UNIWERSYTET ZIELONOGÓRSKI

WYDZIAŁ NAUK ŚCISŁYCH I PRZYRODNICZYCH

kierunek Inżynieria danych

specjalność Projektowanie i obsługa systemów analitycznych

Maksymilian Lewandowski

Sieci konwolucyjne w diagnostyce chorób skóry

Convolutional networks in skin disease diagnostics

Promotor pracy
dr Jacek Bojarski

Zielona Góra, 2025

Spis treści

1 Wstęp

Sztuczne sieci neuronowe stanowią jeden z najbardziej dynamicznie rozwijających się obszarów sztucznej inteligencji. Historia tej technologii sięga lat 50. XX wieku, kiedy to pierwsi badacze, inspirowani strukturą ludzkiego mózgu, próbowali stworzyć systemy zdolne do samodzielnego uczenia się. Jednak dopiero w ostatniej dekadzie, dzięki postępowi w zakresie mocy obliczeniowej i algorytmów głębokiego uczenia, sieci neuronowe stały się praktycznym narzędziem obliczeniowym.

Sieci konwolucyjne, stanowiące jeden z najważniejszych paradygmatów w dziedzinie głębokiego uczenia, zrewolucjonizowały sposób, w jaki maszyny interpretują i analizują obrazy. O ile wcześniejsze metody wymagały ręcznego projektowania cech, sieci konwolucyjne oferują możliwość automatycznego odkrywania i wyodrębniania kluczowych informacji bezpośrednio z danych.

Niniejsza praca skupia się na szczegółowym wyjaśnieniu zasad działania sieci neuronowych. Omówione zostaną zarówno podstawowe elementy budowy sieci, jak i zaawansowane mechanizmy wykorzystywane w nowoczesnych architekturach. Dla lepszego zobrazowania omawianych zagadnień, przedstawiona teoria zostanie zilustrowana praktycznym przykładem klasyfikacji obrazów medycznych.

W części praktycznej wykorzystano dwa uzupełniające się zbiory danych - syntetyczny oraz rzeczywisty. Zbiór syntetyczny pozwala na systematyczne testowanie poszczególnych aspektów działania sieci, podczas gdy zbiór rzeczywisty, zawierający zdjęcia zmian skórnych, demonstruje zachowanie modelu w bardziej wymagających warunkach.

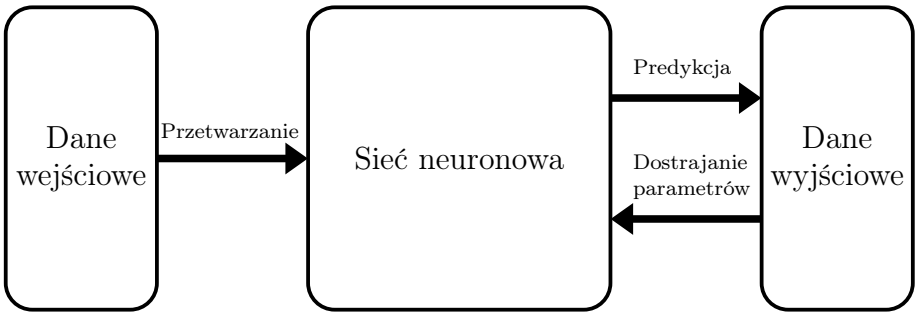
Praca składa się z kilku kluczowych elementów: przedstawienia teoretycznych podstaw sieci neuronowych, szczegółowego opisu architektury sieci konwolucyjnych, metodologii trenowania modelu oraz analizy jego działania na przykładzie wybranego problemu klasyfikacji. W celu zapewnienia pełnej reprodukowalności badań, w dodatku zamieszczono odnośniki do notebooków Google Colab zawierających kompletną implementację opisywanych modeli.

Celem pracy jest dostarczenie kompleksowego opracowania na temat współczesnych sieci neuronowych, które może służyć jako punkt wyjścia do dalszego zgłębiania tej dziedziny.

2 Budowa sieci neuronowej

Sztuczne sieci neuronowe to zaawansowane systemy przetwarzania informacji, zainspirowane działaniem biologicznego układu nerwowego [1]. Ich kluczową cechą jest zdolność uczenia się złożonych wzorców bezpośrednio z danych, bez konieczności jawnego programowania reguł wnioskowania.

To co wyróżnia sieci neuronowe, to ich adaptacyjność - parametry sieci są dostrajane w procesie uczenia, aby jak najdokładniej odwzorować zależności obecne w danych treningowych [2]. Jak pokazano na Rysunku 1, sieć przetwarza dane wejściowe generując predykcję, które są następnie wykorzystywane do dostrajania parametrów w procesie uczenia. Ta właściwość, w połączeniu ze zdolnością do generalizacji na nowe przypadki, sprawia, że sieci neuronowe znajdują zastosowanie w szerokiej gamie problemów - od rozpoznawania obrazów, przez przetwarzanie języka naturalnego, po sterowanie autonomicznymi systemami.



Rysunek 1: Schemat adaptacyjnego cyklu przetwarzania danych w sieci neuronowej. Źródło: opracowanie własne.

W kolejnych sekcjach zostanie szczegółowo omówiona struktura sieci neuronowych - od pojedynczego neuronu, przez warstwy, po kompletne architektury - jak i proces ich uczenia, skupiając się na metodach optymalizacji parametrów oraz technikach efektywnego treningu. Pozwoli to zrozumieć nie tylko budowę, ale i sposób, w jaki sieć adaptuje się do rozwiązywanego zadania.

2.1 Sztuczny neuron

Fundamentem sieci neuronowych jest sztuczny neuron, którego budowa została zainspirowana swoim biologicznym odpowiednikiem. Biologiczny neuron odbiera sygnały przez dendryty, przetwarza je w ciele komórki i przekazuje dalej poprzez akson. Podobnie, sztuczny neuron przyjmuje szereg sygnałów wejściowych, przetwarza je, a następnie generuje sygnał wyjściowy.

Działanie sztucznego neuronu polega na sekwencyjnym wykonaniu dwóch transformacji. Pierwsza z nich, oznaczana symbolem Σ, to agregacja sygnałów wejściowych poprzez iloczyn skalarny wektora wejściowego $\mathbf{x} \in \mathbb{R}^n$ z wektorem wag $\mathbf{w} \in \mathbb{R}^n$ oraz dodanie przesunięcia $b \in \mathbb{R}$ (ang. *bias*). Jak przedstawiono na Rysunku 2, sygnały wejściowe $x_1, x_2, \ldots, x_n$ są ważone przez odpowiadające im wagi $w_1, w_2, \ldots, w_n$. Wynik tej operacji, oznaczany jako z, można zapisać jako:

$$z = \mathbf{w}^T\mathbf{x} + b.$$

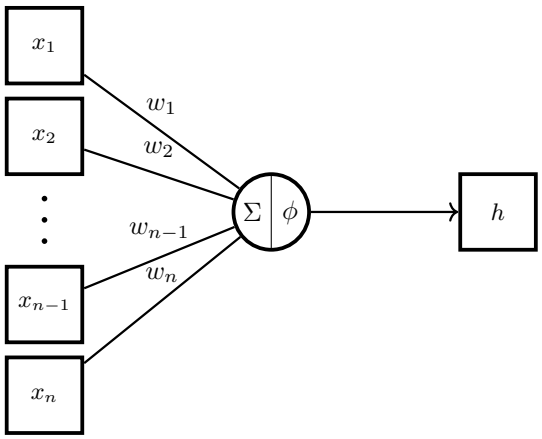
Drugą transformacją jest nieliniowa funkcja aktywacji oznaczana symbolem ϕ, która przekształca sygnał z w końcową wartość wyjściową neuronu h:

$$h = \phi(z).$$

Pełny proces przetwarzania sygnału przez neuron można więc zapisać jako złożenie transformacji:

$$\mathbf{x} \circ \Sigma \circ \phi = h,$$

a jej wynik nazywany jest aktywacją.



Rysunek 2: Schemat struktury i działania sztucznego neuronu. Źródło: opracowanie własne.

2.2 Funkcja aktywacji

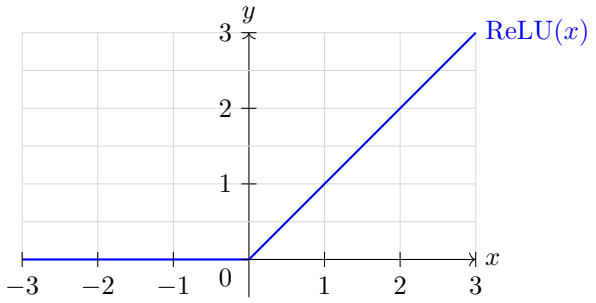
Funkcja aktywacji ϕ pełni kluczową rolę w działaniu neuronu, wprowadzając do modelu nieliniowość niezbędną do aproksymacji złożonych zależności. Obecność nieliniowości jest fundamentalna dla zdolności sieci do modelowania skomplikowanych relacji w danych – bez niej, niezależnie od liczby warstw, sieć realizowałaby jedynie przekształcenia liniowe.

W praktyce istnieje szeroki wachlarz funkcji aktywacji, każda z nich posiadająca swoje unikalne właściwości i obszary zastosowań. Jednakże, w dalszej części opisane zostaną cztery najczęściej stosowane:

- ReLU (Rectified Linear Unit) [3]:

$$\mathrm{ReLU} : \mathbb{R} \to [0, +\infty), \quad \mathrm{ReLU}(x) = \begin{cases} x & \text{dla } x \geq 0, \\ 0 & \text{dla } x < 0. \end{cases}$$

  Funkcja tożsamościowa po części dodatniej oraz stała zero po części ujemnej. Jak pokazano na Rysunku 3, ReLU przepuszcza wartości dodatnie bez zmian, natomiast wszystkie wartości ujemne są zerowane. Jest to obecnie najpopularniejsza funkcja aktywacji ze względu na swoją prostotę obliczeniową oraz efektywność w treningu głębokich sieci. Jej główną zaletą jest eliminacja problemu zanikających gradientów dla wartości dodatnich, jednak może cierpieć na problem „martwych neuronów". Zjawisko to występuje, gdy neuron zaczyna generować wyłącznie wartości ujemne - w takiej sytuacji jego wyjścia są zawsze zerowane przez ReLU, a ponieważ pochodna dla wartości ujemnych wynosi zero, wagi neuronu przestają być aktualizowane w procesie uczenia. W efekcie neuron staje się nieaktywnym elementem sieci.

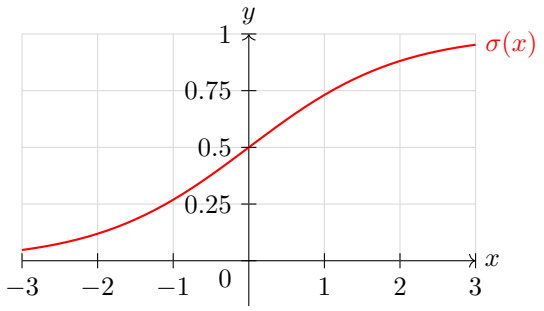


Rysunek 3: Wykres funkcji ReLU. Źródło: opracowanie własne.

- Sigmoid [4]:

$$\sigma : \mathbb{R} \to (0,1), \quad \sigma(x) = \frac{1}{1+e^{-x}}.$$

Funkcja nieliniowa przekształcająca wartości na przedział $(0,1)$. Na Rysunku 4 widoczna jest charakterystyczna krzywa sigmoidalna, która asymptotycznie zbliża się do 0 dla dużych wartości ujemnych i do 1 dla dużych wartości dodatnich, a w punkcie $x = 0$, przyjmuje wartość 0.5. Historycznie była jedną z pierwszych powszechnie stosowanych funkcji aktywacji, inspirowana biologicznym procesem aktywacji neuronów. Jej główną zaletą jest naturalna interpretacja wyjścia jako prawdopodobieństwa, jednak cierpi na problem zanikających gradientów dla wartości znacząco oddalonych od zera, co utrudnia trening głębokich sieci.

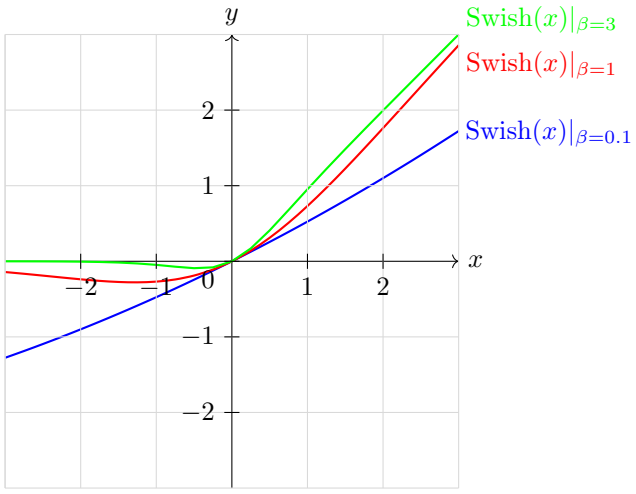


Rysunek 4: Wykres funkcji sigmoid. Źródło: opracowanie własne

- Swish [5]:

$$\text{Swish} : \mathbb{R} \to \mathbb{R} \quad \text{Swish}(x) = x \cdot \sigma(\beta x).$$

Nowoczesna funkcja nieliniowa, wprowadzona jako alternatywa dla ReLU. Na Rysunku 5 przedstawiono zachowanie funkcji Swish dla trzech różnych wartośc parametrów β: 0.1 (niebieska krzywa), 1 (czerwona krzywa) i 3 (zielona krzywa). Jak widać, większa wartości β zbliżają funkcję do charakterystyki ReLU, zachowując jej kluczowe zalty (liniowość dla wysokich wartości dodatnich i efektywna propagacja gradientów podczas treningu), jednocześnie łagodząc problem „martwych neuronów" dzięki gładkiemu przejściu w okolicy zera.

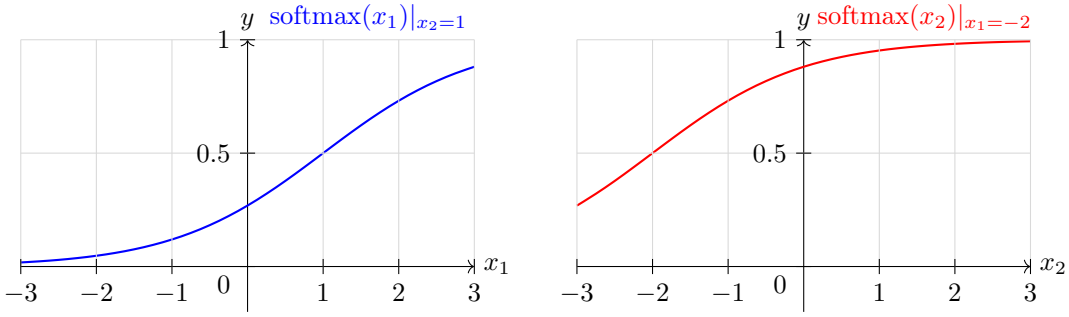


Rysunek 5: Porównanie zachowania funkcji Swish dla różnych wartości parametru β. Źródło: opracowanie własne.

- Softmax [6, s. 184-185]:

$$\text{Softmax} : \mathbb{R}^n \to (0,1)^n, \quad \text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}.$$

Funkcja softmax przekształca wektor liczb rzeczywistych w rozkład prawdopodobieństwa, gdzie suma wszystkich elementów wektora wyjściowego wynosi 1. Jest szczególnie użyteczna w problemach klasyfikacji wieloklasowej, gdzie wyjście sieci interpretujemy jako prawdopodobieństwo przynależności do poszczególnych klas. Na Rysunku 6 przedstawiono zachowanie funkcji softmax dla dwuwymiarowego wektora wejściowego $\mathbf{x} = [-2, 1]$. Lewy wykres pokazuje wartość softmax dla pierwszego elementu (x_1), a prawy dla drugiego (x_2).



Rysunek 6: Odwzorowanie tego samego dwuwymiarowego wektora wejściowego na wartości prawdopodobieństwa przez funkcję softmax dla pierwszej (lewy) i drugiej (prawy) składowej. Źródło: opracowanie własne

Wykresy przedstawiają, jak zmienia się wartość funkcji softmax dla jednego elementu wektora przy różnych wartościach drugiego elementu. Widoczne jest, że wartość softmax dla danego elementu zależy nie tylko od jego własnej wartości, ale również pozostałych elementów wektora wejściowego. Ta właściwość sprawia, że funkcja softmax efektywnie „konkuruje" o prawdopodobieństwo między klasami - wzrost wartości dla jednej klasy automatycznie zmniejsza prawdopodobieństwo pozostałych klas, zachowując ich sumę równą 1.

Warto również zauważyć, że funkcje działające z $\mathbb{R}$ w odpowiednie przeciwdziedziny są funkcjami punktowymi – oznacza to, że mogą być naturalnie rozszerzone na działanie na wektorach $\mathbb{R}^n$ poprzez niezależne przekształcanie każdego elementu $f(\mathbf{x}) = (f(x_1), f(x_2), \ldots, f(x_n))$. Jest to szczególnie istotne w kontekście sieciach neuronowych, gdzie operujemy na wielowymiarowych strukturach danych.

Wybór funkcji aktywacji zależy od specyfiki zadania oraz umiejscowienia neuronu w sieci. Funkcja ReLU jest domyślnym wyborem dla warstw ukrytych ze względu na swoją prostotę obliczeniową i empiryczną skuteczność, podczas gdy sigmoid czy softmax znajdują zastosowanie w warstwach wyjściowych dla zadań klasyfikacji ze względu na swoje właściwości transformujące.

2.3 Warstwa neuronów

W architekturze sieci neuronowych pojedynczy neuron, pomimo swojej fundamentalnej roli jako jednostki obliczeniowej, rzadko występuje samodzielnie. Jego możliwości są zbyt ograniczone dla praktycznych zastosowań. Dlatego w rzeczywistych implementacjach neurony łączy się równolegle, tworząc strukturę zwaną warstwą.

Warstwa neuronowa to funkcja $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ przekształcająca wektor wejściowy w nową reprezentację w postaci wektora o wymiarze m. Jak pokazano na Rysunku 7, każde wejście (x_1 do x_n) jest połączone z każdym neuronem warstwy. Wymiar wyjściowy m, determinowany przez liczbę neuronów w warstwie, jest kluczowym parametrem określającym zdolności aproksymacyjne struktury.

Podobnie jak pojedynczy neuron, warstwa realizuje dwie sekwencyjne transformacje. Pierwsza z nich, agregacja Σ, łączy dane wejściowe $\mathbf{x} \in \mathbb{R}^n$ z parametrami wszystkich neuronów, zapisanymi w postaci macierzy wag $\mathbf{W} \in \mathbb{R}^{m \times n}$ i wektora przesunięć $\mathbf{b} \in \mathbb{R}^m$:

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}.$$

Następnie, funkcja aktywacji ϕ jest aplikowana do wektora $\mathbf{z}$, generując końcowy wektor wyjściowy $\mathbf{h}$:

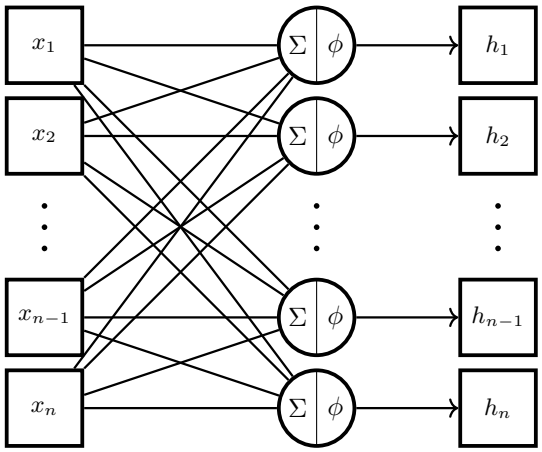
$$\mathbf{h} = \phi(\mathbf{z}).$$

Analogicznie do transformacji neuronu, pełną sekwencję transformacji warstwy można zapisać jako:

$$\mathbf{x} \circ \Sigma \circ \phi = \mathbf{h}.$$

Wiersze macierzy $\mathbf{W}$ odpowiadają wektorom wag poszczególnych neuronów, a elementy wektora $\mathbf{b}$ ich przesunięciom. Co istotne, każdy neuron w warstwie przetwarza te same dane wejściowe, ale z wykorzystaniem własnego, unikalnego zestawu parametrów. Ta różnorodność parametryzacji pozwala warstwie na równoległe wykrywanie różnych cech i wzorców w danych.

Opisana konstrukcja, znana jako warstwa gęsta (ang. *dense layer*) lub w pełni połączona (ang. *fully connected layer*), jest podstawowym elementem wielu architektur sieci neuronowych. W kolejnych rozdziałach poznamy inne typy warstw, w tym warstwę konwolucyjną, która znajduje szczególne zastosowanie w przetwarzaniu obrazów.



Rysunek 7: Architektura warstwy gęstej. Źródło: opracowanie własne.

2.4 Wielowarstwowa sieć neuronowa

Naturalne rozszerzenie architektury warstwowej prowadzi do koncepcji sieci wielowarstwowej – struktury powstałej przez sekwencyjne połączenie warstw neuronowych. Schemat 8 ilustruje, jak w takiej architekturze wyjście każdej warstwy $\mathbf{f}^{(i)}$ staje się wejściem dla warstwy następnej $\mathbf{f}^{(i+1)}$, tworząc kaskadę transformacji od wejścia $\mathbf{x}$ aż do końcowej predykcji $\hat{\mathbf{y}}$. Ten hierarchiczny układ pozwala sieci na stopniową ekstrakcję coraz bardziej złożonych i abstrakcyjnych cech z przetwarzanych danych.

Matematycznie, sieć L-warstwową można opisać jako złożenie funkcji realizowanych przez poszczególne warstwy:

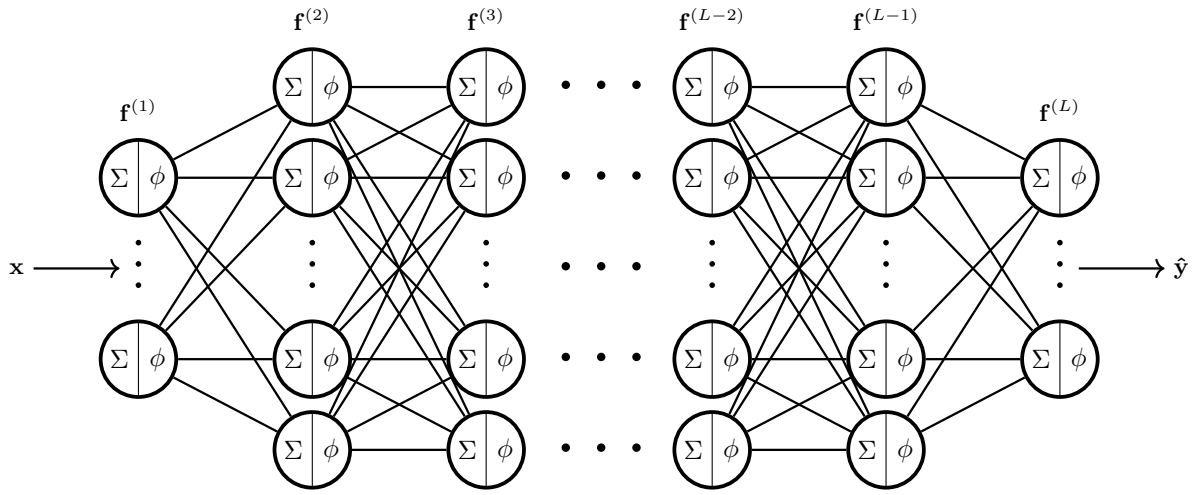
$$\mathbf{x} \circ \mathbf{f}^{(1)} \circ \mathbf{f}^{(2)} \circ \cdots \circ \mathbf{f}^{(L)} = \mathbf{h}^{(L)},$$

gdzie każda warstwa $\mathbf{f}^{(i)}$ realizuje sekwencję dwóch transformacji:

$$\mathbf{z}^{(i)} = \mathbf{W}^{(i)}\mathbf{h}^{(i-1)} + \mathbf{b}^{(i)},$$
$$\mathbf{h}^{(i)} = \phi_i(\mathbf{z}^{(i)}),$$

przy czym $\mathbf{h}^{(0)} = \mathbf{x}$ oznacza początkowe dane wejściowe, a $\mathbf{h}^{(L)} = \hat{\mathbf{y}}$ oznacza predykcję sieci.

Szczególną rolę w architekturze sieci pełni warstwa wyjściowa. Jej struktura musi być dostosowana do specyfiki rozwiązywanego zadania – liczba neuronów określa wymiar przestrzeni wyjściowej, a wybór funkcji aktywacji wpływa na charakter generowanych wyników. Na przykład, w zadaniach klasyfikacji wieloklasowej ostatnia warstwa typowo wykorzystuje funkcję softmax, przekształcającą wyniki na rozkład prawdopodobieństwa po klasach.



Rysunek 8: Architektura sieci wielowarstwowej prezentująca przepływ informacji od wejścia $\mathbf{x}$ do predykcji $\hat{\mathbf{y}}$ poprzez sekwencję warstw ukrytych $\mathbf{f}^{(i)}$. Źródło: opracowanie własne.

Kluczową zaletą architektury wielowarstwowej jest jej zdolność do automatycznego uczenia się hierarchii cech. W miarę przechodzenia przez kolejne warstwy, dane są przekształcane w coraz bardziej abstrakcyjne reprezentacje, dostosowane do rozwiązywanego zadania. Ten proces eliminuje potrzebę ręcznego projektowania cech, co stanowiło istotne ograniczenie tradycyjnych metod uczenia maszynowego. Zamiast tego, sieć sama odkrywa istotne wzorce w danych podczas procesu uczenia, systematycznie transformując przestrzeń cech w sposób optymalny dla danego problemu.

3 Trenowanie sieci

Sieć neuronowa w momencie inicjalizacji nie posiada zdolności do rozwiązywania postawionych zadań. Wynika to z faktu, że jej parametry – wagi i przesunięcia – są początkowo ustawione w sposób losowy. Proces uczenia polega na systematycznej modyfikacji tych parametrów tak, aby sieć nabywała zdolności do generowania pożądanych odpowiedzi dla prezentowanych danych wejściowych.

Uczenie się sieci można postrzegać jako proces optymalizacji, w którym dążymy do minimalizacji błędu popełnianego przez model. W przeciwieństwie do prostszych modeli, jak regresja liniowa, gdzie optymalne parametry można wyznaczyć analitycznie, w przypadku sieci neuronowych złożoność modelu, jego nieliniowość oraz wysoka wymiarowość przestrzeni parametrów wymuszają podejście iteracyjne. W praktyce realizowane jest to poprzez stopniowe dostrajanie parametrów, gdzie w każdym kroku sieć przetwarza porcję danych treningowych i modyfikuje swoje parametry w kierunku poprawy jakości przewidywań.

Do realizacji tego procesu potrzebne są trzy kluczowe komponenty, które zostaną szczegółowo omówione w kolejnych sekcjach: funkcja straty, która pozwala ilościowo ocenić jakość przewidywań sieci, algorytm wstecznej propagacji błędu, który umożliwia obliczenie gradientów, oraz optymalizator odpowiedzialny za aktualizację parametrów. Współdziałanie tych elementów pozwala na efektywną nawigację w złożonej, wielowymiarowej przestrzeni parametrów. Choć takie podejście iteracyjne nie gwarantuje znalezienia globalnego minimum, umożliwia znalezienie satysfakcjonującego rozwiązania przy akceptowalnym koszcie obliczeniowym.

3.1 Inicjalizacja parametrów

Proces inicjalizacji parametrów w sieciach neuronowych odgrywa kluczową rolę w efektywnym trenowaniu. Odpowiednia inicjalizacja ma fundamentalne znaczenie dla procesu uczenia, wpływając zarówno na szybkość zbieżności, jak i jakość końcowego modelu [7, 8].

W praktyce stosuje się kilka standardowych metod inicjalizacji:

1. Inicjalizacja Glorota (znana również jako inicjalizacja Xaviera) [8]:

$$\mathbf{W} \sim \mathcal{N}\left(0, \frac{2}{n+m}\right) \text{ lub } \mathbf{W} \sim U\left(-\sqrt{\frac{6}{n+m}}, \sqrt{\frac{6}{n+m}}\right),$$

   gdzie n i m oznaczają odpowiednio liczbę neuronów w warstwie poprzedniej i następnej. Zaprojektowana dla rodziny funkcji aktywacji sigmoid, aby utrzymać wariancję wejść w takim zakresie, aby aktywacje trafiały w obszar największej czułości funkcji (gdzie pochodna jest największa), zapobiagając tym samym problemowi nasycenia i zanikających gradientów.

2. Inicjalizacja He (znana również jako inicjalizacja Kaiming) [9]:

$$\mathbf{W} \sim \mathcal{N}\left(0, \frac{2}{n}\right) \text{ lub } \mathbf{W} \sim U\left(-\sqrt{\frac{6}{n}}, \sqrt{\frac{6}{n}}\right),$$

   gdzie n oznacza liczbę neuronów w poprzedniej warstwie. Zaprojektowana dla rodziny funkcji aktywacji ReLU, ponieważ uwzględnia fakt, że ReLU wyzerowuje połowę gradientów (dla wejść ujemnych), dlatego wariancja wag jest większa niż w inicjalizacji Glorota (zależna tylko od n, a nie od $n + m$), co pomaga zachować odpowiednią propagację wejść i gradientu w głębokich sieciach wykorzystujących ReLU.

Przesunięcia $\mathbf{b}$ są zazwyczaj inicjalizowane zerami, choć w przypadku sieci z aktywacją ReLU czasem stosuje się małe wartości dodatnie (np. 0.01), aby uniknąć problemu „martwych neuronów" na początku treningu.

3.2 Funkcja straty

W procesie uczenia sieci neuronowej kluczowym elementem jest ocena tego, jak dobrze sieć przewiduje wyniki. Służy do tego funkcja straty [10] (określana również jako funkcja kosztu), która mierzy różnicę między przewidywaniami modelu a rzeczywistymi wartościami. Pełni ona rolę numerycznego wskaźnika, informującego, jak daleko sieć znajduje się od pożądanego zachowania.

Formalnie, dla zbioru treningowego składającego się z par $(\mathbf{x}, \mathbf{y})$, gdzie $\mathbf{x}$ to dane wejściowe, a $\mathbf{y}$ to oczekiwane wyjście, funkcja straty $\mathcal{L}$ określa jakość przewidywań sieci:

$$\mathbf{x} \circ \mathbf{f}^{(1)} \circ \mathbf{f}^{(2)} \circ \cdots \circ \mathbf{f}^{(L)} = \hat{\mathbf{y}},$$
$$\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) \to \mathbb{R}^+,$$

gdzie $f^{(i)}$ to kolejne warstwy sieci, a $\hat{\mathbf{y}}$ to predykcja sieci na podstawie danych wejściowych $\mathbf{x}$.

W praktyce wybór konkretnej funkcji straty zależy od typu rozwiązywanego zadania. Dla problemów regresji, gdzie celem jest przewidywanie wartości ciągłych, standardowym wyborem jest błąd średniokwadratowy (ang. *Mean Squared Error*, MSE) [10, s. 10]:

$$\mathcal{L}_{MSE}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2.$$

MSE silnie „karze" duże odchylenia od wartości rzeczywistej ze względu na swoją kwadratową naturę. Jest szczególnie użyteczny, gdy pojedyncze duże błędy są bardziej kosztowne niż wiele małych odchyleń.

Z kolei w zadaniach klasyfikacji, gdzie celem jest przypisanie danych do określonych kategorii, powszechnie stosowana jest entropia krzyżowa (ang. *Cross Entropy*, CE) [10, s. 14]:

$$\mathcal{L}_{CE}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{i=1}^{c} y_i \log(\hat{y}_i),$$

gdzie c oznacza liczbę klas, a $\hat{y}_i$ i y_i reprezentują odpowiednio przewidywane i rzeczywiste prawdopodobieństwa przynależności do klasy i. Warto zauważyć, że w praktyce wzór ten jest poprawnie określony, ponieważ w zadaniach wykorzystujących entropię krzyżową ostatnią warstwą sieci jest zwykle funkcja aktywacji sigmoid (dla klasyfikacji binarnej) lub softmax (dla klasyfikacji wieloklasowej), które gwarantują, że wartości $\hat{y}_i$ będą należeć do przedziału $[0, 1]$. Entropia krzyżowa jest szczególnie skuteczna w zadaniach klasyfikacji, ponieważ przypisuje wysokie wartości w przypadku dużej rozbieżności między przewidywaniami o wysokiej pewności a wartościami rzeczywistymi.

Wartość funkcji straty można interpretować jako „koszt" błędnych przewidywań modelu – im niższa wartość, tym mniejsza rozbieżność między przewidywaniami a wartościami oczekiwanymi. W procesie treningu dążymy do minimalizacji tej funkcji, systematycznie dostosowując parametry sieci tak, aby generowane przewidywania były jak najbliższe wartościom oczekiwanym.

3.3 Algorytm wstecznej propagacji błędu

Proces uczenia sieci można podzielić na dwa główne etapy: propagację w przód, gdzie dane przepływają przez sieć generując przewidywania, oraz propagację wstecz, w której obliczane są gradienty niezbędne do aktualizacji parametrów. Algorytm wstecznej propagacji błędu jest fundamentalnym elementem procesu trenowania sieci [11].

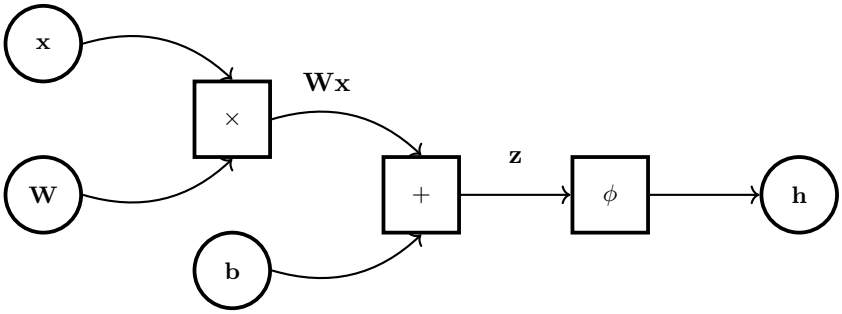
Warto podkreślić, że algorytm wstecznej propagacji odnosi się wyłącznie do metody obliczania gradientów, a nie całego procesu uczenia. Sam proces optymalizacji parametrów realizowany jest przez oddzielny algorytm optymalizacji zwany optymalizatorem. Przykłady najczęściej wykorzystywanych optymalizatorów zostaną przedstawione w dalszej części pracy.

3.3.1 Graf obliczeniowy

Graf obliczeniowy [6, s. 205-206] stanowi reprezentację sieci neuronowej w formie acyklicznego grafu skierowanego. Umożliwia on precyzyjne opisanie algorytmu propagacji wstecz.

Struktura grafu obliczeniowego składa się z węzłów oraz krawędzi. Węzły grafu reprezentują operacje matematyczne (jak dodawanie, mnożenie czy funkcje aktywacji) lub zmienne mogące przechowywać tensory. Krawędzie natomiast reprezentują przepływ danych między operacjami, określając zależności między węzłami oraz wskazując argumenty poszczególnych funkcji.

Wartości w węzłach są obliczane sekwencyjnie, warstwa po warstwie, w taki sposób, że obliczenia dla danego węzła rozpoczynają się dopiero, gdy wszystkie jego dane wejściowe są już dostępne. Graf obliczeniowy przedstawiony na Rysunku 9 obrazuje proces propagacji w przód dla pojedynczej warstwy neuronowej, gdzie kolejne węzły reprezentują operacje matematyczne przekształcające wektor wejściowy $\mathbf{x}$ poprzez mnożenie macierzowe z wagami $\mathbf{W}$, dodanie przesunięć $\mathbf{b}$ i aplikację funkcji aktywacji ϕ, prowadząc do uzyskania wektora wyjściowego $\mathbf{h}$.



Rysunek 9: Wizualizacja propagacji w przód grafu obliczeniowego warstwy neuronowej. Źródło: opracowanie własne.

3.3.2 Działanie algorytmu

Istotą algorytmu jest efektywne zastosowanie reguły łańcuchowej do obliczenia gradientów funkcji straty względem wszystkich parametrów sieci. Algorytm działa w dwóch fazach:

- Propagacja w przód – obliczanie wartości w kolejnych wierzchołkach grafu, aż do uzyskania wartości funkcji straty.
- Propagacja wstecz – obliczanie gradientów, rozpoczynając od funkcji straty o przechodząc wstecz przez graf.

Dla sieci L-warstwowej z parametrami $\mathbf{W}^{(i)}$ i $\mathbf{b}^{(i)}$ oraz funkcją straty $\mathcal{L}$, gdzie $\mathbf{x}$ oznacza wektor wejściowy, a $\mathbf{y}$ to wektor wartości docelowych, propagacja w przód przebiega następująco:

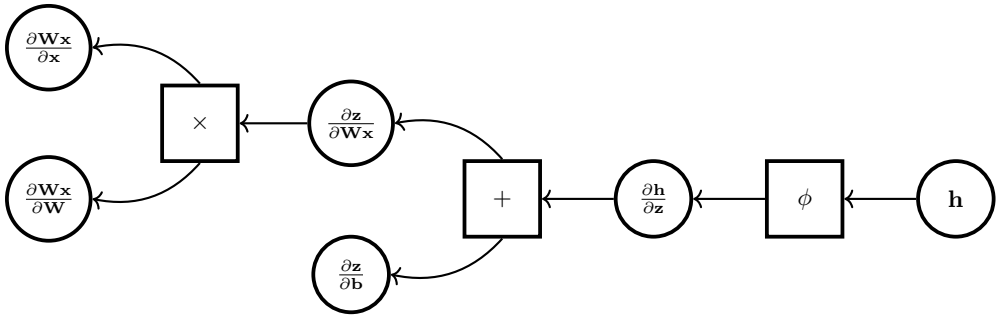
$$\begin{aligned}
\mathbf{h}^{(0)} &= \mathbf{x}, \\
\mathbf{z}^{(i)} &= \mathbf{W}^{(i)}\mathbf{h}^{(i-1)} + \mathbf{b}^{(i)}, \\
\mathbf{h}^{(i)} &= \phi^{(i)}(\mathbf{z}^{(i)}), \\
\ell &= \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}), \\
\mathbf{h}^{(L)} &= \hat{\mathbf{y}},
\end{aligned}$$

gdzie $i = 1, \ldots, L$. Następnie, w fazie propagacji wstecz, gradienty są obliczane rekurencyjnie:

$$\begin{aligned}
\frac{\partial \ell}{\partial \mathbf{z}^{(i)}} &= \frac{\partial \ell}{\partial \mathbf{h}^{(i)}} \odot \phi^{(i)'}(\mathbf{z}^{(i)}), \\
\frac{\partial \ell}{\partial \mathbf{W}^{(i)}} &= \frac{\partial \ell}{\partial \mathbf{z}^{(i)}} (\mathbf{h}^{(i-1)})^T, \\
\frac{\partial \ell}{\partial \mathbf{b}^{(i)}} &= \frac{\partial \ell}{\partial \mathbf{z}^{(i)}}, \\
\frac{\partial \ell}{\partial \mathbf{h}^{(i-1)}} &= (\mathbf{W}^{(i)})^T \frac{\partial \ell}{\partial \mathbf{z}^{(i)}},
\end{aligned}$$

gdzie $\odot$ oznacza mnożenie elementowe.

Proces propagacji wstecznej zilustrowano na Rysunku 10 gdzie widać przepływ gradientów przez graf obliczeniowy warstwy neuronowej, pokazując jak błąd propaguje się wstecz od wyjścia warstwy, przez funkcję aktywacji, aż do parametrów $\mathbf{W}$ i $\mathbf{b}$ zgodnie z regułą łańcuchową.



Rysunek 10: Wizualizacja propagacji wstecz dla grafu obliczeniowego warstwy neuronowej. Źródło: opracowanie własne.

Kluczową zaletą algorytmu wstecznej propagacji jest efektywność obliczeniowa. Dla sieci o n parametrach, złożoność obliczeniowa wynosi $O(n)$. Podczas gdy naiwna implementacja reguły łańcuchowej mogłaby prowadzić do złożoności wykładniczej ze względu na wielokrotne obliczanie tych samych podwyrażeń.

3.3.3 Optymalizator

Optymalizator jest kluczowym elementem procesu trenowania sieci, odpowiedzialnym za aktualizację parametrów modelu na podstawie gradientów obliczonych za pomocą algorytmu propagacji wstecznej. W praktyce stosuje się różne algorytmy optymalizacji, które różnią się sposobem wykorzystania informacji o gradientach do modyfikacji wag i przesunięć sieci.

Najprostszym i fundamentalnym algorytmem jest metoda gradientu prostego. Polega ona na modyfikacji parametrów sieci (zarówno wag w, jak i wyrazów wolnych b, które dalej będziemy oznaczać ogólnie jako θ) w kierunku przeciwnym do gradientu funkcji straty. Dla parametru θ aktualizacja ma postać

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L},$$

gdzie η jest współczynnikiem uczenia.

W praktyce, ze względu na dużą ilość danych treningowych wymaganych do treningu sieci, stosuje się wariant zwany stochastycznym gradientem prostym (SGD). W metodzie tej gradient jest aproksymowany na podstawie losowo wybranej porcji danych treningowych, zwanej mini-batchem. Dokładny opis procesu przetwarzania danych w mini-batchach zostanie przedstawiony w kolejnej sekcji.

Istotnym rozszerzeniem SGD jest technika momentum [12]. Wprowadza ona pojęcie „pędu", które można interpretować jako prędkość w przestrzeni parametrów. Aktualizacja przyjmuje postać:

$$\begin{aligned} v &\leftarrow \alpha v - \eta \nabla_\theta \mathcal{L}, \\ \theta &\leftarrow \theta + v, \end{aligned}$$

gdzie $\alpha \in [0, 1)$ jest współczynnikiem momentum, a v jest „prędkością" inicjalizowaną jako 0. Metoda ta pomaga w zbieżności, szczególnie w przypadku funkcji o wysokiej krzywiźnie.

Nowsze podejścia do optymalizacji wykorzystują adaptacyjne współczynniki uczenia dla poszczególnych parametrów. Jednym z takich algorytmów jest AdaGrad [13], który skaluje współczynniki odwrotnie proporcjonalnie do pierwiastka sumy kwadratów historycznych gradientów:

$$\begin{aligned} r &\leftarrow r + (\nabla_\theta \mathcal{L})^2, \\ \theta &\leftarrow \theta - \frac{\eta}{\sqrt{r + \epsilon}} \nabla_\theta \mathcal{L}, \end{aligned}$$

gdzie r to skumulowana suma kwadratów gradientów inicjowana jako 1, a ϵ to mała stała dla stabilności numerycznej.

RMSProp [14] modyfikuje AdaGrad, zastępując sumę wykładniczo ważoną średnią ruchomą, co pozwala na lepsze działanie w przypadku funkcji wypukłych. Aktualizacja parametrów w RMSProp ma postać:

$$r \leftarrow \rho r + (1-\rho)(\nabla_\theta \mathcal{L})^2,$$
$$\theta \leftarrow \theta - \frac{\eta}{\sqrt{r+\epsilon}} \nabla_\theta \mathcal{L},$$

gdzie ρ jest współczynnikiem zaniku.

Obecnie jednym z najpopularniejszych algorytmów jest Adam [15], łączący cechy RMSProp i momentum. Adam wykorzystuje estymatory pierwszego i drugiego momentu gradientu z korektą obciążenia:

$$m \leftarrow \beta_1 m + (1-\beta_1)\nabla_\theta \mathcal{L},$$
$$v \leftarrow \beta_2 v + (1-\beta_2)(\nabla_\theta \mathcal{L})^2,$$
$$\hat{m} \leftarrow \frac{m}{1-\beta_1^t},$$
$$\hat{v} \leftarrow \frac{v}{1-\beta_2^t},$$
$$\theta \leftarrow \theta - \eta \frac{\hat{m}}{\sqrt{\hat{v}+\epsilon}},$$

gdzie t to indeks kroku, a β_1 i β_2 są współczynnikami eksponencjalnego zaniku dla estymatorów momentów - β_1 kontroluje zanik pierwszego momentu, a β_2 drugiego momentu. Wartości te należą do przedziału $[0, 1)$, ale typowo przyjmuje się $\beta_1 = 0.9$ oraz $\beta_2 = 0.999$. Dzięki nim algorytm łączy zalety adaptacyjnego doboru współczynnika uczenia (poprzez β_2) oraz momentum (poprzez β_1).

Wybór odpowiedniego optymalizatora jest istotną kwestią, gdyż różne algorytmy wykazują zróżnicowaną skuteczność w zależności od charakteru problemu. Badania porównawcze [16] sugerują, że w przypadku algorytmów adaptacyjnych proces dostrajania hiperparametrów jest mniej czasochłonny niż dla klasycznych wariantów SGD. Jednakże, dobrze dostrojony algorytm SGD jest trudny do pokonania pod względem wydajności na większości testowanych przypadków.

3.4 Mini-batche

Podczas trenowania sieci neuronowej rzadko przetwarza się cały zbiór danych na raz. Zamiast tego, dane dzielone są na mniejsze porcje zwane mini-batchami (lub po prostu batchami).

Trenowanie z wykorzystaniem mini-batchy oferuje kilka istotnych zalet:

- **Efektywność obliczeniowa** - współczesne architektury komputerowe i procesory graficzne są zoptymalizowane pod kątem równoległego przetwarzania danych. Mini-batche pozwalają efektywnie wykorzystać te możliwości poprzez jednoczesne przetwarzanie wielu obserwacji.
- **Stabilność gradientów** - gradienty obliczane na podstawie mini-batcha są bardziej stabilne niż te obliczane dla pojedyńczych obserwacji, co przekłada się na bardziej stabilny proces uczenia.
- **Regularyzacja** - wprowadzenie losowości w procesie doboru przykładów do mini-batcha działa jako forma regularyzacji, pomagając w lepszej generalizacji modelu. Każda aktualizacja parametrów opiera się na innym, losowym podzbiorze danych treningowych.

W praktyce, gdy stosujemy podejście oparte na mini-batchach, wszystkie operacje w sieci muszą uwzględniać jednoczesne przetwarzanie wielu obserwacji. Oznacza to, że reprezentacje danych na każdym etapie zyskują dodatkowy wymiar B, odpowiadający rozmiarowi batcha. Na przykład, wektor wejściowy $\mathbf{x} \in \mathbb{R}^n$ staje się macierzą $\mathbf{x} \in \mathbb{R}^{B \times n}$, gdzie każdy wiersz reprezentuje osobny przykład z batcha. Podobnie w przypadku danych o większej liczbie wymiarów, jak obrazy reprezentowane przez tensory $\mathsf{X} \in \mathbb{R}^{H \times W \times C}$ (gdzie H, W, C oznaczają odpowiednio wysokość, szerokość i liczbę kanałów), otrzymujemy reprezentację $\mathsf{X} \in \mathbb{R}^{B \times H \times W \times C}$.

Dla zachowania przejrzystości zapisu i spójności z wcześniejszymi sekcjami, w dalszej części pracy będziemy konsekwentnie używać notacji wektorowej $\mathbf{x}$, mając na uwadze, że w rzeczywistej implementacji operacje wykonywane są równolegle na całych mini-batchach. Ta konwencja pozwoli skupić się na istotnych aspektach architektury i procesu uczenia, bez wprowadzania dodatkowej złożoności notacyjnej.

3.5 Hiperparametry

Innymi kluczowymi elementami, które mają wpływ na wydajność i proces uczenia sieci neuronowych, są hiperparametry [17], które w odróżnieniu od zwykłych parametrów muszą zostać określone przed rozpoczęciem treningu.

3.5.1 Rodzaje hiperparametrów

Do najważniejszych hiperparametrów sieci neuronowej należą:

- Liczba warstw ukrytych – określa głębokość architektury sieci.
- Liczba neuronów w warstwach ukrytych – definiuje złożoność poszczególnych warstw.
- Typ warstw – określa rodzaj transformacji danych w sieci.
- Funkcje aktywacji w poszczególnych warstwach – wprowadzają nieliniowość do modelu.
- Współczynnik uczenia (η) – kontroluje wielkość kroku optymalizatora.
- Liczba epok – określa ile razy model przechodzi przez cały zbiór treningowy.
- Rozmiar mini-batcha – liczba przykładów przetwarzanych jednocześnie w jednej iteracji.
- Dropout – określa prawdopodobieństwo „wyłączenia" aktywacji podczas treningu (więcej o tym w przyszłej sekcji).

3.5.2 Wpływ hiperparametrów na wydajność

Dobór hiperparametrów może znacząco wpłynąć na działanie sieci. Na przykład:

- **Zbyt wysoki współczynnik uczenia** skutkuje za dużymi aktualizacjami parametrów i uniemożliwia zbieżność.
- **Zbyt niski współczynnik uczenia** sprawia, że trening jest bardzo wolny i może utknąć w minimum lokalnym.
- **Zbyt mała liczba neuronów** prowadzi do niedopasowania modelu.
- **Zbyt duża liczba neuronów lub warstw** zwiększa ryzyko nadmiernego dopasowania i wymaga stosowania dodatkowej regularyzacji.

4 Wprowadzanie do problemu

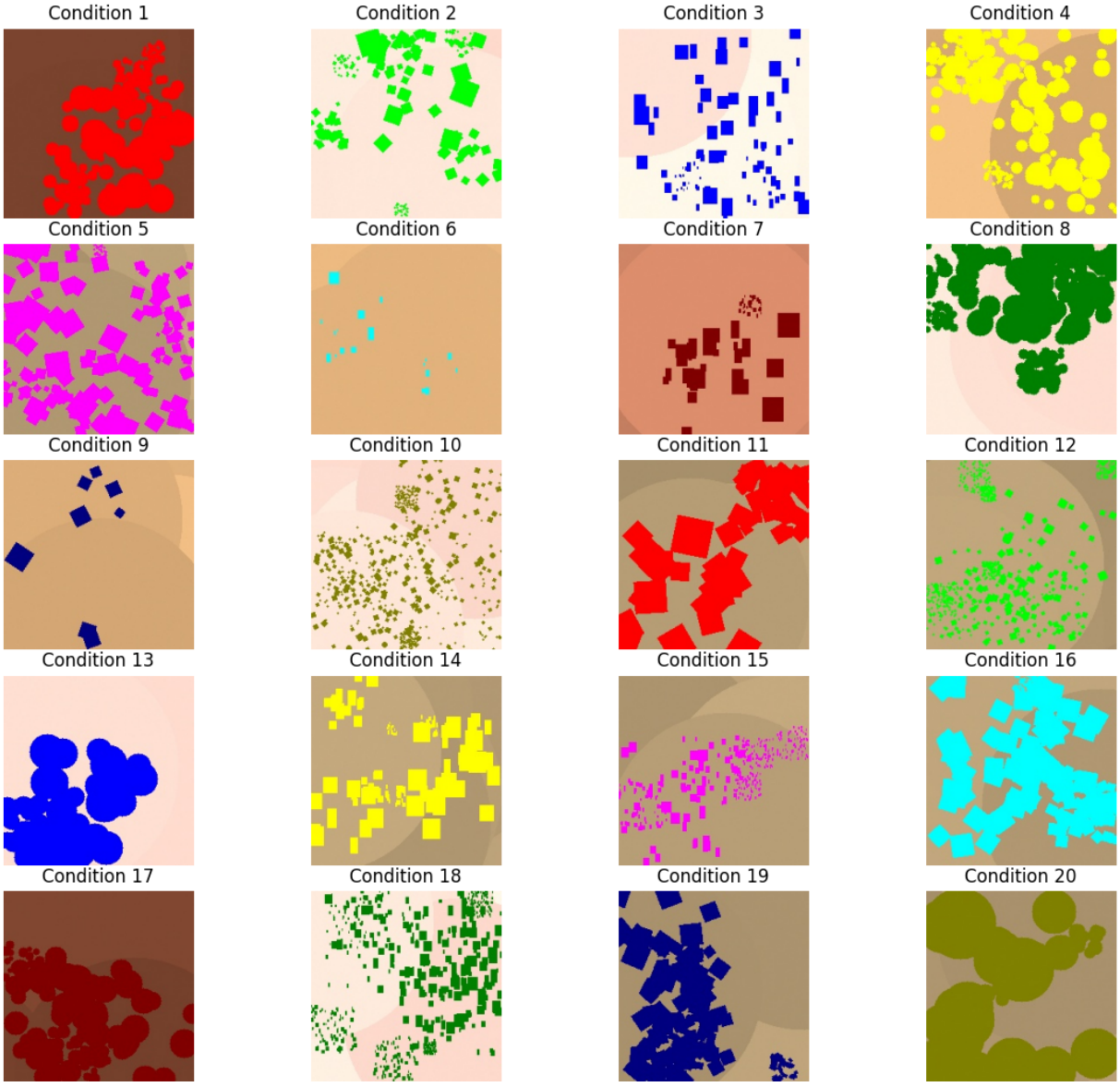
W poprzednich rozdziałach szczegółowo omówiono teoretyczne podstawy działania sieci neuronowych. Przedstawione koncepcje, choć fundamentalne dla zrozumienia działania sieci, nabierają prawdziwego znaczenia dopiero w kontekście konkretnych zastosowań. Aby zilustrować praktyczne możliwości opisanych architektur, w dalszej części pracy skupimy się na dwóch uzupełniających się problemach klasyfikacji obrazów.

Pierwszy z nich polega na klasyfikacji syntetycznie generowanych obrazów o ściśle kontrolowanych parametrach. Problem ten, choć uproszczony w stosunku do rzeczywistych zastosowań, pozwala na systematyczną analizę zdolności sieci do wykrywania i różnicowania określonych cech wizualnych. Wykorzystanie danych syntetycznych daje możliwość systematycznej weryfikacji zdolności sieci do wykrywania kluczowych cech obrazu, zanim zostanie ona zastosowana w rzeczywistej diagnostyce.

Drugim problemem jest klasyfikacja rzeczywistych zdjęć zmian skórnych, reprezentująca praktyczne zastosowanie sieci w diagnostyce medycznej. W przeciwieństwie do danych historycznych, rzeczywiste obrazy medyczne charakteryzują się znaczną zmiennością wynikającą z różnic anatomicznych, warunków oświetlenia czy jakości rejestracji. Ten problem demonstruje zdolność sieci do generalizacji nabytej wiedzy w warunkach rzeczywistych, gdzie kluczowa jest nie tylko precyzja klasyfikacji, ale również odporność na zakłócenia.

4.1 Dane syntetyczne

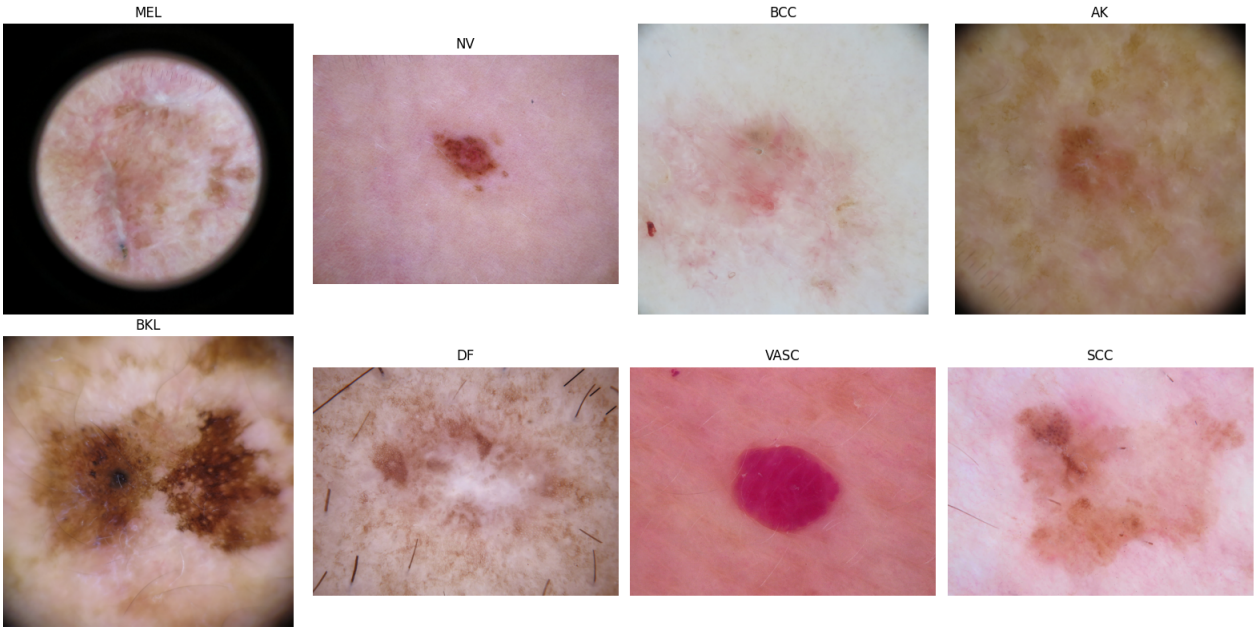
Syntetyczny zbiór danych składa się z 21 klas, z których 20 reprezentuje różne typy zmian skórnych, a jedna klasa odpowiada przypadkom bez zmian chorobowych. W skład zbioru wchodzi 6000 obrazów treningowych oraz 2000 obrazów walidacyjnych, z dobrze zbalansowanym rozkładem klas - 30% rekordów reprezentuje przypadki bez zmian chorobowych, podczas gdy pozostałe 70% jest równomiernie rozdzielone między 20 klas chorób. Na Rysunku 11 przedstawiono przykłady wszystkich klas ze zmianami chorobowymi. Każdy obraz przedstawia unikalną kombinację cech charakterystycznych, takich jak kolor zmian (np. czerwone w wariancie 1, zielone w wariancie 2, czy niebieskie w wariancie 3), kształt (od regularnych kwadratów po nieregularne plamy), a także różną liczbę i gęstość występowania zmian na powierzchni skóry. Ta różnorodność pozwala na systematyczne testowanie zdolności sieci do rozpoznawania i różnicowania kluczowych cech wizualnych występujących w diagnostyce dermatologicznej.



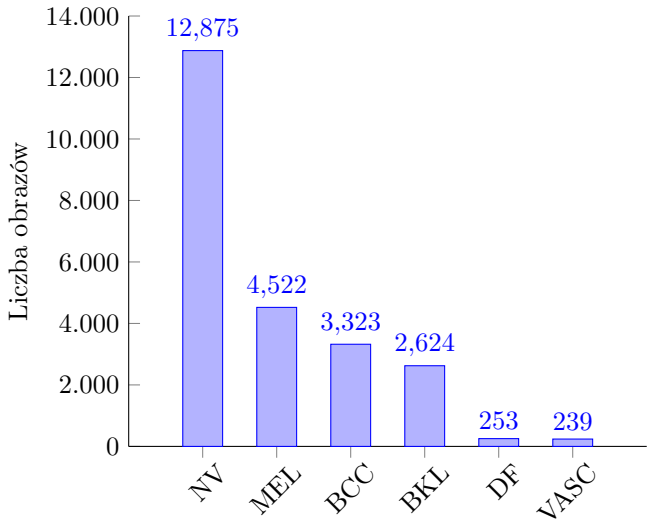
Rysunek 11: Przykładowe zdjęcia dla każdej z 20 klas chorobowych ze zbioru danych syntetycznych. Źródło: opracowanie własne

4.2 Dane rzeczywiste

Drugi zbiór danych zawiera rzeczywiste zdjęcia zmian skórnych, obejmując osiem różnych stanów chorobowych, przedstawionych na Rysunku 12. Zbiór liczy łącznie 25331 obrazów treningowych oraz 8238 obrazów testowych. Jak pokazano na Rysunku 13, rozkład klas w zbiorze treningowym charakteryzuje się znaczącym niezbalansowaniem, odzwierciedlającym rzeczywiste proporcje występowania poszczególnych chorób w praktyce klinicznej. Dominują znamiona melanocytowe (NV, 12875 obrazów), stanowiące ponad 50% wszystkich przypadków, następnie czerniak (MEL, 4522 obrazów), rak podstawnokomórkowy (BCC, 3323 obrazów) oraz brodawka łojotokowa (BKL, 2624 obrazów). Najrzadziej reprezentowane są naczyniaki (VASC, 239 obrazów) oraz przypadki dermatofibromy (DF, 253 obrazów), których liczebność jest niemal 50-krotnie mniejsza od klasy dominującej.

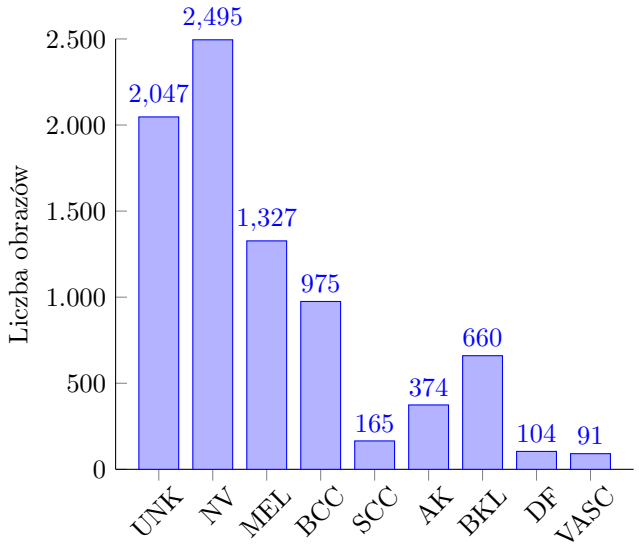


Rysunek 12: Przykładowe zdjęcia dla każdej z 8 klas chorobowych ze zbioru danych rzeczywistych. Źródło danych: [18–21].



Rysunek 13: Rozkład klas w zbiorze treningowym. Źródło: opracowanie własne.

W zbiorze testowym, którego rozkład przedstawiono na Rysunku 14, wprowadzono dodatkową klasę przypadków niesklasyfikowanych (2047 obrazów), stanowiącą około 25% całego zbioru. Pozostałe przypadki zachowały proporcje podobne do zbioru treningowego: znamiona melanocytowe (2495 obrazów), czerniak (1327 obrazów), rak podstawnokomórkowy (975 obrazów), rogowacenie słoneczne (374 obrazów), brodawka łojotokowa (660 obrazów), naczyniak (91 obrazów), dermatofibroma (104 obrazów) oraz rak kolczystokomórkowy (165 obrazów). Istotnym aspektem tego zbioru jest brak reprezentacji klasy "niesklasyfikowany"w danych treningowych. Wymaga to od sieci dodatkowej zdolności do generalizacji wykraczającej poza wprost nauczone wzorce – sieć musi nie tylko poprawnie sklasyfikować znane sobie przypadki, ale również rozpoznawać sytuacje, gdy prezentowana zmiana nie pasuje do żadnej ze znanych kategorii.



Rysunek 14: Rozkład klas w zbiorze testowym. Źródło: opracowanie własne.

5 Opis architektury

Przedstawione wcześniej dwa zbiory danych - syntetyczny oraz rzeczywisty - stawiają przed siecią neuronową odmienne wyzwania. O ile dane syntetyczne pozwalają na systematyczne testowanie podstawowych zdolności sieci do rozpoznawania wzorców wizualnych, o tyle klasyfikacja rzeczywistych obrazów medycznych wymaga znacznie bardziej zaawansowanych mechanizmów przetwarzania. Sieć musi być zdolna do identyfikacji subtelnych różnic w teksturze, kolorze i strukturze zmian chorobowych, które często objawiają się jedynie w ograniczonych obszarach obrazu. Dodatkowo, skuteczna diagnostyka musi uwzględniać nie tylko różnorodność skal i orientacji zmian chorobowych, ale także radzić sobie z maskowaniem cech diagnostycznych przez naturalne zróżnicowanie wyglądu skóry pacjentów oraz zmienne warunki rejestracji obrazu.

W odpowiedzi na te wymagania, w pracy wykorzystano architekturę EfficientNet [22], reprezentującą rodzinę nowoczesnych sieci konwolucyjnych. W przeciwieństwie do wcześniejszych propozycji architektur z lat 2015-2018, które zwiększały dokładność kosztem znaczącego wzrostu liczby parametrów, EfficientNet wprowadza innowacyjne podejście skalowania wielowymiarowego. Dodatkowo, wykorzystanie złożonych bloków konstrukcyjnych, takich jak Mobile Inverted Bottleneck Convolution (MBConv) oraz Squeeze-and-Excitation, umożliwia efektywną ekstrakcję istotnych cech obrazu przy zachowaniu zwartej struktury modelu.

W kolejnych sekcjach szczegółowo zostaną omówione poszczególne elementy składowe architektury, rozpoczynając od fundamentalnych konceptów, jak warstwy konwolucyjne, poprzez mechanizmy normalizacji i regularyzacji, kończąc na wspomnianych wcześniej blokach konstrukcyjnych.

5.1 Warstwy konwolucyjne

Warstwy konwolucyjne [23] stanowią fundamentalny element sieci wykorzystywanych w analizie obrazów, wprowadzając do architektury zdolność do lokalnego przetwarzania informacji. W przeciwieństwie do warstw gęstych, które przetwarzają obraz jako jednowymiarowy wektor wartości, warstwy konwolucyjne zachowują przestrzenną strukturę danych, umożliwiając efektywne wykrywanie wzorców niezależnie od ich położenia w obrazie.

Warstwę konwolucyjną można przedstawić jako złożenie operatorów:

$$\mathsf{X} \circ \mathrm{Conv} \circ \phi = \mathsf{H},$$

gdzie Conv oznacza operator konwolucji:

$$\mathrm{Conv} : \mathbb{R}^{H \times W \times C} \to \mathbb{R}^{H' \times W' \times K},$$

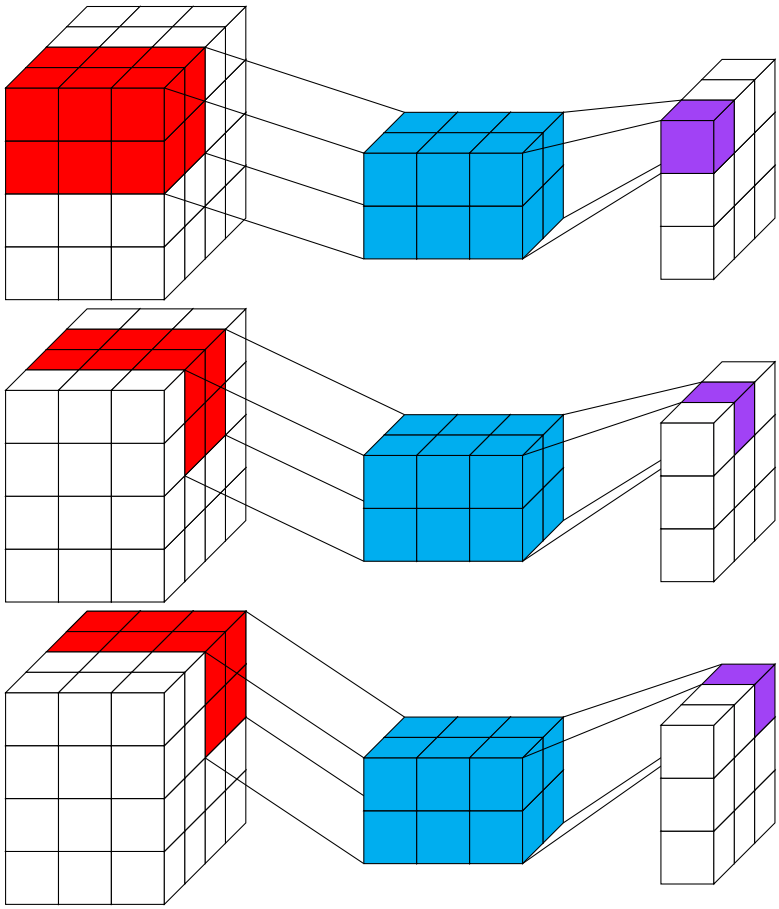
transformujący tensor wejściowy X o wymiarach wysokości H, szerokości W i C kanałach na tensor wyjściowy o wymiarach H', W' i K kanałach, natomiast ϕ oznacza nieliniową funkcję aktywacji.

Operator konwolucji realizuje operację splatania lokalnych regionów obrazu K filtrami konwolucji (nazywanymi również jądrami konwolucji) $\{\mathsf{W}^{(1)}, \ldots, \mathsf{W}^{(K)}\} \in \mathbb{R}^{k \times k \times C}$, gdzie k oznacza rozmiar przestrzenny filtru. Na Rysunku 15 zobrazowano, jak filtr konwolucyjny (niebieski) przesuwa się po tensorze wejściowym, przetwarza lokalne regiony (czerwone) i generuje odpowiednie elementy tensora wyjściowego (fioletowe). Operacja ta dla pojedynczego elementu tensora wyjściowego Z przyjmuje postać:

$$z_{h,w,i} = \sum_{c=1}^{C} \sum_{m=1}^{k} \sum_{n=1}^{k} x_{h+m-1,w+n-1,c} \cdot w_{m,n,c}^{(i)},$$

gdzie indeksy h, w określają pozycję przestrzenną, a i oznacza numer kanału wyjściowego (odpowiadający i-temu filtrowi). Następnie, do otrzymanego tensora Z aplikowana jest funkcja aktywacji ϕ, dając końcowy tensor wyjściowy H:

$$\mathsf{H} = \phi(\mathsf{Z}).$$



Rysunek 15: Wizualizacja procesu przesuwania filtru i tworzenia mapy cech w procesie konwolucji. Źródło: opracowanie własne.

Działanie warstwy jest modyfikowane przez dwa kluczowe parametry:

- Krok przesunięcia (ang. *stride*, oznaczany jako s) - definiuje odległość, o jaką przesuwany jest filtr między kolejnymi obliczeniami.
- Dopełnienie zerami (ang. *padding*, oznaczany jako p) - pozwala zachować wymiary przestrzenne poprzez dodanie zer na brzegach obrazu wejściowego.

Wymiary tensora wyjściowego $\mathsf{H} \in \mathbb{R}^{H' \times W' \times K}$ można obliczyć jako:

$$H' = \left\lfloor \frac{H - k + 2p}{s} \right\rfloor + 1,$$
$$W' = \left\lfloor \frac{W - k + 2p}{s} \right\rfloor + 1$$

gdzie $\lfloor \cdot \rfloor$ oznacza zaokrąglenie w dół do najbliższej liczby całkowitej.

Warto zauważyć, że warstwa konwolucyjna wprowadza do sieci dwie kluczowe właściwości:

- Lokalność przetwarzania – każdy element tensora wyjściowego zależy tylko od małego regionu wejścia.
- Współdzielenie parametrów – te same filtry są używane dla wszystkich lokalizacji w obrazie.

W kontekście klasyfikacji zmian skórnych, warstwy konwolucyjne odgrywają hierarchiczną rolę w ekstrakcji cech charakterystycznych dla różnych jednostek chorobowych. Warstwy początkowe specjalizują się w wykrywaniu podstawowych wzorców wizualnych, takich jak krawędzie czy zmiany tekstury, pod-

czas gdy głębsze łączą te elementarne cechy w celu rozpoznawania złożonych struktur specyficznych dla poszczególnych typów zmian chorobowych.

5.2 Warstwy agregujące

Warstwy agregacyjne (w literaturze znane również jako warstwy poolujące) służą do redukcji wymiarowości map aktywacji wyjściowych z warstw konwolucyjnych poprzez zmniejszenie rozmiaru przestrzennego przy jednoczesnym zachowaniu najistotniejszych cech. Oprócz kompresji danych, operacja agregacji wprowadza częściową niezmienniczość ze względu na translację, co oznacza zmniejszenie wrażliwości sieci na małe przesunięcia w obrazie wejściowym. Operację poolingu można zapisać jako:

$$\mathsf{X} \circ \mathrm{Pool} = \mathsf{H},$$

gdzie Pool oznacza operator agregacji:

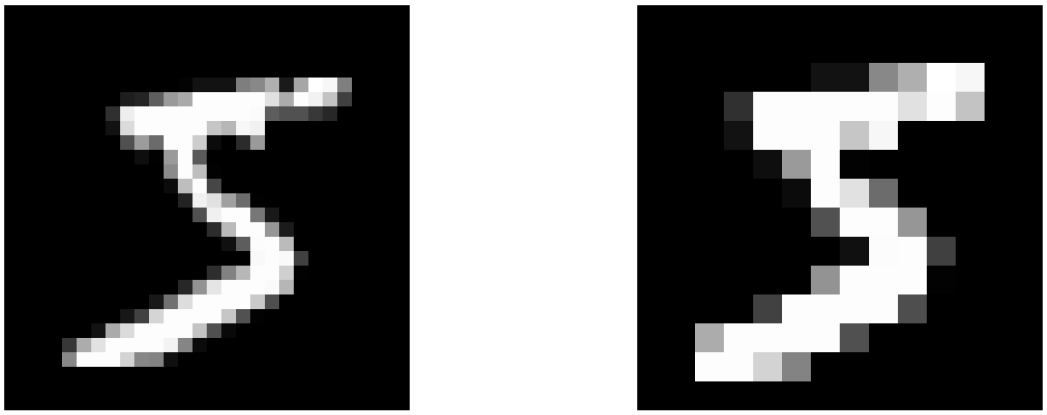
$$\mathrm{Pool} : \mathbb{R}^{H \times W \times C} \to \mathbb{R}^{H' \times W' \times C}$$

W przeciwieństwie do warstw konwolucyjnych, operacja poolingu nie zawiera parametrów podlegających uczeniu, wykorzystując zamiast tego predefiniowane operacje statystyczne. Dla tensora wejściowego X, rozmiaru okna $k \times k$ i przesunięcia s, podstawowe warianty definiuje się jako:

- Max pooling:

$$h_{h,w,c} = \max_{m,n \in \{1,\ldots,k\}} x_{h \cdot s+m-1, w \cdot s+n-1, c},$$

  gdzie indeksy h, w oznaczają pozycję przestrzenną w tensorze wyjściowym, a c oznacza kanał. Max pooling wybiera maksymalną wartość z zadanego regionu, co pozwala zachować wyraziste cechy, takie jak krawędzie czy dominujące elementy tekstur. Jak widać na Rysunku 16, operacja ta skutecznie zredukowała wymiary oryginalnego zdjęcia przy równoczesny zachowaniu najsilniejszych aktywacji, które często odpowiadają istotnym cechom wizualnym.



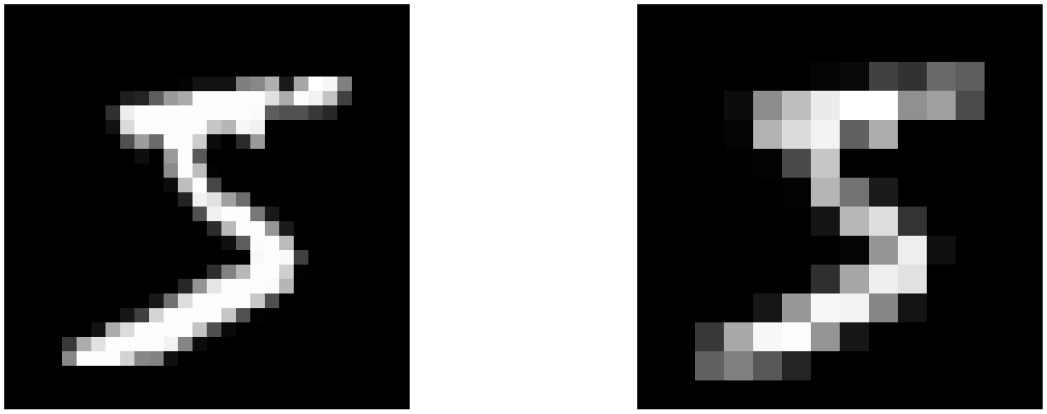
Rysunek 16: Porównanie oryginalnego obrazu przed (po lewej) i po (po prawej) zastosowaniu operacji max pooling. Źródło oryginalnego obrazu: [23], Źródło przetworzonego obrazu: opracowanie własne.

- Average pooling:

$$h_{h,w,c} = \frac{1}{k^2} \sum_{m=1}^{k} \sum_{n=1}^{k} x_{h \cdot s+m-1, w \cdot s+n-1, c},$$

  gdzie indeksy h, w oznaczają pozycję przestrzenną w tensorze wyjściowym, a c oznacza kanał. Average pooling oblicza średnią wartość z zadanego regionu, co prowadzi do zachowania większej ilości

informacji kontekstowych. Rysunek 17 pokazuje, że technika zachowuje informacje o ogólnej strukturze lub intensywności cech w danym obszarze, kosztem potencjalnego rozmycia wyrazistych detali.



Rysunek 17: Porównanie oryginalnego obrazu przed (po lewej) i po (po prawej) zastosowaniu operacji average pooling. Źródło przetworzonego obrazu: opracowanie własne.

Wymiary wyjściowe można obliczyć jako:

$$H' = \left\lfloor \frac{H-k}{s} \right\rfloor + 1,$$
$$W' = \left\lfloor \frac{W-k}{s} \right\rfloor + 1.$$

W typowych implementacjach najczęściej stosuje się max pooling z oknem 2×2 i krokiem $s = 2$, co prowadzi do dwukrotnej redukcji wymiarów przestrzennych przy zachowaniu liczby kanałów. Wybór między max a average poolingiem zależy od specyfiki zadania – max pooling sprawdza się lepiej w zadaniach klasyfikacji, gdzie istotne jest wykrywanie charakterystycznych cech, podczas gdy average pooling może być preferowany w zadaniach wymagających zachowania informacji o teksturach czy gradientach.

5.3 Normalizacja wsadowa

Normalizacja wsadowa [24] (ang. *batch normalization*) to technika wprowadzona w celu przyspieszenia treningu głębokich sieci neuronowych poprzez redukcję zjawiska przesunięcia wewnętrznej kowariancji. Problem polega na ciągłej zmianie rozkładu aktywacji w kolejnych warstwach sieci podczas procesu uczenia, co znacząco spowalnia trening i wymaga ostrożnego doboru hiperparametrów oraz inicjalizacji parametrów.

Warstwę normalizacji wsadowej BatchNorm można przedstawić jako złożenie operacji:

$$\mathsf{X} \circ \mathrm{Norm} \circ \mathrm{Scale} = \mathsf{H},$$

gdzie Norm oznacza operację normalizacji:

$$\mathrm{Norm} : \mathbb{R}^{H \times W \times C} \to \mathbb{R}^{H \times W \times C},$$

transformujący tensor wejściowy do postaci znormalizowanej poprzez standaryzację względem średniej i odchylenia standardowego mini-batcha, natomiast Scale oznacza operację skalowania:

$$\mathrm{Norm} : \mathbb{R}^{H \times W \times C} \to \mathbb{R}^{H \times W \times C}.$$

realizującą liniową transformację znormalizowanych wartości przy użyciu uczonych parametrów.

Operator normalizacji dla pojedynczego kanału c realizuje standaryzację:

$$\hat{x}_{h,w,c} = \frac{x_{h,w,c} - \mu_c}{\sigma_c + \epsilon},$$

gdzie, ϵ to niewielka stała, większa od 0 zachowująca stabilność numeryczną, a μ_c i σ_c oznaczają odpowiednio średnią i wariancję obliczaną dla całego mini-batcha w danym kanale c:

$$\mu_c = \frac{1}{HW} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{h,w,c},$$
$$\sigma_c = \sqrt{\frac{1}{HW} \sum_{h=1}^{H} \sum_{w=1}^{W} (x_{h,w,c} - \mu_c)^2}.$$

Następnie, operator skalowania realizuje transformację:

$$h_{h,w,c} = \gamma_c \hat{x}_{h,w,c} + \beta_c,$$

gdzie γ_c i β_c są uczonymi parametrami, inicjalizowanymi odpowiednio jako $\gamma_c = 1$ i $\beta_c = 0$. Parametry te umożliwiają sieci adaptację stopnia normalizacji, włącznie z możliwością jej całkowitego „cofnięcia", jeśli jest to korzystne dla procesu uczenia.

Warto zaznaczyć, że podczas fazy wnioskowania, zamiast bieżących statystyk mini-batcha, wykorzystywane są przybliżenia średniej i odchylenia standardowego, estymowane w trakcie treningu przy użyciu średniej ruchomej.

Ta technika normalizacji, choć koncepcyjnie prosta, przynosi szereg praktycznych korzyści w treningu głębokich sieci neuronowych. Stabilizacja rozkładu aktywacji pozwala na stosowanie większych współczynników uczenia, co może przyspieszyć proces treningu nawet 14-krotnie [24]. Dodatkowo, technika ta zmniejsza wrażliwość sieci na inicjalizację parametrów i pełni funkcję regularyzacyjną poprzez szum wprowadzany podczas estymacji statystyk.

5.4 Dropout

Dropout [25] to technika regularyzacji modyfikująca przepływ informacji w sieci poprzez losowe zerowanie aktywacji podczas procesu uczenia. W przeciwieństwie do klasycznych metod regularyzacji, które modyfikują funkcję straty, dropout wymusza na sieci uczenie się redundantnych reprezentacji poprzez losowe zakłócenie przepływu informacji.

Mechanizm ten może być zastosowany do dowolnego typu warstwy, działając zawsze na jej aktywacjach. W dalszej części skupimy się na przypadku warstw konwolucyjnych, gdzie danymi wyjściowymi są tensory.

Sekwencję transformacji z dropout można zapisać jako:

$$\mathsf{X} \circ \mathrm{Drop} = \tilde{\mathsf{X}},$$

gdzie Drop realizuje losowe zerowanie aktywacji z poprzedniej warstwy, następnie przekształcone aktywacje są przetwarzane przez warstwę konwolucyjną:

$$\tilde{\mathsf{X}} \circ \mathrm{Conv} \circ \phi = \mathsf{H}.$$

Operator Drop realizuje losowe zerowanie aktywacji poprzez generowanie maski binarnej zgodnej z rozkładem Bernoulliego. W przypadku warstw konwolucyjnych może to być zaimplementowane na dwa sposoby. W podejściu standardowym, dla każdego elementu tensora aktywacji zakłócenie realizowane jest jako:

$$r_{h,w,c} \sim \mathrm{Bernoulli}(p),$$
$$\tilde{x}_{h,w,c} = \frac{r_{h,w,c}}{p} x_{h,w,c},$$

gdzie p jest prawdopodobieństwem zachowania aktywacji, a współczynnik skalujący $\frac{r_{h,w,c}}{p}$ służy do zachowania odpowiedniej wartości oczekiwanej podczas treningu.

Alternatywnie, w wariancie przestrzennym, maska może być generowana na poziomie całych kanałów:

$$r_c \sim \text{Bernoulli}(p),$$
$$\tilde{x}_{h,w,c} = \frac{r_c}{p} x_{h,w,c},$$

gdzie r_c jest wspólne dla wszystkich pozycji przestrzennych w danym kanale.

Kluczową własnością dropout jest to, że działa na poziomie aktywacji, nie modyfikując bezpośrednio wag sieci. Zerowanie aktywacji w danej iteracji treningowej powoduje, że gradienty związane z wyzerowanymi aktywacjami nie wpływają na aktualizację odpowiadających im wag. W efekcie, w każdej iteracji efektywnie trenowany jest inny podzbiór połączeń sieci, co można interpretować jako równoczesne uczenie wielu pod-sieci współdzielących parametry. Ta własność przyczynia się do lepszej generalizacji modelu poprzez redukcję zjawiska przeuczenia.

5.5 Depthwise separable convolutions

Depthwise separable convolutions [26] stanowią wydajną modyfikację standardowych warstw konwolucyjnych, rozkładając operację konwolucji na dwa etapy: konwolucję wgłębną (ang. *depthwise convolution*) oraz konwolucję punktową (ang. *pointwise convolution*). Takie podejście znacząco redukuje liczbę parametrów i złożoność obliczeniową przy zachowaniu porównywalnej zdolności do ekstrakcji cech.

Warstwę typu depthwise separable convolution można przedstawić jako złożenie operacji:

$$\mathsf{X} \circ \text{DepthConv} \circ \text{PointConv} \circ \phi = \mathsf{H},$$

gdzie DepthConv oznacza operację konwolucji wgłębnej:

$$\text{DepthConv} : \mathbb{R}^{H \times W \times C} \to \mathbb{R}^{H' \times W' \times C},$$

realizując niezależne przetwarzanie każdego kanału wejściowego, natomiast PointConv oznacza operację konwolucji punktowej:

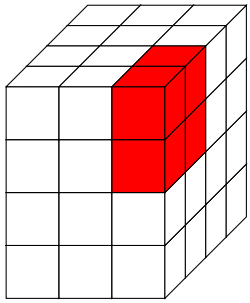
$$\text{PointConv} : \mathbb{R}^{H' \times W' \times C} \to \mathbb{R}^{H' \times W' \times K},$$

łączącą informacje między kanałami.

Operacja DepthConv dla pojedynczego elementu tensora wyjściowego Z' przyjmuje postać:

$$z'_{h,w,c} = \sum_{m=1}^{k} \sum_{n=1}^{k} x_{h+m-1,w+n-1,c} \cdot d^{(c)}_{m,n},$$

gdzie $\mathbf{D}^{(c)} \in \mathbb{R}^{k \times k}$ oznacza wgłębny filtr dla c-tego kanału. Jak pokazano na Rysunku 18, operacja ta przetwarza niezależnie każdy kanał wejściowy, wykorzystując dla niego filtr.

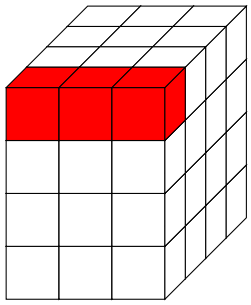


Rysunek 18: Wizualizacja filtru wykorzystywanego w operacji konwolucji wgłębnej. Źródło: opracowanie własne.

Następnie, operacja PointConv przetwarza wyniki konwolucji wgłębnej:

$$z_{h,w,k} = \sum_{c=1}^{C} z'_{h,w,c} \cdot p_c^{(i)},$$

gdzie, $\mathbf{P}^{(i)} \in \mathbb{R}^C$ oznacza i-ty filtr punktowy. Jak widać na Rysunku 19, operacja ta łączy informacje między kanałami poprzez konwolucję z filtrem o wymiarze 1×1.



Rysunek 19: Wizualizacja filtru wykorzystywanego w operacji konwolucji wgłębnej. Źródło: opracowanie własne.

Łączna liczba parametrów w depthwise separable convolution wynosi zatem $k^2 \cdot C + C \cdot K$. Zakładając standardowe wartości $k = 3$, $C = K = 256$, standardowa konwolucja wymagałaby $k^2 \cdot C \cdot K = 589824$ parametrów, podczas gdy depthwise separable convolution wymaga tylko $k^2 \cdot C + C \cdot K = 67840$ parametrów. Ta znacząca redukcja złożoności opiera się na założeniu, że przestrzenne korelacje w obrazie (wykrywane przez konwolucję wgłębną) mogą być modelowane niezależnie od korelacji między kanałami (przetwarzanymi przez konwolucję punktową).

Wbrew intuicji, redukcja liczby parametrów w depthwise separable convolutions nie prowadzi do znaczącego spadku dokładności. Badania empiryczne [26] pokazują, że w głębokich sieciach (powyżej 8 warstw) architektura oparta na tej modyfikacji może osiągać porównywalne lub nawet lepsze wyniki niż standardowe konwolucje.

5.6 Squeeze-and-Excitation

Tradycyjne warstwy konwolucyjne przetwarzają wszystkie kanały wejściowe z jednakową wagą, nie uwzględniając ich względnego znaczenia dla konkretnej obserwacji. Mechanizm squeeze-and-excitation (SE) [27], zobrazowany na Rysunku 20, adresuje to podejście, wprowadzając dynamiczne przeważenie kanałów w zależności od zawartości danych wejściowych. Pozwala to sieci na adaptacyjne modulowanie odpowiedzi poszczególnych filtrów konwolucyjnych, zwiększając jej zdolność do modelowania współzależności między kanałami.

Blok SE można przedstawić jako złożenie operacji:

$$\mathsf{X} \circ \text{Squeeze} \circ \text{Excite} \circ \text{Scale} = \mathsf{H},$$

gdzie Squeeze oznacza operację agregacji informacji przestrzennej:

$$\text{Squeeze} : \mathbb{R}^{H \times W \times C} \to \mathbb{R}^C.$$

Excite to operacja generowania wagi dla kanałów:

$$\text{Excite} : \mathbb{R}^C \to \mathbb{R}^C,$$

a Scale realizuje skalowanie kanałów:

$$\text{Scale} : (\mathbb{R}^{H \times W \times C}, \mathbb{R}^C) \to \mathbb{R}^{H \times W \times C}.$$

Operacja Squeeze wykorzystuje global average pooling do obliczenia statystyk kanałowych:

$$z_c = \frac{1}{HW} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{h,w,c},$$

gdzie z_c oznacza zagregowaną informację dla c-tego kanału.

Operacja Excite przetwarza zagregowane statystyki przez sekwencje dwóch warstw gęstych wraz z odpowiednimi funkcjami aktywacji:

$$\mathbf{z} \circ \mathbf{f}' \circ \phi' \circ \mathbf{f}'' \circ \phi'' = \mathbf{s},$$

gdzie $\phi' = \text{Swish}(\cdot)$ i $\phi'' = \sigma(\cdot)$ to funkcje aktywacji, $\mathbf{s}$ to wektor wag służących do skalowania kanałów, a parametry warstw gęstych $\mathbf{f}'$ i $\mathbf{f}''$ mają wymiary:

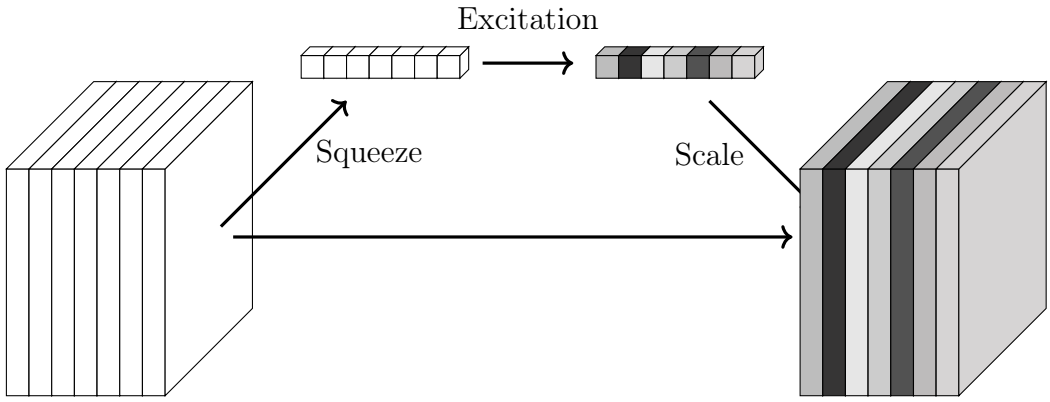
$$\mathbf{W}' \in \mathbb{R}^{(C/r)\times C}, \quad \mathbf{b}' \in \mathbb{R}^{C/r},$$
$$\mathbf{W}'' \in \mathbb{R}^{C\times(C/r)}, \quad \mathbf{b}'' \in \mathbb{R}^{C},$$

gdzie r jest hiperparametrem określającym stopień kompresji reprezentacji pośredniej. Pierwsza warstwa redukuje wymiarowość reprezentacji o współczynnik r, co zmniejsza liczbę parametrów i wymusza na sieci uczenie się bardziej ogólnych zależności między kanałami.

Ostatecznie, operator Scale realizuje rekalibrację tensora wejściowego:

$$h_{h,w,c} = s_c \cdot x_{i,j,c},$$

gdzie $s_c \in [0, 1]$ to waga przypisana c-temu kanałowi.



Rysunek 20: Schemat bloku Squeeze-and-Excitation ilustrujący sekwencję przekształceń: od agregacji informacji przestrzennej, przez generowanie wag, po końcowe przeskalowanie kanałów. Źródło: opracowanie własne na podstawie [27].

5.7 Połączenia rezydualne

Głębokie sieci neuronowe stają przed fundamentalnym wyzwaniem związanym z efektywnym przepływem informacji przez kolejne warstwy. W tradycyjnej architekturze każda warstwa przetwarza wyniki poprzedniej. Taka konstrukcja wymusza na każdej warstwie uczenie się kompletnej transformacji danych wejściowych. W głębokich sieciach prowadzi to do szeregu problemów: trudności w propagacji informacji przez wiele warstw, niestabilności procesu uczenia oraz degradacji dokładności – zjawiska, w którym dodawanie kolejnych warstw pogarsza, zamiast poprawiać wydajność sieci.

Połączenia rezydualne [28, 29] wprowadzają alternatywne podejście poprzez dodanie bezpośredniego połączenia między wejściem a wyjściem bloku, co zobrazowano na Rysunku 21. Zamiast uczenia się pełnej transformacji, sieć uczy się jedynie różnicy (rezyduum) między pożądaną transformacją a funkcją tożsamościową.

W dalszej części skupimy się na przypadku, gdy zarówno dane wejściowe, jak i wyjściowe są w postaci tensora, co jest typowe dla warstw konwolucyjnych. Blok rezydualny można przedstawić jako złożenie operacji:

$$\mathsf{X} \circ \text{Transform} \circ \text{AddSkip}[\mathsf{X}] = \mathsf{H},$$

gdzie Transform oznacza dowolną sekwencję transformacji realizowanych przez warstwy bloku:

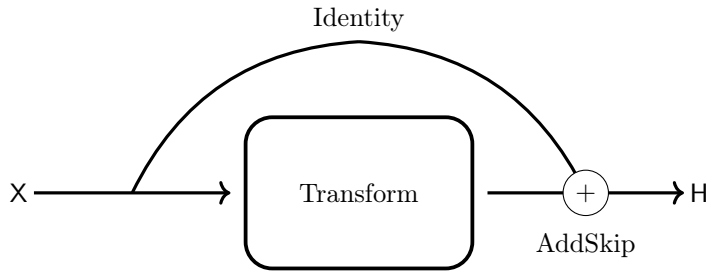
$$\text{Transform} : \mathbb{R}^{H\times W\times C} \to \mathbb{R}^{H\times W\times C},$$

a AddSkip to operator dodający do przetworzonego tensora oryginalny tensor wejściowy:

$$\text{AddSkip} : (\mathbb{R}^{H\times W\times C}, \mathbb{R}^{H\times W\times C}) \to \mathbb{R}^{H\times W\times C}.$$

Dla tensora wejściowego X, wyjście bloku rezydualnego można zapisać jako:

$$\mathsf{H} = \text{Transform}(\mathsf{X}) + \mathsf{X}.$$



Rysunek 21: Struktura bloku rezydualnego z połączeniem tożsamościowym (identity) omijającym główną ścieżkę transformacji, umożliwiającym efektywny przepływ gradientów. Źródło: opracowanie własne.

Kluczową obserwacją jest to, że możemy wyodrębnić efekt transformacji poprzez przegrupowanie elementów:

$$\text{Transform}(\mathsf{X}) = \mathsf{H} - \mathsf{X}.$$

Jest to znacznie prostsze zadanie niż uczenie się pełnej transformacji, szczególnie jeżeli optymalne wyjście jest bliskie wejściu. W takim przypadku rezyduum jest bliskie zeru, co sieć może łatwo osiągnąć poprzez wyzerowanie wag w głównej ścieżce.

Szczególnie istotny jest wpływ połączenia rezydualnego na propagację wsteczną gradientów. Dla bloku rezydualnego gradient funkcji straty ℓ względem wejścia X można zapisać jako:

$$\frac{\partial \ell}{\partial \mathsf{X}} = \frac{\partial \ell}{\partial \mathsf{H}} \left(\frac{\partial}{\partial \mathsf{X}} \text{Transform}(\mathsf{X}) + \text{Identity}(\mathsf{X}) \right),$$

gdzie Identity to operacja tożsamościowa. Obecność członu Identity w równaniu gwarantuje, że nawet jeśli gradienty przepływające przez Transform staną się bardzo małe, to gradient nadal może efektywnie propagować się wstecz przez połączenie rezydualnego. Jest to szczególnie ważne w głębokich sieciach, gdzie kaskada wielu nieliniowych transformacji może znacząco tłumić przepływ gradientów.

Popularną modyfikacją podstawowego bloku rezydualnego jest wzw. wariant bottleneck, który wprowadza dodatkową kompresję reprezentacji w danych w celu zwiększenia efektywności obliczeniowej. W tym wariancie operator Transform realizuje sekwencję trzech transformacji:

$$\text{Transform} = \text{Conv}(1\times 1) \circ \phi \circ \text{Conv}(3\times 3) \circ \phi \circ \text{Conv}(1\times 1) \circ \phi,$$

gdzie pierwsza konwolucja 1×1 redukuje liczbę kanałów (typowo do $1/4$ oryginalnej liczby), środkowa konwolucja 3×3 przetwarza dane w przestrzeni o zredukowanej wymiarowości, a ostatnia konwolucja 1×1 przywraca oryginalną liczbę kanałów.

W praktyce, zarówno w podstawowym wariancie, jak i wariancie bottleneck, często zachodzi potrzeba modyfikacji wymiarów tensora wejściowego, np. gdy zmieniamy liczbę kanałów, czyli:

$$\text{Transform} : \mathbb{R}^{H \times W \times C} \to \mathbb{R}^{H \times W \times K}.$$

W takich przypadkach stosuje się wariant z projekcją liniową:

$$\mathsf{H} = \text{Transform}(\mathsf{X}) + \text{Project}(\mathsf{X}),$$

gdzie Project to konwolucja 1×1 dostosowująca wymiary tensora wejściowego:

$$\text{Project} : \mathbb{R}^{H \times W \times C} \to \mathbb{R}^{H \times W \times K}.$$

5.8 Mobile Inverted Bottleneck Convolution

Mobile Inverted Bottleneck Convolution (MBConv) [30, 31] reprezentuje modyfikację standardowego bloku rezydualnego typu bottleneck. W przeciwieństwie do klasycznego wariantu bottleneck, który najpierw redukuje wymiarowość w celu efektywnego przetwarzania, MBConv odwraca tę kolejność, rozpoczynając od ekspansji przestrzeni cech, a następnie jej kompresji. Dodatkowo, blok ten intensywnie wykorzystuje opisane wcześniej depthwise separable convolutions oraz mechanizm squeeze-and-excitation.

Blok MBConv można przedstawić jako złożenie operacji:

$$\mathsf{X} \circ \text{Expand} \circ \text{BN} \circ \phi \circ \text{DepthConv} \circ \text{BN} \circ \phi \circ \text{SE} \circ \text{Project} \circ \text{BN} \circ \text{AddSkip}[\mathsf{X}] = \mathsf{H},$$

gdzie:

- Expand – konwolucja punktowa zwiększająca liczbę kanałów:

$$\text{Expand} : \mathbb{R}^{H \times W \times C} \to \mathbb{R}^{H \times W \times tC},$$

- DepthConv – konwolucja wgłębna działająca na dane w rozszerzonej przestrzeni:

$$\text{DepthConv} : \mathbb{R}^{H \times W \times tC} \to \mathbb{R}^{H \times W \times tC},$$

- SE – blok Squeeze-and-Excitation:

$$\text{SE} : \mathbb{R}^{H \times W \times tC} \to \mathbb{R}^{H \times W \times tC},$$

- Project – konwolucja punktowa redukująca liczbę kanałów:

$$\text{Project} : \mathbb{R}^{H \times W \times tC} \to \mathbb{R}^{H \times W \times K},$$

- AddSkip – dodanie połączenia rezydualnego:

$$\text{AddSkip} : (\mathbb{R}^{H \times W \times C}, \mathbb{R}^{H \times W \times K}) \to \mathbb{R}^{H \times W \times K},$$

- BN – normalizacja wsadowa,
- ϕ – funkcja aktywacji, najczęściej Swish.

Współczynnik ekspansji t jest hiperparametrem kontrolującym stopień rozszerzenia przestrzeni cech w bloku. Typowe wartości to $t \in \{4, 6, 8\}$, przy czym większe wartości pozwalają na bardziej złożone transformacje kosztem zwiększonej liczby obliczeń.

Blok MBConv może operować z różnym krokiem przestrzennym s. W przypadku gdy $s > 1$ lub gdy liczba kanałów wejściowych C różni się od liczby kanałów wyjściowych K, konieczne jest dostosowanie połączenia rezydualnego poprzez projekcję liniową, analogicznie jak w standardowym bloku rezydualnym.

5.9 EfficientNet

W poprzednich sekcjach szczegółowo omówiono podstawowe komponenty nowoczesnych sieci konwolucyjnych. Komponenty te, odpowiednio połączone, tworzą architekturę EfficientNet - rodzinę modeli konwolucyjnych zorganizowanych w sekwencję etapów przetwarzania. Każdy etap wykorzystuje bloki MBConv o różnej konfiguracji. Model podstawowy, oznaczany jako EfficientNet-B0, służy jako baza do tworzenia większych wariantów modeli z tej rodziny poprzez systematyczne skalowanie trzech wymiarów sieci: głębokości (liczby warstw), szerokości (liczby kanałów) oraz rozdzielczości przetwarzanych map cech.

5.9.1 Architektura bazowa

Model EfficientNet-B0 realizuje systematyczną transformację tensora wejściowego poprzez trzy główne komponenty: warstwę wejściową, sekwencję etapów zawierających powtarzające się bloki MBConv oraz warstwę wyjściową. Pełną strukturę sieci można zapisać jako:

$$\mathsf{X} \circ \text{Stem} \circ \text{Stages} \circ \text{Head} = \hat{\mathbf{y}},$$

gdzie Stem realizuje wstępne przetwarzanie poprzez sekwencję operacji:

$$\text{Stem} = \text{Conv}_{3\times 3} \circ \text{BN} \circ \text{Swish},$$

Stages to sekwencja siedmiu etapów, z których każda składa się z powtarzających się bloków MBConv o określonej konfiguracji (podanych w Tabeli 1):

$$\text{Stages} = \text{Stage}_1 \circ \text{Stage}_2 \circ \cdots \circ \text{Stage}_7,$$

gdzie każdy etap Stage_i jest złożeniem L_i identycznych bloków MBConv:

$$\text{Stage}_i = \underbrace{\text{MBConv}_i \circ \text{MBConv}_i \circ \cdots \circ \text{MBConv}_i}_{L_i \text{ razy}},$$

a Head to końcowy etap klasyfikacji:

$$\text{Head} = \text{Conv}_{1\times 1} \circ \text{BN} \circ \text{Swish} \circ \text{Pool} \circ \text{Dropout} \circ \text{FC}.$$

Numer etapu	Wsp. ekspansji	Rozmiar filtru	Liczba powtórzeń (L_i)	Stride	Kanały wyjściowe
1	1	3×3	1	1	16
2	6	3×3	2	2	24
3	6	5×5	2	2	40
4	6	3×3	3	2	80
5	6	5×5	3	1	112
6	6	5×5	4	2	192
7	6	3×3	1	1	320

Tabela 1: Konfiguracja bloków MBConv w architekturze EfficientNet-B0.

5.9.2 Metoda skalowania modelu

EfficientNet dodatkowo wprowadza systematyczne podejście do zwiększania złożoności modelu, realizowane poprzez jednoczesne skalowanie trzech wymiarów sieci:

- głębokości (d) - określającej liczbę powtórzeń bloków w każdym etapie przetwarzania,
- szerokości (w) - definiującej liczbę kanałów w warstwach,
- rozdzielczości (r) - kontrolującej wymiary przestrzenne przetwarzanych map cech.

Współczynniki skalowania są wyznaczane według zasady compound scaling:

$$d = \alpha^{\phi}, \quad w = \beta^{\phi}, \quad r = \gamma^{\phi},$$

gdzie α, β i γ są stałymi określającymi relatywne tempo wzrostu w każdym wymiarze, spełniającymi warunek:

$$\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2,$$

a ϕ jest globalnym współczynnikiem skalowania kontrolującym złożoność modelu.

W praktyce skalowanie realizowane jest poprzez:

- Zwiększenie liczby powtórzeń bloków w każdym etapie: $L_i' = \lceil d \cdot L_i \rceil$,
- Zwiększenie liczby kanałów: $C' = \lceil w \cdot C \rceil$,
- Zwiększenie wymiarów przestrzennych: $H' = \lceil r \cdot H \rceil$, $W' = \lceil r \cdot W \rceil$.

Na podstawie eksperymentów empirycznych ustalono optymalne wartości współczynników jako $\alpha = 1.2$, $\beta = 1.1$ i $\gamma = 1.15$. Dla modelu bazowego (B0) wszystkie współczynniki skalowania są równe 1, a kolejne warianty (B1-B7) powstają przez zwiększanie współczynnika ϕ.

6 Wyniki

W niniejszym rozdziale przedstawiono szczegółową analizę działania sieci konwolucyjnej w zadaniu klasyfikacji obrazów medycznych. Na podstawie zbiorów danych omówionych w rozdziale „Wprowadzenie do problemu", przeprowadzono dwuetapowe badanie - najpierw na danych syntetycznych, co pozwoliło na systematyczne zbadanie zdolności sieci do wykrywania określonych cech wizualnych, a następnie na danych rzeczywistych, weryfikując jej działanie w warunkach klinicznych. Dla każdego zbioru przedstawiono zarówno szczegóły procesu uczenia, jak i kompleksową analizę uzyskanych wyników, co pozwala lepiej zrozumieć praktyczne aspekty działania omawianych wcześniej mechanizmów sieci neuronowych.

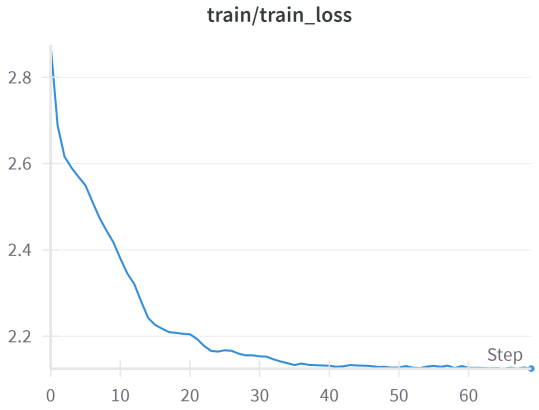
6.1 Dane syntetyczne

Proces uczenia Spośród rodziny modeli EfficientNet wybrano wariant B0 jako bazową architekturę do eksperymentów. Wybór ten podyktowany był względną prostotą modelu przy zachowaniu wysokiej efektywności obliczeniowej, co pozwoliło na przeprowadzanie systematycznych eksperymentów w rozsądnym czasie i niskim koszcie.

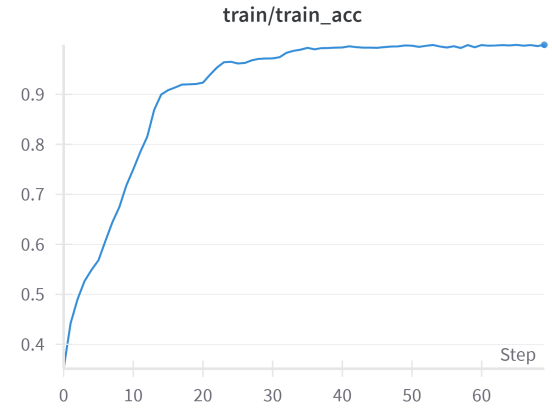
Model był trenowany przez 70 epok z wykorzystaniem optymalizatora SGD. Hiperparametry procesu uczenia zostały dobrane empirycznie:

- Współczynnik uczenia: $\eta = 0.05$,
- Rozmiar batcha: $B = 64$,
- Dropout: $p = 0.2$.

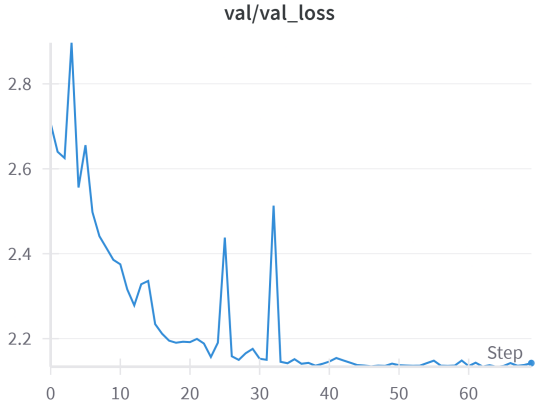
Jak pokazano na Rysunkach 22 oraz 23 przedstawiono przebieg procesu uczenia w postaci dwóch wykresów: wartości funkcji straty (lewy wykres) oraz wartości dokładności (prawy wykres) dla zbioru treningowego. Analogicznie, na Rysunkach 24 oraz 25 widać te same wykresy, ale dla zbioru walidacyjnego. Każdy z tych wykresów pokazuje wartości odpowiednich metryk w funkcji liczby epok treningu.



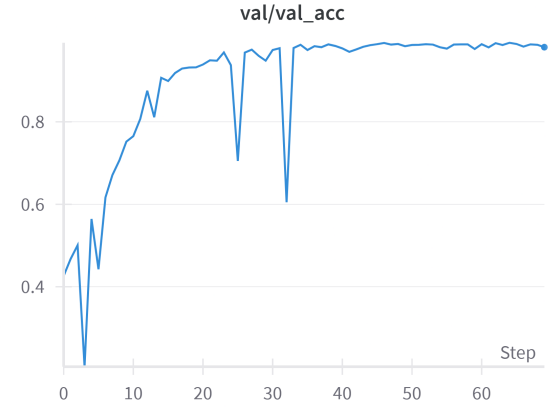
Rysunek 22: Wykres straty na zbiorze treningowym. Źródło: opracowanie własne na podstawie [32].



Rysunek 23: Wykres dokładności na zbiorze treningowy. Źródło: opracowanie własne na podstawie [32].



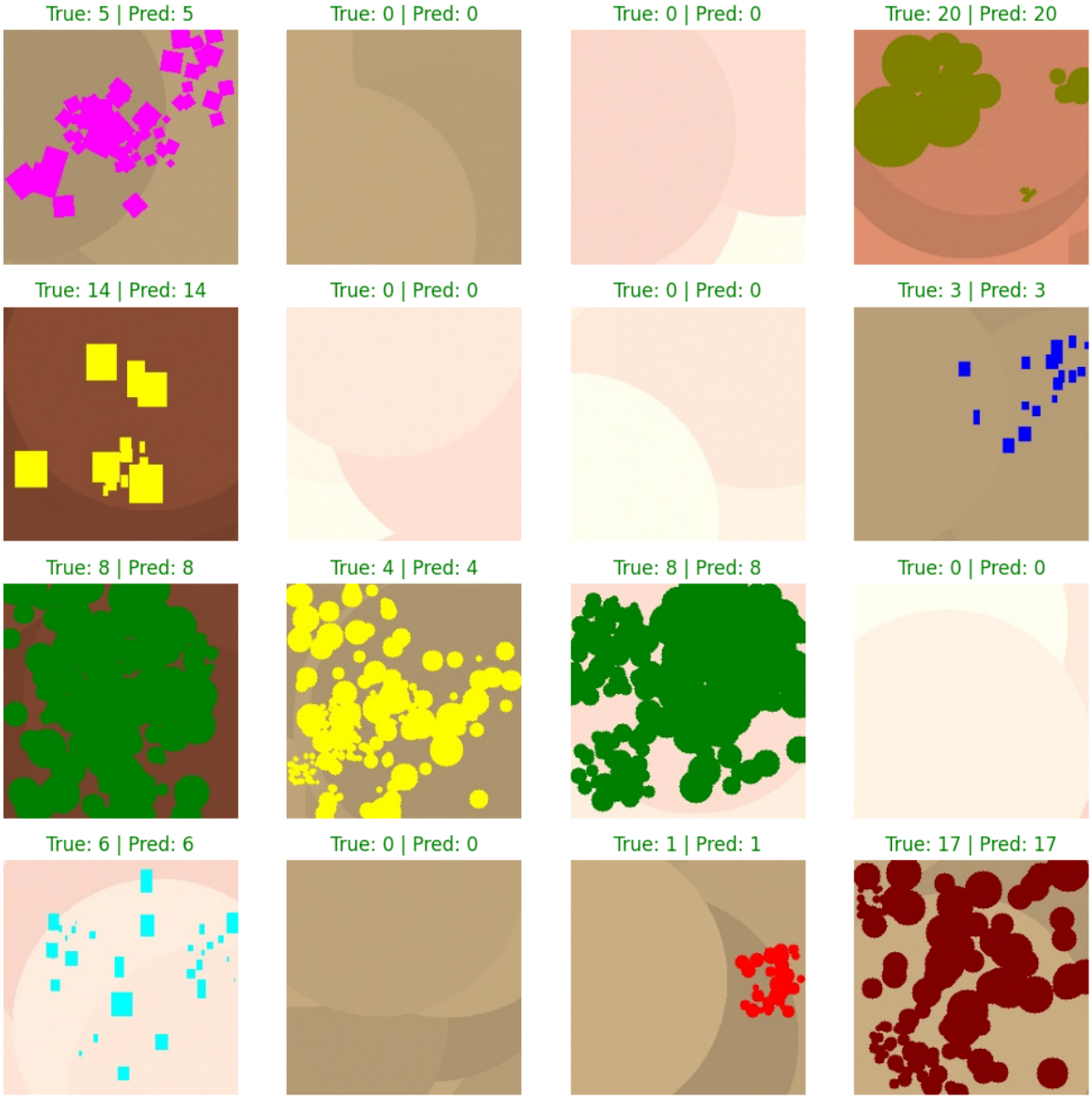
Rysunek 24: Wykres straty na zbiorze walidacyjnym. Źródło: opracowanie własne na podstawie [32].



Rysunek 25: Wykres dokładności na zbiorze walidacyjnym. Źródło: opracowanie własne na podstawie [32].

Na podstawie przedstawionych wykresów proces uczenia charakteryzował się typowymi fazami. Początkowy etap (pierwsze 10 epok) cechował się szybkim wzrostem dokładności i spadkiem funkcji straty zarówno dla zbioru treningowego, jak i walidacyjnego. W kolejnych epokach tempo zmian znacząco spadło, a krzywe zaczęły się stabilizować. Na wykresie walidacyjnym można zaobserwować kilka nagłych spadków dokładności (około 25 i 32 epoki), jednak model szybko powracał do poprzedniego poziomu skuteczności. Pomimo tych chwilowych fluktuacji, ogólny trend wskazuje na stabilne uczenie się modelu, z funkcją straty systematycznie malejącą i utrzymującą się na niskim poziomie w końcowych epokach. Po zakończeniu treningu model osiągnął dokładność około 99.9% na zbiorze treningowym oraz 98.8% na zbiorze walidacyjnym, przy wartościach funkcji straty wynoszących odpowiednio 2.1 i 2.15.

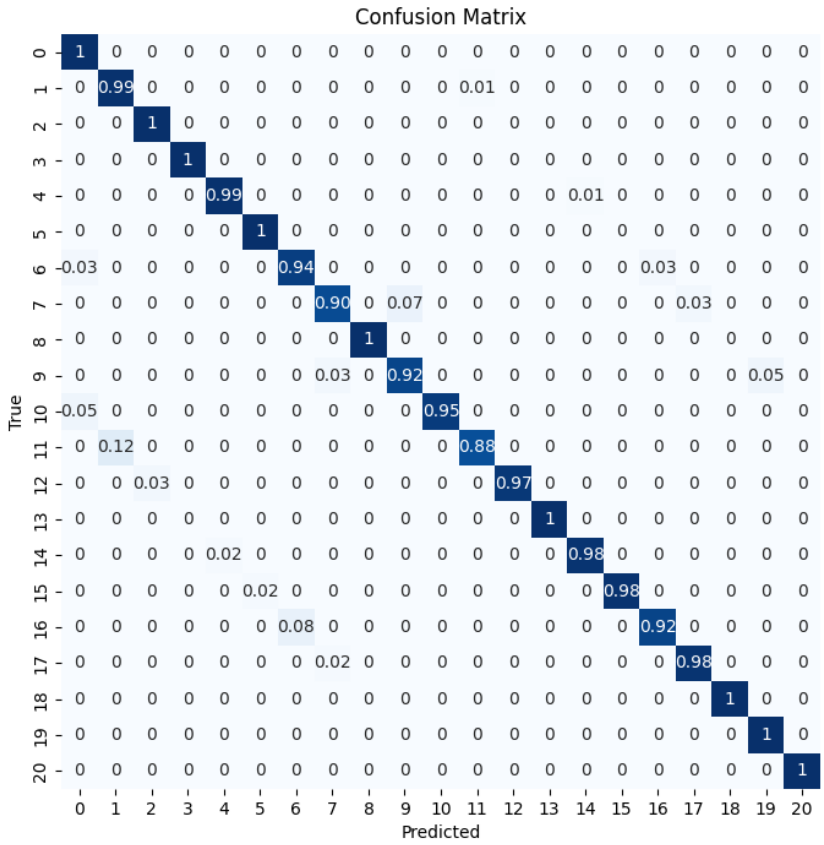
Analiza wyników Skuteczność wytrenowanego modelu została poddana szczegółowej analizie jakościowej i ilościowej. Na Rysunku 26 przedstawiono reprezentatywne przykłady klasyfikacji dla różnych typów zmian chorobowych. Dla każdego obrazu pokazano zarówno prawdziwą klasę (True) jak i predykcję modelu (Pred), co pozwala na bezpośrednią ocenę trafności klasyfikacji.



Rysunek 26: Przykłady klasyfikacji 16 przypadków chorobowych ze zbioru testowego. Źródło: opracowanie własne.

Wizualna analiza przykładów demonstruje, że model skutecznie radzi sobie z szerokim spektrum możliwych wariantów zmian chorobowych, niezależnie od ich charakterystyki przestrzennej czy stopnia nasilenia. Ta uniwersalność jest szczególnie istotna w kontekście przyszłego zastosowania modelu do klasyfikacji rzeczywistych zmian skórnych, które również charakteryzują się dużą różnorodnością form i postaci.

Podczas gdy powyższa analiza wizualna została przeprowadzona na niewielkiej próbie obrazów, pełniejszy obraz skuteczności modelu przedstawia macierz pomyłek na Rysunku 27, wygenerowana na podstawie całego zbioru testowego. Macierz ujawnia wysoką skuteczność klasyfikacji - zdecydowana większość predykcji skupia się na przekątnej macierzy, co oznacza poprawną klasyfikację. Dla większości klas model osiąga dokładność powyżej 95%.



Rysunek 27: Macierz pomyłek dla zbioru testowego. Źródło: opracowanie własne.

Te wyniki potwierdzają i rozszerzają wcześniejsze obserwacje z analizy wizualnej, demonstrując wyjątkowo wysoką skuteczność modelu w klasyfikacji syntetycznych obrazów medycznych na pełnym zbiorze testowym. Osiągnięte wyniki sugerują, że model skutecznie nauczył się rozróżniać charakterystyczne cechy poszczególnych typów zmian chorobowych.

6.2 Dane rzeczywiste

Proces uczenia W przypadku rzeczywistego zbioru danych medycznych proces uczenia wymagał wprowadzenia istotnych modyfikacji w stosunku do podejścia zastosowanego dla danych syntetycznych. Głównym wyzwaniem było znaczące niezbalansowanie klas - niektóre jednostki chorobowe występowały znacznie częściej niż inne, co mogło prowadzić do stronniczości modelu w kierunku klas dominujących. Dodatkową trudność stanowił brak reprezentacji klasy "nieznanej choroby"w zbiorze treningowym, co wymagało od modelu zdolności do efektywnej generalizacji, aby mógł rozpoznawać przypadki niespotykane podczas treningu.

Próbkowanie ważone Problem niezbalansowania klas został zaadresowany poprzez wprowadzenie próbkowania ważonego podczas treningu. Dla każdej obserwacji i w zbiorze treningowym, prawdopodobieństwo jej wylosowania do batcha określone jest jako:

$$P(i) = \frac{w_{y_i}}{\sum_j w_{y_j}},$$

gdzie w_{y_j} oznacza wagę przypisaną klasie obserwacji i. Wagi zostały wyznaczone jako:

$$w_k = \log\left(1 + \frac{N}{\text{count}(k) + \epsilon}\right),$$

gdzie N to całkowita liczba obserwacji, count(k) to liczba wystąpień klasy k, a $\epsilon = 10^{-6}$ to stała zapewniająca stabilność numeryczną. Następnie wagi zostały znormalizowane tak, aby ich suma była równa liczbie klas:

$$w_k = \frac{w_k}{\sum_j w_j} \cdot c,$$

gdzie c oznacza liczbę klas.

Modyfikacja funkcji straty Standardowa funkcja straty dla problemu klasyfikacji wieloklasowej oparta na entropii krzyżowej została rozszerzona o dwa mechanizmy: focal loss [[33] oraz margines pewności. Focal loss modyfikuje standardową entropię krzyżową tak, aby model skupiał się bardziej na trudnych przypadkach:

$$\mathcal{L}_{FL}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{i=1}^{c} w_i y_i (1 - \hat{y}_i)^{\gamma} \log(\hat{y}_i),$$

gdzie w_i to wagi klas, a γ jest hiperparametrem sterującym koncentracją na trudnych przypadkach. Warto zaznaczyć, że wagi w_i użyte w funkcji straty są inne niż te wykorzystane w mechanizmie próbkowania ważonego. Ich konkretne, dobrane empirycznie wartości zostaną przedstawione wraz z pozostałymi hiperparametrami modelu w kolejnej sekcji.

Focal loss pomaga w uczeniu się trudnych przypadków, jednak sam nie jest wystarczający do wykrywania przypadków nieznanych. W tym celu wprowadzono dodatkowy komponent straty marginesu:

$$\mathcal{L}_{margin}(\hat{\mathbf{y}}, \mathbf{y}) = \max(0, p_{max} - p_t + m),$$

gdzie p_t to prawdopodobieństwo przewidywane dla prawdziwej klasy, a p_{max} to najwyższe przewidywane prawdopodobieństwo spośród pozostałych klas, a m to zadany margines. Komponent ten działa podczas treningu, wykorzystując informację o prawdziwej klasie do wymuszenia zdecydowanych predykcji – prawdopodobieństwo przewidywane dla prawdziwej klasy powinno być znacząco wyższe niż najwyższe prawdopodobieństwo wśród pozostałych klas.

Końcowa funkcja straty łączy oba komponenty:

$$\mathcal{L}_{FL\,+\,margin} = \mathcal{L}_{FL} + \lambda \mathcal{L}_{margin},$$

gdzie λ to hiperparametr kontrolujący względną wagę składnika marginesu. Dodatkowo zastosowano technikę label smoothing - mechanizm regularyzacji, który zamiast używać jednoznacznych etykiet (0 lub 1), wprowadza "miękkie"etykiety poprzez przypisanie małej wartości prawdopodobieństwa klasom niepoprawnym. Dla etykiety reprezentowanej przez wektor jednostkowy $\mathbf{y}$, wersja wygładzona $\tilde{\mathbf{y}}$ jest obliczana jako:

$$\tilde{y}_i = \begin{cases} 1 - \varepsilon & \text{dla } i = y, \\ \frac{\varepsilon}{k-1} & \text{dla } i \neq y, \end{cases}$$

gdzie ε jest parametrem wygładzania, a k to liczba klas. Technika ta zapobiega nadmiernemu wzrostowi pewności modelu i poprawia jego zdolność do generalizacji.

O ile podczas treningu komponent marginesu wykorzystuje informację o prawdziwej klasie, w fazie wnioskowania potrzebny jest mechanizm działający bez tej wiedzy. W tym celu wprowadzono porównanie dwóch najwyższych prawdopodobieństw:

$$\text{Przewidywana klasa} = \begin{cases} \arg\max_i \hat{y} & \text{jeśli } \max_i \hat{y}_i - \max_{j \neq \arg\max_i \hat{y}_i} \hat{y}_j > \tau, \\ \text{nieznana} & \text{w przeciwnym razie,} \end{cases}$$

Jeśli różnica między nimi jest mniejsza niż ustalony próg τ, przypadek klasyfikowany jest jako "nieznana choroba". Pozwala to na identyfikację sytuacji, gdzie nie ma jednej wyraźnie dominującej klasy w przewidywaniach.

Optymalizacje techniczne W trakcie pracy nad modelem wprowadzono dwie istotne optymalizacje techniczne, które znacząco usprawniły proces treningu. Pierwszą z nich jest akumulacja gradientów - rozwiązanie, które pozwala efektywniej wykorzystać dostępne zasoby obliczeniowe. W praktyce oznacza to, że zamiast przetwarzać od razu duże porcje danych (co wymagałoby znacznej ilości pamięci GPU), model przetwarza mniejsze porcje i kumuluje ich wyniki przed aktualizacją parametrów. W implementacji zastosowano akumulację gradientów z czterech kolejnych batchy, każdy o rozmiarze 128 obrazów, przed wykonaniem pojedynczego kroku optymalizacji. Takie podejście pozwala efektywnie symulować trening na batchu rozmiaru 512 obrazów, zachowując przy tym wymagania pamięciowe na poziomie przetwarzania batcha o rozmiarze 128 obrazów.

Drugą wprowadzoną optymalizacją jest trening z mieszaną precyzją [34] (ang. *mixed precision training*). Technika ta polega na inteligentnym wykorzystaniu różnych poziomów precyzji obliczeń - część operacji wykonywana jest z pełną precyzją (32-bitową), podczas gdy pozostałe używają precyzji obniżonej (16-bitowej). Takie podejście znacząco przyspiesza obliczenia i zmniejsza wykorzystanie pamięci, szczególnie na nowoczesnych kartach graficznych. W naszym przypadku model zachowuje pełną precyzję dla krytycznych operacji (jak aktualizacja wag), podczas gdy większość pozostałych obliczeń wykonywana jest z precyzją obniżoną.

Zastosowanie obu tych technik pozwoliło na znaczące przyspieszenie procesu treningu przy jednoczesnym zmniejszeniu wymagań pamięciowych i braku negatywnych wpływów na uzyskane wyniki.

Augmentacja zdjęć W celu dodatkowej regularyzacji, dla zbioru rzeczywistego zastosowano rozbudowaną augmentację zdjęć w czasie treningu, zwiększającą różnorodność zbioru treningowego i redukującą przeuczenie. Transformacje były aplikowane z prawdopodobieństwem 0.8 i obejmowały dwa typy przekształceń: geometryczne oraz fotometryczne.

Przekształcenia geometryczne obejmowały:

- Losowe odbicia poziome oraz pionowe (p=0.5 dla każdego kierunku),
- Losowe obroty do 30 stopni,
- Przesunięcia w zakresie ±15% wymiarów obrazu,
- Skalowanie w zakresie 85% do 115% oryginalnego obrazu,
- Deformacje perspektywiczne w zakresie ±5 stopni.

Przekształcenia fotometryczne zawierały:

- Modyfikacje jasności, kontrastu i saturacji (±30% dla każdego parametru),
- Zmiana odcienia (±10%),
- Zwiększenie ostrości (o 50% z prawdopodobieństwem 0.3),
- Rozmycie gaussowskie (kernel 3×3, $\sigma \in [0.1, 0.2]$).

Losowy charakter transformacji zapobiega zapamiętywaniu przez sieć konkretnych cech obrazu, wymuszając skupienie się na istotnych cechach diagnostycznych.

Hiperparametry Model EfficientNet-B0 analogicznie jak w przypadku danych syntetycznych, został wykorzystany jako bazowa architektura do przeprowadzania eksperymentów. Model był trenowany przez 100 epok z wykorzystaniem optymalizatora SGD z momentum. Hiperparametry procesu uczenia zostały dobrane empirycznie:

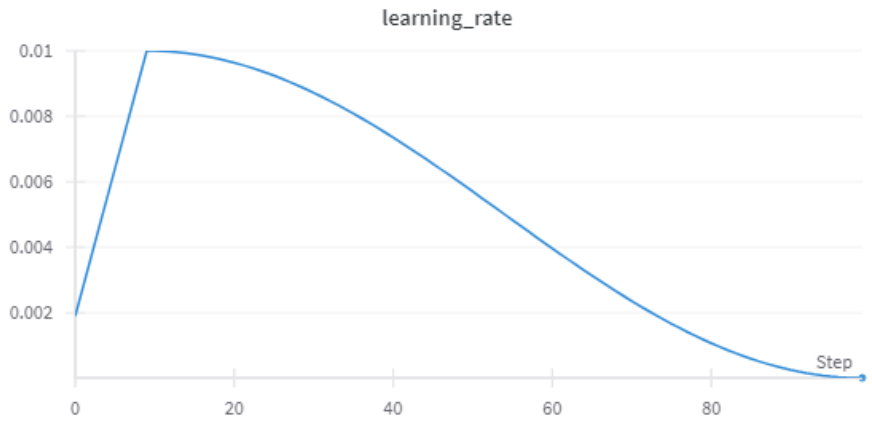
- Współczynnik uczenia: $\eta = 0.01$,
- Rozmiar efektywnego batch: $B = 512$ (realizowany poprzez akumulację gradientów),
- Dropout: $p = 0.5$,
- Parametr focal loss: $\gamma = 2$,
- Margines pewności: $m = 0.03$,
- Próg klasyfikacji przypadków nieznanych: $\tau = 0.05$,
- Wagi klas: [1.0, 1.0, 0.25, 0.7, 1.1, 0.55, 1.5, 1.5, 1.1],

- Parametr label smoothing: $\varepsilon = 0.05$.

W celu kontroli procesu uczenia zastosowano harmonogram zmian współczynnika uczenia składający się z dwóch faz:

- Faza rozgrzewki trwająca 10 epok, w której współczynnik uczenia liniowo wzrasta od 0.1η do η, pomagając uniknąć niestabilności na początku treningu,
- Faza zaniku cosinusowego, w której współczynnik uczenia stopniowo maleje do wartości minimalnej $\eta_{min} = 10^{-5}$ według funkcji cosinusa umożliwiając dokładniejszą eksplorację przestrzeni parametrów w końcowych etapach treningu.

Wykres zmian współczynnika uczenia przedstawiony jest na Rysunku 28. Dodatkowo, parametr λ w funkcji straty był stopniowo zwiększany podczas fazy rozgrzewki od 0 do zadanej wartości, co również pozwoliło na stabilniejszy początek treningu.



Rysunek 28: Wykres zmiany współczynnika uczenia. Źródło: opracowanie własne na podstawie [32].

6.2.1 Analiza wyników

Proces uczenia modelu można prześledzić analizując wykresy funkcji straty, dokładności oraz współczynnika kappa dla zbiorów treningowego oraz walidacyjnego. Współczynnik kappa Cohena jest szczególnie istotną metryką w przypadku niezbalansowanych zbiorów danych, gdyż uwzględnia możliwość przypadkowej zgodności klasyfikacji. Przyjmuje wartości od -1 do 1, gdzie 1 oznacza idealną zgodność, 0 – zgodność na poziomie losowym, a wartości ujemne wskazują na zgodność gorszą niż losowa.

Matematycznie, współczynnik kappa wyrażany jest wzorem:

$$\kappa = \frac{p_o - p_e}{1 - p_e},$$

gdzie p_o oznacza zaobserwowaną zgodność (dokładność), a p_e oznacza oczekiwaną zgodność wynikającą z przypadku. Wartość p_e obliczana jest jako:

$$p_e = \sum_{i=1}^{c} \hat{p}_i p_i,$$

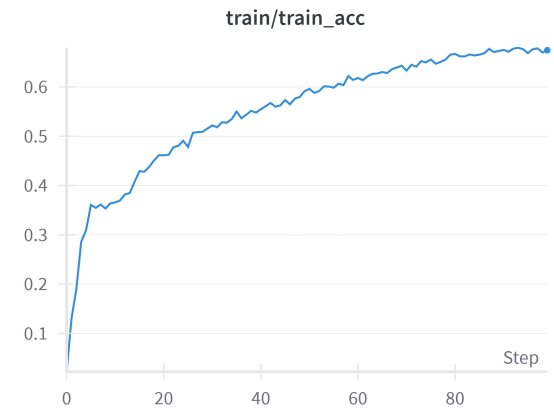
gdzie $\hat{p}_i$ oznacza proporcję przypadków zaklasyfikowanych przez model do klasy i, a p_i oznacza rzeczywistą proporcję przypadków należących do klasy i w zbiorze danych, a c to liczba klas. Intuicyjnie, współczynnik kappa można interpretować jako miarę tego, o ile lepszy jest nasz model od przypadkowego zgadywania. Wartość $\kappa = 0.5$ oznacza, że model jest o 50% lepszy od losowego przypisywania klas, uwzględniając strukturę danych. Jest to szczególnie istotne w kontekście rozwiązywanego problemu, gdzie występuje znaczące niezbalansowanie klas - współczynnik kappa dostarcza bardziej wiarygodnej oceny skuteczności modelu niż prosta dokładność, szczególnie dla rzadziej występujących jednostek chorobowych.

Według powszechnie przyjętej skali interpretacji [35], wartości współczynnika kappa można interpretować następująco:

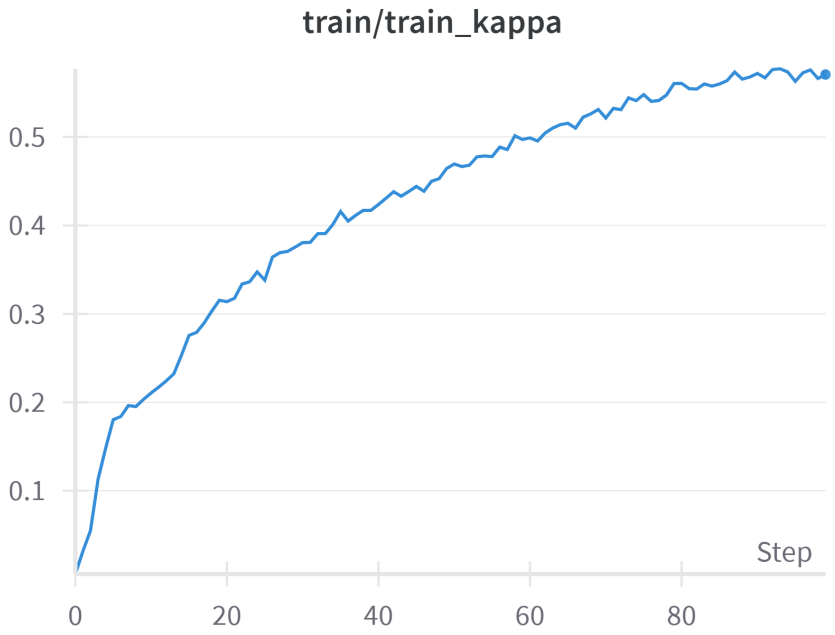
- poniżej 0: zgodność gorsza niż losowa,
- 0.0 - 0.20: słaba zgodność,
- 0.21 - 0.40: zadowalająca zgodność,
- 0.41 - 0.60: umiarkowana zgodność,
- 0.61 - 0.80: dobra zgodność,
- 0.81 - 1.00: bardzo dobra zgodność.

Na Rysunkach 29, 30 i 31 przedstawiono ewolucję metryk podczas treningu. Strata treningowa (Rysunek 29) systematycznie malała z początkowej wartości około 3.9 do końcowej 1.2. Dokładność (Rysunek 30) wzrosła do około 67%, natomiast współczynnik kappa (Rysunek 31) osiągnął wartość około 0.57, co wskazuje na umiarkowaną skuteczność klasyfikacji przekraczającą znacząco poziom losowy.

Rysunek 29: Wykres straty na zbiorze treningowym. Źródło: opracowanie własne na podstawie [32]

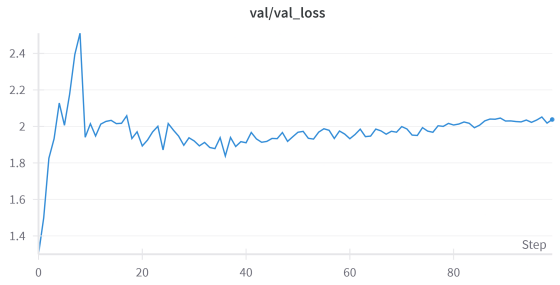


Rysunek 30: Wykres dokładności na zbiorze treningowy. Źródło: opracowanie własne na podstawie [32].

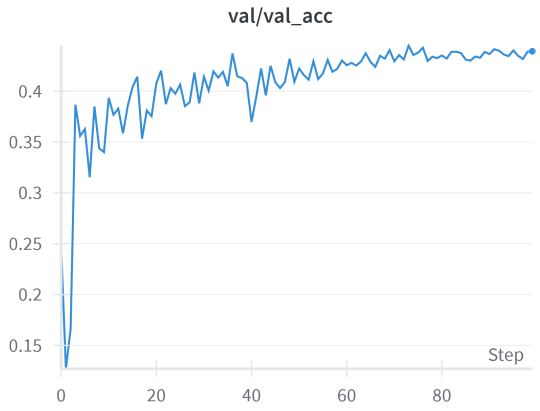


Rysunek 31: Wykres współczynnika kappa na zbiorze treningowy. Źródło: opracowanie własne na podstawie [32].

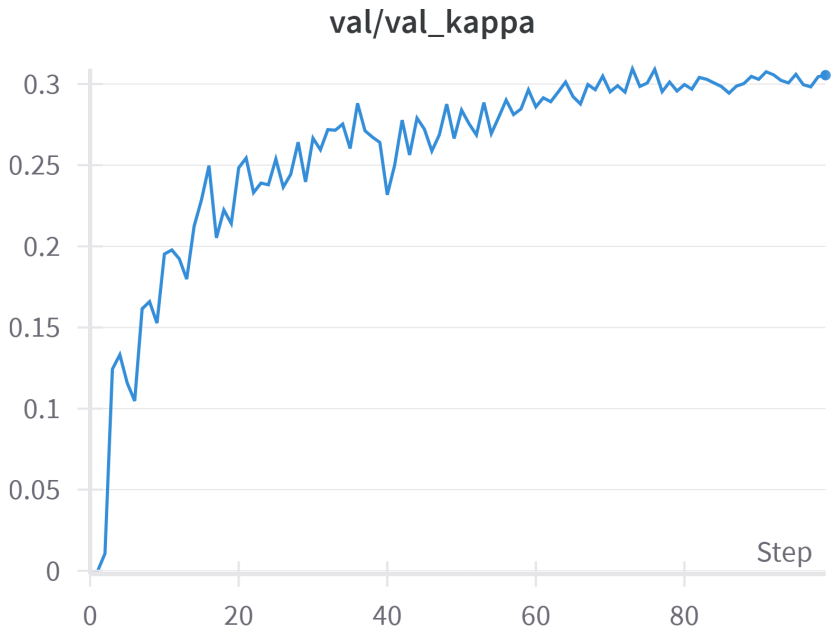
Analogicznie metryki dla zbioru walidacyjnego przedstawiono na Rysunkach 32, 33 i 34. Pomimo zastosowania rygorystycznych technik regularyzacji, można zaobserwować znaczące różnice między wynikami na zbiorze treningowym i walidacyjnym. Strata walidacyjna (Rysunek 32) stabilizuje się na poziomie ok. 2.0, podczas gdy dokładność (Rysunek 33) osiąga jedynie 45%, a współczynnik kappa (Rysunek 34) stabilizuje się w okolicach 0.3. Ta wyraźna różnica z wynikami treningowymi wskazuje na istotne przeuczenie modelu, co sugeruje, że zastosowanie techniki regularyzacji, choć rozbudowane, nie były w pełni skuteczne w przeciwdziałaniu temu zjawisku.



Rysunek 32: Wykres straty na zbiorze walidacyjnym. Źródło: opracowanie własne na podstawie [32].



Rysunek 33: Wykres dokładności na zbiorze walidacyjnym. Źródło: opracowanie własne na podstawie [32].

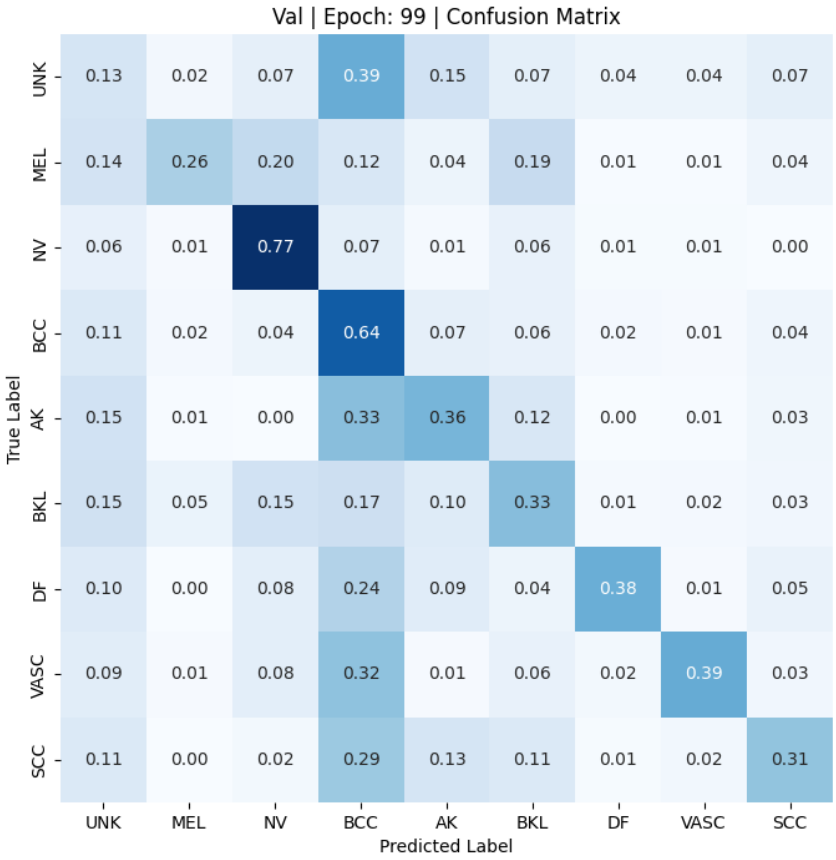


Rysunek 34: Wykres współczynnika kappa na zbiorze walidacyjnym. Źródło: opracowanie własne na podstawie [32].

Przeanalizujmy teraz szczegółowo skuteczność klasyfikacji poszczególnych klas, wykorzystując macierze pomyłek dla zbiorów treningowego (Rysunek 35) i walidacyjnego (Rysunek 36). Na zbiorze treningowym model osiągnął znaczącą skuteczność w klasyfikacji kilku typów zmian skórnych. Szczególnie wysoką dokładność uzyskano w przypadku naczyniaków (VASC, 92%) oraz dermatofibromy (DF, 84%). Dobre wyniki zaobserwowano również dla znamion melanocytowych (NV, 76%), raka podstawnokomórkowego (BCC, 71%) oraz raka kolczystokomórkowego (SCC, 62%). Niższą skuteczność model wykazał w przypadku rogowacenia słonecznego (AK, 59%) i brodawki łojotokowej (BKL, 45%).

Rysunek 35: Macierz pomyłek dla zbioru treningowego. Źródło: opracowanie własne na podstawie [32].

Analiza macierzy pomyłek dla zbioru walidacyjnego ujawnia znaczący spadek skuteczności dla większości klas. Najwyższą skuteczność model utrzymał jedynie dla znamion melanocytowych (NV, 77%) oraz raka podstawnokomórkowego (BCC, 64%). Dla pozostałych klas skuteczność klasyfikacji spadła do poziomu 30-40%. Szczególnie problematyczna okazała się klasyfikacja przypadków nieznanych (UNK), które często były błędnie identyfikowane jako rak podstawnokomórkowy (39% przypadków). Co więcej, widoczna jest wyraźna tendencja do nadmiernej klasyfikacji przypadków jako BCC, co sugeruje, że model mógł nauczyć się pewnych uniwersalnych cech wizualnych charakterystycznych dla zmian skórnych, które często interpretuje jako oznaki raka podstawnokomórkowego.



Rysunek 36: Macierz pomyłek dla zbioru walidacyjnego. Źródło: opracowanie własne na podstawie [32].

Ta analiza macierzy pomyłek potwierdza wcześniejsze obserwacje dotyczące przeuczenia modelu. Mimo zastosowania zaawansowanych technik regularyzacji, model wykazuje znacząco gorszą skuteczność na danych walidacyjnych, szczególnie w przypadku rzadziej występujących klas chorób. Jednocześnie tendencja do nadmiernej klasyfikacji przypadków jako BCC wskazuje na potencjalne problemy z reprezentatywnością zbioru treningowego, niewystarczającą skuteczność zastosowanych technik zrównoważenia klas lub nieodpowiedni dobór wag klas.

7 Podsumowanie

Celem pracy było zbadanie możliwości wykorzystania sieci konwolucyjnych w diagnostyce zmian skórnych. Autor przeprowadził kompleksowe badania na syntetycznym i rzeczywistym zbiorze danych, demonstrując potencjał oraz ograniczenia sieci neuronowych w klasyfikacji obrazów medycznych.

Model wykazał wysoką skuteczność na danych syntetycznych (niemal 99% dokładności). Na rzeczywistych danych medycznych osiągnął dokładność około 45% oraz współczynnik kappa na poziomie 0.3, co wskazuje na znaczące wyzwania związane z różnorodnością i niezbalansowaniem zbioru. Badania ujawniły kluczowe aspekty budowy efektywnych modeli klasyfikacyjnych, takie jak dobór architektury, augmentacja danych oraz techniki regularyzacji.

Przeprowadzone badania rzucają światło na praktyczne aspekty wykorzystania sieci neuronowych w zadaniach klasyfikacji obrazów medycznych. Uzyskane wyniki sugerują, że mimo obiecujących rezultatów na danych syntetycznych, zastosowanie tych modeli w rzeczywistej diagnostyce wymaga dalszych badań i udoskonaleń, szczególnie w kontekście obsługi niezbalansowanych zbiorów danych oraz generalizacji na nieznane przypadki. Szczegółowa implementacja wszystkich opisanych rozwiązań, wraz z kodem źródłowym i instrukcjami reprodukcji eksperymentów, dostępna jest w notebookach Google Colab, do których odnośniki zamieszczono w dodatku.

A Skrypty Google Colab z implementacją

Przedstawione modele zostały zaimplementowane i wytrenowane z wykorzystaniem środowiska Google Colab. Pełny kod źródłowy dostępny jest w następujących notebookach:

- Model dla danych syntetycznych:
  `https://colab.research.google.com/drive/1vbyTibINo9ArrmlvYjQm-UqqSPgQx-ND?usp=sharing`
- Model dla danych rzeczywistych:
  `https://colab.research.google.com/drive/1Kt7CT2KZwAIi7WvPAm8EzFfVJfAs0Vz6?usp=sharing`

Kod został zoptymalizowany pod kątem wykonania w środowisku Google Colab z wykorzystaniem akceleracji GPU.

Bibliografia

[1] Warren S. McCulloch i Walter Pitts. „A logical calculus of the ideas immanent in nervous activity". W: *The bulletin of mathematical biophysics* 5.4 (grud. 1943), s. 115–133. ISSN: 1522-9602. DOI: 10.1007/BF02478259.

[2] Frank Rosenblatt. „The perceptron: a probabilistic model for information storage and organization in the brain." W: *Psychological review* 65 6 (1958), s. 386–408.

[3] Xavier Glorot, Antoine Bordes i Yoshua Bengio. „Deep Sparse Rectifier Neural Networks". W: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Red. Geoffrey Gordon, David Dunson i Miroslav Dudík. T. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, list. 2011, s. 315–323.

[4] Chigozie Nwankpa i in. *Activation Functions: Comparison of trends in Practice and Research for Deep Learning*. 2018. arXiv: 1811.03378 [cs.LG].

[5] Prajit Ramachandran, Barret Zoph i Quoc V. Le. *Searching for Activation Functions*. 2017. arXiv: 1710.05941 [cs.NE].

[6] Ian J. Goodfellow, Yoshua Bengio i Aaron Courville. *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[7] Ilya Sutskever i in. „On the importance of initialization and momentum in deep learning". W: *Proceedings of the 30th International Conference on Machine Learning*. Red. Sanjoy Dasgupta i David McAllester. T. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, s. 1139–1147.

[8] Xavier Glorot i Yoshua Bengio. „Understanding the difficulty of training deep feedforward neural networks". W: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Red. Yee Whye Teh i Mike Titterington. T. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, s. 249–256.

[9] Kaiming He i in. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. 2015. arXiv: 1502.01852 [cs.CV]. URL: https://arxiv.org/abs/1502.01852.

[10] Lorenzo Ciampiconi i in. *A survey and taxonomy of loss functions in machine learning*. 2023. arXiv: 2301.05579 [cs.LG].

[11] David E. Rumelhart, Geoffrey E. Hinton i Ronald J. Williams. „Learning representations by back-propagating errors". W: *Nature* 323.6088 (paź. 1986), s. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0.

[12] B.T. Polyak. „Some methods of speeding up the convergence of iteration methods". W: *USSR Computational Mathematics and Mathematical Physics* 4.5 (1964), s. 1–17. ISSN: 0041-5553. DOI: https://doi.org/10.1016/0041-5553(64)90137-5.

[13] John Duchi, Elad Hazan i Yoram Singer. „Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". W: *Journal of Machine Learning Research* 12.61 (2011), s. 2121–2159.

[14] G. Hinton. *Neural Networks for Machine Learning*. Coursera Lecture 6e. 2012.

[15] Diederik P. Kingma i Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[16] Tom Schaul, Ioannis Antonoglou i David Silver. *Unit Tests for Stochastic Optimization*. 2014. arXiv: 1312.6055 [cs.LG].

[17] Yoshua Bengio. *Practical recommendations for gradient-based training of deep architectures*. 2012. arXiv: 1206.5533 [cs.LG].

[18] Philipp Tschandl, Cliff Rosendahl i Harald Kittler. „The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions". W: *Scientific Data* 5 (2018), s. 180161. DOI: 10.1038/sdata.2018.161.

[19] Noel C. F. Codella i in. „Skin Lesion Analysis Toward Melanoma Detection: A Challenge at the 2017 International Symposium on Biomedical Imaging (ISBI), Hosted by the International Skin Imaging Collaboration (ISIC)". W: *arXiv preprint arXiv:1710.05006* (2017).

[20] Clara Hernández-Pérez i in. „BCN20000: Dermoscopic lesions in the wild". W: *Scientific Data* 11.1 (2024), s. 641.

[21] Department of Dermatology, Hospital Clínic de Barcelona, ViDIR Group, Department of Dermatology, Medical University of Vienna i Memorial Sloan Kettering Cancer Center. *ISIC 2019: Training Dataset*. Dataset consists of BCN_20000, HAM10000, and MSK datasets. 2019.

[22] Mingxing Tan i Quoc V. Le. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. 2020. arXiv: 1905.11946 [cs.LG].

[23] Y. Lecun i in. „Gradient-based learning applied to document recognition". W: *Proceedings of the IEEE* 86.11 (1998), s. 2278–2324. DOI: 10.1109/5.726791.

[24] Sergey Ioffe i Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG].

[25] Nitish Srivastava i in. „Dropout: A Simple Way to Prevent Neural Networks from Overfitting". W: *Journal of Machine Learning Research* 15.56 (2014), s. 1929–1958.

[26] François Chollet. *Xception: Deep Learning with Depthwise Separable Convolutions*. 2017. arXiv: 1610.02357 [cs.CV].

[27] Jie Hu i in. *Squeeze-and-Excitation Networks*. 2019. arXiv: 1709.01507 [cs.CV].

[28] Kaiming He i in. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[29] Kaiming He i in. *Identity Mappings in Deep Residual Networks*. 2016. arXiv: 1603.05027 [cs.CV].

[30] Mark Sandler i in. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. 2019. arXiv: 1801.04381 [cs.CV].

[31] Andrew Howard i in. *Searching for MobileNetV3*. 2019. arXiv: 1905.02244 [cs.CV].

[32] Lukas Biewald. *Experiment Tracking with Weights and Biases*. Software available from wandb.com. 2020. URL: https://www.wandb.com/.

[33] Tsung-Yi Lin i in. *Focal Loss for Dense Object Detection*. 2018. arXiv: 1708.02002 [cs.CV].

[34] Paulius Micikevicius i in. *Mixed Precision Training*. 2018. arXiv: 1710.03740 [cs.AI].

[35] J Richard Landis i Gary G. Koch. „The measurement of observer agreement for categorical data." W: *Biometrics* 33 1 (1977), s. 159–74.