

PROJEKT ZESPOŁOWY

KLASYFIKACJA ZDJĘĆ OWOCÓW I WARZYW

PROJEKT REALIZOWANY ZE WSPÓŁPRACĄ Z FIRMĄ
INSERT

Mateusz Biedka 263527, Cezary Storczyk 263487

Filip Bimkiewicz 263464, Joanna Rogula 263536

Spis treści

1 Wprowadzenie	2
1.1 Cel Projektu	2
1.2 Zastosowanie komercyjne	2
1.3 Wykorzystane technologie	2
2 Zbiory danych	3
3 Wstęp teoretyczny	3
3.1 Teoria sieci	3
3.2 Funkcje straty	4
3.3 Transfer learning	5
4 Implementacja modelu	5
4.1 Biblioteki	5
4.2 Hiperparametry	5
4.3 Generowanie par/trójkę	5
4.4 Pobieranie modelu do transfer learningu	6
4.5 Definiowanie funkcji straty	6
4.6 Zdefiniowanie zachowania wag	6
4.7 Wczesne zatrzymanie i redukcja learning rate	6
4.8 Optymalizator	7
4.9 Wybieranie reprezentantów	7
5 Podjęte decyzje	7
5.1 Obsługa danych	7
5.2 Metryka	7
5.3 Reprezentanci	8
5.4 Problem overfittingu	8
5.5 Fine-tuning	8
6 Eksperymenty	9
7 Wyniki	14
8 Wnioski	16

1 Wprowadzenie

1.1 Cel Projektu

Celem projektu jest implementacja i rozwój systemu opartego na modelu one-shot learning do rozpoznawania produktów spożywczych na podstawie zdjęć. Architektura Siamese Network, którą wykorzystano, umożliwia rozpoznawanie nowych typów obiektów, które nie były częścią zestawu treningowego, dzięki czemu, w praktyce, można dodawać nowe produkty spożywcze bez konieczności ponownego trenowania modelu, co znacznie ogranicza potrzebę zaangażowania programistów oraz zasobów czasowych i sprzętowych. Wystarczy dostarczenie kilku zdjęć nowego produktu.

1.2 Zastosowanie komercyjne

Rozwijany system może być zaimplementowany w kasach samoobsługowych, które dynamicznie zyskują popularność w sklepach spożywczych. Zastosowanie modelu do rozpoznawania owoców znacznie przyspieszyłoby zakupy i wyeliminowało potrzebę ręcznego wyszukiwania produktu, dzięki czemu sklepy mogłyby skierować więcej klientów do kas samoobsługowych, co odciążałoby pracowników. Dzięki architekturze Siamese Network, sklep może łatwo dodawać nowe produkty, wgrywając ich zdjęcia do systemu bez potrzeby angażowania specjalistów technicznych.

Przykład użycia: klient korzysta z kasy samoobsługowej, kładzie na wagę wybrany owoc, a kamera robi zdjęcie i przesyła je do modelu. Na ekranie natychmiast wyświetlanych jest kilka najbardziej podobnych owoców z bazy danych.

Korzyści dla sklepów:

- Satysfakcja klienta - brak konieczności ręcznego przeglądania katalogów z produktami i szukania odpowiedniego produktu
- Mniejsze kolejki - szybciej zrealizowane zakupy przez klientów
- Reklamowanie nowego rozwiązania jako innowacyjnego systemu z wykorzystaniem sztucznej inteligencji
- zachęcanie klientów do nie używania plastikowych reklamówek - model dokładniej określi rodzaj produktu bez dodatkowych przedmiotów na kasie

1.3 Wykorzystane technologie

- Środowisko - Conda, Kaggle
- Język - Python 3.12.2
- Narzędzia sieci neuronowych: TensorFlow 2.16.1, Keras 3.3.3
- Główna architektura sieci neuronowej: Siamese Network
- Wstępnie przetrenowane sieci jako baza Siamese Network: VegFru, VGG, EfficientNet, Xception, ResNet
- Prezentacja wyników: Jupyter Notebook 8.6.0, matplotlib 3.8.4
- Obsługa wielowymiarowych danych: numpy 1.26.4

2 Zbiory danych

- Fruit-360 dataset - zawiera 90380 zdjęć owoców i warzyw bez tła i w różnych konfiguracjach. Najprostszy zbiór danych wykorzystany został podczas pisania funkcji przetwarzających dane, skonstruowania modelu i jego wstępnego przetestowania.



- Fruit Recognition - zawiera 44406 zdjęć owoców umieszczonych na srebrnej tacę. Zdjęcia wykonane z różnymi ilościami owoców i w różnych konfiguracjach co jest istotne z perspektywy projektu. Zdjęcia zawarte w zbiorze danych mocno odzwierciedlają docelowe warunki komercyjne.



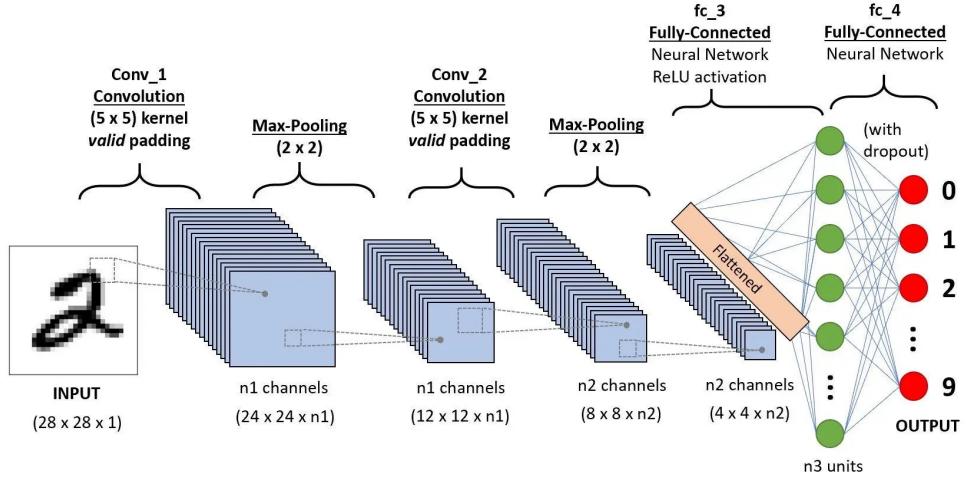
- VegFru - zawiera 160737 zdjęć podzielonych na 92 kategorie owoców oraz 200 kategorii warzyw. Zdjęcia są zrobione w różnych miejscach i w różnych konfiguracjach. Docelowym rozwiązańiem było przetrenowanie modelu na tym zbiorze danych ze względu na jego rozmiar i różnorodność a następnie przetestowanie na zbiorze Fruit Recognition. Takie podejście eliminuje wystąpienie overfittingu związanego z trenowaniem i testowaniem na tym samym zbiorze danych.



3 Wstęp teoretyczny

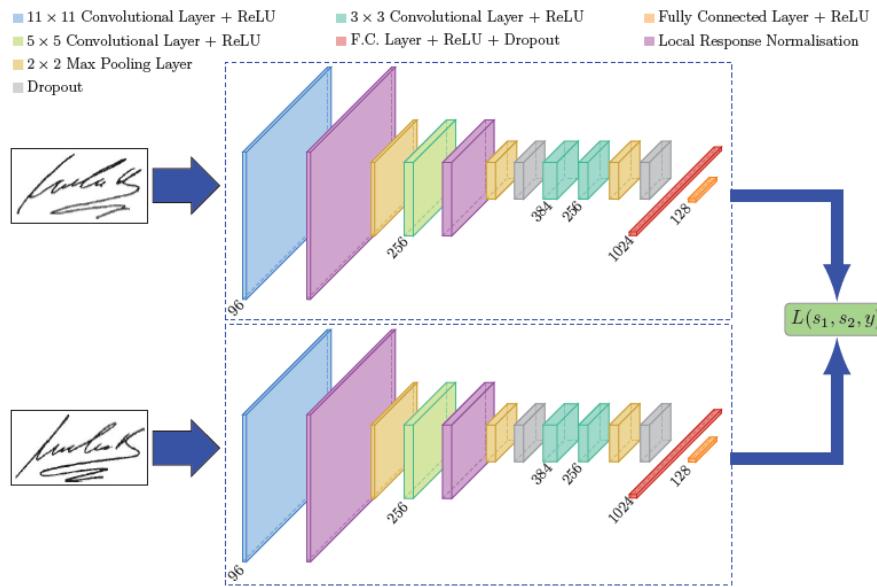
3.1 Teoria sieci

Konwolucyjna sieć neuronowa (CNN) to rodzaj architektury sieci neuronowej głębokiego uczenia się powszechnie stosowanej w przetwarzaniu obrazu i wideo. Warstwa konwolucyjna nakłada filtry na obraz wejściowy, aby wyodrębnić wzorce, warstwa Max-Pooling zmniejsza próbki obrazu, aby zmniejszyć ilość obliczeń, a warstwa Fully-Connected dokonuje ostatecznej prognozy. Sieć uczy się optymalnych filtrów poprzez propagację wsteczną i opadanie gradientowe.



Rysunek 1: Budowa sieci konwolucyjnej.

Syjamska sieć neuronowa (inaczej bliźniacza sieć neuronowa) to architektura sieci wykorzystująca te same wagi podczas pracy równoległej pracy na dwóch różnych wektorach wejściowych w celu obliczenia porównywalnych wektorów wyjściowych.



Rysunek 2: Budowa sieci syjamskiej.

3.2 Funkcje straty

Funkcje straty które umożliwiają nauczenie się problemu przez sieć są "Contrastive Loss" i "Triplet Loss". Są to funkcje straty które umożliwiają interpretację produktów na podstawie ich położenia w n-wymiarowej przestrzeni.

Contrastive loss bierze wyjście sieci za pozytywny przykład, oblicza jej odległość do przykładu tej samej klasy i kontrastuje to z odległością do przykładów negatywnych. Innymi słowy, błąd jest niski, jeśli próbki dodatnie są zapisane do podobnych (bliższych) reprezentacji, a negatywne przykłady są zapisane do różnych (dalszych) reprezentacji. Po procesie trenowania tworzą się n-wymiarowe chmury punktów, gdzie każda z chmur to jedna klasa.

Triplet loss jest funkcją straty, w której wejście referencyjne (anchor) jest porównywane z pasującym wejściem (positive) i niepasującym wejściem (negative). Odległość od anchora do pozytywnego przykładu jest minimalizowana, a odległość od anchora do negatywnego przykładu jest maksymalizowana.

3.3 Transfer learning

Transfer learning polega na wykorzystaniu wiedzy modelu zdobytej podczas wykonywania jakiegoś zadania, w celu zwiększenia wydajności realizacji powiązanego zadania. Na przykład w przypadku klasyfikacji obrazów wiedzę zdobytą podczas nauki rozpoznawania samochodów można zastosować przy próbie rozpoznawania ciężarówek. Ponowne wykorzystanie informacji z wcześniej wyuczonych zadań znaczaco poprawia efektywność uczenia się. Dzięki takiemu podejściu możliwe jest wytrenowanie modelu pomimo posiadania stosunkowo małej ilości danych. W projekcie wykorzystano tą technikę, korzystając z udostępnionych przez Keras modeli takich jak VegFru, VGG, EfficientNet, Xception czy ResNet.

4 Implementacja modelu

4.1 Biblioteki

Wykorzystane biblioteki:

- Tensorflow i Keras - do budowy i trenowania modelu
- numpy - do obliczeń
- livelossplot - do wizualizacji wyników

4.2 Hiperparametry

- image_size - rozmiar zdjęć, który należy dostosować do zainportowanego modelu
- epochs - liczba epok, należy dobrąć tak aby model miał wystarczająco czasu na nauczenie, jednocześnie trzeba mieć na uwadze możliwość przeuczenia (overfitting)
- batch_size - liczba przetwarzanych obrazów w jednej paczce, im większa paczka tym mamy stabilniejsze aktualizacje gradientów, tym samym stabilniejszą naukę modelu
- margin - margines, czyli zakres odległości dobrany do funkcji straty, wpływa na uczenie podczas porównywania zdjęć między sobą
- training_pairs, validation_pairs, training_triplets, validation_triplets - liczba par/trójk do trenowania i walidacji modelu, im większa ilość par/trójk tym mamy lepszą reprezentację tego czego chcemy model nauczyć

4.3 Generowanie par/trójk

Do generowania par/trójk użyto generatora, dzięki czemu nie wczytujemy wszystkich danych do pamięci RAM komputera, a tylko ścieżki plików ze zdjęciami, co znaczaco przyśpiesza proces uczenia. Generator podczas uczenia wczytuje jedynie kolejną paczkę danych zamiast całego zbioru danych. Budowanie par/trójk zdjęć obsługiwane jest w

klasie DatasetFactory. Metoda budująca generator w tej klasie, pobiera przy wywołaniu 5 wartości:

- ds_path - ścieżka do zdjęć
- image_size - rozmiar zdjęć po konwersji
- batch_size - ilość par/trójk generowanych każdorazowo do nauczania
- max_iterations - sumaryczna ilość zdjęć do wygenerowania
- method - funkcja straty, z której aktualnie korzystamy: "contrastive_loss" dla par, "triplet_loss" dla trójk

Aby przetestować działanie generatora, pobieramy pojedynczy zestaw par/trójk poprzez: `next(iter(train_dataset))` i wyświetlamy wygenerowane zdjęcia.

4.4 Pobieranie modelu do transfer learningu

Z `tensorflow.keras.applications` importujemy wybrany model. Następnie definiujemy wagi które chcemy wgrać do modelu, używamy zawsze wag "imagenet". W celu nadbudowania modelu, usuwamy najwyższą warstwę, aby móc dodać naszą warstwę gęstą.

Nadbudowujemy "od dołu", poprzez zdefiniowanie jakie rozmiary wejść chcemy wrzucać do modelu oraz "od góry", przez dorzucenie warstw gęstych i normalizację embeddin-gów.

4.5 Definiowanie funkcji straty

W celu skonstruowania sieci Syjamskiej, użyliśmy warstwy Lambda, która pozwoliła nam na zdefiniowanie własnej funkcji przekształcenia. Używamy jej do tego aby obliczyć odległość między dwoma osadzeniami (embeddings) zdjęć na wejściu.

4.6 Zdefiniowanie zachowania wag

Model który wgraliśmy domyślnie ma ustawione wagi jako trenowalne. Aby skutecznie wytrenować model, niezbędne jest „zamrożenie” wag w warstwach z modelu bazowego, a trenowanie jedynie dodanych warstw gęstych. Dzięki takiemu podejściu nie jest naruszana zdolność sieci bazowej do rozpoznawanie kształtów, a jedynie poszerzamy jej umiejętności o rozpoznawanie konkretnych obiektów, w tym przypadku owoców.

4.7 Wczesne zatrzymanie i redukcja learning rate

Wczesne zatrzymanie (EarlyStop) służy do zatrzymania procesu uczenia sieci i cofnięcia się do najlepszego wyniku, w przypadku gdy model zaczął się przetrenowywać (overfitting). Żeby ocenić czy wystąpił overfitting, podczas uczenia śledzona jest wartość błędu walidacyjnego. Jeśli błąd rośnie lub utrzymuje się na tym samym poziomie od przy-najmniej 3 epok, przy jednoczesnym spadku błędu trenowania, jest to oznaka overfittingu.

4.8 Optymalizator

Przy wybieraniu optymalizatora sugerowaliśmy się ogólną sprawnością. Podczas pierwszego uczenia sieci wybrany został optymalizator Adam. Jest to jeden z najlepszych optymalizatorów ze względu na zmienny współczynnik uczenia. Z kolei w procesie Fine Tuningu wybraliśmy SGD. SGD jest bezpiecznym rozwiązaniem w tym przypadku, z racji stałego learning rate.

4.9 Wybieranie reprezentantów

Nasz projekt opiera się o łatwość użycia w zastosowaniu komercyjnym. Głównym celem jest łatwe wprowadzenie produktów do bazy produktów do rozpoznania. Rozwiązanie które uznaliśmy za słuszne w naszym projekcie jest wybranie 5 reprezentantów każdego z gatunku produktu. To rozwiązanie ułatwia późniejszy proces wprowadzenia nowego produktu do sklepu (a więc tym samym do zbioru danych kasy sklepowej), przy jednoczesnym utrzymaniu bardzo dobrych wyników. Jedynym zadaniem pracownika sklepu będzie zrobienie 5 zdjęć o określonych wytycznych i wprowadzenie ich do brazy.

Średni embedding z 5 produktów jednej klasy dobrze reprezentuje tą klasę. Dając w ten sposób przedstawiciela, który jest "w miarę blisko" środka chmury punktów. Zdjęcia reprezentantów powinny spełniać następujące kryteria:

- zdj*ę*cie pojedynczego owocu
- zdj*ę*cie wielu owoców
- zdj*ę*cie jaśniejsze
- zdj*ę*cie ciemniejsze
- zdj*ę*cie z różnymi odcieniami owoców

5 Podjęte decyzje

5.1 Obsługa danych

Podczas realizacji projektu kilkukrotnie trzeba było podjąć decyzję związane z dalszym rozwojem modelu. Pierwszą była obsługa danych. Ważne było, aby nie wczytywać wszystkich zdjęć do pamięci RAM komputera, aby nie spowalniać procesu uczenia. Odnowionym rozwiązaniem okazało się wykorzystanie generatora, który na bieżąco generuje zdjęcia, a w pamięci przechowywane są tylko ścieżki do plików.

5.2 Metryka

Po zaimplementowaniu pierwszego działającego modelu przyszedł czas na jego udoskonalanie. Dobrym wyjściem było najpierw skonstruowanie odpowiedniej do naszego zastosowania metryki, która umożliwiłaby rzetelną ocenę modeli. Tutaj zdecydowano, że metryka Top5 będzie najlepiej odpowiadała faktycznemu zastosowaniu systemu. Metryka ta bierze pod uwagę 5 najbliższych wyników otrzymanych przez model, co przekłada się na 5 wyświetlonych zdjęć na kasie samoobsługowej.

5.3 Reprezentanci

Kolejnym etapem było wyznaczenie reprezentantów każdej klasy. Po realizacji kilku testów, doszliśmy do wniosku, że 5 zdjęć jest wystarczające do tego, aby model był w stanie dobrze określać daną klasę. Kolejną kwestią było ustalenie z jakich zdjęć powinni składać się reprezentanci. Aby najlepiej odzwierciedlić całą klasę danego produktu reprezentanci powinni składać się z jak najbardziej różnorodnych zdjęć. Tak więc ustalono krótkie wytyczne dotyczące wyboru reprezentantów: pojedynczy produkt w różnych miejscach na tacce, różnej ilości produkty w różnych konfiguracjach na tacce, produkty o różnych odcieniach (np. jabłko jednego gatunku może być czerwone, żółte lub zielone), a także należy uwzględnić zdjęcia robione pod różnym oświetleniem, ponieważ w rzeczywistych warunkach w sklepie możemy spotkać się ze zmiennym oświetleniem. Wytyczne te odnoszą się także do sytuacji, kiedy dodajemy całkowicie nową klasę do modelu. Zdjęcia, na które będzie się składać ta klasa powinny spełniać powyższe wytyczne, aby model był w stanie dobrze rozpoznawać nowe produkty.

Kiedy mamy już wybranych reprezentantów, należy ustalić sposób w jaki porównujemy ich do nowego zdjęcia. Zdecydowaliśmy, że najlepszym sposobem będzie obliczenie średniej ze wszystkich embeddingów danej klasy.

5.4 Problem overfittingu

Głównym datasetem, którym się posługiwano był zbiór "Fruit Recognition", gdzie zdjęcia owoców i warzyw są zrobione na tle srebrnych tac, ponieważ dobrze odzwierciedla to sytuację, w której model ma być zaaplikowany. Szybko jednak doszło do sytuacji, gdzie model osiągał dokładność około 99%, co świadczyło o overfittingu związanym z danym datasetem, ponieważ do nauki i testów wykorzystywano ten sam zbiór. Groziło to sytuacji, w której model jest świetnie przystosowany do danego zbioru zdjęć, jednak jeśli doszłaby jakaś zmienność, na przykład inne tło, to model mógłby sobie nie poradzić z prawidłowym klasyfikowaniem. W związku z tym, zdecydowano się na trenowanie sieci na innym zbiorze, a "Fruit Recognition" posłużył jedynie za zbiór testowy. Do trenowania użyto zbioru "VegFru" zawierającego wiele kategorii owoców i warzyw w najróżniejszych konfiguracjach, co eliminowało overfitting.

5.5 Fine-tuning

Fine Tuning to kolejna część uczenia dobrego modelu. Jest to etap, na którym częściowo naruszamy sieć bazową, zmuszając ją do aktualizacji wag. Jest to etap pozytywnie wpływający na wydajność modelu, często dający bardzo dobre rezultaty. Celem jest nauczenie ostatnich najbardziej abstrakcyjnych warstw sieci konwolucyjnej, aby model dopasował się do naszego problemu. W tym celu zamrożone zostały wszystkie warstwy sieci bazowej oprócz kilku ostatnich. Do uczenia wybrano optymalizator SGD.

Sieć bazowa, którą używamy, była trenowana na "imagenet" - jest to ogromny zbiór zdjęć obejmujący miliony różnych zdjęć, podzielonych na tysiące klas. Zawarte są tam zdjęcia budynków, samochodów, ludzi i wielu innych kategorii. My potrzebujemy rozpoznawać tylko owoce i warzywa, więc pewne rzeczy które umie robić gotowy model są nam nieprzydatne, lub być może nawet trochę przeszkadzają.

Dobrze przeprowadzony fine-tuning sprowadza się do tego, że model nauczy się lepiej rozpoznawać owoce i warzywa, w zamian za stracenie umiejętności rozróżniania innych, niezwiązanych z naszym projektem kategorii.

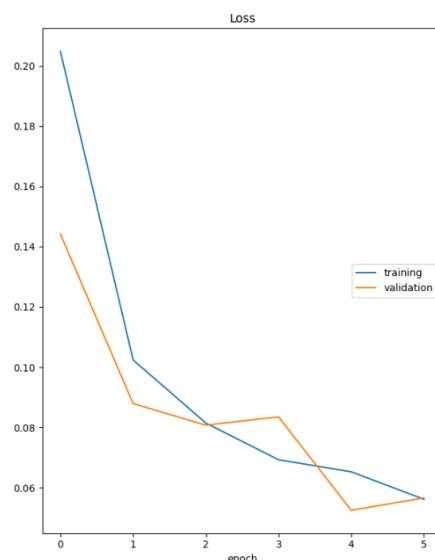
6 Eksperymenty

Pierwsze eksperymenty przeprowadzono na zbiorze fruit-360, aby sprawdzić czy napisany kod jest poprawny i czy model działa prawidłowo. Istotne parametry, które wpływają na naukę modelu to: wartość współczynnika uczenia, wielkość nadbudowanych wartości gęstych - co za tym idzie embeddingu, wielkość paczki danych (batch size), ilość epok, margines, ilość douczonych warstw w sieci bazowej (fine tuning). To te parametry ulegały modyfikacji podczas eksperymentów aby uzyskać najlepsze rezultaty.

Kolejne eksperymenty, już przy użyciu transfer learningu, przeprowadzono na zbiorze Fruit Recognition. Poniżej kilka przykładów uzyskanych rezultatów.

1. eksperyment:

- Model bazowy: VGG16
- Funkcja straty: Triplet Loss
- Epoki: 6
- Batch size: 16
- Margines: 0.5
- Optymalizator: Adam 0.001
- Nadbudowane warstwy gęste: $512 + 16$
- Training triplets: 4800
- Validation triplets: 1200
- Ilość danych testowych: 6705

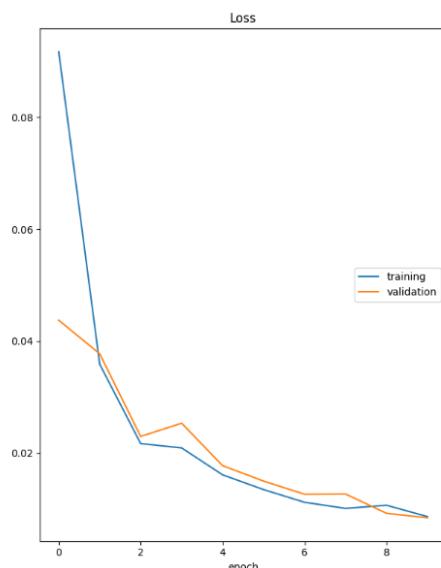


Skuteczność sieci wyniosła **0.1148**.

2. eksperyment:

Wartości parametrów:

- Epoki = 10
- Batch size = 16
- Margines = 0.5
- Optymalizator = Adam 0.001
- Nadbudowane warstwy gęste = $512 + 16$
- Training triplets = 31000
- Validation triplets = 6700
- Ilość danych testowych = 6705

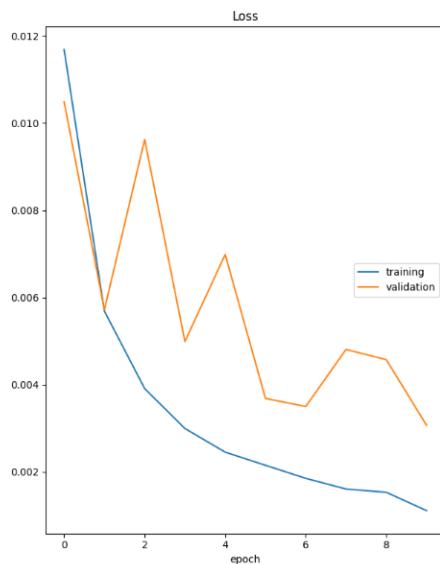


Skuteczność sieci wyniosła **0.8577**.

3. eksperyment:

Sieć z poprzedniego przykładu po przeprowadzeniu fine-tuningu na 4 ostatnich warstwach z optymalizatorem SGD 0.001.

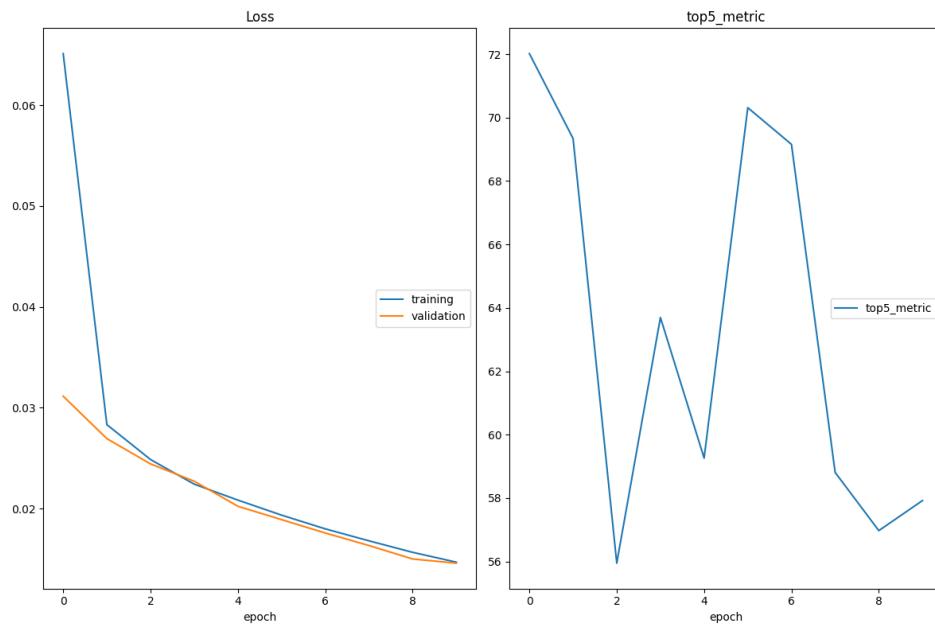
Model trenowany i testowany był na danych ze zbioru Fruit Recognition. Skuteczność sieci wyniosła **0.9441**. Jak widać wartość metryki modelu jest bardzo wysoka, jednak jest to spowodowane overfittingiem modelu do danych ze zbioru Fruit Recognition.



Po uzyskaniu bardzo wysokich wyników na zbiorze Fruit Recognition, trening sieci został wykonywany na większym zbiorze VegFru.

4. eksperyment:

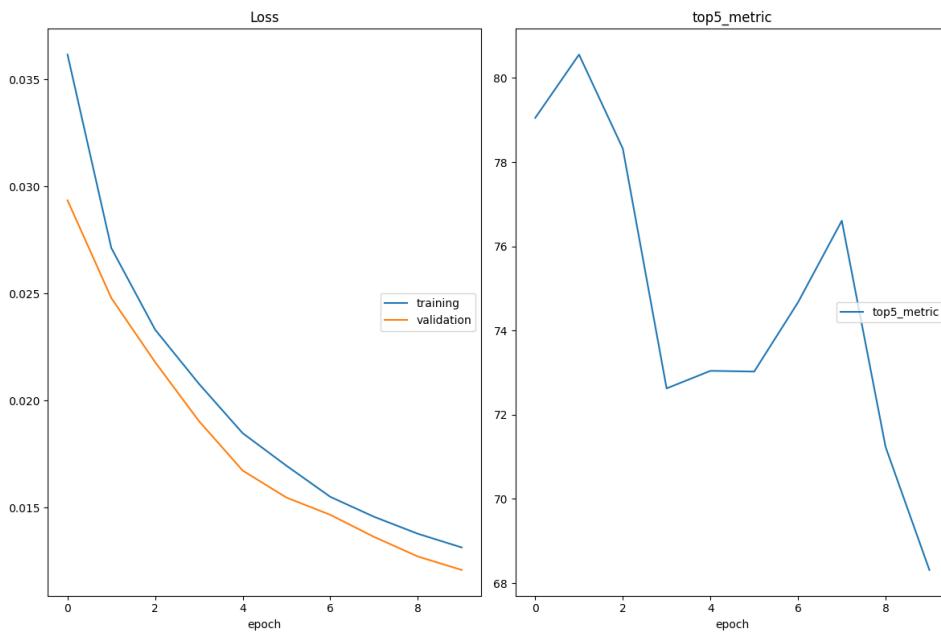
- Model bazowy: Xception
- Funkcja straty: Contrastive Loss
- Epoki: 10
- Batch size: 128
- Margines: 0.5
- Optymalizator: Adam 0.001
- Nadbudowane warstwy geste: 256 + 64
- Training pairs: 72000
- Validation pairs: 9000
- Ilość danych testowych: 9000



Model testowany był na danych ze zbioru Fruit Recognition. Skuteczność sieci wyniosła **0.59**.

5. eksperyment:

- Model bazowy: Xception
- Funkcja straty: Contrastive Loss
- Epoki: 10
- Batch size: 128
- Margines: 0.5
- Optymalizator: Adam 0.001
- Nadbudowane warstwy gęste: 256 + 64
- Training pairs: 50000
- Validation pairs: 6500
- Ilość danych testowych: 9000



Model testowany był na danych ze zbioru Fruit Recognition.

Skuteczność sieci wyniosła **0.696**.

Istotną informacją podczas przeprowadzania eksperymentów była również efektywność modelu dla poszczególnych klas. Jeśli model miałby dobrą skuteczność ale źle sobie radził z jedną, konkretną klasą mogłoby to na przykład świadczyć o źle dobranych reprezentantach tej klasy.

```

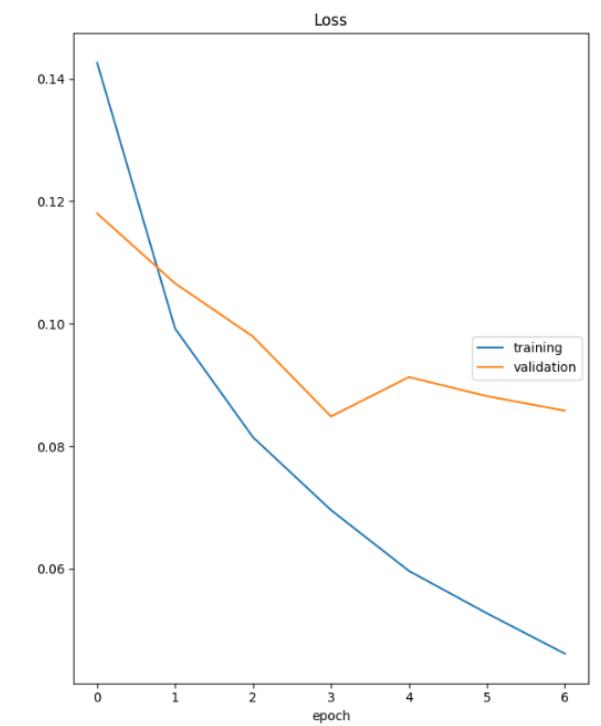
Efficiency for Kiwi B is 38.44999999999999
Efficiency for Orange is 97.179600886918
Efficiency for guava B is 67.54732510288058
Efficiency for Apple B is 99.70909090909092
Efficiency for Pomegranate is 57.63692307692298
Efficiency for Apple C is 79.94666666666667
Efficiency for Plum is 72.29651162790691
Efficiency for Apple E is 59.33333333333333
Efficiency for Carambola is 98.68167202572349
Efficiency for kiwi A is 51.90273556231002
Efficiency for Pear is 75.99113082039915
Efficiency for Persimmon is 86.30322580645148
Efficiency for guava A is 53.66371681415924
Efficiency for Tomatoes is 85.68615384615372
Efficiency for Apple A is 28.73786407766992
Efficiency for Apple D is 72.55844155844157
Efficiency for muskmelon is 34.63022508038583
Efficiency for Peach is 77.98477157360402
Efficiency for Banana is 98.57268722466962
Efficiency for Apple F is 81.36842105263156
Efficiency for Pitaya is 97.83466666666668
Efficiency for Mango is 54.83146067415727
Efficiency for Kiwi C is 36.44444444444446
Efficiency of model in our metric: 69.88222038398206

```

7 Wyniki

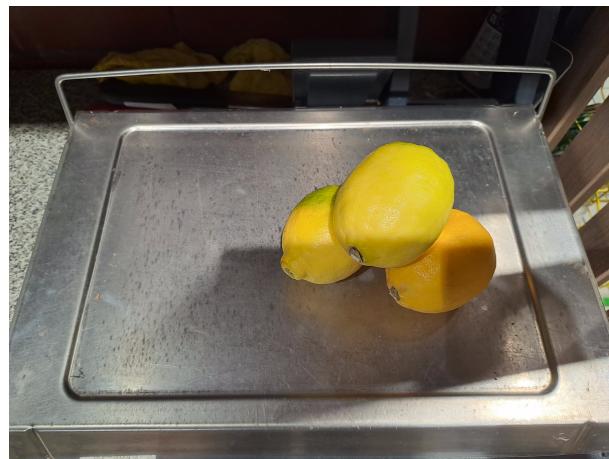
Najlepszy wynik modelu przetrenowanego na zbiorze VegFru i przetestowanego na zbiorze FruitRecognition otrzymano dla poniższych parametrów:

- Model bazowy: Xception
- Funkcja straty: Triplet loss
- Epoki: 10
- Batch size: 128
- Margin: 0.5
- Optimizer: Adam 0.001
- Nadbudowane warstwy: 512 + 256
- Training triplets: 72779 (Veg) + 55638 (Fru)
- Validation triplets: 8999 (Veg) + 6914 (Fru)
- Ilość danych testowych: 6705



Skuteczność sieci wyniosła **0.8677**. Po przetestowaniu modelu na większej ilości danych ze zbioru Fruit Recognition uzyskano skuteczność **0.8702**. Przetrenowany model przetestowano również na nowych klasach. Zrobiono kilka zdjęć owoców, które nie występowały w zbiorze treningowym i testowym, a następnie obliczono ich średnie embeddingi. Zasymulowano w ten sposób docelowe użycie komercyjne.

Testowanym produktem było zdjęcie cytryny.



Najbliższe klasy wyznaczone przez model to kolejno: Pomarańcz, Cytryna, Kiwi A, Guawa B, Guawa A. Zdjęcia, które wyświetliłyby się na ekranie kasy samoobsługowej:



Drugim przetestowanym produktem było zdjęcie arbuzu.



Najbliższe klasy wyznaczone przez model to kolejno: Arbuz, Karambola, Kiwi B, Mango, Kiwi A. Zdjęcia, które wyświetliłyby się na ekranie kasy samoobsługowej:



8 Wnioski

- Jednym z najistotniejszych elementów w machine learningu jest odpowiedni dobór i przygotowanie danych. Należy rozważyć wiele ważnych aspektów takich jak jakość odwzorowania danych do docelowego przypadku użycia, możliwość wystąpienia overfittingu czy tzw. data leakage, który występuje kiedy model jest trenowany i testowany na tych samych danych.
- Zastosowanie sieci syjamskiej z architekturą *Triplet loss* dało lepsze wyniki niż sieć z architekturą *Contrastive loss*.
- Trenowanie sieci konwolucyjnych wymaga obszernej wiedzy i wielu eksperymentów w celu dobrania odpowiednich parametrów modelu.
- Wykorzystanie metody *Transfer learning* pozwala na uzyskanie wysokiej jakości modelu pomimo posiadania stosunkowo małej ilości danych treningowych.
- Ważne jest aby testować model na danych z innego zbioru niż podczas trenowania.