prostsze bodźce. Działanie neuronów biologicznych w korze wzrokowej, zgodnie z powyższym opisem, zostało zobrazowane na rysunku 1.5.



Rysunek 1.5: Działanie neuronów biologicznych w korze wzrokowej

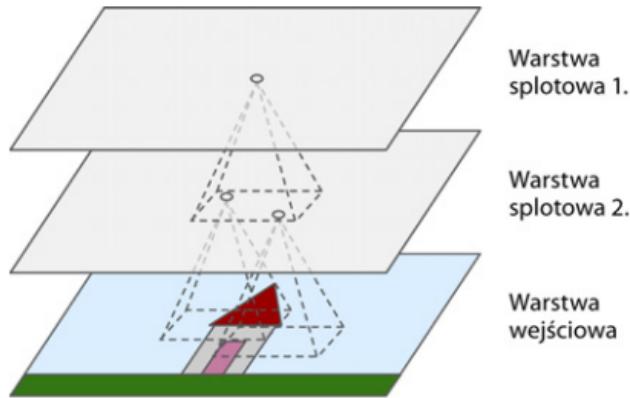
Źródło: [6]

1.2.2. Elementy sieci splotowej

Warstwy splotowe

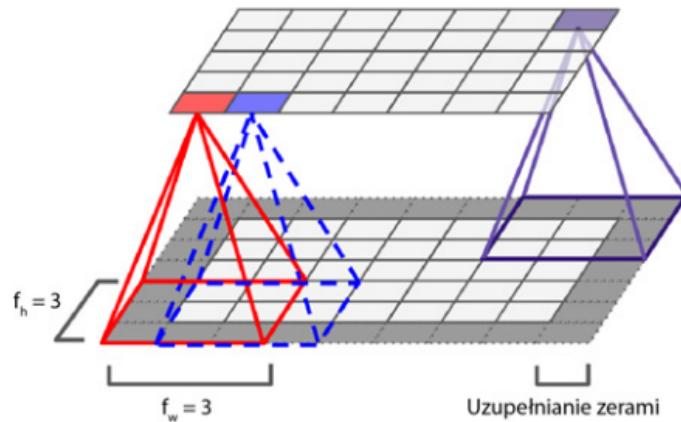
Przejdziemy teraz do opisu najistotniejszej części sieci CNN, jaką jest warstwa splotowa (*convolutional layer*). Na rysunku 1.6 widoczny jest przykład warstw splotowych z prostokątnymi polami recepcyjnymi. W pierwszej warstwie splotowej, neurony nie są połączone z każdym pikselem obrazu wejściowego (w przeciwieństwie do opisanych wcześniej warstw gęstych), lecz jedynie z pikselami znajdującymi się w polu recepcyjnym danego neuronu. W kolejnej warstwie każdy neuron łączy się wyłącznie z neuronami z niewielkiego obszaru pierwszej warstwy. Dzięki temu sieć koncentruje się na pewnych cechach w pierwszej warstwie, a w drugiej warstwie może je łączyć w bardziej złożone kształty. Taka hierarchiczna struktura w naturalny sposób występuje na zdjęciach, co przyczyniło się do dużej skuteczności sieci splotowych w rozpoznawaniu obrazu.

Neuron znajdujący się w wierszu i oraz kolumnie j danej warstwy jest połączony z wyjściami neuronów poprzedniej warstwy zlokalizowanymi w rzędach od i do $i + f_h - 1$ i

**Rysunek 1.6:** Warstwy splotowe z prostokątnymi lokalnymi polami recepcyjnymi*Źródło:* [6]

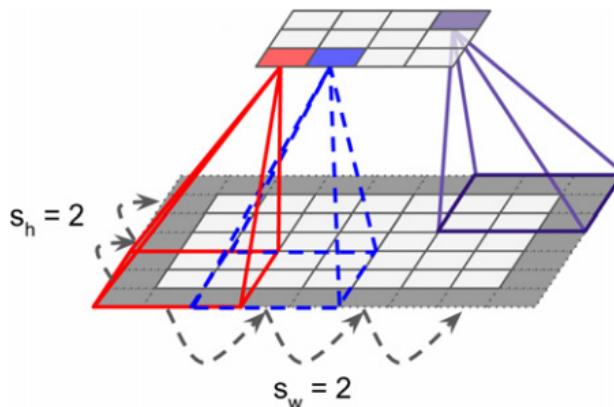
kolumnach od j do $j + f_w - 1$, gdzie f_h i f_w oznaczają, odpowiednio, wysokość i szerokość pola recepcyjnego (rysunek 1.7). W celu uzyskania takich samych wymiarów każdej warstwy najczęściej są dodawane zera wokół wejść, co zostało pokazane na rysunku 1.7. Proces ten nazywamy **uzupełnianiem zerami** (ang. zero padding).

Uwaga 1.2 (Dopełnianie). Dopełnianie (padding) jest ściśle związane z parametrem kroku (który zostanie opisany w kolejnym akapicie). Zapewnia poprawność obliczeń w warstwie konwolucyjnej [10].

**Rysunek 1.7:** Związek pomiędzy warstwami a uzupełnianiem zerami*Źródło:* [6]

Możliwe jest również łączenie bardzo dużej warstwy wejściowej ze znacznie mniejszą kolejną warstwą poprzez rozdzielenie pól recepcyjnych, tak jak zaprezentowano na rysunku 1.8. Rozwiązanie to zmniejsza drastycznie złożoność obliczeniową modelu. Odległość pomiędzy dwoma kolejnymi polami recepcyjnymi nosi nazwę **kroku** (ang. stride). Na widocznym schemacie warstwa wejściowa o wymiarach 5×7 (plus uzupełnianie zerami) łączy się z warstwą o rozmiarze 3×4 za pomocą pól recepcyjnych będących kwadratami 3×3 i

kroku o wartości 2 (w omawianym przykładzie krok jest taki sam w obydwu wymiarach, ale nie jest to wcale regułą). Neuron zlokalizowany w rzędzie i oraz kolumnie j górnej warstwy łączy się z wyjściami neuronów dolnej warstwy mieszczącymi się w rzędach od $i \times s_h$ do $i \times s_h + f_h - 1$ i w kolumnach od $j \times s_w$ do $j \times s_w + f_w - 1$, gdzie s_h i s_w definiują wartości kroków odpowiednio w kolumnach i rzędach.



Rysunek 1.8: Warstwa splotowa z krokiem o długości 2

Źródło: [6]

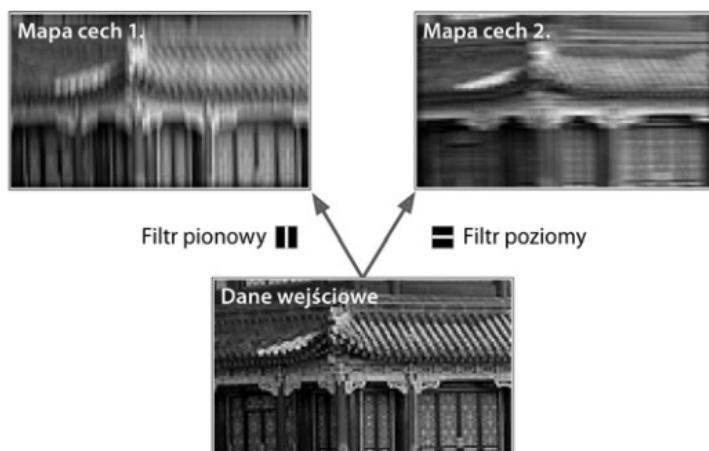
Uwaga 1.3 (Długość kroku). Długość kroku (stride length) - oznacza odległość, o jaką jądro przesuwa się po obrazie. Często stosowaną wielkością jest długość jednego piksela. Często wybieraną długością są również dwa piksele, rzadziej trzy. Dłuższe kroki nie są stosowane, ponieważ jądro może wtedy pomijać obszary obrazu potencjalnie wartościowe dla modelu. Z drugiej strony, im dłuższy krok, tym większa szybkość uczenia się modelu, ponieważ jest mniej obliczeń do wykonania. Trzeba znajdować kompromis między tymi efektami [10].

Filtry

Wagi neuronu mogą być przedstawiane jako niewielki obraz o rozmiarze pola recepcyjnego. Na przykład na rysunku 1.9 widzimy dwa możliwe zbiory wag, tak zwane **filtry** (lub **jadra splotowe**; ang. *convolution kernels*). Pierwszy filtr jest symbolizowany jako czarny kwadrat z białą pionową linią przechodzącą przez jego środek (jest to macierz o wymiarach 7×7 wypełniona zerami oprócz środkowej kolumny, która zawiera jedynki); neurony zawierające te wagi będą ignorować wszystkie elementy w polu recepcyjnym oprócz znajdujących się w środkowej pionowej linii (dane wejściowe znajdujące się poza tą linią będą przemnajżane przez 0). Drugi filtr wygląda podobnie; różnica polega na tym, że środkowa linia jest ułożona poziomo. Także w tym wypadku będą brane pod uwagę jedynie dane wejściowe znajdujące się w tej linii.

Jeśli wszystkie neurony w danej warstwie będą korzystać z tego samego filtra „pionowego” (i takiego samego członu obciążenia), a my wczytamy do sieci obraz zaprezentowany

na dole rysunku 1.9, to uzyskamy obraz widoczny w lewym górnym rogu rysunku. Zauważ, że po zastosowaniu tego filtru pionowe białe linie stają się wyraźniej widoczne, natomiast pozostała część obrazu zostaje rozmazana. Na zasadzie analogii otrzymujemy obraz widoczny w prawym górnym rogu rysunku po zastosowaniu filtru „poziomego”; teraz z kolei poziome linie zostają wyostrzone, a reszta obrazu ulega zamazaniu. Zatem warstwa wypełniona neuronami wykorzystującymi ten sam filtr daje nam **mapę cech** (ang. *feature map*), dzięki której możemy dostrzec elementy najbardziej przypominające dany filtr. Sieć CNN w czasie uczenia wyszukuje filtry najbardziej przydatne do danego zadania i uczy się łączyć je w bardziej złożone wzorce.



Rysunek 1.9: Uzyskiwanie dwóch map cech za pomocą dwóch różnych filtrów

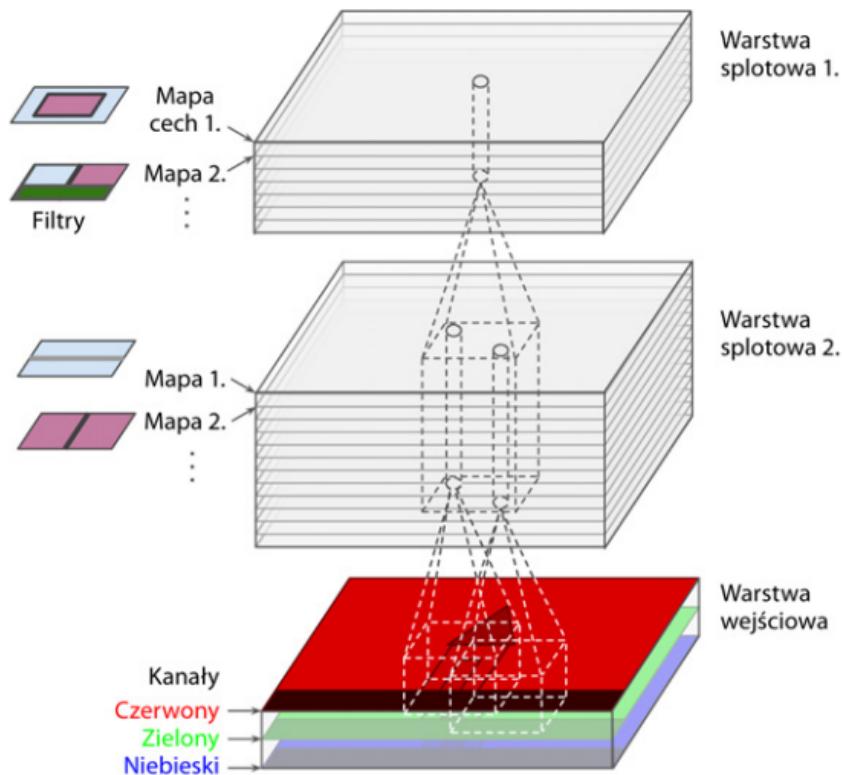
Źródło: [6]

Uwaga 1.4 (Wielkość jądra). Jądro (zwane również filtrem lub polem receptywnym) typowo ma wysokość i szerokość trzech pikseli. Rozmiar ten okazał się optymalny w szerokim zakresie zastosowań widzenia maszynowego w nowoczesnych sieciach konwolucyjnych. Popularna jest również wielkość 5x5 pikseli, a maksymalny stosowany rozmiar to 7x7 pikseli. Jeśli jądro jest zbyt duże w stosunku do obrazu, wtedy w polu receptywnym pojawia się zbyt wiele cech i warstwa konwolucyjna nie jest w stanie się skutecznie uczyć. Jeżeli jądro jest zbyt małe, np. ma wymiary 2x2 piksele, nie jest w stanie dopasować się do żadnej struktury, przez co jest bezużyteczne [10].

Stosy map cech

Do tej pory dla uproszczenia przedstawialiśmy każdą warstwę splotową w postaci cienkiej, dwuwymiarowej warstwy, ale w rzeczywistości składa się ona z kilku map cech o identycznych rozmiarach, dlatego trójwymiarowe odwzorowanie jest bliższe rzeczywistości (rysunek 1.10). W zakresie jednej mapy cech każdy neuron jest przydzielony do jednego piksela, a wszystkie tworzące ją neurony współdzielą te same parametry (wagi i człon obciążenia).

Neurony w innych mapach cech mają odmienne wartości parametrów. Pole recepcyjne neuronu nie ulega zmianie, ale „przebiega” przez wszystkie mapy cech poprzednich warstw. Krótko mówiąc, warstwa splotowa równocześnie stosuje różne filtry na wejściach, dzięki czemu jest w stanie wykrywać wiele cech w dowolnym obszarze obrazu.



Rysunek 1.10: Warstwy splotowe zawierające wiele map cech, a także zdjęcie z trzema kanałami barw

Źródło: [6]

Co więcej, obrazy wejściowe także składają się z kilku warstw podrzędnych, po jednej na każdy **kanał barw** (ang. *color channel*). Standardowo występują trzy kanały barw — czerwony, zielony i niebieski (ang. red, green, blue — RGB). Obrazy czarno-białe (w odcieniach szarości) zawierają tylko jeden kanał, ale istnieją też takie zdjęcia, które mogą mieć ich znacznie więcej — np. fotografie satelitarne utrwalające dodatkowe częstotliwości fal elektromagnetycznych (takie jak podczerwień).

W szczególności neuron zlokalizowany w rzędzie i oraz kolumnie j mapy cech k w danej warstwie splotowej l jest połączony z neuronami wcześniejszej warstwy $l - 1$ umieszczonymi w rzędach od $i \times s_h$ do $i \times s_h + f_h - 1$ i kolumnach od $j \times s_w$ do $j \times s_w + f_w - 1$ we wszystkich mapach cech (warstwy $l - 1$). Wszystkie neurony znajdujące się w tym samym rzędzie i oraz kolumnie j , ale w innych mapach cech są połączone z wyjściami dokładnie tych samych neuronów poprzedniej warstwy.

Powyższy opis został podsumowany następującym wzorem, służącym do obliczania wyniku danego neuronu w warstwie splotowej:

$$z_{i,j,k} = b_k \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k'=0}^{f_{n'}-1} x_{i',j',k'} \cdot w_{u,v,k',k}, \quad \text{gdzie } \begin{cases} i' = i \times s_h + u \\ j' = j \times s_w + v \end{cases}$$

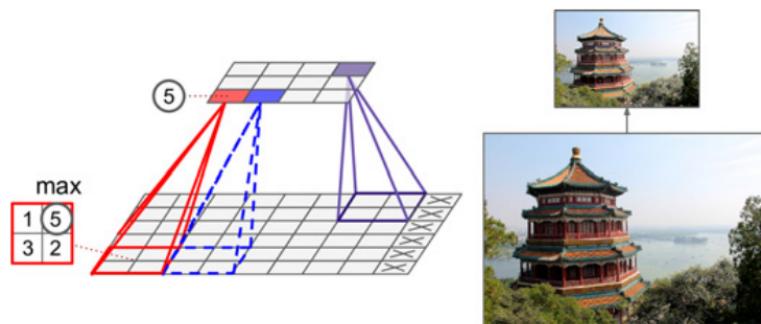
W tym równaniu:

- $z_{i,j,k}$ jest wyjściem neuronu znajdującego się w rzędzie i , kolumnie j i mapie cech k warstwy splotowej l ;
- jak już zostało wyjaśnione, s_h i s_w to kroki pionowy i poziomy, f_h i f_w są wysokością i szerokością pola receptivejnego, natomiast $f_{n'}$ oznacza liczbę map cech w poprzedniej warstwie ($l - 1$);
- $x_{i',j',k'}$ jest wyjściem neuronu zlokalizowanego w warstwie $l - 1$, rzędzie i' , kolumnie j' , mapie cech k (lub kanale k' , jeżeli poprzednia warstwa była warstwą wejściową);
- b_k to człon obciążenia dla mapy cech k (w warstwie l); można go interpretować jako „pokrętło jasności” mapy cech k ;
- $w_{u,v,k',k}$ jest wagą połączenia pomiędzy dowolnym neuronem w mapie cech k warstwy l a jego wejściem mieszącym się w wierszu u , kolumnie v (względem pola receptivejnego neuronu) a mapą cech k' .

Warstwa łącząca

Przejdźmy teraz do drugiego elementu budulcowego sieci CNN — **warstwy łączącej**, zwanej czasem redukującą (ang. pooling layer). Warstwa konwolucyjna może zawierać dowolną liczbę jąder, z których każde generuje mapę aktywacji. Zatem wyjście warstwy konwolucyjnej jest trójwymiarowa tablica aktywacji, której głębokość jest równa liczbie filtrów. Warstwa redukująca zmniejsza przestrzenny wymiar mapy aktywacji, pozostawiając jej głębokość bez zmian [10]. Jej celem jest podpróbkowanie (ang. subsample; tj. zmniejszenie) obrazu wejściowego w celu zredukowania obciążenia obliczeniowego, wykorzystania pamięci i liczby parametrów (a tym samym ograniczenia ryzyka przetrenowania). Podobnie jak w przypadku warstw splotowych, każdy neuron stanowiący część warstwy łączącej łączy się z wyjściami określonej liczby neuronów warstwy poprzedniej, mieszącej się w obszarze niewielkiego, prostokątnego pola receptivejnego. Podobnie jak wcześniej, musimy definiować tu rozmiar tego pola, wartość kroku, rodzaj uzupełniania zerami itd. Jednakże warstwa łącząca nie zawiera żadnych wag; jej jedynym zadaniem jest gromadzenie danych wejściowych za pomocą jakiejś funkcji agregacyjnej, np. maksymalizującej lub średniającej. Na rysunku 1.11 widzimy najpopularniejszy rodzaj — **maksymalizującą warstwę łączącą** (ang. max pooling layer). W tym przykładzie korzystamy z **jądra łączącego** (ang. pooling kernel) o rozmiarze 2×2 , kroku o wartości 2 i z pominięciem uzupełniania zerami.

Jedynie maksymalna wartość z każdego jądra zostaje przekazana do następnej warstwy natomiast pozostałe wartości wejściowe zostają odrzucone. Na przykład w lewym dolnym polu recepcyjnym na rysunku 1.11 widzimy wartości wejściowe 1, 5, 3, 2, zatem tylko wartość maksymalna, czyli 5, zostanie przekazana do następnej warstwy. Z powodu kroku równego 2 obraz wyjściowy ma szerokość i wysokość o połowę mniejsze w porównaniu do obrazu wejściowego (zaokrąglamy tu w dół, ponieważ nie korzystamy z uzupełniania zerami).

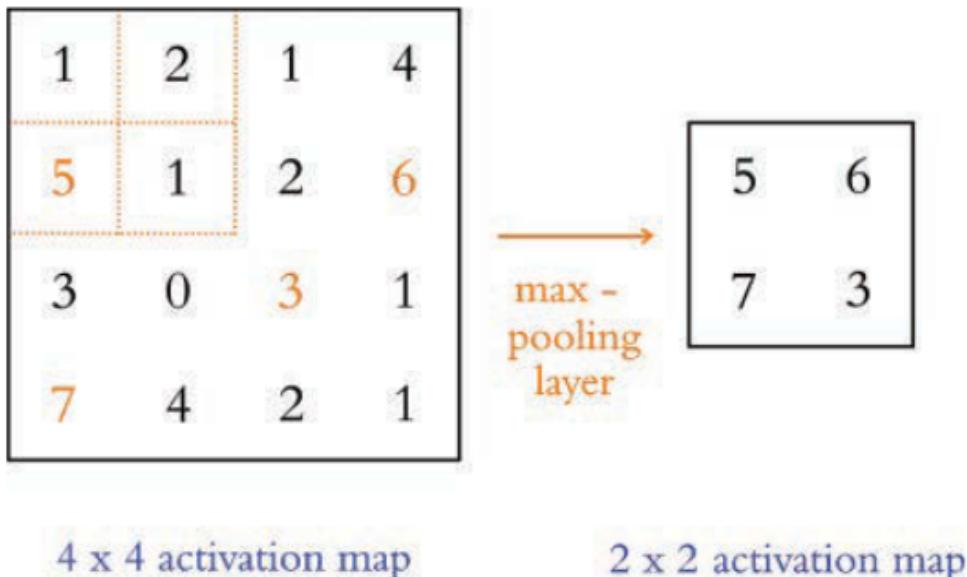


Rysunek 1.11: Maksymalizująca warstwa łącząca (jądro łączące: 2×2 , krok: 2, brak uzupełniania zerami)

Źródło: [6]

Uwaga 1.5 (Parametry warstwy redukującej). Filtr w warstwie redukującej ma zazwyczaj wymiary 2×2 piksele, a krok ma długość dwóch pikseli. W takim wypadku filtr w każdej pozycji przetwarza cztery wartości aktywacji, wybiera największą i w efekcie czterokrotnie redukuje liczbę aktywacji [10].

Na rysunku 1.12 widoczne jest działanie maksymalizującej warstwy redukującej na mapie aktywacji o wymiarze 4×4 . Filtr przesuwa się nad danymi wejściowymi od lewej do prawej, od góry do dołu, tak jak w warstwie konwolucyjnej, i w każdej zajmowanej pozycji przeprowadza operację redukcji danych. W tym przykładzie filtr i krok mają rozmiar 2×2 . Skutkuje to otrzymaniem mapy cztery razy mniejszej niż oryginalna.

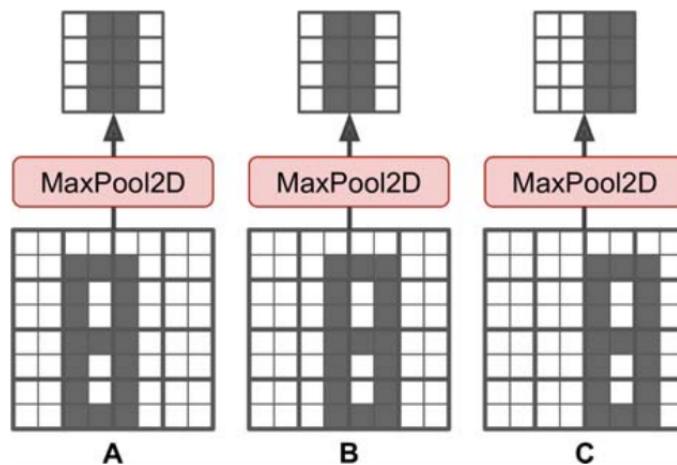


Rysunek 1.12: Warstwa *max-pooling* z filtrem i krokiem rozmiaru 2×2 , zastosowana do mapy aktywacji o rozmiarze 4×4 (widoczna po lewej stronie) skutkuje uzyskaniem mapy czterokrotnie mniejszej niż oryginalna

Źródło: [10]

Oprócz ograniczania liczby obliczeń, zużycia pamięci i liczby parametrów maksymalizująca warstwa łącząca wprowadza także pewien stopień **niezmienniczości** w stosunku do drobnych przesunięć, co widać na rysunku 1.13. Zakładamy tu, że piksele jasne mają mniejszą wartość od pikseli ciemnych, trzy obrazy (A, B i C) przechodzą przez maksymalizującą warstwę łączącą o jądrze 2×2 i kroku równym 2. Obrazy B i C wyglądają tak samo jak obraz A, ale są przesunięte o, odpowiednio, jeden i dwa piksele w prawo. Jak widać, rezultaty wygenerowane w maksymalizującej warstwie łączącej z obrazów A i B są identyczne. Na tym polega **niezmienniczość przesunięć** (ang. translation invariance). W przypadku obrazu C wynik jest odmienny: jest on przesunięty o jeden piksel w prawo (nadal jednak pozostaje niezmieniony w mniej więcej 75%). Poprzez wstawianie maksymalizującej warstwy łączącej co kilka warstw sieci CNN możliwe jest uzyskanie ograniczonej niezmienniczości przesunięć w większej skali. Ponadto warstwa ta zapewnia niewielki stopień niezmienniczości rotacyjnej i drobną niezmienniczość skalowania. Tego typu niezmienniczość (mimo że jest

ograniczona) przydaje się wszędzie tam, gdzie prognozy nie powinny być zależne od tych szczegółów, na przykład w zadaniach klasyfikacji.



Rysunek 1.13: Niezmienność związana z drobnymi przesunięciami

Źródło: [6]

Jednak maksymalizująca warstwa łącząca jest niepozbawiona również wad. Przede wszystkim jest ona bardzo niszczycielska: nawet w przypadku niewielkiego jądra o rozmiarze 2×2 i kroku o wartości 2 dane wyjściowe będą dwukrotnie mniejsze w każdym kierunku (zatem obszar obrazu będzie zmniejszony czterokrotnie), co oznacza porzucenie 75% wartości wejściowych. Z kolei w pewnych zastosowaniach niezmienność jest niepożądana, na przykład w segmentacji semantycznej (zadaniu klasyfikowania każdego piksela obrazu zgodnie z jego przynależnością do danego obiektu): jest oczywiste, że jeżeli obraz wejściowy zostanie przesunięty o jeden piksel w prawo, to wynik również powinien być przesunięty w taki sam sposób. Wówczas celem staje się **ekwiwariancja** (ang. equivariance), a nie niezmienność: mała zmiana w sygnale wejściowym powinna prowadzić do powiązanej z nią niewielkiej zmiany w sygnale wyjściowym.

Podobnie do maksymalizującej warstwy łączącej jest zdefiniowana **uśredniająca warstwa łącząca** (average pooling layer), która zamiast maksimum używa średniej. Jest ona rzadziej wybierana w zastosowaniach niż warstwa maksymalizująca, z powodu słabszej wydajności. Obliczanie średniej zazwyczaj powoduje mniejszą utratę informacji niż obliczanie maksimum, ale za to warstwa maksymalizująca zachowuje wyłącznie najistotniejsze cechy i ignoruje te mniej ważne, dlatego kolejne warstwy otrzymują coraz czystszy sygnał.

Ostatnim rodzajem warstwy łączącej często spotykanym we współczesnych architekturach jest **globalna uśredniająca warstwa łącząca** (ang. global average pooling layer). Mechanizm jej działania jest całkiem odmienny: oblicza ona jedynie średnią każdej mapy cech (przypomina to działanie uśredniającej warstwy łączącej, w której jądro ma takie same wymiary przestrzenne jak dane wejściowe). Oznacza to, że generuje ona na wyjściu poje-

dynczą wartość na każdą mapę cech i na każdy przykład. Jest to oczywiście rozwiążanie skrajnie destrukcyjne (większość informacji zawartych w mapie cech zostaje utraconych), ale, jak przekonasz się w dalszej części rozdziału, bywa przydatne na wyjściu modelu.

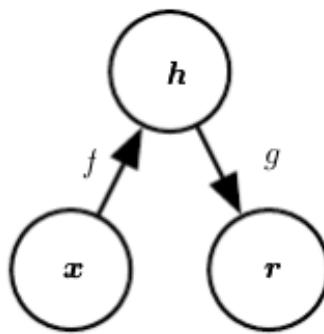
1.3. Autoenkodery

1.3.1. Czym jest autoenkoder

Autoenkoder (czasem także nazywany autokoderem, z ang. *autoencoder*, *auto-encoder*) jest rodzajem sieci neuronowej przeznaczonym głównie do kodowania danych wejściowych do skompresowanej i znaczącej reprezentacji, a następnie dekodowania ich z powrotem w taki sposób, aby zrekonstruowane dane były jak najbardziej podobne do oryginalnych [2]. Autoenkodery uczą się gęstych reprezentacji danych, tzw. **reprezentacji ukrytych** (ang. *latent representations*) lub **kodowań** (ang. *codings*) bez jakiegokolwiek formy nadzorowania (tzn. zbiór danych nie zawiera etykiet). Wyjściowe kodowania zazwyczaj mają mniejszą wymiarowość od danych wejściowych, dzięki czemu autoenkodery mogą z powodzeniem służyć do **redukcji wymiarowości**. Mają też zastosowanie w **modelach generatywnych** (ang. *generative models*), które potrafią losowo generować nowe dane przypominające zbiór uczący, choć warto zaznaczyć, że często lepszej jakości dane można uzyskać przy użyciu generatywnych sieci przeciwnych, czyli GAN (ang. *Generative Adversarial Networks*). [6]. Autoenkoder jest zatem siecią neuronową szkoloną po to, aby kopiować dane wejściowe do wyjścia. Zawiera ukrytą warstwę h , która opisuje kodowanie używane do reprezentowania wejścia. Sieć można postrzegać jako składającą się z dwóch części: kodującej funkcji $h(x)$ i dekodera, który tworzy rekonstrukcję $r = g(h)$ [7]. Ogólna struktura autoenkodera jest przedstawiona na rysunku 1.14.

Gdyby autoenkoder nauczył się po prostu, aby na wyjściu ustawać zawsze $g(f(x)) = x$, to nie byłby zbyt przydatny. Z tego powodu autoenkodery są projektowane tak, aby nie potrafiły kopiować w sposób doskonały. Zazwyczaj nakładane są ograniczenia, aby autoenkoder mógł kopiować jedynie w przybliżeniu, i tylko takie dane, które są podobne do danych ze zbioru uczącego. Dzięki temu model musi wybierać jedynie pewne aspekty danych wejściowych, które powinny być kopiowane. W ten sposób może on zdobyć użyteczne informacje o strukturze danych.

Autoenkodery można wyobrazić sobie jako specjalny przypadek sieci jednokierunkowych i można je szkolić, używając wszystkich tych samych technik, zwykle minipakietowego spadku gradientu po gradientach obliczonych przez propagację wstecz. W przeciwnieństwie do ogólnych sieci jednokierunkowych, autoenkodery można szkolić za pomocą recyrkulacji,



Rysunek 1.14: Struktura autoenkuera odwzorowującego wejście x na wyjście r (nazywane rekonstrukcją) poprzez reprezentację ukrytą (kodowanie) h . Autoenkoder składa się z dwóch składników: kodera f (odwzorowującego x na h) i dekodera g (odwzorowującego h na r)

Źródło: [7]

czyli algorytmu uczącego się na bazie porównywania aktywacji sieci na oryginalnym wejściu z aktywacjami na zrekonstruowanym wejściu [7].

1.3.2. Rodzaje autoenkoderów

Jak wspomniano wcześniej, aby autoenkoder był użyteczny, zamiast jedynie kopiować wejście do wyjścia, można na niego nałożyć różne ograniczenia, jak na przykład limit na rozmiar reprezentacji ukrytej. Ze względu na nakładane ograniczenia, wyróżniamy wiele rodzajów autoenkoderów, a wśród nich:

- autoenkodery niedopełnione (ang. *undercomplete*), w których wyjście musi mieć mniejszy wymiar niż wejście
- autoenkodery z regularyzacją (ang. *regularized*), w których wyjście ma taki sam lub większy (ang. *overcomplete*) wymiar niż wyjście, ale używają specjalnie dopasowanych funkcji straty. Wśród autoenkoderów z regularyzacją można rozróżnić na przykład:
 - autoenkodery rzadkie (ang. *sparse*), które dążą do rzadkiej reprezentacji ukrytej
 - autoenkodery odszumiające (ang. *denoising*), które na wejściu dostają zniekształcone dane, a starają się odzyskać pierwotne, niezaszumione informacje
 - autoenkodery kurczliwe (ang. *contractive*) dążące do małego rozmiaru pochodnej
- autoenkodery stosowe (ang. *stacked*) nazywane również głębkimi (ang. *deep*)
- autoenkodery splotowe (ang. *convolutional*)
- autoenkodery rekurencyjne (ang. *recurrent*)
- autoenkodery wariacyjne (ang. *variational*)
- autoenkodery przeciwwstawne (ang. *adversarial*)
- autoenkodery stochastyczne (ang. *stochastic*)

W następnych podrozdziałach zostaną przybliżone cechy charakterystyczne tych rodzajów autoenkoderów.

Autoenkodery niedopełnione

Kopiowanie wejścia do wyjścia może wydawać się bezużyteczne, ale wyjście dekodera nas nie interesuje. Mamy za to nadzieję, że skutkiem przeszkoletnia autoenkodera do kopowania będzie kod h , mający przydatne właściwości. Jednym ze sposobów, aby uzyskać przydatne cechy z autoenkodera, jest ograniczenie h do mniejszego wymiaru niż x . Autoenkoder, w którym wymiar kodu jest mniejszy niż wymiar wejściowy, jest nazywany niekompletnym (niedopełnionym). Poznawanie niekompletnych reprezentacji zmusza autoenkoder do przechwycenia najistotniejszych cech danych szkoleniowych. Proces poznawania jest opisywany po prostu jako minimalizowanie funkcji straty

$$L(x, g(f(x)))$$

gdzie L jest funkcją straty karzącą $g(f(x))$ za niepodobieństwo do x jak np. błąd średniokwadratowy.

Gdy dekoder jest liniowy, a L to błąd średniokwadratowy, niekompletny autoenkoder uczy się obejmować tą samą podprzestrzeń co PCA. W tym przypadku poznanie zasadniczej podprzestrzeni danych szkoleniowych przez szkolony do kopowania autoenkoder jest efektem ubocznym. Autoenkodery z nieliniowymi funkcjami kodowania f i nieliniowymi funkcjami dekodowania g mogą więc poznawać potężniejsze, nieliniowe uogólnienie PCA. Niestety, jeśli koder i dekoder będą mieć zbyt dużą pojemność, autoenkoder może nauczyć się wykonywać kopowanie bez wyodrębniania pożytecznych informacji o rozkładzie danych. Teoretycznie można sobie wyobrazić, że autoenkoder z jednowymiarowym kodem, ale bardzo potężnym nieliniowym koderem, może nauczyć się reprezentować każdy przykład szkoleniowy $x^{(i)}$ za pomocą kodu i . Dekoder mógłby nauczyć się odwzorowywać te całkowitoliczbowe indeksy z powrotem na wartości konkretnych przykładów szkoleniowych. Ten konkretny przykład nie występuje w praktyce, ale pokazuje wyraźnie, że autoenkoder przeszkoletny do wykonywania kopowania może zawieść, jeśli chodzi o poznanie czegoś przydatnego na temat zbioru danych, jeśli pozwoli się, aby miał za dużą pojemność [7].

Autoenkodery z regularyzacją

W przypadku, gdy wymiar wyjścia jest większy (autoenkodery nadkompletne) lub równy niż wymiar wejścia, nawet liniowy koder i dekoder mogą nauczyć się kopować wejście do wyjścia bez uczenia się niczego przydatnego na temat rozkładu danych.

Ideałem byłaby możliwość udanego szkolenia dowolnej architektury autoenkodera przy wyborze wymiaru kodu oraz pojemności kodera i dekodera na podstawie złożoności rozkładu modelowanego. Można to zrobić, stosując regularyzację. Zamiast ograniczać pojemność

modelu przez zachowywanie płytkości kodera i dekodera oraz małego rozmiaru kodu, autoenkodery z regularyzacją używają funkcji straty, dzięki której model może posiadać inne właściwości oprócz możliwości kopiowania swojego wejścia do wyjścia. Do tych właściwości należą:

- rzadkość reprezentacji
- mały rozmiar pochodnej reprezentacji
- odporność na szum lub brakujące dane wejściowe

Autoenkoder z regularyzacją może być nieliniowy i nadkompletny, a mimo to nauczyć się czegoś wartościowego o rozkładzie danych, nawet jeśli pojemność modelu jest na tyle duża, aby poznać trywialną funkcję tożsamościową [7].

Rzadkie autoenkodery

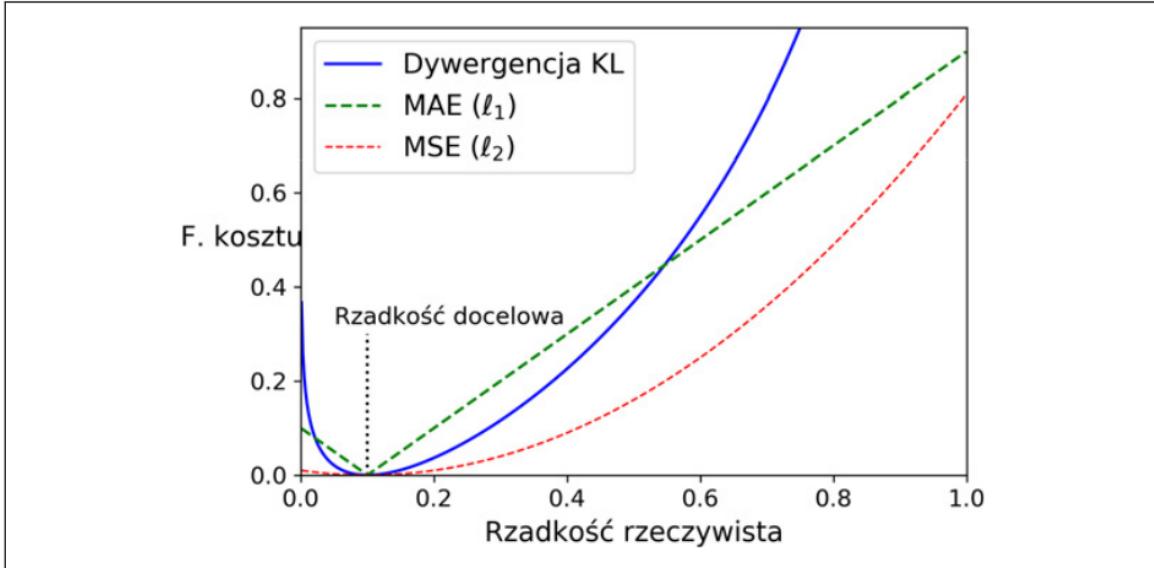
Rzadkie autoenkodery są zwykle używane do tego, aby uczyć się cech do innego zadania, takiego jak klasyfikacja. Autoenkoder, który dzięki regularyzacji jest rzadki, musi reagować na unikatowe statystyczne cechy zbioru danych, na którym został wyszkolony, a nie tylko działać jak funkcja tożsamościowa.

Kryterium szkolenia autoenkodera rzadkiego obejmuje karę rzadkości $\Omega(h)$ na warstwie kodu h oprócz błędu rekonstrukcji

$$L(x, g(f(x))) + \Omega(h)$$

gdzie $g(h)$ to wyjście dekodera, a zwykle mamy $h = f(x)$, czyli wyjście kodera [7]. Dodanie tego składnika do funkcji kosztu zmusza autoenkoder do zmniejszenia liczby aktywnych neuronów w warstwie kodowania. W ten sposób każde wejście musi być reprezentowane jako kombinacja niewielkiej liczby pobudzeń. Dzięki temu każdy neuron warstwy kodowania zazwyczaj uczy się wykrywać jakąś przydatną cechę [6].

W każdym przebiegu uczenia następuje pomiar rzeczywistej rzadkości warstwy kodowania i karanie modelu, gdy zmierzona rzadkość różni się od docelowej. W tym celu obliczania jest średnia aktywacja każdego neuronu w warstwie dla całej grupy przykładów uczących. Rozmiar tej grupy nie może być zbyt mały, aby wyliczona wartość średniej była dokładna. Następnie nakładana jest kara na zbyt aktywne neurony, poprzez dodanie funkcji straty rzadkości (ang. *sparsity loss*) $\Omega(h)$ do funkcji kosztu. Dla przykładu, jeśli średnia wartość aktywacji neuronu to 0.3, ale docelowo powinna wynosić 0.1, musimy ją zmniejszyć. Jednym ze sposobów jest dodanie kwadratu błędu $(0.3 - 0.1)^2$ do funkcji kosztu. Lepszym rozwiązaniem w praktyce jest zastosowanie dywergencji Kullbacka-Leiblera, której gradienty są znacznie większe niż w błędzie średniokwadratowym (rysunek 1.15).

**Rysunek 1.15:** Funkcje straty rzadkości*Źródło:* [6]

Mając dwa dyskretne rozkłady prawdopodobieństwa P i Q , możemy obliczyć rozbieżność pomiędzy nimi $D_{KL}(P||Q)$ za pomocą dywergencji Kullbacka-Leiblera:

$$D_{KL}(P||Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$

W przypadku autoenkodera rzadkiego naszym celem jest zmierzenie rozbieżności pomiędzy docelowym prawdopodobieństwem p aktywacji neuronu w warstwie kodowania, a rzeczywistym prawdopodobieństwem q (które jest średnią aktywacją dla danych uczących). Dywergencję KL można wówczas zapisać jako:

$$D_{KL}(p||q) = p \log \frac{p}{q} + (1-p) \log \frac{1-p}{1-q}$$

Po obliczeniu funkcji straty rzadkości dla każdego neuronu w warstwie kodowania, należy je zsumować i wynik dodać do funkcji kosztu. W celu regulowania względnej istotności funkcji straty rzadkości i funkcji straty rekonstrukcji, można tą pierwszą pomnożyć przez hiperparametr wagi rzadkości. Jeśli wartość tego hiperparametru będzie zbyt duża, to model pozostanie blisko rzadkości docelowej, ale jednocześnie nie będzie w stanie prawidłowo rekonstruować danych wejściowych. Przy zbyt małej wartości tego parametru, model będzie ignorował cel rzadkości [6].

Autoenkodery z odszumianiem

Zamiast dodawać karę Ω do funkcji kosztów, możemy uzyskać autoenkoder, który uczy się czegoś przydatnego, zmieniając składnik błędu rekonstrukcji w funkcji kosztów. Tradycyjne autoenkodery minimalizują jakąś funkcję

$$L(x, g(f(x)))$$

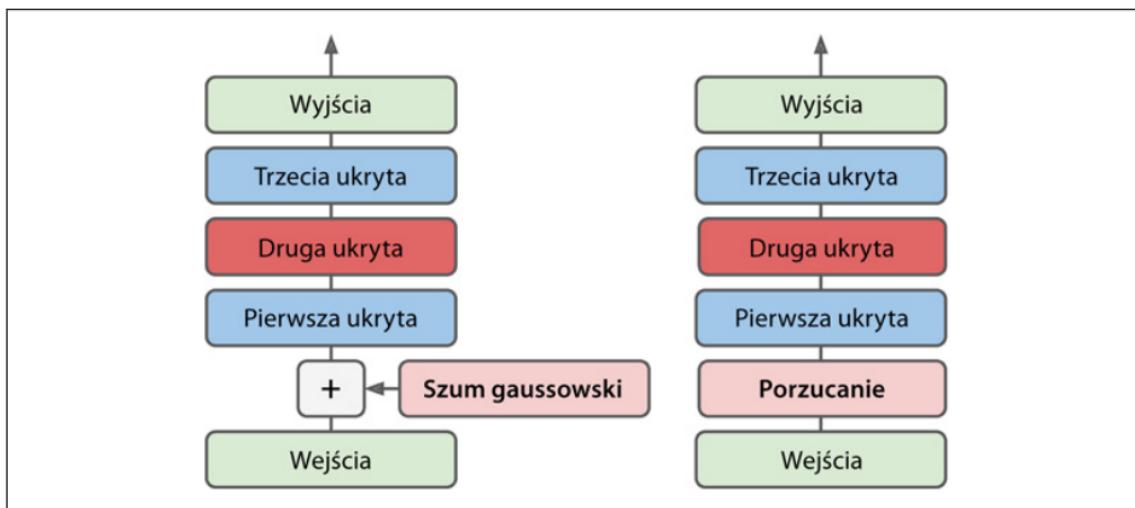
gdzie L to funkcja straty karząca $g(f(x))$ za niepodobieństwo do x , jak np. norma L^2 ich różnicy. Sprzyja to temu, aby $g \circ f$ uczyła się być jedynie funkcją tożsamościową, jeśli ma do tego odpowiednią pojemność.

Autoenkoder z odszumianiem zamiast tego minimalizuje

$$L(x, g(f(\tilde{x})))$$

gdzie \tilde{x} to kopia x , która została znieksztalcona przez jakiegoś rodzaju postać szumu. Autoenkodery mają tym samym za zadanie odwrócić to znieksztalconie, a nie po prostu przekopować swoje wejście [7].

Znieksztalconie może być szumem gaussowskim dodawanym do danych wejściowych lub może przybrać postać losowo wyłączanych wejść za pomocą metody porzucania. Autoenkodery z takimi znieksztalconiami zostały przedstawione na rysunku 1.16.

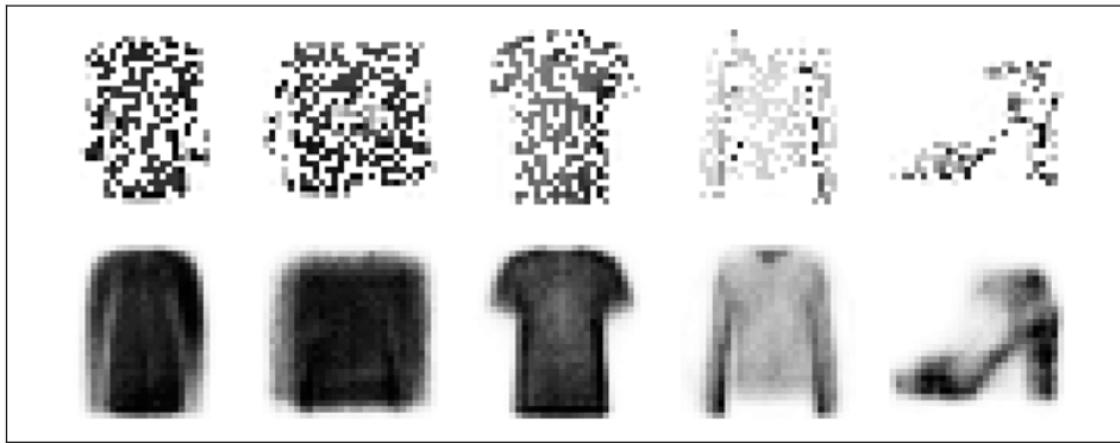


Rysunek 1.16: Autokodery odszumiające: wykorzystujące szum gaussowski (po lewej) lub metodę porzucania (po prawej)

Źródło: [6]

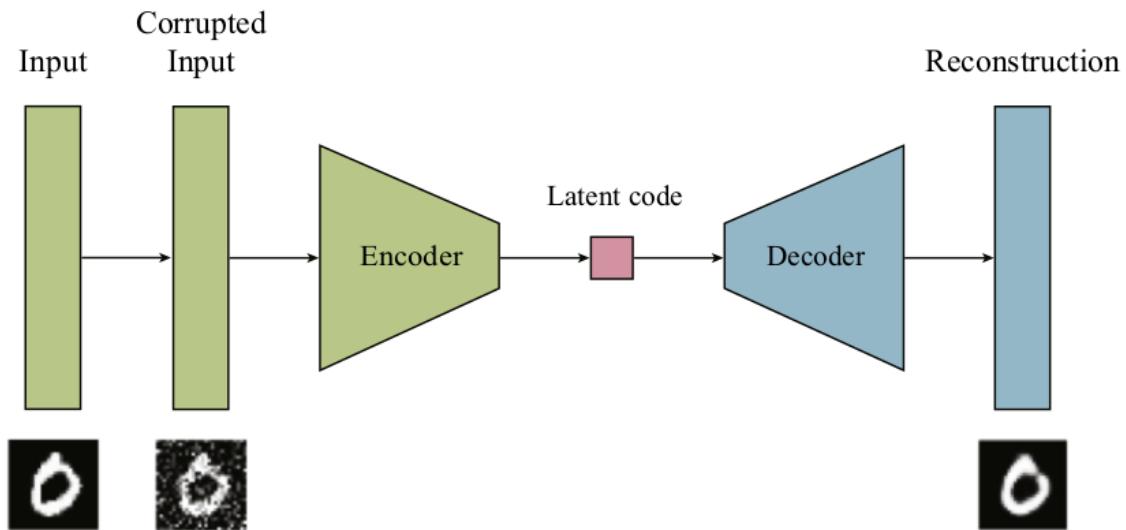
Rysunek 1.17 przedstawia przykłady zaszumionych obrazów (połowa pikseli została „wyłączona”), a także ich rekonstrukcje uzyskane za pomocą autokodera odszumiającego (bazującego na warstwie porzucania). Autokoder „odgaduje” szczegóły niewystępujące w obrazach wejściowych, na przykład górną część białej sukienki (czwarty obraz w dolnym rzędzie).

Ogólna struktura autoenkodera odszumiającego została przedstawiona na rysunku 1.18. Wejściem jest zanieczyszczony obraz, a docelową wartością jest oryginał bez znieksztalconia. Sieć uczy się rozpoznawać i usuwać znieksztalconia, aby wygenerować rekonstrukcję. Autoenkoder nnie widzi oryginalnego, czystego obrazu, jedynie ten ze znieksztalconiem.



Rysunek 1.17: Zaszumione obrazy (na górze) i ich rekonstrukcje (na dole)

Źródło: [6]



Rysunek 1.18: Struktura autoenkodera odszumiającego

Regularyzacja poprzez karanie pochodnych

kolejną strategią regularyzacji autoenkodera jest użycie kary Ω tak, jak w rzadkich autoenkoderach

$$L(x, g(f(x))) + \Omega(h, x)$$

ale z inną postacią Ω :

$$\Omega(h, x) = \lambda \sum_i ||\nabla_x h_i||^2$$

Zmusza to model do poznania funkcji, która nie zmienia się za bardzo przy niewielkich zmianach x . Ponieważ kara ta jest stosowana tylko w przykładach szkoleniowych, zmusza autoenkoder do poznawania cech, które przechwytyują informacje o rozkładzie szkoleniowym.

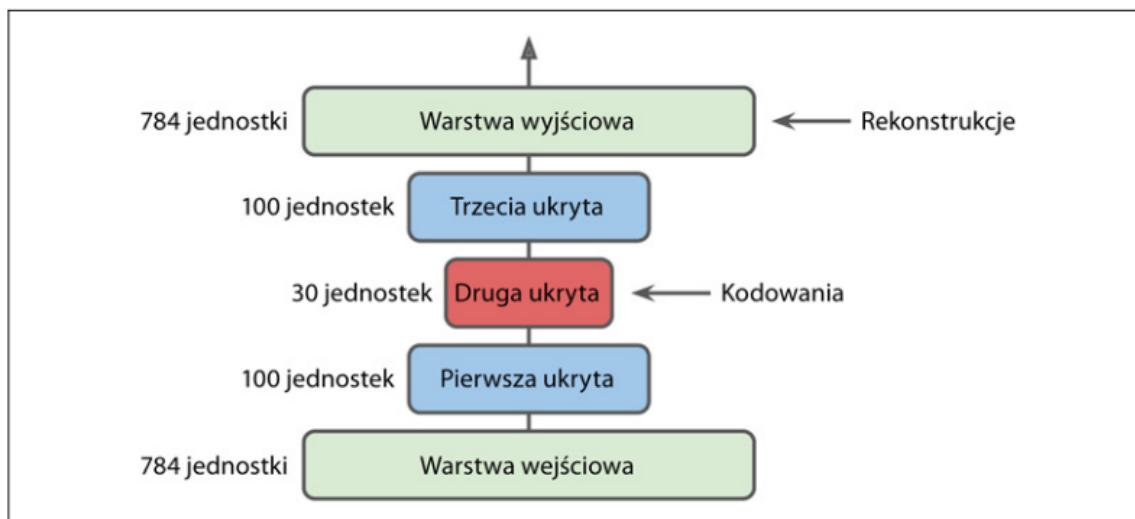
Autoenkoder z taką regularyzacją jest nazywany **kurczliwym** [7].

Autoenkodery stosowe

Podobnie jak inne sieci neuronowe, również autoenkodery mogą mieć wiele warstw ukrytych. Takie autoenkodery są nazywane **stosowymi** (ang. *stacked*) lub **głębokimi** (ang. *deep*). Dzięki wielu warstwom ukrytym, autoenkoder może się uczyć bardziej skomplikowanych kodowań. Pojawia się jednak niebezpieczeństwo stworzenia modelu zbyt potężnego, podobnie jak zostało to opisane przy autoenkoderach niedopełnionych, gdzie podano teoretyczny przykład autoenkodera przekształcającego i -ty przykład uczący na pojedynczą liczbę.

Przykładowa struktura autoenkodera stosowego została przedstawiona na rysunku 1.19. Zazwyczaj architektura autoenkodera stosowego jest symetryczna względem jego centralnej warstwy ukrytej, tak jak widać na rysunku. W przypadku symetrycznych autoenkoderów, często stosowane jest **wiązanie wag** (ang. *tying weights*) warstw kodera z wagami warstw dekodera. W ten sposób liczba wag w modelu zostaje zredukowana o połowę, co przyspiesza proces uczenia i zmniejsza ryzyko przetrenowania modelu. Jeśli autoenkoder zawiera N warstw (oprócz warstwy wejściowej), a W_L oznacza wagę połączeń w L -tej warstwie, to wagi warstwy dekodera można zdefiniować jako $W_{N-L+1} = W_L^T$ (dla $L = 1, 2, \dots, N/2$).

W celu zaoszczędzenia czasu można zamiast uczyć cały autoenkoder stosowy na raz, trenować pojedyncze składowe osobno, a na koniec połączyć je w całość. To rozwiązanie nazywane jest „zachłannym uczeniem warstwowym”. Obecnie jest rzadko stosowane [6].



Rysunek 1.19: Przykładowa struktura autoenkodera stosowego

Źródło: [6]

Autoenkoder splotowy

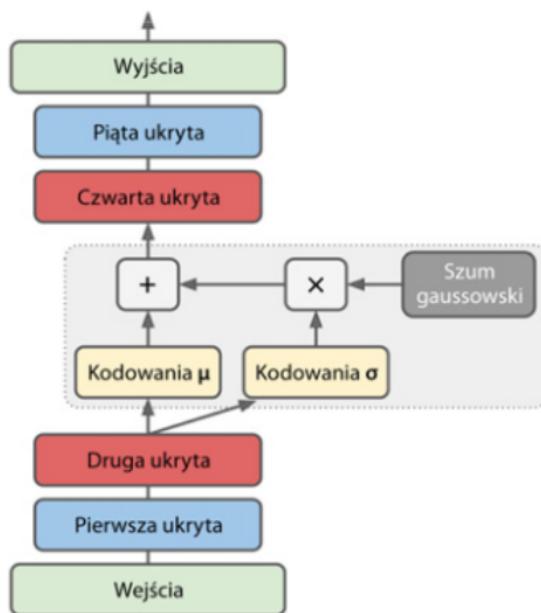
Autoenkodery splotowe są typem autoenkoderów, który najlepiej nadaje się do przetwarzania obrazów. W takim rodzaju autoenkodera, koderem jest sieć konwolucyjna składająca się z warstw splotowych i redukujących. Zwykle zmniejsza ona wymiarowość obrazu (wysokość i szerokość), a zwiększa głębokość (liczbę map cech). Dekoder przeprowadza operację odwrotną: musi zwiększyć rozdzielcość i zredukować głębokość do pierwotnego wymiaru. W tym celu można wykorzystać transponowane warstwy splotowe [6].

Autoenkoder rekurencyjny

Autoenkodery rekurencyjne zwykle służą do przetwarzania danych sekwencyjnych, takich jak szeregi czasowe czy tekst. W przypadku takiego autoenkodera, koderem zwykle jest sieć sekwencyjno-wektorowa która kompresuje sekwencję wejściową do pojedynczego wektora. Dekoderem jest sieć wektorowo-sekwencyjna odwracająca tą operację [6].

Autoenkoder wariancyjny

Autoenkodery wariancyjne istotnie różnią się od opisywanych wcześniej rodzajów autoenkoderów. Są one autoenkoderami probabilistycznymi, czyli generują częściowo losowe wyniki. Stanowią one klasę modeli generatywnych, co oznacza, że są w stanie tworzyć nowe dane przypominające te ze zbioru uczącego.



Rysunek 1.20: Przykładowa struktura autoenkodera wariancyjnego

Źródło: [6]

Na rysunku 1.20 przedstawiona jest przykładowa struktura autoenkodera wariancyjnego. Można zauważyć tutaj elementy podstawowej architektury autoenkoderów: koder i de-

koder składają się z dwóch warstw ukrytych. Mamy też do czynienia z pewną modyfikacją: koder nie generuje bezpośredniego kodowania próbki wejściowej, lecz **uśrednione kodowanie** μ oraz odchylenie standardowe σ . Rzeczywiste kodowanie jest następnie losowane z rozkładu normalnego o parametrach właśnie μ i σ . Następnie dekoder w standardowy sposób dekoduje wylosowane kodowanie. W czasie uczenia funkcja kosztu zmusza kodowania do stopniowego poruszania się po przestrzeni kodowania (nazywanej również przestrzenią ukrytą, z angielskiego *latent space*) w poszukiwaniu miejsca wewnątrz obszaru przypominającego chmurę punktów gaussowskich [6].

Funkcja kosztu autoenkodera wariancyjnego składa się z dwóch członów. Pierwszy jest tradycyjną funkcją straty rekonstrukcji, która zmusza autoenkoder do rekonstruowania danych wejściowych. Drugi element nazywany jest **funkcją straty ukrytej** (ang. *latent loss*). Sprawia on, że autoenkoder uzyskuje reprezentacje przypominające te uzyskiwanie z rozkładu normalnego. Stosujemy w tym celu dywergencję Kullbacka-Leiblera pomiędzy docelowym rozkładem (normalnym) a rzeczywistym rozkładem kodowań. Funkcja straty ukrytej wygląda wówczas następująco:

$$-\frac{1}{2} \sum_{i=1}^n [1 + \log(\sigma_i^2) - \sigma_i^2 - \mu_i^2],$$

gdzie n - wymiarowość kodowań, μ_i i σ_i to średnia i odchylenie standardowe i -tej składowej kodowania. Koder generuje na wyjściu wektory μ i σ , przechowujące wszystkie wartości μ_i i σ_i .

Często spotykaną modyfikacją jest zastąpienie σ na wyjściu kodera przez $\gamma = \log(\sigma^2)$. Rozwiążanie to jest stabilniejsze numerycznie i przyspiesza proces uczenia [6]. Funkcja straty ukrytej ma wówczas postać:

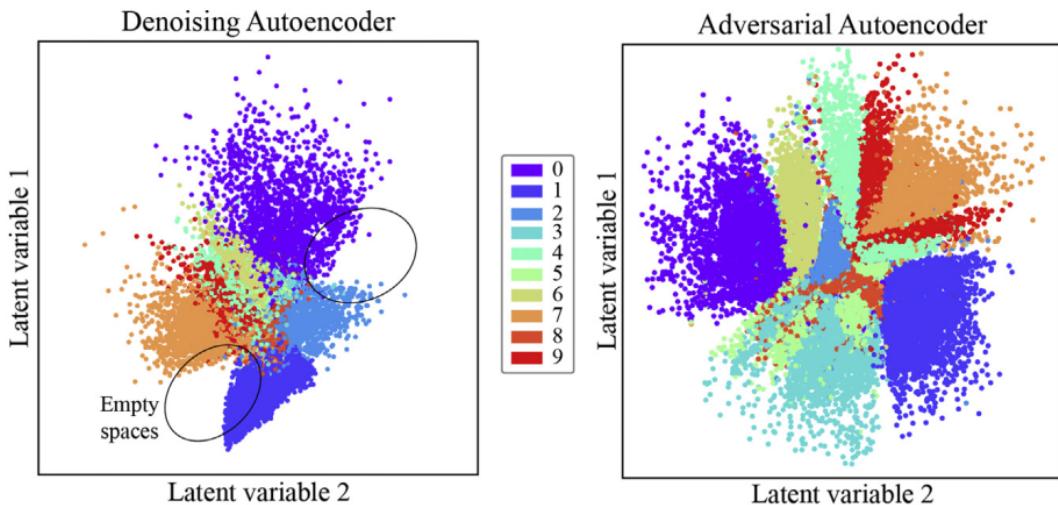
$$-\frac{1}{2} \sum_{i=1}^n [1 + \gamma_i - \exp(\gamma_i) - \mu_i^2].$$

Warto zauważyć, że po wytrenowaniu autoenkodera wariancyjnego generowanie nowych próbek jest bardzo łatwe - wystarczy wylosować kodowanie z rozkładu normalnego, a następnie rozkodować je przy użyciu dekodera [6].

Autoenkoder przeciwnostawny

Ograniczeniem wielu rodzajów autoenkoderów jest to, że nie generują one dobrze ustrukturyzowanej przestrzeni ukrytej. Dzieje się tak dlatego, że podczas uczenia, różne obserwacje ze zbioru zostają zakodowane w sposób losowo rozproszony w przestrzeni ukrytej. W rezultacie rozmieszczenie reprezentacji ukrytych czasami zawiera puste miejsca w przestrzeni ukrytej, co widać na rysunku 1.21. W tych pustych miejscach znajdują się kodowania h , których dekoder nigdy nie uczył się rekonstruować. Dlatego też proces generowania nowych danych z losowo wybranego kodowania h może być trudny. W praktyce rekonstrukcja

próbek z tych pustych miejsc zwykle jest błędna. Ogranicza ona zastosowanie niektórych rodzajów autoenkoderów jako modeli generatywnych [14].

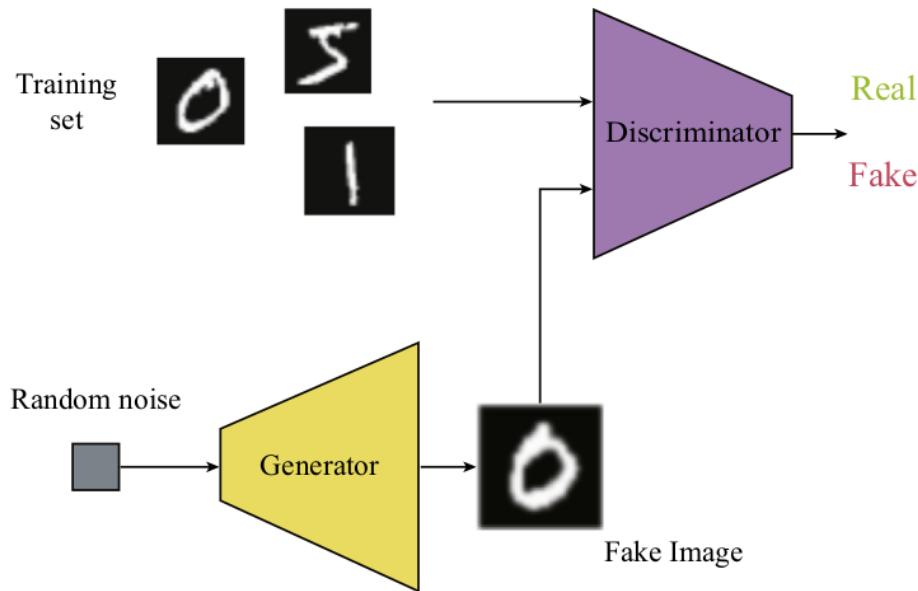


Rysunek 1.21: Reprezentacje ukryte. W tym przykładzie, obrazy cyfr od 0 do 9 były użyte do trenowania autoenkodera odszumiającego i przeciwnego. W obu przypadkach dane uczące są przedstawione w dwuwymiarowej przestrzeni ukrytej. Ponieważ nie nakładamy żadnych ograniczeń na autoenekoder odszumiający, to jego przestrzeń ukryta zawiera puste miejsca. Z drugiej strony, autoenekoder przeciwny ogranicza reprezentacje do podobnych według rozkładu (w tym przypadku dwuwymiarowego normalnego), co w rezultacie oznacza brak pustych miejsc

Źródło: [14]

Jedną z metod pozwalających na kontrolowanie struktury przestrzeni ukrytej jest stosowanie autoenkodera przeciwnego. Ten rodzaj autoenkodera jest połączeniem ogólnego modelu autoenkodera z podejściem trenowania przeciwnego, typowego dla generatywnych sieci przeciwnych (GAN). Generatywne sieci przeciwnie składają się z dwóch sieci neuronowych, które konkurują ze sobą, aby polepszyć swoje działanie (rysunek 1.22). Pierwsza sieć to generator, który na wejściu dostaje losowe wartości i stara się wygenerować sztuczne dane, możliwe jak najbardziej podobne do prawdziwych danych. Drugą siecią jest dyskryminator, który decyduje, czy jego wejście pochodzi z generatora czy z prawdziwego zbioru. Celem generatora jest „oszukać” dyskryminatora. Obie sieci są trenowane równocześnie, a generator nie ma bezpośredniego dostępu do prawdziwego zbioru danych. Jedynym jego sposobem na poprawę działania jest interakcja z dyskryminatorem. Na podstawie informacji zwrotnych z dyskryminatora, generator stara się dopasować swoją sieć tak, by produkować wyjścia będące dla dyskryminatora trudne do zidentyfikowania jako fałszywe, czyli bardziej podobne do realnych danych.

Autoenkodery przeciwnie wykorzystują trenowanie przeciwnie, by ukształtować rozkład danych wejściowych tak, aby był jak najbardziej podobny do predefiniowanego



Rysunek 1.22: Struktura generatywnej sieci przeciwnostawnej (GAN)

Źródło: [14]

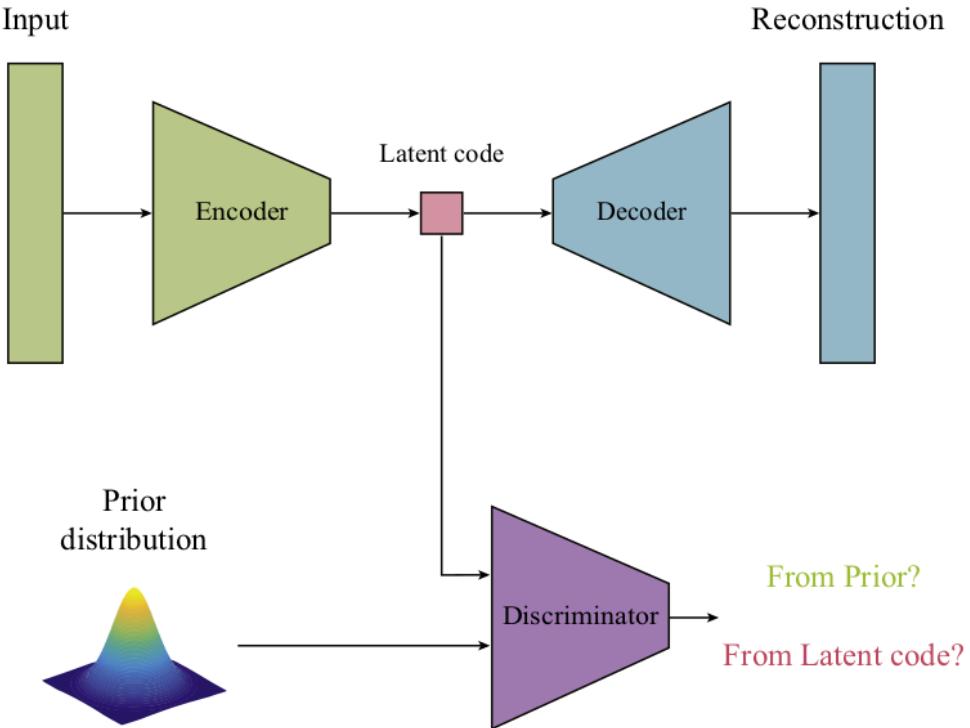
rozkładu. Jest to osiągane poprzez dodanie sieci dyskryminatora do struktury autoenkodera (rysunek 1.23). W przypadku autoenkoderów przeciwnostawnych, dyskryminator otrzymuje dwa rodzaje danych wejściowych: wartości próbkiowane z pożądanego rozkładu (przykładowo losowe wartości z rozkładu normalnego) oraz ukryte reprezentacje h obserwacji ze zbioru uczącego. Co ważne, oba muszą mieć takie same wymiary. Pożądany rozkład może być tu traktowany jako rozkład a priori. Podczas procesu uczenia, dyskryminator dokonuje klasyfikacji dotyczącej tego, czy dane pochodzą z rozkładu a priori czy z kodowań ukrytych. Gdy w strukturze znajduje się dyskryminator, enkoder jest zmuszony robić jednocześnie dwie rzeczy:

- (i) generować przestrzeń ukrytą, na której może pracować dekoder w celu stworzenia rekonstrukcji wejść
- (ii) generować przestrzeń ukrytą, która oszuka dyskryminator tak, aby klasyfikował kodowania jako próbki z rozkładu a priori

Innymi słowy, enkoder pełni także rolę generatora.

Trenowanie autoenkodera przeciwnostawnego ma trzy kroki w każdej epoce:

1. Autoenkoder aktualizuje parametry enkodera i dekodera, na podstawie standardowej funkcji straty rekonstrukcji
2. Model aktualizuje parametry sieci dyskryminującej, aby rozróżnić „prawdziwe” próbki (generowane z rozkładu a priori) od wygenerowanych próbek (ukrytych reprezentacji z enkodera)



Rysunek 1.23: Struktura autokodera przeciwnego. Sieć dyskryminująca jest dodana do autoenkodera, aby zmusić go do generowania reprezentacji ukrytych podobnych do rozkładu a priori

Źródło: [14]

3. Enkoder na podstawie informacji zwrotnej z dyskryminatora aktualizuje swoje parametry, aby poprawić generowane reprezentacje ukryte, w celu „oszukania” dyskryminatora.

Jeśli proces uczenia się powiedzie, enkoder nauczy się przekształcić rozkład danych wejściowych do rozkładu a priori, dyskryminator nigdy ni ebędzie pewny, czy dane wejściowe są prawdziwe czy nie, a dekoder uczy się modelu generatywnego, który odwzorowuje narzucony rozkład a priori na rozkład danych wejściowych. Ostatecznie jest możliwe generowanie nowych danych poprzez próbkowanie z rozkładu a priori, przekazanie go do dekodera i pozwolenie mu na wygenerowanie danych.

1.3.3. Zastosowania autoenkoderów

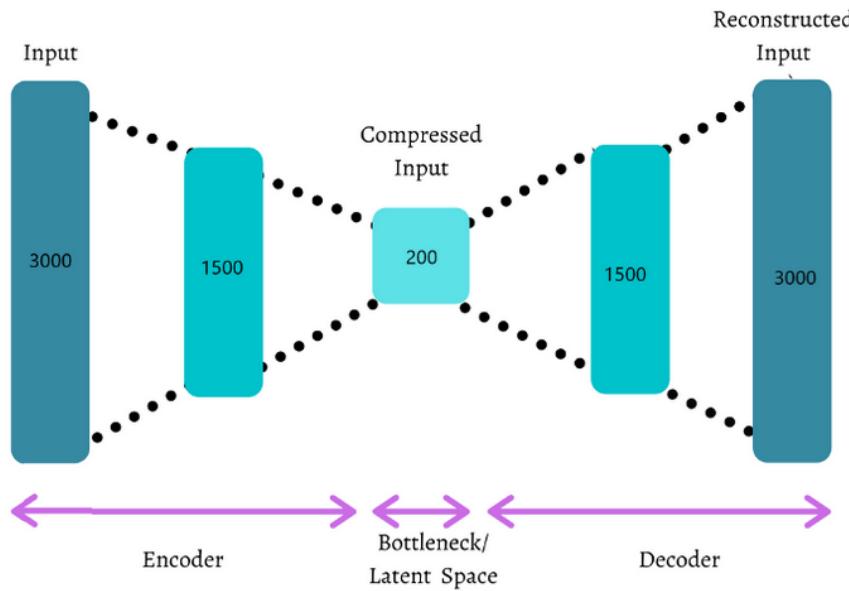
W zadaniach klasyfikacji lub regresji autoenkodery mogą być wykorzystywane do wyodrębniania cech z surowych danych w celu zwiększenia odporności modelu [11]. Istnieje wiele innych zastosowań sieci autoenkoderów, które można wykorzystać w różnym kontekście. Omówimy krótko zastosowania takie jak:

1. Zmniejszenie wymiarowości (ang. *Dimensionality Reduction*)
2. Wyodrębnianie cech (ang. *Feature Extraction*)
3. Odszumianie obrazu (ang. *Image Denoising*)

4. Kompresja obrazu (ang. *Image Compression*)
5. Wyszukiwanie obrazu (ang. *Image Search*)
6. Wykrywanie anomalii (ang. *Anomaly Detection*)
7. Uzupełnianie brakujących danych (*Missing Value Imputation*)

Zmniejszenie wymiarowości

Autoenkodery trenują sieć w celu wyjaśnienia naturalnej struktury danych w efektywnej reprezentacji niskowymiarowej. Osiaga się to poprzez zastosowanie strategii dekodowania i kodowania w celu zminimalizowania błędu rekonstrukcji [11].



Rysunek 1.24: Autodenkoder w zastosowaniu do redukcji wymiarowości

Źródło: [11]

Jak widać na rysunku 1.24, autoenkoder składa się z trzech elementów:

- koder - funkcja służąca do kompresji danych do ich reprezentacji w niższym wymiarze.
- wąskie gardło (ang. *bottleneck*) - nazywane również przestrzenią ukrytą, gdzie nasze początkowe dane są reprezentowane w niższym wymiarze.
- dekoder - funkcja dekompresji lub rekonstrukcji danych o niskim wymiarze z powrotem do wymiaru początkowego.

W przykładzie na rysunku 1.24 warstwa wejściowa i warstwa wyjściowa mają wymiar 3000, a pożądanym wymiar zredukowany wynosi 200. Możemy stworzyć sieć pięciowarstwową, w której koder ma 3000 i 1500 neuronów, podobnie jak w przypadku sieci dekodera. Reprezentacje ukryte można traktować jako reprezentację o zredukowanym wymiarze.

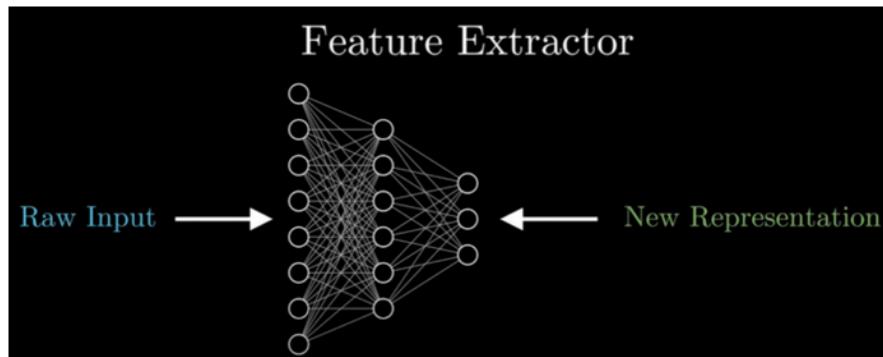
Popularną metodą redukowania wymiarowości jest analiza składowych głównych. Porównamy tą metodę ze stosowaniem autoenkodów [12]:

- PCA jest liniową transformacją danych, podczas gdy autoenkodery mogą być liniowe lub nieliniowe
- PCA jest szybsze niż autodenkody, które są trenowane przy użyciu algorytmu spadku gradientu
- PCA gwarantuje ortogonalność wynikowej przestrzeni zmiennych, autoenkoder dąży jedynie do osiągnięcia jak najmniejszej wartości funkcji straty
- autodenkody są w stanie modelować bardziej złożone lub nieliniowe zależności, podczas gdy PCA jest prostym przekształceniem liniowym

- przy decyzji między użyciem PCA i autodenkodera, warto wybierać PCA dla mniejszych zbiorów danych, a autodenkodery dla większych
- PCA ma tylko jeden hiperparametr - liczba ortogonalnych wymiarów, które chcemy użyć. W przypadku autodenkoera występują wszystkie hiperparametry architektury sieci neuronowej
- autoenkoder z jedną warstwą i liniową funkcją aktywacji ma działanie podobne do PCA. Autoenkodery z wieloma warstwami i nieliniowymi funkcjami aktywacji (Głęboki Autoenkoder) są skłonne do przeuczenia, mogą być poprawiane przez regularyzację i odpowiednie zaprojektowanie sieci neuronowej

Wyodrębnianie cech

Autokodery mogą być używane jako ekstraktory cech w zadaniach klasyfikacji lub regresji. Autokodery pobierają nieoznakowane dane i uczą się efektywnych kodowań struktury danych, które mogą być wykorzystane w zadaniach uczenia nadzorowanego. Po wytrenowaniu sieci autoenkodera na próbce danych treningowych można zignorować dekoder, a jedynie użyć kodera do przekształcenia surowych danych wejściowych o wyższym wymiarze na przestrzeń zakodowaną o niższym wymiarze (rysunek 1.25). Ten niższy wymiar danych może być wykorzystany jako cecha w zadaniach uczenia nadzorowanego [11].

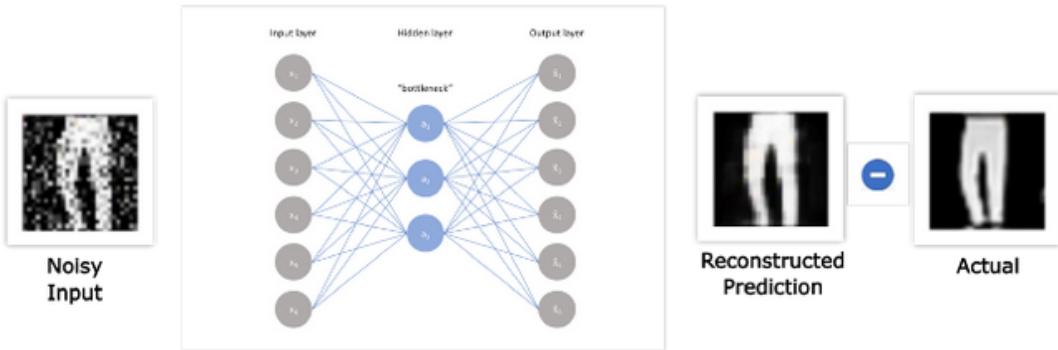


Rysunek 1.25: Autoenkoder stosowany do ekstrakcji cech

Źródło: [11]

Odszumianie obrazów

Surowe dane wejściowe ze świata rzeczywistego są często zaszumione, a wytrenowanie dobrego modelu nadzorowanego wymaga danych oczyszczonych i pozabawionych szumu. Do odszumiania danych można wykorzystać autoenkodery. Jednym z popularnych zastosowań jest odszumianie obrazów, w którym autoenkodery próbują zrekonstruować obraz pozabawiony szumu z zaszumionego obrazu wejściowego [11].



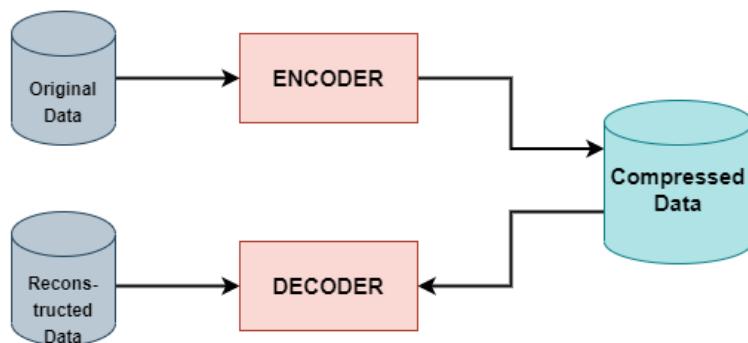
Rysunek 1.26: Stosowanie autoenkodera do odszumiania obrazu

Źródło: [11]

Zakłócony obraz wejściowy jest podawany do autoenkodera jako wejście, a bezszumowe wyjście jest rekonstruowane przez minimalizację straty rekonstrukcji w stosunku do oryginalnego wyjścia docelowego (bezszumowego). Po wytrenowaniu wag autoenkodera można je dalej wykorzystać do odszumiania obrazu surowego. Schemat działania autoenkodera odszumiającego widać na rysunku 1.26.

Kompresja obrazu

Innym zastosowaniem sieci autoenkodów jest kompresja obrazów. Surowy obraz wejściowy można przekazać do sieci kodera i uzyskać skompresowany wymiar zakodowanych danych. Wagi sieci autoenkodera mogą być uczone przez rekonstrukcję obrazu ze skompresowanego kodowania za pomocą sieci dekodera [11]. Schemat takiego zastosowania jest pokazany na rysunku 1.27.



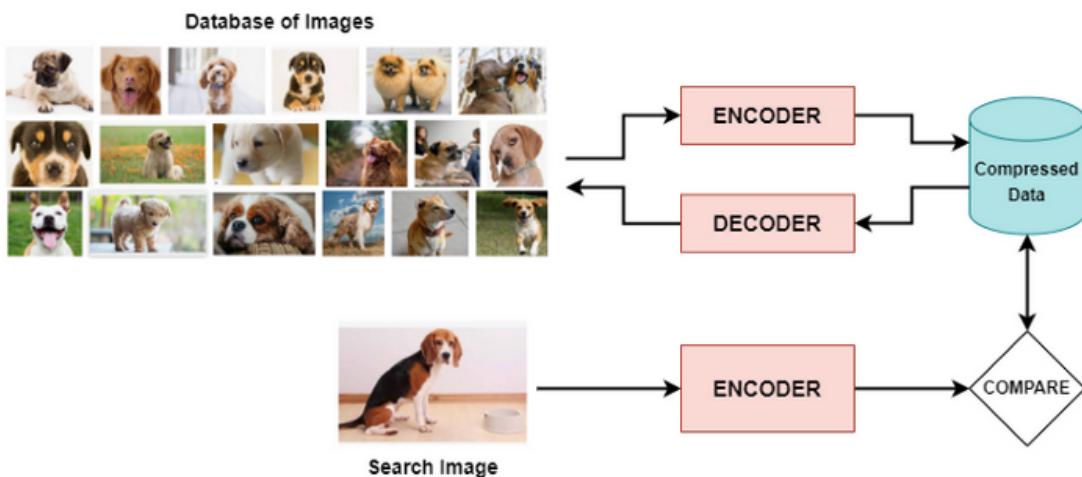
Rysunek 1.27: Zastosowanie autoenkodera do kompresji obrazu

Źródło: [11]

Zazwyczaj autoenkodery nie nadają się zbyt dobrze do kompresji danych, lepiej sprawdzają się raczej podstawowe algorytmy kompresji.

Wyszukiwanie obrazu

Do kompresji bazy danych obrazów można użyć autokoderów. Skompresowana reprezentacja może być przy przeszukiwaniu porównywana z zakodowaną wersją szukanego obrazu [11].



Rysunek 1.28: Zastosowanie autoenkodera do wyszukiwania obrazu

Źródło: [11]

Wykrywanie anomalii

Innym użytecznym zastosowaniem sieci autoenkodów jest wykrywanie anomalii. Model wykrywania anomalii można wykorzystać do wykrywania oszukańczych transakcji lub wszelkich zadań nadzorowanych o wysokim stopniu nierównowagi.

Idea polega na trenowaniu autoenkodów tylko na próbkach danych jednej klasy (klasy większościowej). W ten sposób sieć jest w stanie zrekonstruować dane wejściowe z dobrą lub mniejszą stratą rekonstrukcji. Jeśli przez sieć autoenkodera przepuści się próbę danych innej klasy docelowej, spowoduje to porównywalnie większą stratę rekonstrukcji. Można określić wartość progową straty rekonstrukcji (wynik anomalii), której przekroczenie będzie uznawane za anomalię [11].

Uzupełnianie brakujących danych

Do imputacji brakujących wartości w zbiorze danych można wykorzystać autoenkodery odszumiające. Idea polega na trenowaniu sieci autoenkodów poprzez losowe umieszczenie brakujących wartości w danych wejściowych i próbę odtworzenia oryginalnych danych surowych poprzez minimalizację straty rekonstrukcji.

Po wytrenowaniu wag autoenkodera rekordy zawierające brakujące wartości mogą być przepuszczane przez sieć autoenkodera w celu zrekonstruowania danych wejściowych, także z imputowanymi brakującymi cechami [11].

Rozdział 2

Część praktyczna

Zbiorem, na przykładzie którego będziemy pokazywać zastosowania autoenkoderów, jest zbiór danych MNIST. Przykładowe dane z części treningowej tego zbioru są przedstawione na rysunku 2.1. W zbiorze uczącym znajduje się 60 tysięcy obserwacji, a w testowym 10 tysięcy.

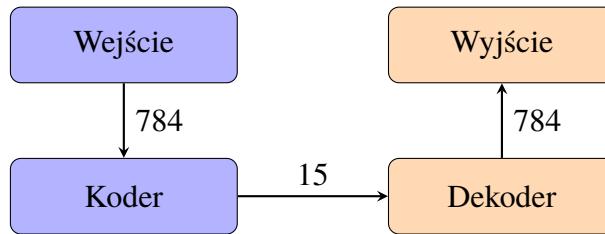


Rysunek 2.1: Pięć pierwszych obserwacji ze zbioru cyfr MNIST

Źródło: Opracowanie własne

2.1. Prosty autoenkoder

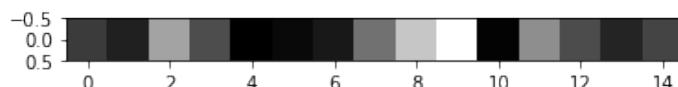
Pierwszym przykładem jest tak zwany prosty autoenkoder - nazywany tak w odróżnieniu od autoenkodera splotowego. Składa się on z kodera i dekodera, z których każdy jest pojedynczą warstwą gęstą. Warstwa wyjściowa będzie przyjmować obrazy „spłaszczone” do wektora o długości 784 (obrazy 2D były wymiaru 28×28). Reprezentacja ukryta ma wymiar 15, jest to więc autoenkoder niedopełniony (kodowania mają mniejszy wymiar niż dane wejściowe). Następnie dekoder przywraca wejściowy wymiar 784. Struktura użytego autoenkodera jest przedstawiona na rysunku 2.2.



Rysunek 2.2: Struktura autoenkodera prostego oraz wymiary kolejnych warstw

Źródło: Opracowanie własne

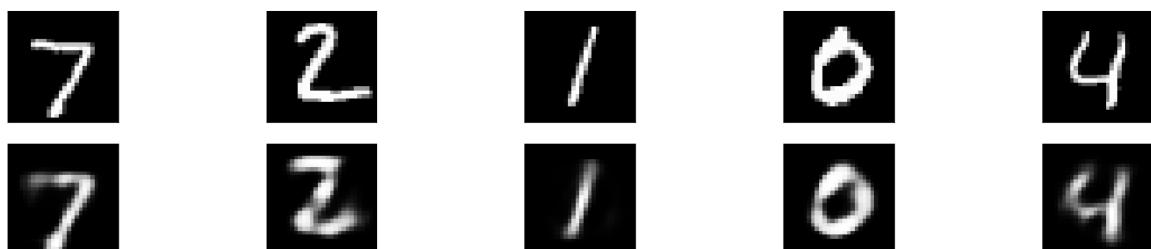
Model autoenkodera jest uczyony przez 15 epok. Rozmiar pakietu (*batch size*) wynosi 256. Zbiorem walidacyjnym jest zbiór testowy.



Rysunek 2.3: Reprezentacja 15-wymiarowa dla pierwszej obserwacji ze zbioru testowego MNIST

Źródło: Opracowanie własne

Rysunek 2.3 przedstawia kodowanie w ukrytej przestrzeni 15-wymiarowej dla pierwszej obserwacji ze zbioru testowego. Rysunek 2.4 przedstawia pięć pierwszych obrazów ze zbioru testowego (górny rząd) oraz ich rekonstrukcje po odkodowaniu przez dekoder (dolny rząd). Można zauważyć, że po przejściu przez autoenkoder niedopełniony, obrazy znacznie tracą na jakości.

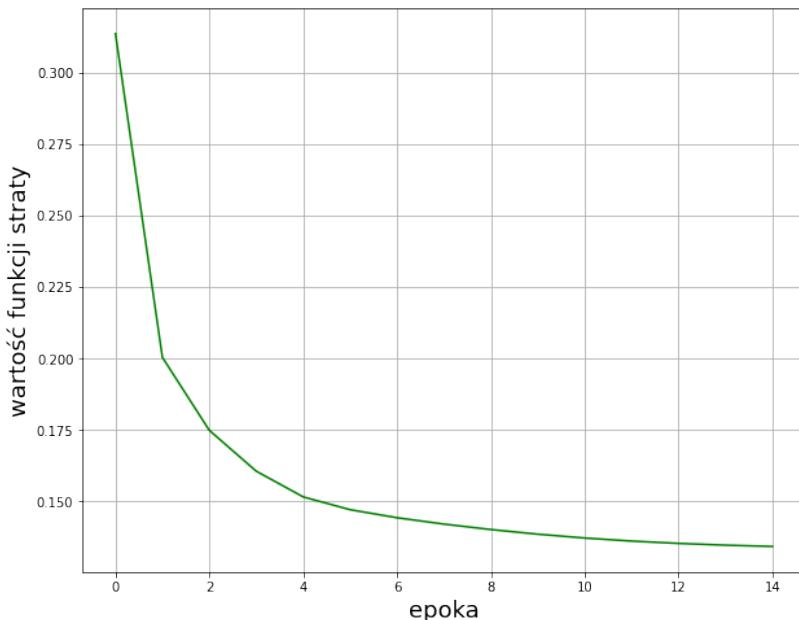


Rysunek 2.4: Pięć pierwszych obserwacji ze zbioru testowego MNIST: w górnym rzędzie

oryginalne obrazy, w dolnym rekonstrukcje z autoenkodera niedopełnionego

Źródło: Opracowanie własne

Rysunek 2.5 przedstawia wartość funkcji straty w kolejnych epokach uczenia autoenkodera prostego. W pierwszej epoce wartość funkcji straty wynosiła około 0.3. Widać, że największy spadek wartości funkcji straty następował w pierwszych kilku iteracjach. Po około 10 epoce wartość funkcji straty utrzymywała się na podobnym poziomie (około 0.13).



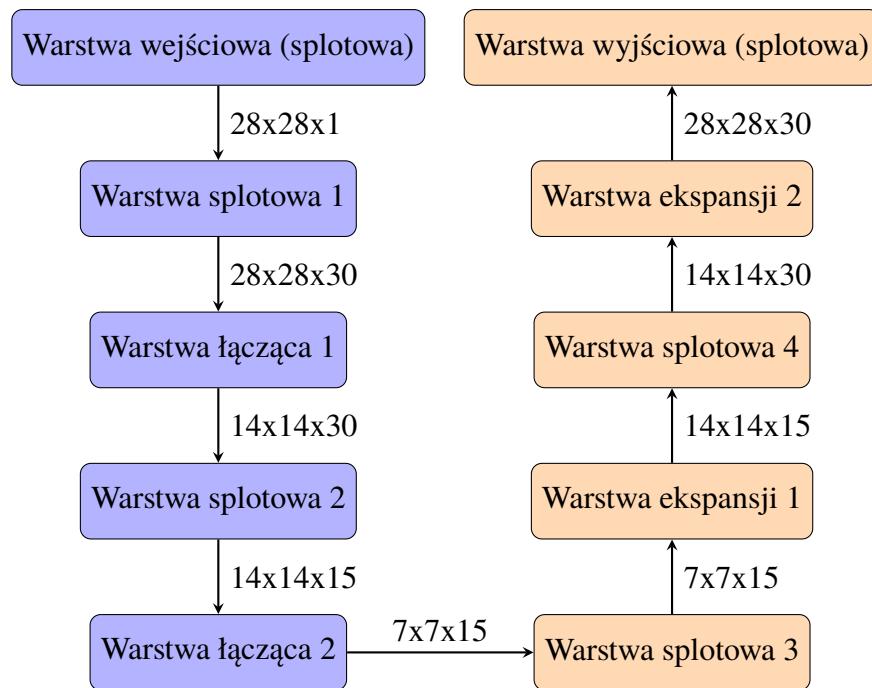
Rysunek 2.5: Wartość funkcji straty w kolejnych epokach trenowania autoenkodera niedopełnionego
Źródło: Opracowanie własne

2.2. Autoenkoder splotowy

W przypadku danych ze zbioru MNIST obrazy są dość małe, więc nawet prosty autoenkoder dał dość dobre wyniki. Jednak w zastosowaniu do danych wejściowych będących obrazami, zamiast autoenkodera prostego można użyć autoenkodera splotowego, który jest przeznaczony przede wszystkim do takiego typu danych. W autoenkoderze splotowym sieć kodera składa się z warstw splotowych i łączących - zazwyczaj ma ona na celu zmniejszenie wymiarowości danych (wysokości i szerokości obrazu) oraz zwiększenie głębokości (liczby map cech). Dekoder powinien przeprowadzić operację odwrotną (zwiększyć wysokość i szerokość obrazu, a zmniejszyć liczbę map cech). W tym celu można wykorzystać transponowane warstwy splotowe lubłączyć zwykłe warstwy splotowe z warstwami ekspansji [6].

Rysunek 2.6 przedstawia strukturę użytego autoenkodera splotowego. Koder składa się z dwóch warstw splotowych i dwóch warstw łączących. Warstwy łączące zmniejszają wymiary obrazu z 28x28 kolejno do 14x14 i 7x7. Dekoder składa się z dwóch warstw splotowych i dwóch warstw ekspansji, które zwiększają wymiary skompresowanego obrazu kolejno do

14×14 i 28×28 , osiągając oryginalny wymiar. W przypadku warstw splotowych, jądro jest wymiaru 3×3 . Warstwy 1 i 4 mają po 30 filtrów, a warstwy 2 i 3 po 15 filtrów. Warstwa wyjściowa również jest warstwą splotową. Ma ona jeden filtr. W warstwach łączących stosowana jest metoda redukowania *MaxPooling*, gdzie jądro łączące ma rozmiar 2×2 . W warstwach ekspansji współczynnik nadpróbkowania jest równy 2 dla wierszy i tyle samo dla kolumn.



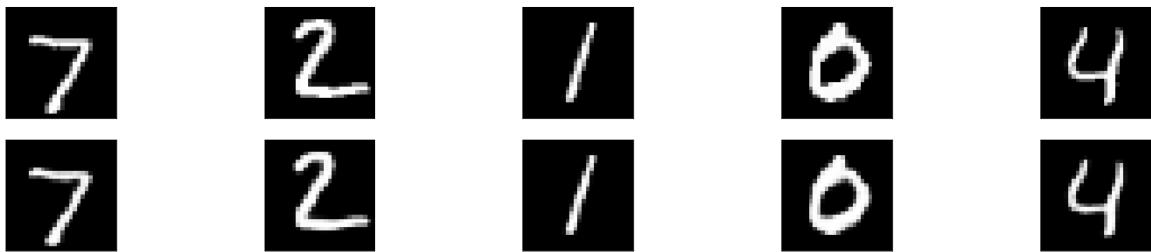
Rysunek 2.6: Struktura autoenkodera splotowego oraz wymiary kolejnych warstw

Źródło: Opracowanie własne

Autoenkoder splotowy jest trenowany przez 15 epok. Rozmiar pakietu (*batch size*) wynosi 128. Zbiorem walidacyjnym jest zbiór testowy.

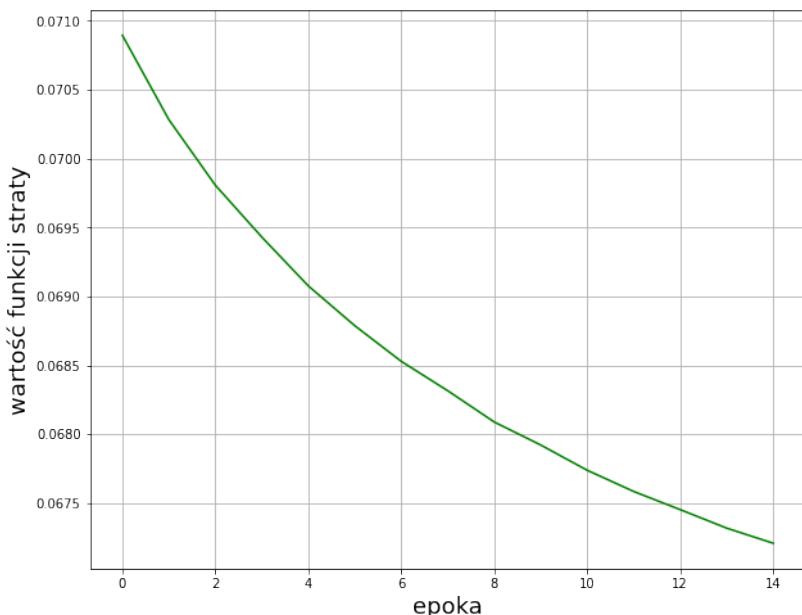
Na rysunku 2.7 widać efekty działania autoenkodera splotowego. W górnym rzędzie przedstawione są oryginalne obrazy ze zbioru testowego MNIST, a w dolnym rzędzie ich rekonstrukcje z autoenkodera. Widać, że obrazy w obu rzędach są do siebie podobne - strata jakości jest mniejsza niż w przypadku autoenkodera niedopełnionego.

Rysunek 2.8 przedstawia wartości funkcji straty w kolejnych epokach trenowania autoenkodera splotowego. Widać, że wartości funkcji straty maleją przez wszystkie 15 epok, co oznacza, że raczej nie wystąpiło przeuczenie modelu. Można też zauważyć, że już w pierwszej epoce wartość funkcji straty jest niższa niż w 15 epokach w przypadku autoenkodera niedopełnionego. Oznacza to, że rekonstrukcje z autoenkodera splotowego znacznie mniej różnią się od oryginalnych danych niż rekonstrukcje z autoenkodera niedopełnionego.



Rysunek 2.7: Pięć pierwszych obserwacji ze zbioru testowego MNIST: w górnym rzędzie oryginalne obrazy, w dolnym rekonstrukcje z autoenkodera splotowego

Źródło: Opracowanie własne



Rysunek 2.8: Wartość funkcji straty w kolejnych epokach trenowania autoenkodera splotowego

Źródło: Opracowanie własne

2.3. Autoenkoder odszumiający

Pokażemy teraz zastosowanie autoenkodera splotowego do odszumiania zanieczyszczonych obrazów. W poprzednich przykładach zbiór uczący stanowił zarówno dane wejściowe, jak i oczekiwane dane wyjściowe w modelu. W przypadku autoenkodera odszumiającego zbiór uczący również jest traktowany jako docelowy wynik, ale danymi wejściowymi są zaszumione obserwacje z tego zbioru.

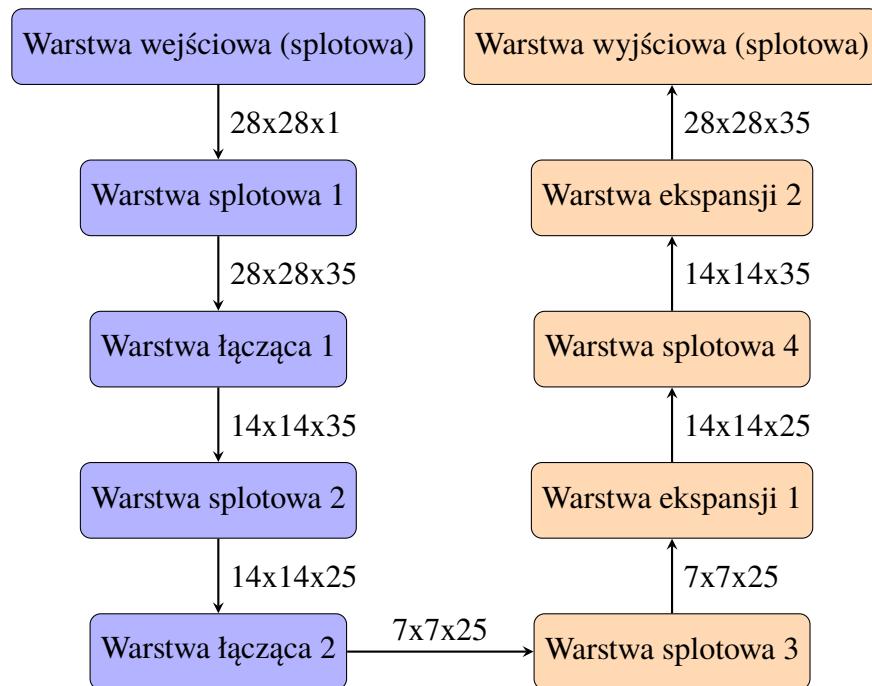


Rysunek 2.9: Zaszumione obrazy ze zbioru testowego MNIST

Źródło: Opracowanie własne

Rysunek 2.9 przedstawia obrazy ze zbioru testowego MNIST po zaszumieniu. Zaszumienie polegało na dodaniu do wartości każdego piksela losowej liczby z rozkładu $\mathcal{N}(0, 1)$, pomnożonej przez 0.7 (parametr 0.7 został przyjęty arbitralnie i można go zmieniać w celu dalszych eksperymentów).

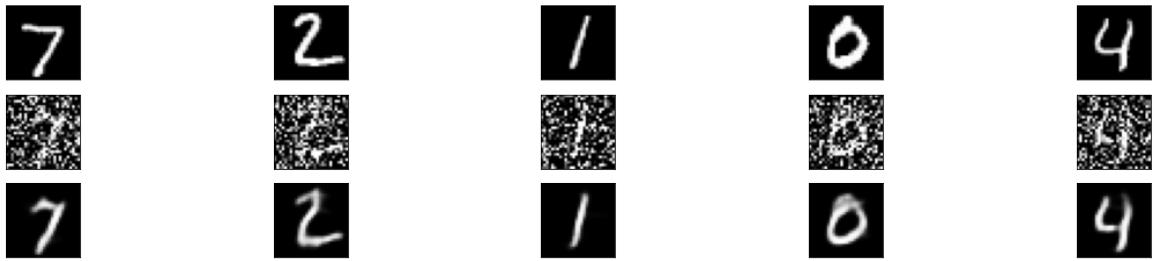
Struktura autoenkodera odszumiającego jest w tym przypadku analogiczna jak autoenkodera splotowego (rysunek 2.6). Poszczególne warstwy różnią się jedynie liczbą filtrów, co jest pokazane na schemacie 2.10



Rysunek 2.10: Struktura autoenkodera splotowego oraz wymiary kolejnych warstw

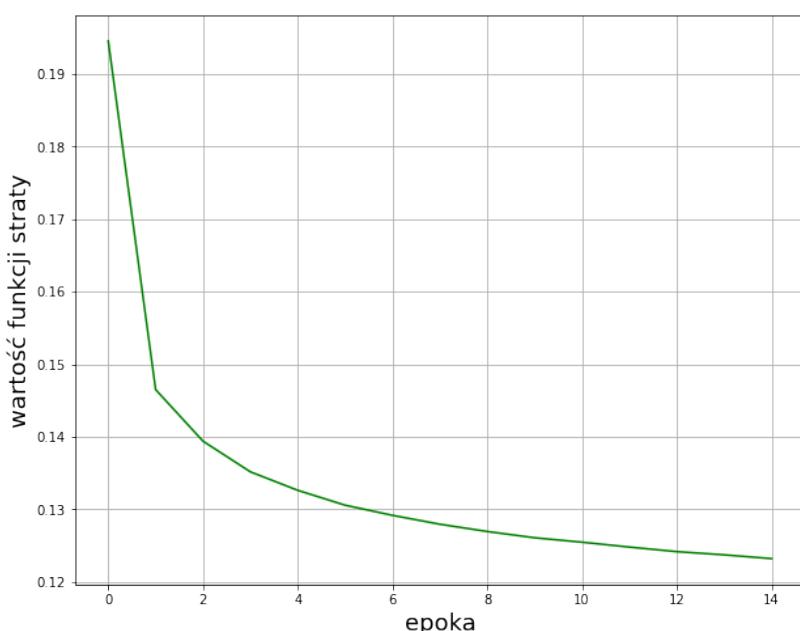
Źródło: Opracowanie własne

Rysunek 2.11 przedstawia efekt działania autoenkodera odszumiającego. W górnym rzędzie widać oryginalne, niezaszumione obrazy ze zbioru testowego. Środkowy rzząd przedstawia zanieczyszczone obrazy, które stanowią dane wejściowe dla autoenkodera. W dolnym rzędzie widać obrazy bez szumu, zrekonstruowane przez autoenkoder.



Rysunek 2.11: Pięć pierwszych obserwacji ze zbioru testowego MNIST: w górnym rzędzie oryginalne obrazy, w środkowym zaszumione, w dolnym rekonstrukcje z autoenkodera odszumiającego
Źródło: Opracowanie własne

Na rysunku 2.12 przedstawiona jest wartość funkcji straty w kolejnych epokach uczenia autoenkodera odszumiającego.



Rysunek 2.12: Wartość funkcji straty w kolejnych epokach trenowania autoenkodera odszumiającego
Źródło: Opracowanie własne

2.4. Wyszukiwanie obrazu

Do wyszukiwania obrazu zostanie użyty autoenkoder prosty opisany w podrozdziale 2.1. Elementy ze zbioru treningowego zostały przy użyciu kodera skompresowane do wymiaru 15. Dla skompresowanych danych budowany jest model k najbliższych sąsiadów, gdzie parametr k oznacza również liczbę podobnych obrazów do zadanego, jakie zostaną wyszukane. Zbiorem testowym są w tym przypadku obrazy, do których chcemy wyszukać

obrazy podobne. Użyty zbiór testowy jest przedstawiony na rysunku 2.13. Dla każdego z tych obrazów chcemy wyszukać $k = 5$ obrazów podobnych.



Rysunek 2.13: Zbiór testowy dla zadania wyszukiwania obrazu

Źródło: Opracowanie własne

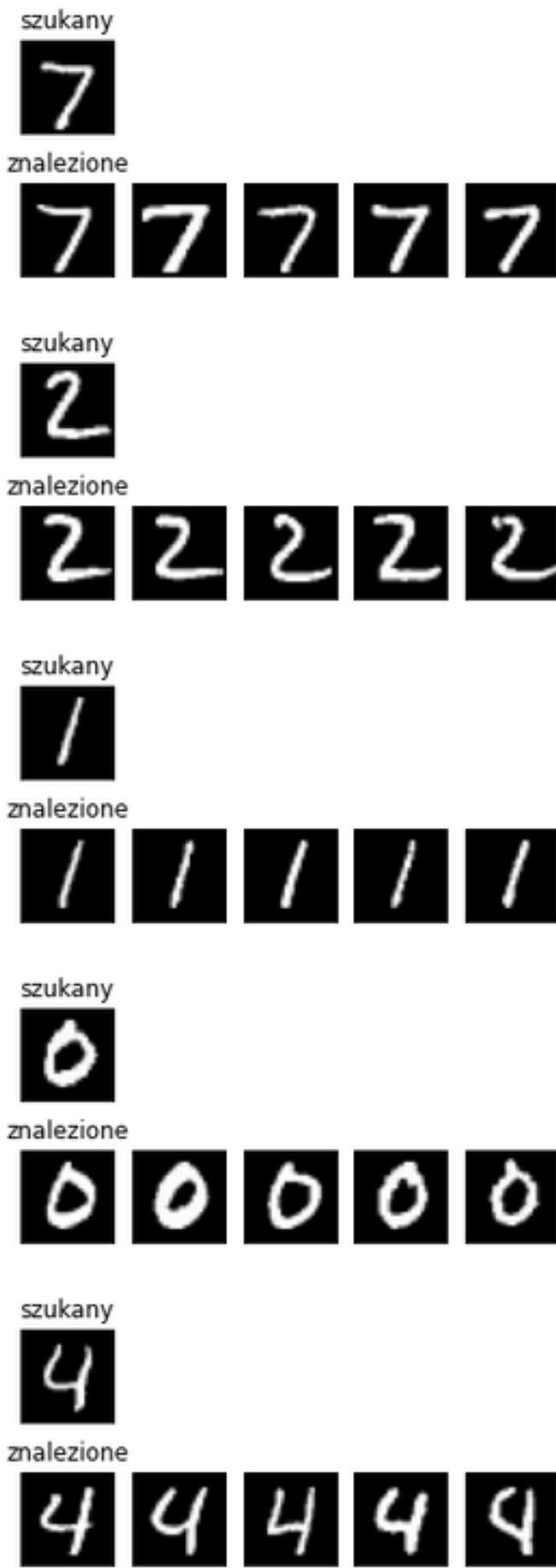
Obrazy ze zbioru testowego również są kompresowane przy użyciu kodera. Następnie dla skompresowanej formy obrazów znajdujemy $k = 5$ najbliższych sąsiadów według modelu KNN. Jako wynik wyświetlane są oryginalne wersje znalezionych obrazów. Wynik dla opisanego zbioru testowego znajduje się na rysunku 2.14. Widać, że dla każdego elementu ze zbioru testowego, znalezione obrazy zawierają odpowiednią cyfrę, zgodną z wyszukiwaniem (np. dla pierwszego przykładu z liczbą 7, wszystkie znalezione obrazy również przedstawiają tę cyfrę).

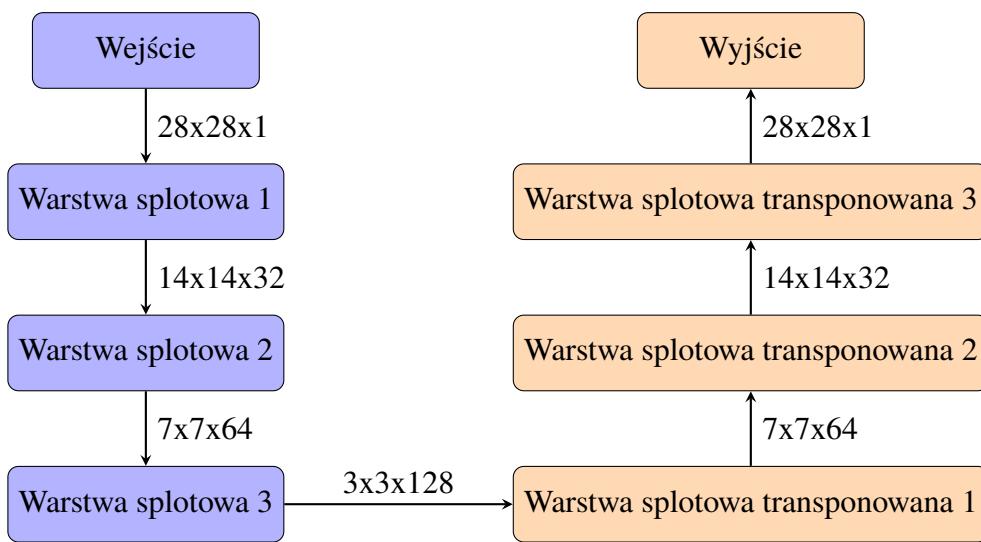
2.5. Wykrywanie anomalii przy użyciu autoenkodera

W każdej warstwie jądro ma rozmiar 3×3 , a krok wynosi 2. Funkcją aktywacji w każdej warstwie jest funkcja ReLU. Zastosowano uzupełnianie zerami. Funkcją straty jest funkcja SSIM. Autoenkoder jest trenowany przez 5 epok.

Graniczną wartością straty, przy której uznamy obserwację za anormalną, jest dziesięćdziesiąty dziewiąty percentyl wartości funkcji straty ze zbioru treningowego (wynosi on około 0.017821).

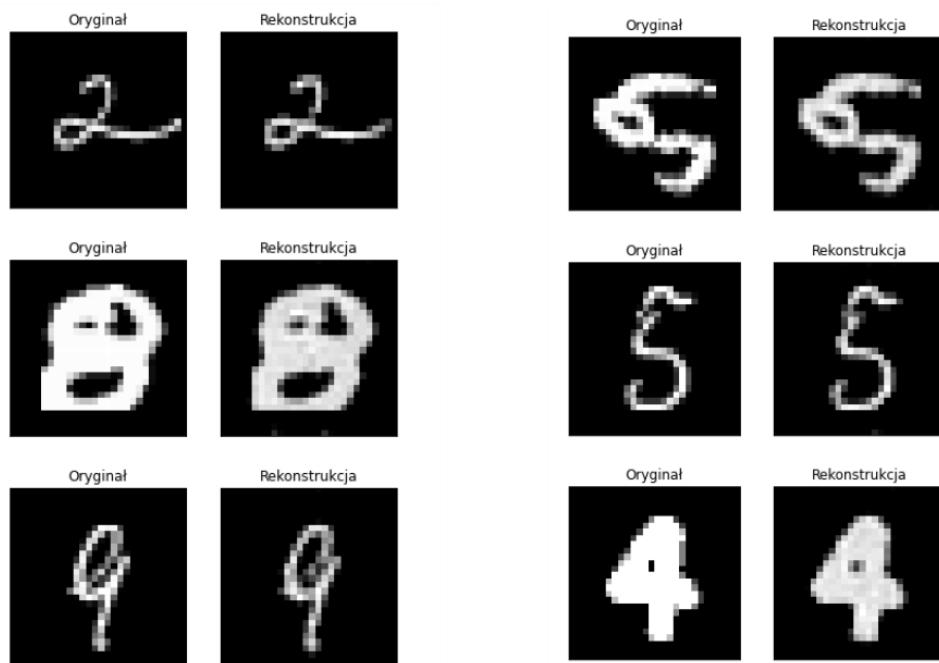
2.6. Generowanie obrazów (autoenkoder wariancyjny)





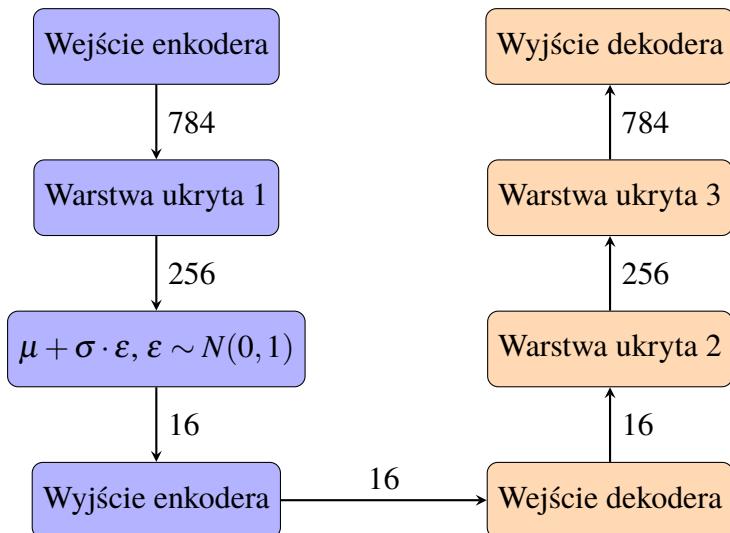
Rysunek 2.15: Struktura autoenkodera splotowego zastosowanego do wykrywania anomalii

Źródło: Opracowanie własne



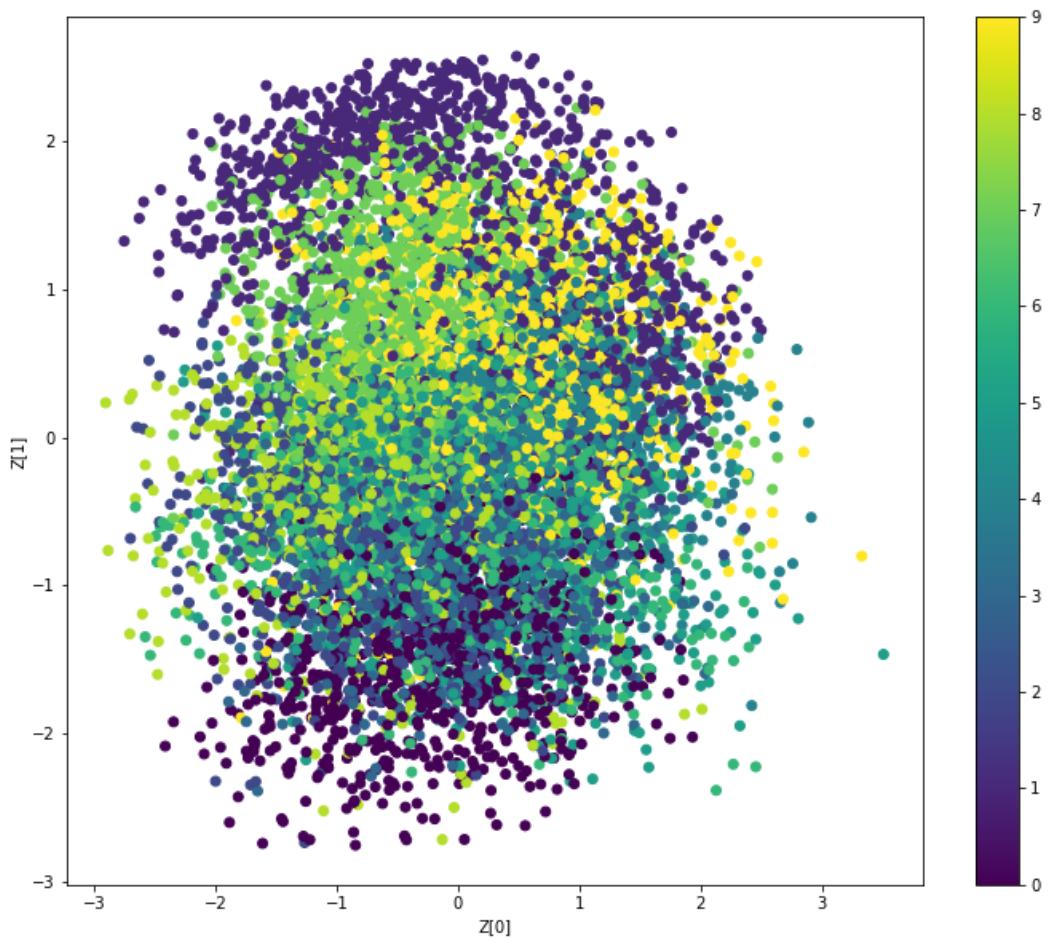
Rysunek 2.16: Przykłady anomalii w zbiorze MNIST wykrytych za pomocą autoenkodera

Źródło: Opracowanie własne



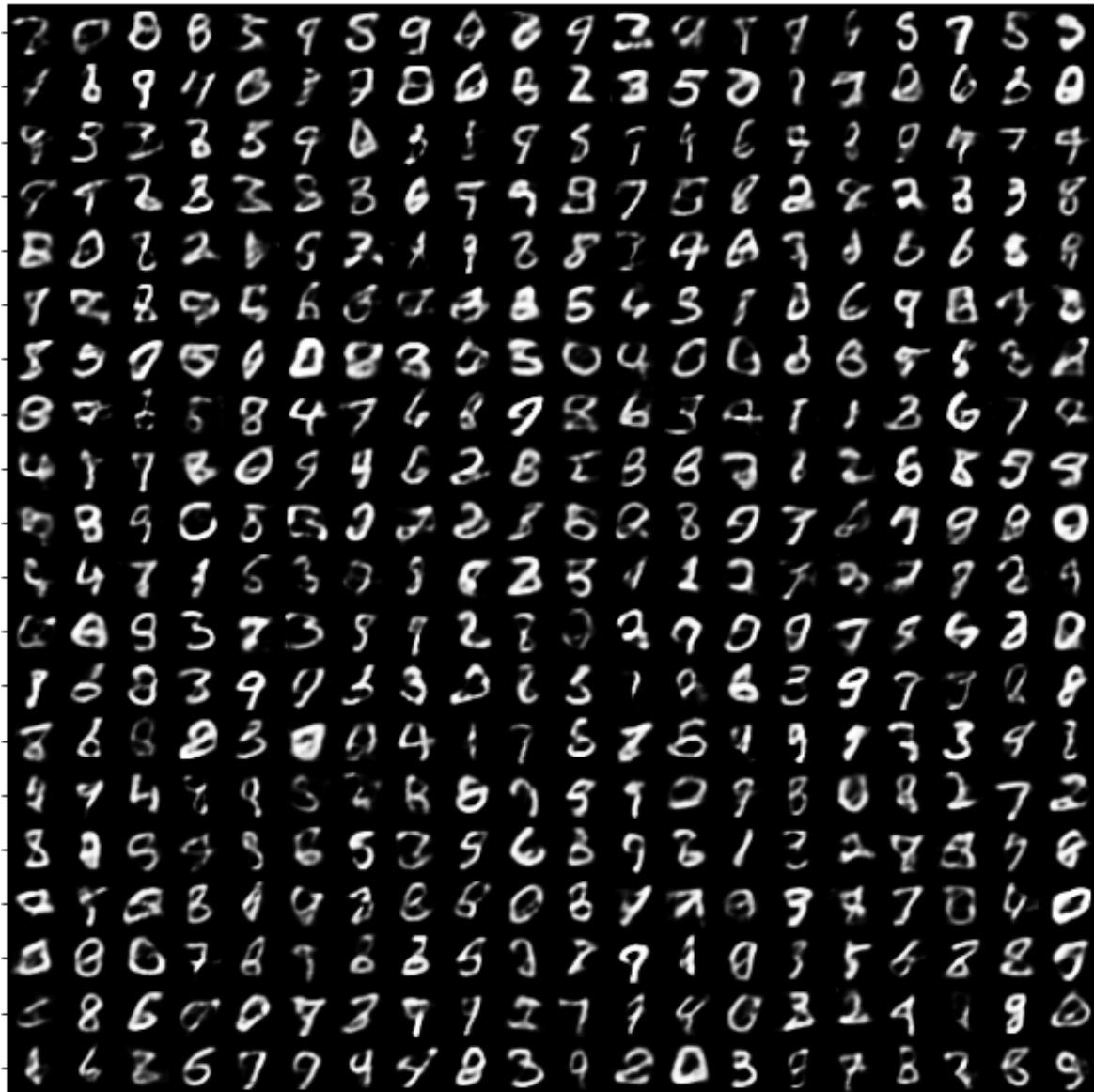
Rysunek 2.17: Struktura autoenkodera wariacyjnego użytego do generowania obrazów przypominających te ze zbioru uczącego

Źródło: Opracowanie własne



Rysunek 2.18: Przestrzeń ukryta autoenkodera wariacyjnego

Źródło: Opracowanie własne



Rysunek 2.19: Przykłady obrazów cyfr wygenerowanych przez autoenkoder wariancyjny

Źródło: Opracowanie własne

Podsumowanie i wnioski

tu będzie jakieś podsumowanie a może nawet jakieś wnioski

Bibliografia

- [1] Aggarwal, C. C. (2018). *Neural Networks and Deep Learning*, Springer International Publishing
- [2] Bank D., Koenigstein N., Giryes R. (2021), *Autoencoders*, arXiv:2003.05991v2
- [3] Chollet F., Allaire J. J. (2019) *Deep Learning. Praca z językiem R i biblioteką Keras*, Helion SA
- [4] Edureka, Autoencoders Tutorial. Autoencoders in Deep Learning. Tensorflow Training https://www.youtube.com/watch?v=nTt_ajul8NY
- [5] Ertel W. (2017) *Introduction to Artificial Intelligence. Second Edition*, Springer International Publishing
- [6] Géron A. (2020) *Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow. Wydanie II*, Helion SA
- [7] Goodfellow I., Bengio Y., Courville A. (2018), *Deep Learning. Systemy uczące się*, PWN, Warszawa
- [8] Hubel D. H., *Single Unit Activity in Striate Cortex of Unrestrained Cats*, J. Physiol. (1959) 147, 226-238
- [9] Hubel D. H., Wiesel T. N., *Receptive Fields of Single Neurones in the Cat's Striate Cortex*, J. Physiol., (1959), 148, 574-591
- [10] Krohn J., Beyleveld G., Bassens A., *Uczenie głębokie i sztuczna inteligencja. Interaktywny przewodnik ilustrowany*, Helion 2022
- [11] Kumar S., (2021) *7 Applications of Auto-Encoders every Data Scientist should know. Essential guide to Auto-Encoders and its usage*, Towards Data Science, <https://towardsdatascience.com/6-applications-of-auto-encoders-every-data-scientist-should-know-dc703cbc892b> (dostęp: 01.04.2022)
- [12] Mungoli A., (2020) *Dmiensionality Reduction: PCA versus Autoencoders. Comparison of PCA and AutoEncoders for Dimensionality Reduction*, Towards Data Science, <https://towardsdatascience.com/dimensionality-reduction-pca-versus-autoencoders-338fcacf3297d> (dostęp: 24.04.2022)
- [13] Osinga D. (2019) *Deep Learning. Receptury*, Helion SA

Bibliografia

- [14] Pinaya W. H. L., Vieira S., Garcia-Dias R., Mechelli A., *Machine Learning. Methods and Applications to Brain Disorders*, Academic Press, 2020
- [15] Skansi S. (2018) *Introduction to Deep Learning. From Logical Calculus to Artificial Intelligence*, Springer International Publishing

Spis rysunków

1.1	Przykładowe sztuczne sieci neuronowe rozwiązuające proste zadania logiczne	7
1.2	Struktura sztucznego neuronu, który stosuje funkcję skokową f na ważonej sumie sygnałów wejściowych	8
1.3	Perceptron z trzema neuronami wejściowymi i trzema wyjściami	8
1.4	Przykładowe funkcje aktywacji wraz z pochodnymi	11
1.5	Działanie neuronów biologicznych w korze wzrokowej	12
1.6	Warstwy splotowe z prostokątnymi lokalnymi polami recepcyjnymi	13
1.7	Związek pomiędzy warstwami a uzupełnianiem zerami	13
1.8	Warstwa splotowa z krokiem o długości 2	14
1.9	Uzyskiwanie dwóch map cech za pomocą dwóch różnych filtrów	15
1.10	Warstwy splotowe zawierające wiele map cech, a także zdjęcie z trzema kanałami barw .	16
1.11	Maksymalizująca warstwa łącząca (jądro łączące: 2×2 , krok: 2, brak uzupełniania zerami)	18
1.12	Warstwa <i>max-pooling</i> z filtrem i krokiem rozmiaru 2×2 , zastosowana do mapy aktywacji o rozmiarze 4×4 (widoczna po lewej stronie) skutkuje uzyskaniem mapy czterokrotnie mniejszej niż oryginalna	19
1.13	Niezmienniczość związana z drobnymi przesunięciami	20
1.14	Struktura autoenkoera odwzorowującego wejście x na wyjście r (nazywane rekonstrukcją) poprzez reprezentację ukrytą (kodowanie) h . Autoenkoder składa się z dwóch składników: kodera f (odwzorowującego x na h) i dekodera g (odwzorowującego h na r)	22
1.15	Funkcje straty rzadkości	25
1.16	Autokodery odszumiające: wykorzystujące szum gaussowski (po lewej) lub metodę porzucania (po prawej)	26
1.17	Zaszumione obrazy (na górze) i ich rekonstrukcje (na dole)	27
1.18	Struktura autoenkodera odszumiającego	27
1.19	Przykładowa struktura autoenkodera stosowego	28
1.20	Przykładowa struktura autoenkodera wariancyjnego	29

Spis rysunków

1.21 Reprezentacje ukryte. W tym przykładzie, obrazy cyfr od 0 do 9 były użyte do trenowania autoenkodera odszumiającego i przeciwnego. W obu przypadkach dane uczące są przedstawione w dwuwymiarowej przestrzeni ukrytej. Ponieważ nie nakładamy żadnych ograniczeń na autoenekoder odszumiający, to jego przestrzeń ukryta zawiera puste miejsca. Z drugiej strony, autoenekoder przeciwny ogranicza reprezentacje do podobnych według rozkładu (w tym przypadku dwuwymiarowego normalnego), co w rezultacie oznacza brak pustych miejsc	31
1.22 Struktura generatywnej sieci przeciwniej (GAN)	32
1.23 Struktura autokodera przeciwnego. Sieć dyskryminująca jest dodana do autoenkodera, aby zmusić go do generowania reprezentacji ukrytych podobnych do rozkładu a priori	33
1.24 Autodenkoder w zastosowaniu do redukcji wymiarowości	35
1.25 Autoenekoder stosowany do ekstrakcji cech	36
1.26 Stosowanie autoenkodera do odszumiania obrazu	37
1.27 Zastosowanie autoenkodera do kompresji obrazu	37
1.28 Zastosowanie autoenkodera do wyszukiwania obrazu	38
2.1 Pięć pierwszych obserwacji ze zbioru cyfr MNIST	39
2.2 Struktura autoenkodera prostego oraz wymiary kolejnych warstw	40
2.3 Reprezentacja 15-wymiarowa dla pierwszej obserwacji ze zbioru testowego MNIST	40
2.4 Pięć pierwszych obserwacji ze zbioru testowego MNIST: w górnym rzędzie oryginalne obrazy, w dolnym rekonstrukcje z autoenkodera niedopełnionego	40
2.5 Wartość funkcji straty w kolejnych epokach trenowania autoenkodera niedopełnionego	41
2.6 Struktura autoenkodera splotowego oraz wymiary kolejnych warstw	42
2.7 Pięć pierwszych obserwacji ze zbioru testowego MNIST: w górnym rzędzie oryginalne obrazy, w dolnym rekonstrukcje z autoenkodera splotowego	43
2.8 Wartość funkcji straty w kolejnych epokach trenowania autoenkodera splotowego	43
2.9 Zaszumione obrazy ze zbioru testowego MNIST	43
2.10 Struktura autoenkodera splotowego oraz wymiary kolejnych warstw	44
2.11 Pięć pierwszych obserwacji ze zbioru testowego MNIST: w górnym rzędzie oryginalne obrazy, w środkowym zaszumione, w dolnym rekonstrukcje z autoenkodera odszumiającego	45
2.12 Wartość funkcji straty w kolejnych epokach trenowania autoenkodera odszumiającego	45
2.13 Zbiór testowy dla zadania wyszukiwania obrazu	46
2.14 Wyniki zastosowania autoenkodera przy wyszukiwaniu obrazu	48
2.15 Struktura autoenkodera splotowego zastosowanego do wykrywania anomalii	49
2.16 Przykłady anomalii w zbiorze MNIST wykrytych za pomocą autoenkodera	49
2.17 Struktura autoenkodera wariancyjnego użytego do generowania obrazów przypominających te ze zbioru uczącego	50
2.18 Przestrzeń ukryta autoenkodera wariancyjnego	50

2.19 Przykłady obrazów cyfr wygenerowanych przez autoenkoder wariancyjny 51

Spis tabel

Załączniki

1. Płyta CD z niniejszą pracą w wersji elektronicznej.

Streszczenie (Summary)

Przegląd autoenkoderów stosowanych w nienadzorowanym uczeniu maszynowym

An overview of autoencoders used in unsupervised machine learning