Klasyfikacja kwiatków

Julia Chomicka, s188873
Tomasz Krezymon, s189642
Robert Kalinowski, s188685

Wstęp - opis problemu

Jednym z popularnych zastosowań sztucznej inteligencji jest klasyfikacja obrazów przy użyciu sieci neuronowych. W naszym projekcie skupimy się na klasyfikacji kwiatków przy użyciu sieci neuronowych. Klasyfikacja kwiatków jest ważnym problemem w dziedzinie ogrodnictwa, natomiast ręczna analiza cech każdego gatunku kwiatu byłaby bardzo czasochłonna. Zastosowanie sztucznej inteligencji może przyspieszyć ten proces i umożliwić automatyczną klasyfikację.

Spróbujemy zająć się tym problemem i stworzymy 3 modele sieci neuronowych, które będą próbowały sklasyfikować kwiat z obrazu na podstawie jego cech i zobaczymy na ile każdy z modeli będzie w stanie przewidzieć, do którego gatunku należy kwiatek. Skupimy się również na porównaniu 3 zbudowanych przez nas sieci, oraz jak baza zdjęć wpływa na wynik klasyfikacji. Spróbujemy również zbudować prostą sieć neuronową "od zera", aby poznać podstawy sieci neuronowych i zobaczyć, jak ona poradzi sobie w klasyfikacji.

Teoretyczny opis użytych metod

sieć konwolucyjna

Rodzaj sieci, który najczęściej stosowany jest do przetwarzania obrazu. Typowa sieć konwolucyjna składa się z następujących warstw:

- **warstwa konwolucyjna** - miejsce, w którym dokonana zostaje operacja konwolucji. Polega ona na przejściu przez dane wejściowe macierzy o małym rozmiarze z pewnymi ustalonymi wagami, przez które piksele obrazu zostają przemnożone oraz posumowane i uzyskujemy w ten sposób przefiltrowany obraz. Mała macierz z wagami najczęściej określana jest jako filtr bądź Kernel. Cała operacja stosowana jest w celu uzyskania cech, które mogą być przydatne w dalszej obróbce.
- **warstwa poolingu** - zmniejszenie rozmiaru obrazu poprzez wyodrębnienie najważniejszych cech dokonując decyzji na podstawie lokalnych przestrzeni pikseli. Najczęstszą metodą jest tzw. max pooling, w którym to z małego określonego okienka wybierany zostaje piksel o największej wartości i służy on jako wyjście nowego obrazu.
- **flatten** - ta warstwa służy do przekształcenia wyniku warstw konwolucyjnych, na jednowymiarowy wektor. Jest to niezbędne, ponieważ warstwy takie jak w pełni połączone (dense) oczekują wektora jako wejścia.
- **warstwa w pełni połączona** - stosowana najczęściej na końcu sieci neuronowej, na jej podstawie zostanie dokonana decyzja o klasyfikacji danego obrazu wzorując się na wyodrębnionych cechach z poprzednich warstw
- **warstwa Dropout** - wyklucza losowo neurony poprzez wyzerowanie ich. Jej celem jest zapobieganie przeuczenia się modelu.

model teacher-student

Najważniejszym elementem tego modelu jest tzw. knowledge distillation, czyli proces przekazywania wiedzy z dużego modelu na mniejszy. Używany jest on w celu zmniejszenia czasu trenowania modelu, jak i w celu redukcji zasobów. Ponadto, duża sieć teacher mimo swoich wielkich rozmiarów, nie wykorzystuje całej swojej pojemności na wiedzę, dlatego właśnie korzysta się z modelu student. Klasycznie cały model składa się na dwa podmodele:

- **teacher** - zaawansowany i dokładny model, który uczy się na danych treningowych, i którego celem jest przekazywanie parametrów studentowi
- **student** - model, zazwyczaj o uproszczonej budowie, który uczy się na podstawie obrazów oraz predykcji, cech lub etykiet modelu nauczyciela, ale na innej bazie niż teacher. Teacher jest wytrenowanym modelem, a student uczy się dodatkowo soft outputu teachera czyli wyniku teachera w

postaci prawdopodobieństwa dla każdej klasy. Do nauki modelu studenta korzysta się z cross entropy czyli miary odległości (strat) między dwoma rozkładami prawdopodobieństwa. W przypadku korzystania z etykiet od nauczyciela, do oceny jakości modelu predykcyjnego bierze się przewidywane wartości studenta i porównuje zarówno z rzeczywistym wyjściem jak i tym uzyskanym przez teachera - na tej podstawie obliczana jest funkcja straty.

Celem modelu teacher-student jest to, aby model studenta dzięki przekazanej wiedzy przez nauczyciela osiągnął podobne wyniki jak model nauczyciela a nawet i lepsze w znacznie krótszym czasie mając prostszą budowę niż model nauczyciela.

technika przycinania (pruningu)

Metoda polegająca na usuwaniu wag lub neuronów, które nie mają znaczącego wpływu na działanie sieci, w celu poprawy efektów jaki i zmniejszeniu rozmiaru sieci. Przycinanie wag polega na usunięciu ich lub wyzerowaniu w momencie kiedy ich wartości znajdują się poniżej ustalonego progu, natomiast przycinanie neuronów polega na usunięciu konkretnych neuronów, które nie mają dużego znaczenia w całej sieci. Ogólny proces działania dla przycinania wag wygląda następująco:

1. **trenowanie modelu** - model powinien zostać wytrenowany zanim zostanie na nim wykonana operacja przycinania
2. **ustalenie ważności wag** - może zostać dokonany np. przy pomocy sortowania po wartościach rosnąco
3. **usuwanie wag** - usunięty zostaje jakiś procent wag o najmniejszej wartości, czyli takich które są najmniej znaczące
4. **Fine-tuning** - po usunięciu wag może być zauważalne pogorszenie się wyników co poprawiane jest poprzez ponowne trenowanie modelu
5. Opcjonalnym krokiem jest powtarzanie kroków od 2. do 4. jeżeli końcowy efekt nie jest zadowalający. Taką metodę nazywamy **iteracyjnym przycinaniem**.

Dzięki przycięciu znacznej liczby wag, możemy osiągnąć dużo mniejszą sieć, która daje niemal takie same wyniki jak ta sieć bez pruningu.

Baza danych

Do przeanalizowania działania powyżej trzech opisanych modeli skorzystaliśmy z trzech baz danych. Na dane składają się różnorodne zdjęcia kwiatków konkretnych gatunków. Pierwsza baza składa się z 5 gatunków kwiatów, gdzie każdy gatunek jest reprezentowany przez około 700-1000 zdjęć, druga baza składa się z 16 gatunków kwiatów, gdzie każdy gatunek również zawiera po około 700-1000 zdjęć, natomiast ostatnia baza zawiera 7 gatunków kwiatów i każdy gatunek jest reprezentowany przez 1600 zdjęć. Każdy z trzech modeli został przetestowany na trzech bazach zdjęć.

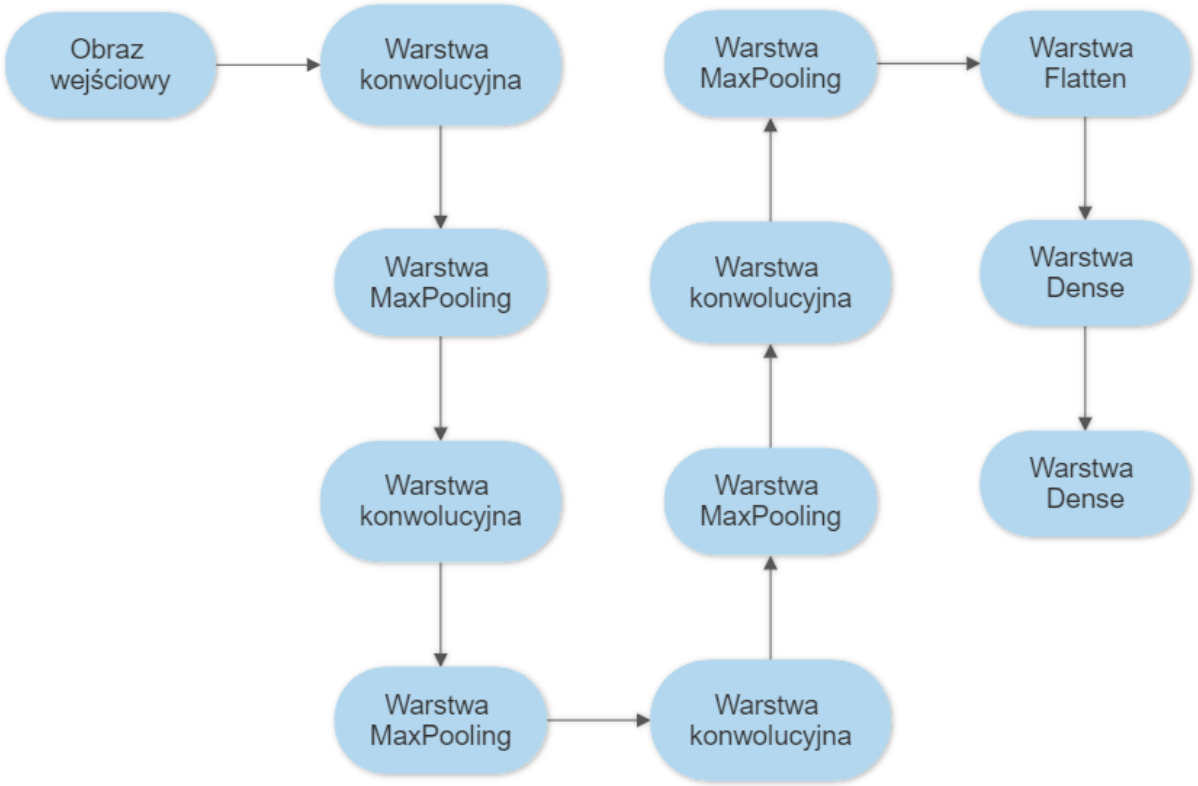
Aby móc wykorzystać bazę zdjęć w naszej prostej sieci musieliśmy zdefiniować funkcję, która przekonwertuje tę bazę do pliku .csv. W tym celu aby każdy obraz miał te same cechy, zmieniliśmy rozmiar zdjęć, tak aby każdy z nich był wielkości 128 x 128 pikseli, oraz przekonwertowaliśmy barwy RGB na HSV, w celu użycia parametru Hue jako jedynej wartości opisującej kolor, gdyż sieć nie jest przystosowana do działania na wektorach. Następnie dla każdego kwiatka dodaliśmy kolumnę, która mówi o tym, jakiego jest on gatunku, by móc w kolejnych krokach wyłuskać dla każdego kwiatu jego label, by stworzyć y_train oraz y_test.

Dla modeli zbudowanych przy pomocy biblioteki Keras, wszystkie wczytane zdjęcia zostały przeskalowane do wielkości 150x150, (w przypadku modelu referencyjnego do 300x300) a następnie przekształcone na wektory pikseli w formacie RGB. Tak przygotowane dane zostały zapisane w macierzy X. Ponadto macierz X została znormalizowana dzieląc ją przez 255, aby otrzymać wartości pikseli w przedziale od 0 do 1. Następnie przygotowaliśmy wektor Y etykiet, który został stworzony za pomocą funkcji LabelEncoder(). Przekształca ona nazwy kwiatów w liczby 1-5 /1-16 / 1-7 w zależności od użytej bazy. W ten sposób każda liczba będzie etykietą innego kwiatka. Następnie za pomocą funkcji *to_caterogical(Y,i)* (gdzie i to liczba gatunków kwiatów w danej bazie zdjęć) etykiety zostały przekształcone na format one-hot encoding. Technika ta polega na przekształceniu etykiet klas na wektory binarne o długości równej liczbie klas, gdzie tylko jedna wartość jest ustawiona na 1, a reszta na 0.
Ostatecznie stworzyliśmy zestawy treningowe i testowe dla macierzy X oraz Y za pomocą funkcji train_test_split() z parametrem test_size=0.25, który dzieli zbiór na stosunek 3:1, gdzie 25% stanowi zbiór danych testowych.

Realizacja zadania - implementacja modeli

Sieć konwolucyjna

Nasza sieć konwolucyjna składa się z 11 warstw. Struktura tej sieci konwolucyjnej została przedstawiona poniżej:



- **warstwa konwolucyjna**: w pierwszej warstwie konwolucyjnej zastosowaliśmy 32 filtry konwolucyjne, i jądro filtru konwolucyjnego o rozmiarze (5, 5). Sposób wypełniania brzegów został tak ustawiony, by rozmiar wyjściowy był taki sam jak wejściowy. W tej warstwie została użyta funkcja ReLU jako funkcja aktywacji. Kolejne trzy warstwy konwolucyjne została stworzone prawie tak samo - została tylko zwiększona dwukrotnie liczba filtrów konwolucyjnych (64) w drugiej warstwie oraz trzykrotnie (96) w trzeciej i czwartej. Rozmiar danych wejściowych to (150, 150, 3), gdzie 150 x 150 to rozmiar obrazu w pikselach, a wartość 3 tyczy się liczby kanałów tzn. kolorystyki RGB.
- **warstwa MaxPooling,** gdzie rozmiar okna dla operacji ustaliliśmy na (2, 2) dla obu użytych warstw.
- **warstwa Flatten:** ta warstwa konwertuje dane z dwuwymiarowej macierzy na jednowymiarową tablice.

- **warstwa Dense(w pełni połączona):** Pierwsza warstwa Dense jest warstwą w pełni połączoną, w której każdy neuron jest połączony z każdym neuronem z poprzedniej warstwy. Ta warstwa ma na celu przekształcenie danych z poprzedniej warstwy Flatten w odpowiednią reprezentację dla klasyfikacji. Liczba neuronów w pierwszej warstwie została ustalona na 512. Funkcją aktywacyjną jakiej użyliśmy była ReLU. W drugiej warstwie dense określiliśmy liczbę klas tzn. liczbę gatunków kwiatów; tutaj będzie przyjmowana wartość 5, 7 lub 16 w zależności od tego która baza kwiatków będzie używana. W tej warstwie użyliśmy funkcji aktywacji ***softmax*** (Pozwala ona w naszym przypadku interpretować wyjścia jako prawdopodobieństwa przynależności kwiatka do danego gatunku). Druga warstwa dense jest odpowiedzialna już za wygenerowanie ostatecznych wyników klasyfikacji.

Następnie skorzystaliśmy z funkcji kompilującej model. Pozwoliła ona nam także określić współczynnik uczenia (learning rate), który kontroluje szybkość uczenia modelu. Przyjęliśmy wartość tego współczynnika na *0.001*. Algorytmem optymalizacyjnym jakim użyliśmy był Adam (Adaptive Moment Estimation), który efektywnie dostosowuje tempo uczenia dla każdego parametru w sieci *(W praktyce optimizer Adam, po wyliczeniu gradientu dla wag, oblicza samodzielnie współczynniki uczenia dla każdego parametru w sieci, uwzględniając historyczne informacje o gradientach, a następnie wykorzystuje te współczynniki do aktualizacji wag modelu)*. Zdefiniowaliśmy także w funkcji kompilującej model funkcję straty (loss), która pozwala nam mierzyć różnicę pomiędzy przewidywaniami modelu a rzeczywistymi etykietami. Dokonujemy tego za pomocą funkcji *categorical_crossentropy (Ta funkcja straty jest odpowiednia dla problemów, w których wyjście modelu jest kodowane w postaci one-hot encoding, a dane należą do jednej z wielu klas. Categorical crossentropy mierzy odległość między prawdziwym rozkładem prawdopodobieństwa etykiet a przewidywanym rozkładem prawdopodobieństwa przez model, i stara się minimalizować tę odległość w trakcie treningu).* Dodatkowo parametr *metrics będzie* mierzyć procent poprawnych predykcji w stosunku do wszystkich próbek oraz służy do oceny modelu.

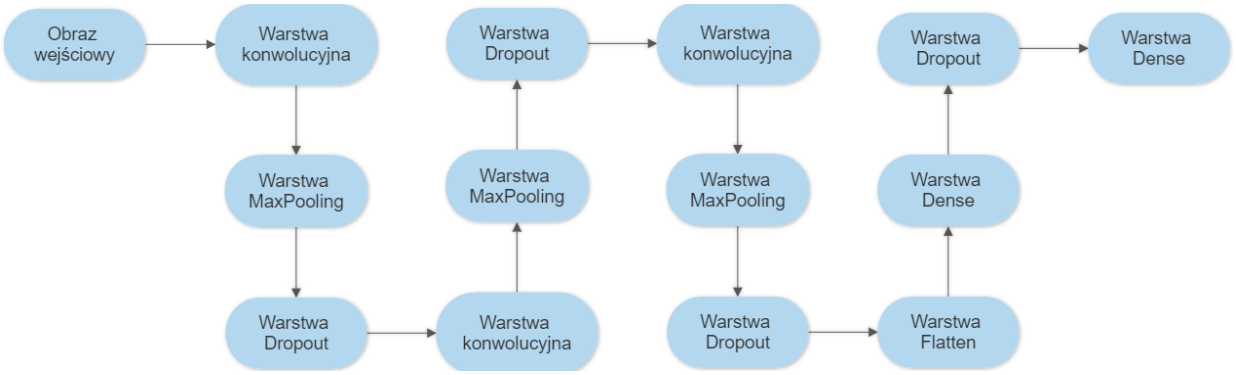
Po zdefiniowaniu całej sieci korzystamy z funkcji **fit**, aby dopasować model do danych treningowych. Ustaliliśmy liczbę epok na 50, czyli liczbę określającą ile razy model przejdzie przez cały zbiór treningowy. Ponadto, ustaliliśmy *batch_size* na 128, który określa liczbę próbek treningowych, które są używane przez model jednocześnie, zanim nastąpi aktualizacja wag. Do funkcji fit podajemy również dane treningowe wraz z ich prawdziwymi etykietami.

Wykorzystaliśmy jeszcze funkcje summary która pozwala nam zobaczyć całą architekturę modelu. Możemy zobaczyć dzięki niej liczbę parametrów do trenowania w każdej warstwie oraz ogólną liczbę parametrów.

Na samym końcu korzystając z funkcji save zapisujemy nasz stworzony model, aby mieć możliwość go przeanalizować.

Model teacher - student

W modelu teacher - student za nauczyciela przyjęliśmy zbudowaną przez nas wcześniej **sieć konwolucyjną**. Jako, że sieć studenta z założenia ma być zdecydowanie prostsza, składa się ona z trzech warstw konwolucyjnych wraz z maxpoolingiem. Po każdej warstwie poolingowej występuje warstwa dropoutowa.



Ponadto, parametry modelu zostały uproszczone do następujących:

- pierwsza warstwa konwolucyjna posiada 32 filtry, a dwie pozostałe 64. Dodatkowo użyto parametru *strides=3* dla warstwy drugiej i trzeciej, który określa przesunięcie okna kernela o 3 piksele w każdym kroku.
- w pierwszej warstwie Dense liczba neuronów została zredukowana ośmiokrotnie w porównaniu do nauczyciela (z 512 na 64)
- po pierwszej warstwie konwolucyjnej i poolingu dodano warstwę dropout o parametrze 0.1, po drugiej i trzeciej warstwie konwolucyjnej i poolingu dodano warstwę dropout o parametrze 0.2, i po pierwszej warstwie dense dodano warstwę dropout o parametrze 0.3. Parametry te mówią o prawdopodobieństwie odrzucenia neuronu.

Reszta parametrów warstw została ustalona tak, jak w modelu teacher.

W ten sposób udało nam się otrzymać **model ponad 56 razy mniejszy** (72,965 tysięcy do 4,143 miliona parametrów).
W celu stworzenia modelu studenta, który uczy się na bazie predykcji nauczyciela oraz swoich własnych, utworzyliśmy klasę Distiller. Klasa Distiller dziedziczy po klasie Model z biblioteki Keras i przeciąża dwie funkcje - train_step() oraz test_step(). Metoda train_step() uzyskuje predykcje z modelu nauczyciela oraz predykcję studenta.
Funkcja straty studenta obliczana jest za pomocą domyślnej funkcji CaterogicalCrossentropy(), natomiast funkcja straty przekazywania wiedzy

jest obliczana za pomocą KLDivergence(), która oblicza dywergencję Kullbacka-Leiblera między dwoma rozkładami prawdopodobieństwa, czyli w naszym przypadku mierzy różnicę między rozkładem prawdopodobieństwa predykcji nauczyciela, a rozkładem prawdopodobieństwa predykcji studenta. Wzór na kldivergence() został przedstawiony poniżej:

$$E(x|t) = -\sum_i (\widehat{y_i}(x|t) * log\, y_i(x|t))$$

Do ustalenia wpływu etykiet prawdziwych i tych od teachera na dalsze epoki definiujemy współczynnik **alpha**, który odpowiada za to, który z tych parametrów ma większe znaczenie w procesie uczenia:

$$loss = alpha * loss_{student} + (1 - alpha) * loss_{distillation}$$

Ostatnim zdefiniowanym parametrem jest **temperatura**, której wartość odpowiada za działanie funkcji aktywacyjnej softmax. W zależności od niej rozkład prawdopodobieństwa może być znormalizowany (temperatura równa 1), może być wygładzony im większa jest temperatura, lub bardziej ostry, gdy temperatura jest mniejsza od 0.

technika przycinania (pruningu)

W celu prezentacji tej techniki posłużymy się ponownie zbudowaną przez nas wcześniej siecią konwolucyjną.

Naszą sieć będziemy przycinać za pomocą wag. Do wykonania tej czynności posłuży nam pierwsza funkcja *weight_prune_dense_layer()*, która jako parametry przyjmuje wagi i biasy wcześniej stworzonego modelu oraz **k_sparsity**, która mówi nam o tym, jaki procent wag ma być usunięty, czyli ustawiony na wartość 0. Zadaniem funkcji obcinającej wagi jest posortowanie rosnąco indeksów elementów macierzy wag oraz biasów na podstawie ich wartości bezwzględnych. Następnie ilość wag oraz biasów określona poprzez k_sparsity zostaje ustawiona na 0.

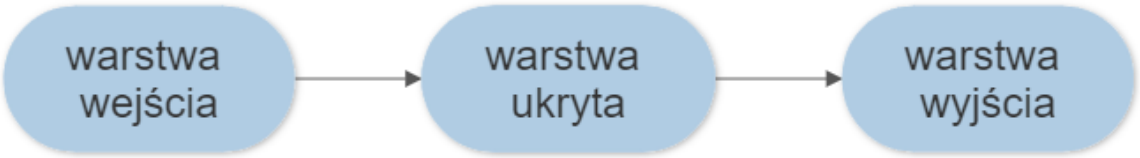
Aby zbudować nową sieć korzystamy z funkcji *sparsify_model(). Funkcja ta najpierw klonuje poprzedni model, a następnie* pobierając jej wagi i biasy, przycina je na ostatnich dwóch warstwach Dense stosując wyżej wspomnianą taktykę. Następnie stworzonemu modelowi nadpisujemy zmienione wagi, pozostawiając te inne bez zmian. Model ten jest kompilowany stosując wspomniany w powyższych metodach optimizer Adam, metryki oraz crossentropy, a następnie podawana jest jego dokładność po wykonaniu pruningu. Warstwy konwolucyjne nie są prunowane, ponieważ odpowiadają one

za obróbkę obrazu wejściowego i zmiana ich powoduje zbyt radykalne efekty co może prowadzić do niepożądanych konsekwencji.

W celu znalezienia k_sparsity dającego najlepsze efekty wykonujemy pruning dla przedziału [0, 0.95]. Po znalezieniu najlepszej wartości dla modelu zostaje dokonany fine-tuning, czyli dotrenowanie w 5 epokach w celu uzyskania jeszcze lepszego rozwiązania.

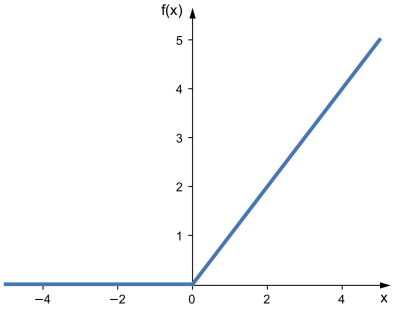
Prosta sieć neuronowa

Na samym końcu postanowiliśmy stworzyć "ręcznie" prostą sieć neuronową w celu zrozumienia podstawowego ich działania. Struktura tej sieci została przedstawiona poniżej:



- **warstwa wejścia** - wartości 16384 pikseli obrazu 128x128, podanych w wartościach modelu przestrzeni barw HSV.
- **warstwa ukryta** - warstwa 5 neuronów, których wagi oraz biasy przyjmują wartości w przedziale [-0.5, 0.5], a wykorzystywaną funkcją aktywacyjną jest ReLU

ReLU - funkcja aktywacyjna, która przyjmuje wartość wejścia, jeżeli jest ono większe od zera, przeciwnym razie wynosi 0.



$$f(x) = \max(0, x)$$

- **warstwa wyjścia** - warstwa 5 neuronów, których wagi oraz biasy przyjmują wartości w przedziale [-0.5, 0.5], a wykorzystywaną funkcją aktywacyjną jest softmax, na podstawie którego zwracane są prawdopodobieństwa, do jakiej klasy pasuje obraz wejściowy

---

softmax - funkcja aktywacyjna, która jako wejście przyjmuje wektor wartości i normalizuje go do postaci rozkładu prawdopodobieństwa dla każdego możliwego rezultatu. Normalizuje ona wartości wyjściowe tak, aby suma wszystkich wartości była równa 1.
Określony jest on wzorem:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \; for\; i = 1, \dots, K \;\wedge\; z = (z_1, \dots, z_k) \in R^K$$

Uczenie tego modelu będzie postępowało w następujący sposób:

1. wyliczenie parametrów aktualizacyjnych:
    a. parametr dZ_2 - error warstwy wyjścia - różnica pomiędzy outputem i poprawnymi labelami
    b. parametr dW_2 - wpływ wag warstwy wyjścia na error - iloczyn parametru dZ_2 i wartości aktywowanej warstwy ukrytej podzielonej przez ilość zdjęć wejściowych
    c. parametr db_2 - wpływ biasów warstwy wyjścia na error - suma wartości parametru dZ podzielona przez ilość zdjęć wejściowych
    d. parametr dZ_1 - error warstwy ukrytej - wagi warstwy warstwy wyjścia pomnożone przez parametr dZ_2 i wartości pomnożone jeszcze przez przez pochodną funkcji ReLU z nieaktywowanej warstwy ukrytej
    e. parametr dW_1 - wpływ wag warstwy ukrytej na error - parametr dZ_1 pomnożony przez wartości warstwy wejściowej, podzielony przez ilość danych wejściowych
    f. parametr db_1 - wpływ biasów warstwy ukrytej na error - suma wartości parametru dZ_1 podzielony przez ilość danych wejściowych

2. zaktualizowanie parametrów warstw:
    a. W_1 - wagi warstwy ukrytej - różnica pomiędzy aktualnymi wagami i iloczynu parametru dW_1 pomnożony przez learning rate
    b. b_1 - biasy warstwy ukrytej - różnica pomiędzy aktualnymi biasami i iloczynu parametru db_1 pomnożony przez learning rate
    c. W_2 - wagi warstwy wyjściowej - różnica pomiędzy aktualnymi wagami i iloczynu parametru dW_2 pomnożony przez learning rate
    d. b_1 - biasy warstwy wyjściowej - różnica pomiędzy aktualnymi biasami i iloczynu parametru db_2 pomnożony przez learning rate

Prezentacja osiągniętych wyników

model referencyjny - sieć konwolucyjna

Wyniki dokładności dla tego modelu dla każdej bazy zostały przedstawione poniżej. Jak widzimy, model ten osiąga bardzo dobre wyniki klasyfikacji.

Nr Bazy	**accuracy [%]**
[1]	89%
[2]	90%
[3]	92%

sieć konwolucyjna

Baza [1]

Po wytrenowaniu modelu na tej bazie uzyskał on *accuracy* **77.66%** Strata natomiast osiągnęła wartość **0.7397.**

Baza [2]

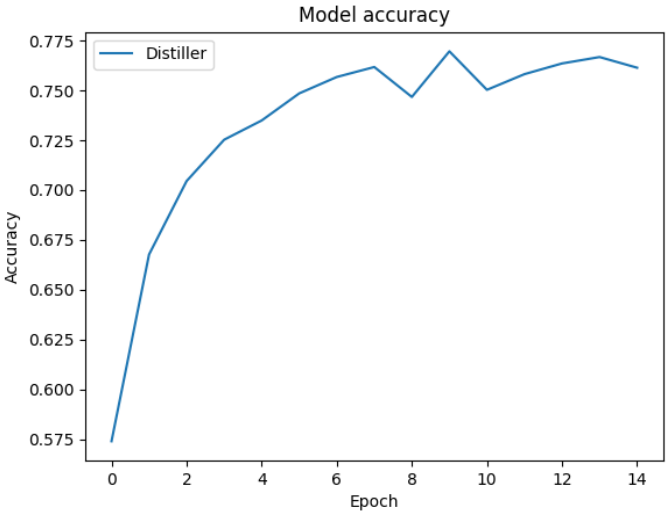
Po wytrenowaniu modelu na tej bazie uzyskał on *accuracy* **66.98%.** Strata natomiast osiągnęła wartość **3.0133**.

Wynik ten jest trochę mniej zadowalający natomiast wciąż nie najgorszy, w porównaniu z modelem na bazie pierwszej. Może to wynikać na przykład z gorszych do rozpoznania zdjęć. Baza ta jest także trudniejsza w porównaniu z pierwszą, ponieważ ma ponad trzykrotnie więcej gatunków, co także może powodować gorsze wyniki.

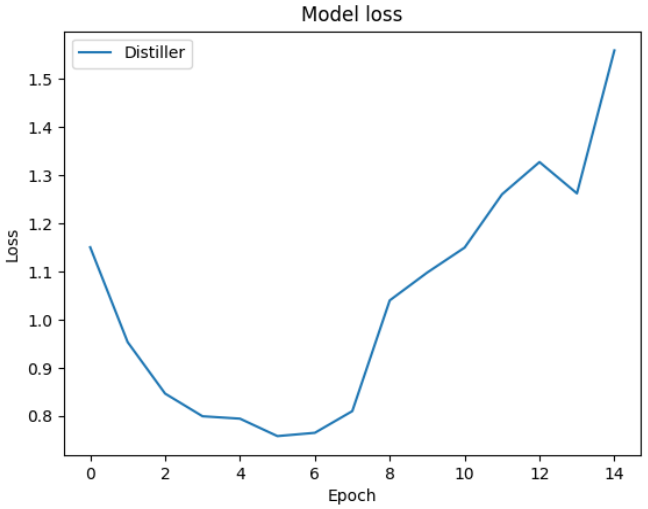
Baza [3]

Po wytrenowaniu modelu na tej bazie uzyskał on *accuracy* **76.32%.** Strata natomiast osiągnęła wartość **1.61.**

W celu dokładniejszej analizy modelu skorzystaliśmy z bazy trzeciej i stworzyliśmy wykresy prezentujące jak zmieniała się dokładność oraz strata dla kolejnych epok:



W analizowanym modelu, który trenował się przez 15 epok, zaobserwowano wzorzec na wykresie dokładności. Począwszy od pierwszej epoki, odnotowano stopniowy, lecz stabilny wzrost wartości validate accuracy, która początkowo wynosiła **0.575**, a po siedmiu epokach osiągnęła poziom 0.76. Jednakże, po siódmej epoce, dokładność treningowa dla tego tej bazy wyniosła 99%, a model nie był w stanie dostosować się do nowych danych.



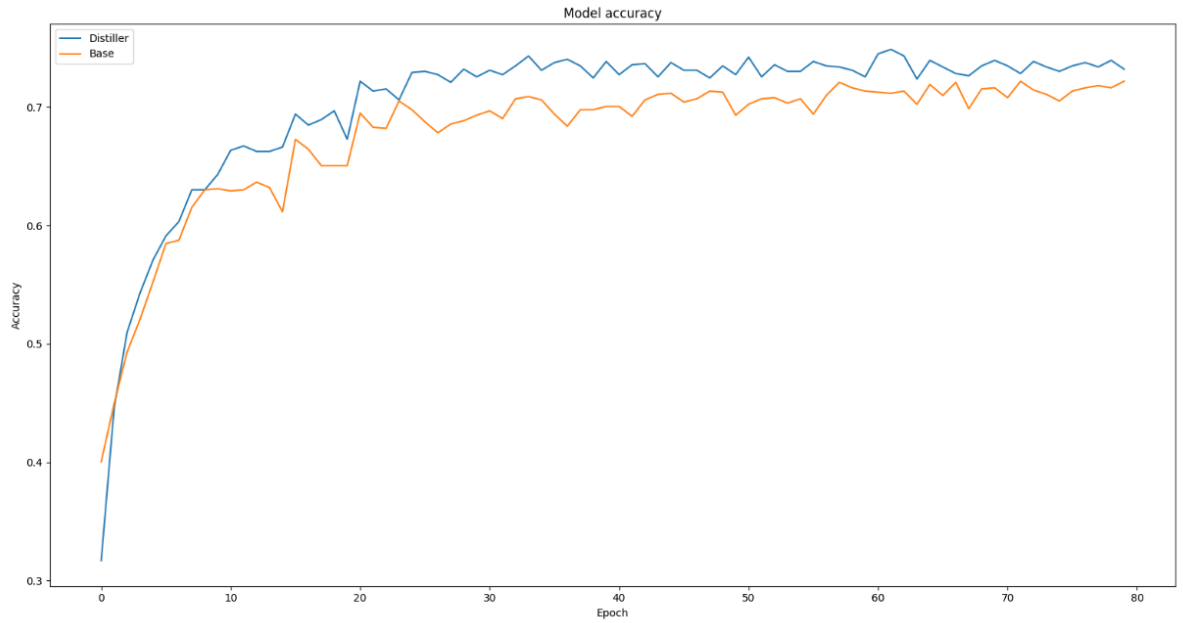
Taki scenariusz może wskazywać na **zjawisko przeuczenia (overfitting)**, gdzie model nauczył się idealnie dopasowywać do danych treningowych, ale nie potrafił ogólnie radzić sobie z nowymi danymi. W przypadku naszego modelu, osiągnięcie 99% dokładności treningowej sugeruje, że model nauczył się "zapamiętywać" dane treningowe, co może prowadzić do niewłaściwych predykcji na nowych, nieznanych danych.

Z naszą teorią pokrywa się wykres straty nowych danych, który również od 7 epoki zaczął drastycznie rosnąć. Jedną z przyczyn może być to, że model

zaczyna popełniać błędy na danych walidacyjnych, które wcześniej nie występowały.

model teacher-student

W celu zauważenia różnic między modelem uczonym z nauczycielem a modelem, który nie korzystał z tej wiedzy, przeprowadziliśmy eksperyment na bazie 5-klasowej, w którym oba modele były trenowane przez 80 epok.



Zauważyliśmy, że model uczony z nauczycielem osiągał lepsze rezultaty już po 10 epokach treningu, w porównaniu do modelu, który nie korzystał z tej dodatkowej wiedzy.

Aby ustalić wartości alpha i temperaturę dla działania modelu dla każdej z baz monitorowaliśmy jej działanie dla różnych wartości, aby postarać się uzyskać najlepsze wyniki. Dla każdej z baz przedstawiliśmy wykresy, które przedstawiają jak zmieniały się wartości dokładności studenta z pomocą teachera jak i studenta bez tej pomocy dla kolejnych epok.

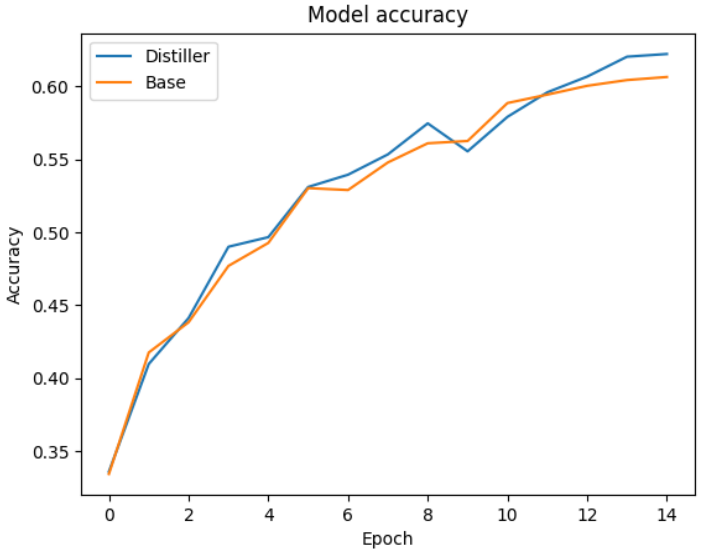
Baza [1]

Dla tego modelu wartości alpha i temperatury ustaliliśmy ich wartości kolejno na **0.02** i **23**.
Najlepszym rezultatem studenta była dokładność na poziomie **74.88%**, natomiast sieci studenta bez wsparcia teachera- **70.43%**. Taki wynik został osiągnięty po 15 epokach.

Baza [2]

W tym wypadku wartość alpha w porównaniu z pierwszą bazą nie zmieniła się tzn, przyjęła także wartość **0.02**, natomiast temperaturę zmieniliśmy na wartość **10.**

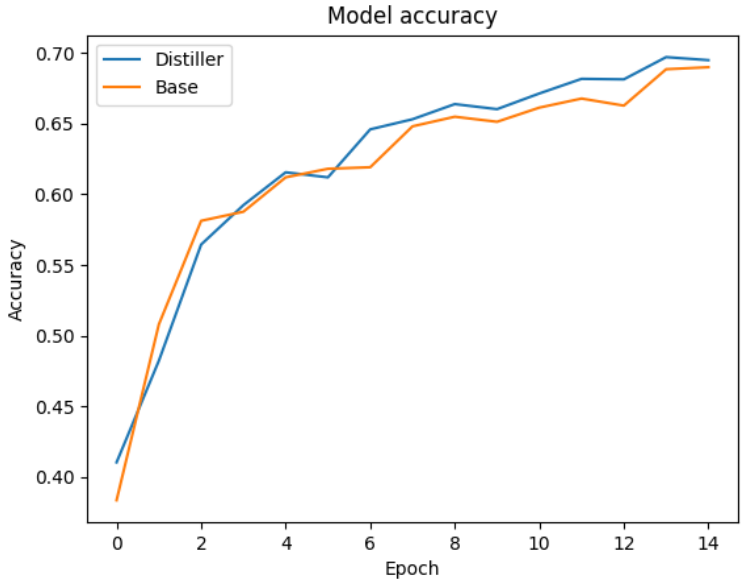
Najlepszym rezultatem studenta była dokładność na poziomie **62.23%**, natomiast sieci bez wsparcia teachera- **60.65%**. Taki wynik został osiągnięty po 15 epokach.



Baza [3]

Wartości alpha i temperatury zostały ustalone takie same jak dla bazy drugiej, czyli **0.02** i **10**.

Najlepszym rezultatem studenta była dokładność na poziomie **70.23%**, natomiast sieci bez wsparcia teachera- **68.99%**. Taki wynik został osiągnięty po 15 epokach.



Na powyższych wynikach widzimy, że student z dodatkową wiedzą zawsze poradził sobie lepiej niż model ten bez pomocy. Różnice pomiędzy dokładnościami modelu teachera, a studenta z uczeniem dla każdej z baz wyniosły kolejno: **2.78%, 4.75%, 6.09%** . Możemy zauważyć, że dokładności studenta zawsze są gorsze od nauczyciela, natomiast różnica ta nie jest bardzo duża. Dzięki temu uzyskaliśmy dużo prostsze modele, które z pomocą teachera pozwoliły osiągnąć tylko trochę gorsze wyniki od niego.

Poniżej przedstawiliśmy przykładowe, bardziej szczegółowe porównanie sieci studenta i teachera dla bazy pierwszej:

5 klas	Sieć konwolucyjna(nauczyciel)	Sieć studenta
Dokładność	77.66%	74.88%
Czas	122 ms/step	41 ms/step
Strata	0.7397	0.5067

Sieć nauczyciela osiąga wyższą dokładność (77.66%) niż sieć studenta (74.88%). To sugeruje, że sieć nauczyciela jest bardziej zaawansowana i lepiej radzi sobie z klasyfikacją danych. Jednak warto zauważyć, że sieć studenta jest znacznie szybsza, wykonując pojedynczy krok w czasie 41 ms, podczas gdy sieć nauczyciela potrzebuje 122 ms. Ta różnica w czasie może być ważna, zwłaszcza jeśli wymagane jest szybkie przetwarzanie (np. na smartfonach).

Oprócz tego, sieć studenta osiąga niższą wartość straty (0.5067) niż sieć nauczyciela (0.7397). Sugeruje to, że model studenta lepiej dopasowuje się do danych, niż model teachera.

pruning

Każdą bazę kwiatów monitorowaliśmy dla k_sparcity od 0 do 0.95 o krok 0.05, aby sprawdzić jaka wartość będzie dawała optymalne wyniki i równocześnie będzie znacznie zmniejszała sieć. Po tym kroku ustaliliśmy k_sparcity kolejno dla baz kwiatków 1, 2, 3: 0.65, 0.6, 0.6. Są to wartości, które pozwalają osiągnąć nam prawie tak samo dobry wynik, przy znacznym obcięciu wag.
W celu pokazania różnic między sieciami bez pruningu (k = 0.0), a pruningiem z naszym wybranym k optymalnym, oraz pruningiem obcinającym 99% wag przedstawiliśmy wykresy dla każdej bazy.

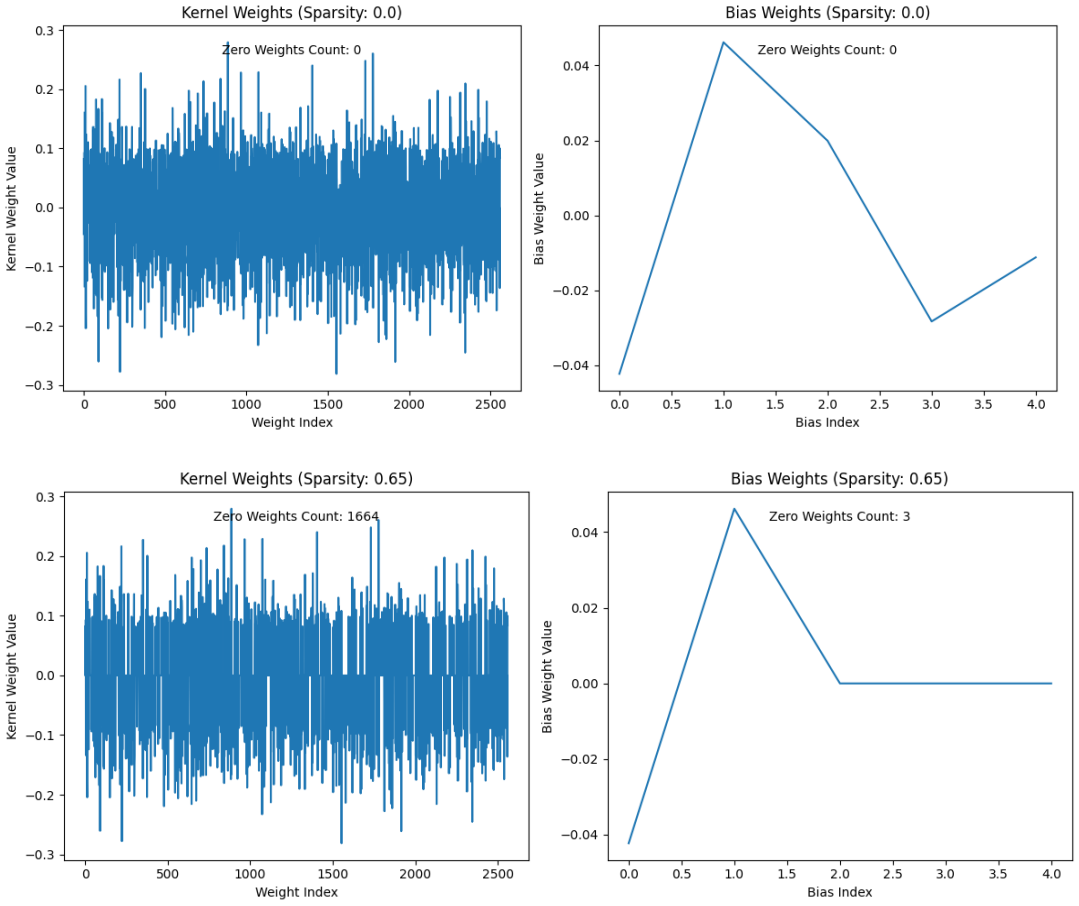
Baza [1]

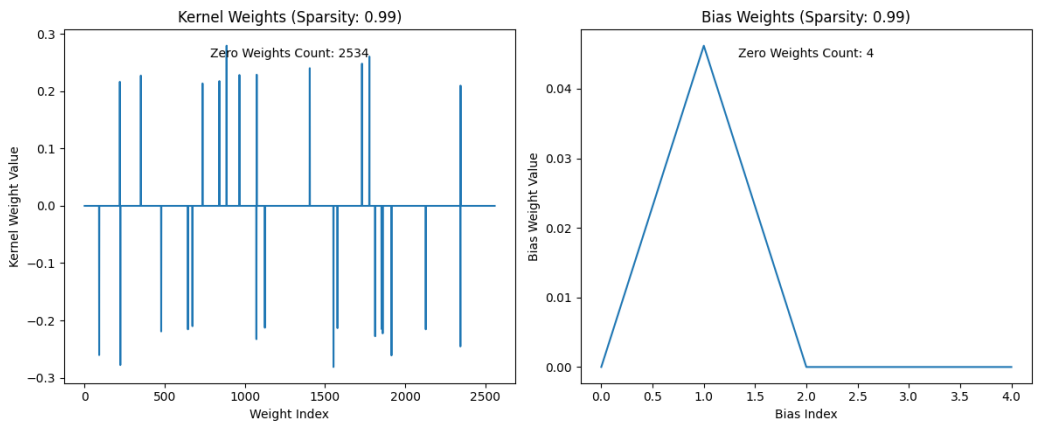
Bezpośrednio po obcięciu 65% wag sieci dała ona dokładność **76.27%**, a jej strata wyniosła **0.68662** Wynik ten jest o **1.39%** gorszy niż modelu bez obcięcia, mimo, że model z obcięciem stracił znaczną część wag. Po dotrenowaniu modelu przez 5 epok dał on ostatecznie accuracy **79.52%** oraz wartość straty **1.1072**.

```
cutoff:  332
cutoff:  3
k% weight sparsity:  0.65       Test loss: 0.68662      Test accuracy: 76.27 %%
Epoch 1/5
26/26 [==============================] - 45s 2s/step - loss: 0.2442 - accuracy: 0.9153 - val_loss: 0.7760 - val_accuracy: 0.7924
Epoch 2/5
26/26 [==============================] - 52s 2s/step - loss: 0.1228 - accuracy: 0.9598 - val_loss: 0.7804 - val_accuracy: 0.8054
Epoch 3/5
26/26 [==============================] - 48s 2s/step - loss: 0.0736 - accuracy: 0.9762 - val_loss: 0.9536 - val_accuracy: 0.7915
Epoch 4/5
26/26 [==============================] - 51s 2s/step - loss: 0.0522 - accuracy: 0.9849 - val_loss: 0.9386 - val_accuracy: 0.7961
Epoch 5/5
26/26 [==============================] - 51s 2s/step - loss: 0.0397 - accuracy: 0.9892 - val_loss: 1.1072 - val_accuracy: 0.7952
```

Po douczeniu modelu obciętego okazało się, że daje on o **1.86%**. większą dokładność niż model bez obcięć. Udało nam się zatem uzyskać model, który dość, że wykorzystuje niecałą połowę wag to dodatkowo uzyskuje lepsze wyniki niż model bez obcięć. W przypadku douczania wartość loss dla zbioru walidacyjnego wraz z kolejną epoką wzrasta, co może świadczyć o **overfittingu,** szczególnie, że dokładność dla danych treningowych wynosi prawie 100%.

Wykresy przedstawiające różnice w obcięciach:



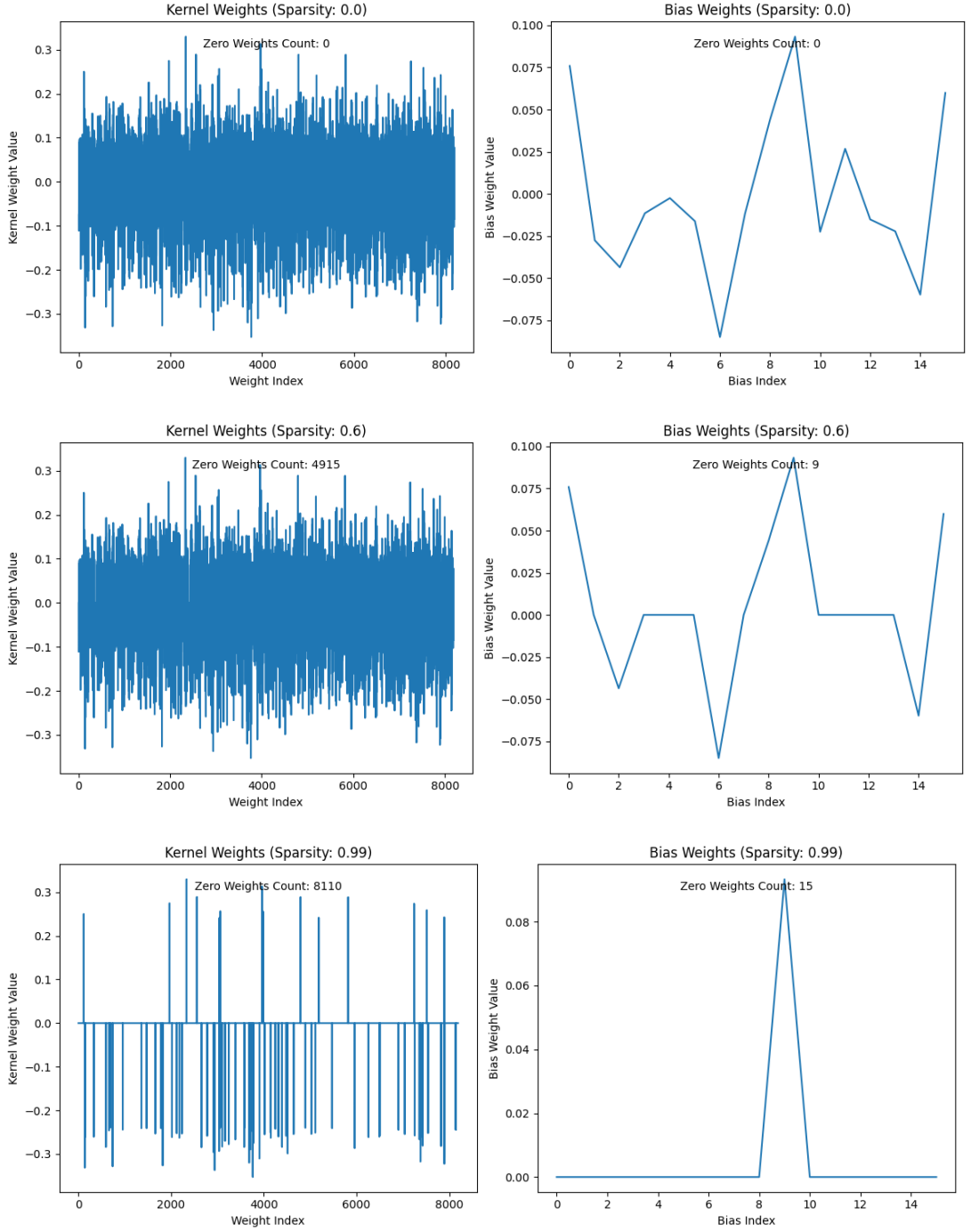


Baza [2]

Bezpośrednio po obcięciu 60% wag sieci dała ona dokładność **65.12%**, a jej strata wyniosła **2.78797**. Jest to o **1.86%** gorszy wynik niż modelu bez obcięcia (accuracy modelu bez obcięcia: 66.98%), mimo, że model z obcięciem stracił znaczną część wag. Po dotrenowaniu modelu przez 5 epok dał on ostatecznie accuracy **65.25%** oraz wartość straty **2.8351** (wartość straty bez obcięcia: 3.0133). Widzimy w tym wypadku, że wynik ten nieznacznie wzrósł. Możemy dostrzec, że douczając model wartość straty rosła dosyć gwałtownie, co oznacza, że model ten mógł wpaść w minimum lokalne, skąd może wynikać minimalny efekt poprawy wyniku po dotrenowaniu. Ostatecznie jednak widzimy, że model z obcięciem zawierający o ponad połowę mniej wag nadal potrafi prawie tak samo dobrze sklasyfikować kwiatka jak duży model bez obcięcia. Różnica ta wynosi zaledwie **1.72%**.

```
cutoff:  307
cutoff:  9
k% weight sparsity:  0.6        Test loss: 2.78797      Test accuracy: 65.12 %%
Epoch 1/5
90/90 [==============================] - 185s 2s/step - loss: 0.0418 - accuracy: 0.9879 - val_loss: 2.7975 - val_accuracy: 0.6667
Epoch 2/5
90/90 [==============================] - 927s 10s/step - loss: 0.0192 - accuracy: 0.9941 - val_loss: 2.9816 - val_accuracy: 0.6612
Epoch 3/5
90/90 [==============================] - 152s 2s/step - loss: 0.0077 - accuracy: 0.9976 - val_loss: 2.8744 - val_accuracy: 0.6690
Epoch 4/5
90/90 [==============================] - 159s 2s/step - loss: 0.0131 - accuracy: 0.9960 - val_loss: 3.0999 - val_accuracy: 0.6596
Epoch 5/5
90/90 [==============================] - 163s 2s/step - loss: 0.0368 - accuracy: 0.9884 - val_loss: 2.8351 - val_accuracy: 0.6525
```

Wykresy przedstawiające różnice w obcięciach:



Baza [3]

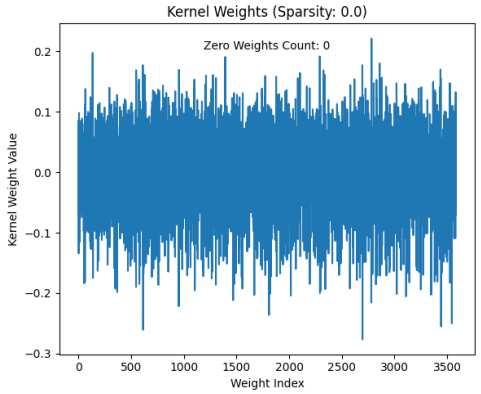
Bezpośrednio po obcięciu 60% wag sieci dała ona dokładność **76.6%**, a jej strata wyniosła **1.26970.** Dokładność ta okazała się być się **0.47% lepsza** niż modelu bez obcięcia (76.13%), a ponadto model z obcięciem stracił znaczną część wag. Po dotrenowaniu modelu przez 5 epok dał on ostatecznie accuracy **76.85%** oraz wartość straty **1.5292**. Dokładność po douczeniu jeszcze trochę

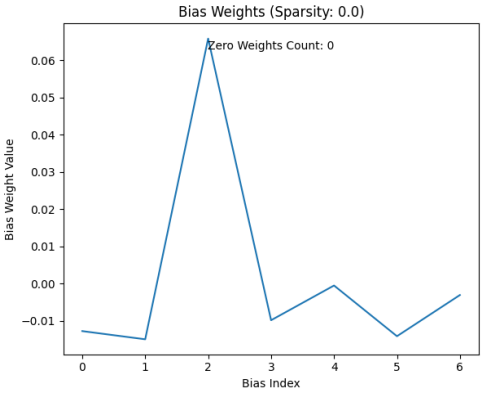
zwiększyła się i różnica pomiędzy studentem douczonym a teacherem wynosi teraz **0.72%**. Dla modelu z tymi danymi wejściowymi udało nam się otrzymać dokładność nawet lepszą od teachera, mimo, że model ten ma obciętą znaczną część wag.

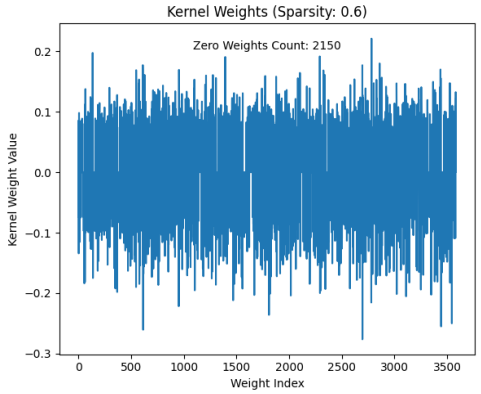
```
88/88 [==============================] - 8s 83ms/step - loss: 1.5597 - accuracy: 0.7613
```

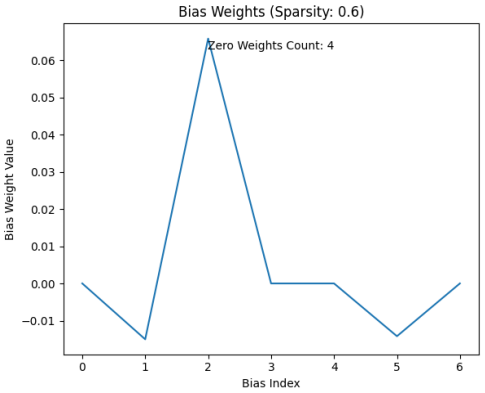
```
cutoff:  307
cutoff:  4
k% weight sparsity:  0.6        Test loss: 1.26970      Test accuracy: 76.60 %%
Epoch 1/5
66/66 [==============================] - 123s 2s/step - loss: 0.0739 - accuracy: 0.9792 - val_loss: 1.4958 - val_accuracy: 0.7717
Epoch 2/5
66/66 [==============================] - 116s 2s/step - loss: 0.0524 - accuracy: 0.9834 - val_loss: 1.3608 - val_accuracy: 0.7506
Epoch 3/5
66/66 [==============================] - 380s 6s/step - loss: 0.0407 - accuracy: 0.9888 - val_loss: 1.4383 - val_accuracy: 0.7728
Epoch 4/5
66/66 [==============================] - 113s 2s/step - loss: 0.0246 - accuracy: 0.9931 - val_loss: 1.4400 - val_accuracy: 0.7713
Epoch 5/5
66/66 [==============================] - 117s 2s/step - loss: 0.0292 - accuracy: 0.9929 - val_loss: 1.5292 - val_accuracy: 0.7685
```

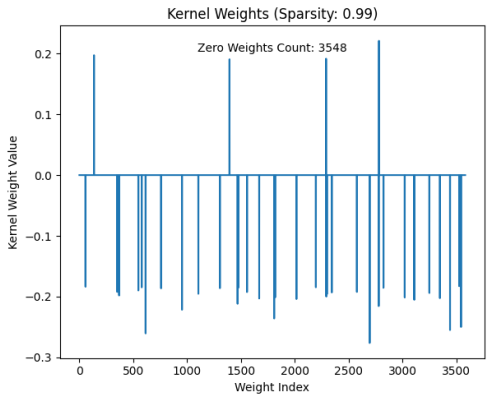
Wykresy przedstawiające różnice w obcięciach:

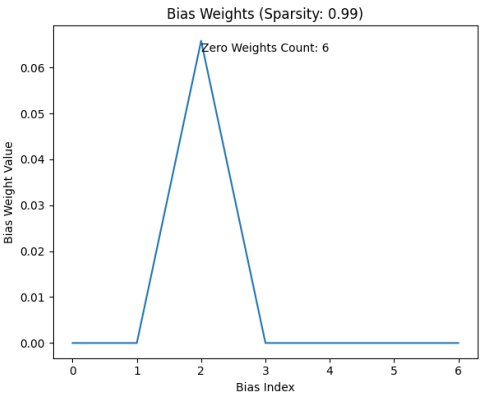












prosta sieć neuronowa

W celu oceny działania zbudowanej przez nas prostej sieci neuronowej, wytrenowaliśmy ją na pierwszej bazie. Wynikiem tego uczenia była dokładność wynosząca **27.76%.**

Niestety zdecydowanie widać, że taka prosta sieć nie sprawdza się do takiej klasyfikacji zdjęć. Dokładność tej sieci odbiega od wartości uzyskanych przez nas w pozostałych trzech modelach. Budowa takiej sieci natomiast pozwoliła nam bardziej zrozumieć je od samych podstaw, aby móc lepiej posługiwać się zdecydowanie bardziej zaawansowanymi modelami sieci.

Dyskusja

Sieć konwolucyjna może być skutecznym narzędziem do klasyfikacji obrazów. Zastosowanie tej sieci w klasyfikacji kwiatków ze zdjęć okazało się efektywne i przyniosło poprawne wyniki dla każdej z trzech baz. Najmniej efektywny okazał się model na drugiej bazie zdjęć (około 68%), a najlepsza okazała się dla bazy pierwszej (około 77%).
Sieć konwolucyjna jest szczególnie przydatna w zadaniach przetwarzania wizyjnego, ponieważ może wykrywać wzorce i cechy w obrazach, co umożliwia skuteczną klasyfikację różnych klas kwiatów.

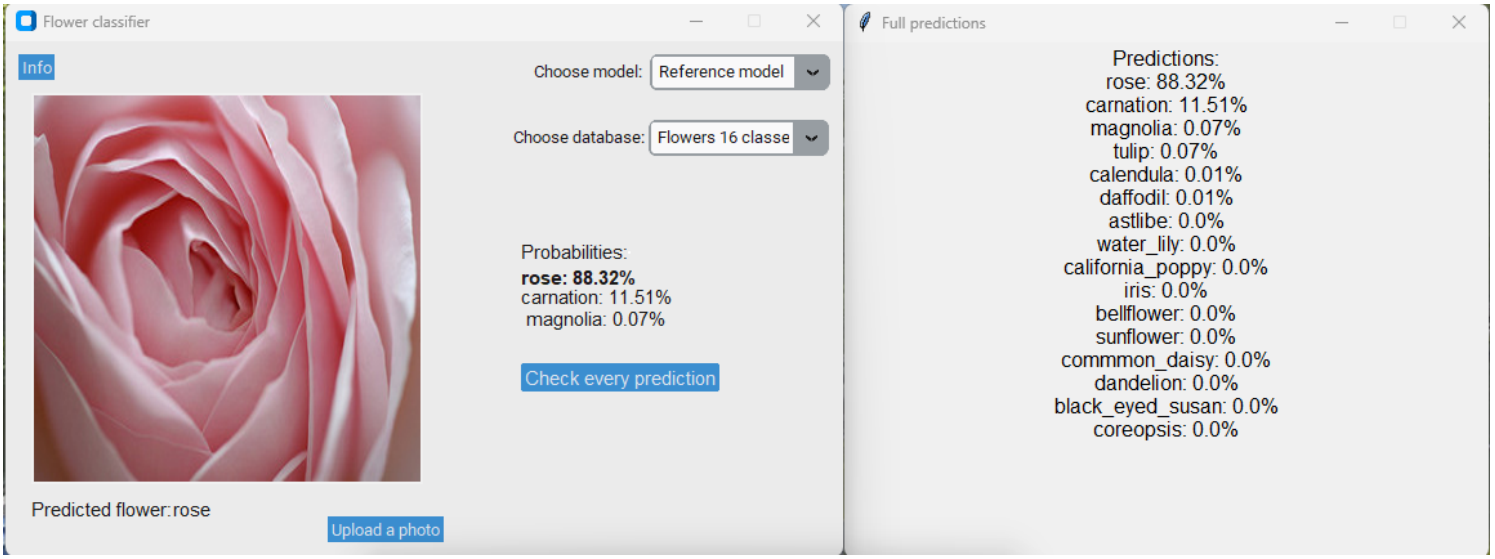
Zastosowanie modelu **teacher-student** pozwoliło nam na transfer wiedzy pomiędzy sieciami. Nauczyciel przekazywał swoją wiedzę o klasyfikacji kwiatów studentowi, co umożliwiło mu osiągnięcie lepszych wyników w klasyfikacji niż gdyby uczył się samodzielnie. Student zawsze osiągał trochę gorsze wyniki niż teacher, ale różnice te były tylko paru-procentowe, a sieć studenta była znacznie mniejsza i szybsza. Podsumowując, sieć nauczyciela jest bardziej skomplikowana, osiąga wyższą dokładność, ale działa wolniej. Sieć studenta, mimo nieco niższej dokładności, jest szybsza i może osiągać niższą wartość straty. Zatem model teacher-student może być przydatny w przypadkach, **gdy mamy ograniczone zasoby obliczeniowe** lub **chcemy przyspieszyć proces uczenia się** poprzez wykorzystanie istniejącego, bardziej rozbudowanego modelu jako nauczyciela.

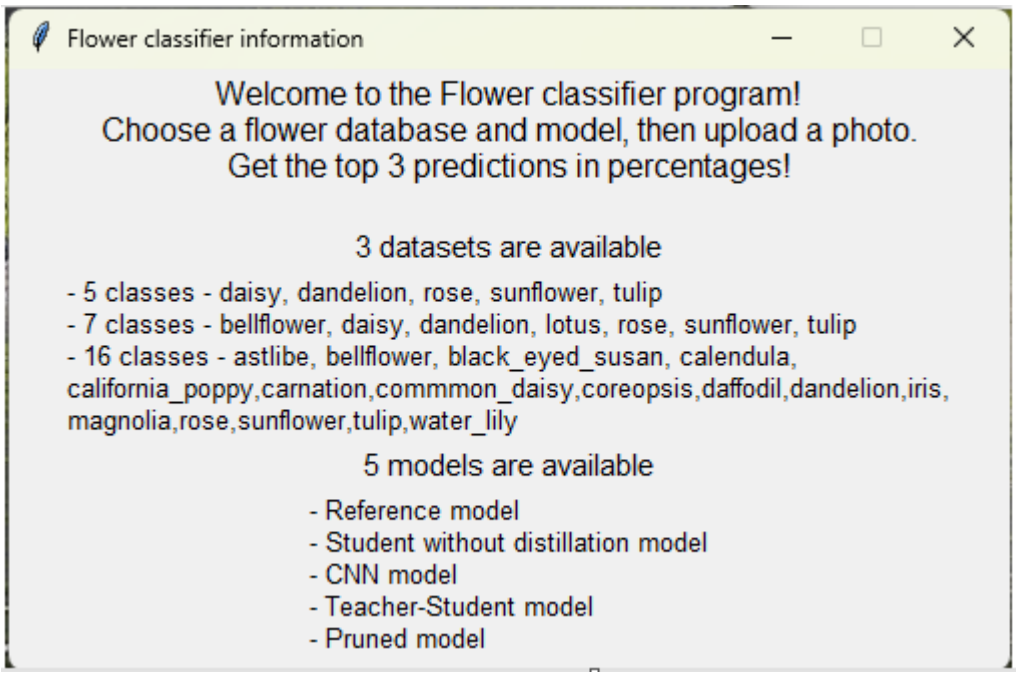
Proces **pruningu** sieci konwolucyjnej pozwolił na redukcję liczby parametrów i rozmiaru sieci, zachowując jednocześnie jej zdolność do skutecznej klasyfikacji kwiatów. Doprowadziło to do zmniejszenia złożoności obliczeniowej i zużycia pamięci, bez większego pogorszenia wydajności. Udało nam się nawet dla dwóch danych wygenerować lepsze dokładności, przy dużo mniejszej liczbie wag.

Przycinanie sieci może być przydatne, gdy chcemy **zoptymalizować wykorzystanie zasobów** sprzętowych lub **uczynić sieć bardziej mobilną**.

klasyfikacja w praktyce

Aby zaprezentować działanie klasyfikacji kwiatów stworzyliśmy aplikację okienkową, w której użytkownik ma możliwość wyboru dowolnego zdjęcia na komputerze, wybrania bazy kwiatów oraz modelu na którym predykcja ma zostać oszacowana. Po dokonaniu tych kroków użytkownik otrzymuje 3 klasy, których prawdopodobieństwo dopasowania obrazu było najwyższe. W zakładce "Info" można dowiedzieć się więcej na temat klas i modeli w aplikacji.
Pod przyciskiem "Check every prediction" użytkownik może sprawdzić rozkład prawdopodobieństwa dla wszystkich klas bazy.





Bibliografia

1) Baza zdjęć kwiatków nr 1:
https://www.kaggle.com/datasets/alxmamaev/flowers-recognition

2) Baza zdjęć kwiatków nr 2:
https://www.kaggle.com/datasets/l3llff/flowers

3) Baza zdjęć kwiatków nr 3:
https://www.kaggle.com/datasets/nadyana/flowers

4) Modele referencyjne:
- https://www.kaggle.com/code/aryashah2k/flower-image-classification-transfer-learning

5) Inne:
- Funkcja straty sieci teacher-student:
http://cs231n.stanford.edu/reports/2016/pdfs/120_Report.pdf

- Wzorzec knowledge distillation:
https://keras.io/examples/vision/knowledge_distillation/

- Wzorzec pruning:
https://colab.research.google.com/github/matthew-mcateer/Keras_pruning/blob/master/Model_pruning_exploration.ipynb#scrollTo=BXrVJhdZ5P4P