

Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI  
I TECHNIK INFORMACYJNYCH



# Sieci neuronowe

(projekt)

Wykorzystanie sieci VGG19  
do klasyfikacji owoców

Drelich Ewelina, Dziurlikowski Krzysztof,  
Pawlak Iga, Pierczyk Krzysztof

Warszawa, 22 stycznia 2021

# Spis treści

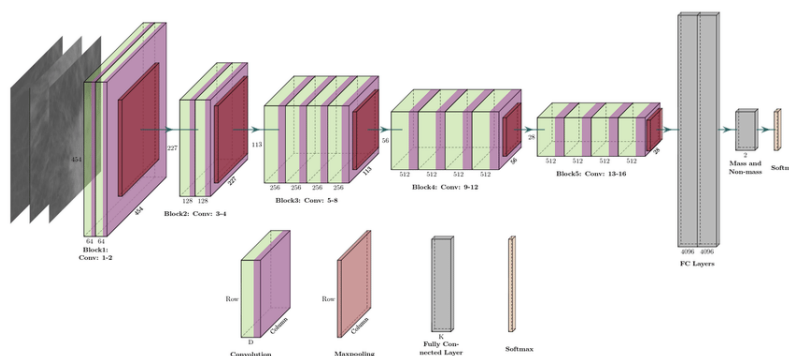
<b>1</b>	<b>Architektura VGG19</b>	<b>3</b>
<b>2</b>	<b>Zbiór danych</b>	<b>4</b>
2.1	Potok . . . . .	4
2.2	Augmentacja . . . . .	5
<b>3</b>	<b>Klasyfikatory</b>	<b>6</b>
3.1	Klasyfikator perceptronowy . . . . .	8
3.2	Maszyna Wektorów Wspierających . . . . .	8
3.3	Porównanie wyników . . . . .	8
<b>4</b>	<b>Sieci głębokie</b>	<b>8</b>
4.1	Uczenie ostatniej warstwy splotowej . . . . .	8
4.2	Uczenie dwóch ostatnich warstw splotowych . . . . .	8
4.3	Uczenie pełnej sieci . . . . .	8
4.4	Uczenie sieci o uproszczonej strukturze . . . . .	8
4.5	Porównanie wyników . . . . .	8
<b>5</b>	<b>Wizualizacja</b>	<b>8</b>
5.1	Class Activation Map . . . . .	8
5.2	Deep Dream . . . . .	8
<b>6</b>	<b>Podsumowanie</b>	<b>8</b>

Sztuczne sieci neuronowe na stałe zadomowiły się w dziedzinie, którą dzisiaj powszechnie określamy mianem sztucznej inteligencji. Algorytmy tworzone przez firmy jak Google potrafią już same uczyć się operowania w tak złożonych grach jak szachy czy Starcraft II znacząco przewyższając wynikami ludzi [1]. Coraz częściej pojawiają się również w bardziej egzotycznych obszarach sterując balonami stratosferycznymi [2] czy przewidując struktury przestrzenne długich łańcuchów aminokwasowych [3].

Jednym z klasycznych zastosowań sieci neuronowych jest klasyfikacja obrazów. Wśród najpowszechniej używanych w tym celu architektur znajduje się od dłuższego czasu zaproponowana w 2014 roku *VGG*. Niniejsza praca skupia się na jednym z wariantów tego modelu - *VGG19* - analizując jego możliwości w kontekście klasyfikacji obrazów owoców ze zbioru *Fruits-360*. Pierwsze trzy rozdziały stanowią opis postawionego problemu, wykorzystanej architektury oraz zbioru danych. Rozdział 4 opisuje przypadki uczenia klasyfikatorów typu perceptronowego oraz SVM bazujących na cechach generowanych przez warstwy splotowe sieci *VGG19* uprzednio wytrenowanej na zbiorze *ImageNet*. Następnie przedstawiony został trening części klasyfikującej (typu perceptronowego) wraz z częścią lub wszystkimi warstwami splotowymi. Przedostatni rozdział zgłębia analizę wytrenowanych sieci wykorzystując techniki wizualizacji obszarów uwagi oraz stopnia aktywacji poszczególnych warstw sieci.

## 1 Architektura VGG19

Akronim *VGG* pochodzi od nazwy grupy badawczej *Visual Geometry Group* z uniwersytetu w Oxfordzie będącej autorem analizowanej sieci. W 2014 roku zaproponowana przez naukowców z *VGG* architektura zdobyła pierwsze i drugie miejsce kolejno w kategorii lokalizacji i klasyfikacji konkursu *ImageNet* [4]. Kluczową cechą ich modelu było zminimalizowanie wielkości filtrów warstw splotowych (a co za tym idzie ilości związanych z nimi parametrów) na rzecz zwiększenia ilości warstw.



**Rysunek 1:** Struktura sieci VGG19. Źródło: [5]

*VGG* występuje w kilku głównych wariantach oznaczanych przez *VGGx*, gdzie *x* określa liczbę warstw sieci, których parametry są modyfikowane w czasie treningu. Rys. 1 prezentuje strukturę *VGG19*. Składa się ona z trójwarstwowej części perceptronowej poprzedzonej kilkoma blokami konwolucyjnymi. W skład każdego bloku wchodzi szereg warstw

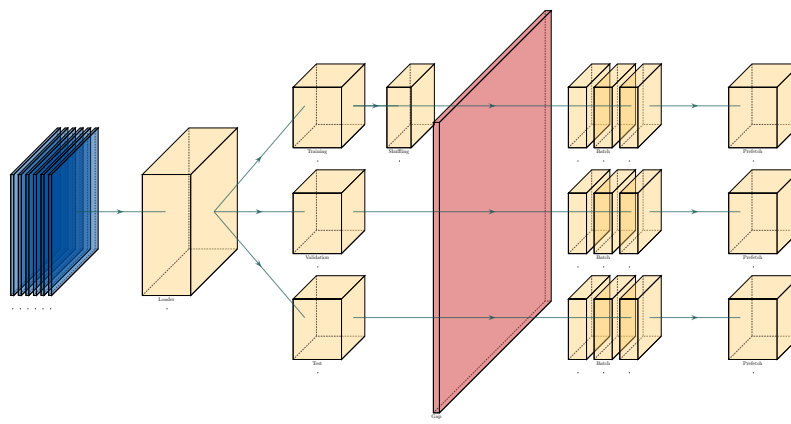
splotowych zakończony warstwą agregującą typu *max pooling*. Rozmiar pola recepcyjnego wynosi odpowiedni  $3 \times 3$  dla warstw splotowych i  $2 \times 2$  dla warstw agregujących. Warto w tym miejscu zauważyć, że *efektywne* pole recepcyjne dwóch połączonych warstw konwolucyjnych  $3 \times 3$  ma wymiar  $5 \times 5$  [6]. Podejście zaproponowane w VGG ma jednak tę zaletę, że wykorzystuje większą ilość warstw nieliniowych, co według twórców przekłada się na ubogacenie funkcji decyzyjnej.

Ilość kanałów w warstwach splotowych zwiększa się dwukrotnie w każdym kolejnym bloku począwszy od 64 w bloku wejściowym. Parametr *stride* został ustalony na 1 w warstwach splotowych oraz na 2 w agregujących. Część perceptronowa składa się z trzech warstw w pełni połączonych o odpowiednio 4096, 4096 i 1000 neuronach. W warstwach ukrytych zastosowano nieliniowość typu *ReLU*. Jedynie warstwa wyjściowa (klasyfikująca) wykorzystuje funkcję typu *softmax*.

## 2 Zbiór danych

W projekcie wykorzystana została baza danych **Fruits-360** [7], która inkorporuje zbiór ponad 90 tys. zdjęć owoców podzielonych na 131 klas. Zdjęcia dostarczone są w formacie JPEG, a ich wielkość została ustandaryzowana do wymiaru  $100 \times 100$  pikseli. Warto nadmienić, że różne warianty tych samych gatunków owoców zostały umieszczone w oddzielnych klasach. Dane zostały podzielone przez autorów na dwa podzbiory: *Training* (67692 zdjęć) i *Test* (22688 zdjęć).

### 2.1 Potok



**Rysunek 2:** Struktura potoku danych wejściowych

Prace nad projektem rozpoczęto od wyboru bibliotek dostarczających zestaw podstawowych narzędzi uczenia maszynowego. Decyzja padła na popularny framework *Tensorflow*, dzięki któremu możliwe było przygotowanie wygodnego, wysoce konfigurowalnego potoku danych wejściowych. Wykorzystanie *tf.data* API pozwoliło na dynamiczne ładowanie danych do pamięci, co znacznie zredukowało jej zużycie w procesie uczenia. Możliwość zrównoleglenia tego procesu względem obliczeń wykonywanych na procesorze graficznym

usunęła występujące początkowo wąskie gardło w postaci procesu przenoszenia danych z pamięci operacyjnej do VRAMu. Całość rozwiązania została zamknięta w postaci klasy, której parametry mogą być ustalane z poziomu plików konfiguracyjnych. Takie podejście wyeliminowało potrzebę ingerowania w kod źródłowy na etapie eksploatacji potoku.

Rys. 2 przedstawia graficzną reprezentację przepływu danych. Na wejściu następuje przydzielenie danych do trzech zbiorów. Podzbiór *Training* jest w całości wykorzystywany jako zbiór treningowy. Z kolei *Test* **dzielony jest w stosunku 1 : 1** na dane walidacyjne i testowe. W kolejnym kroku następuje losowe przemieszanie danych treningowych. Obszar zaznaczony na rysunku na czerwono oznacza lukę. W tym miejscu na zbiory można nałożyć arbitralne modyfikacje, co wykorzystywane jest na etapie augmentacji. Następnie wszystkie zbiory zostają podzielone na porcje (ang. *mini-batches*). Ostatecznie część danych zostaje pobrana do bufora w pamięci operacyjnej (faza *prefetch*).

## 2.2 Augmentacja

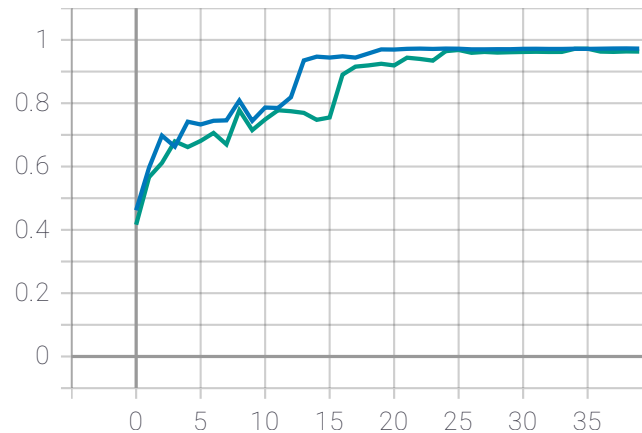
Po stworzeniu potoku przyszedł czas na zaprojektowanie mechanizmów zwielokrotniających dane treningowe (ang. *data augmentation*). W pierwszej kolejności należało wybrać pomiędzy podejściem *online* a *offline*. Pierwsze z nich polega na generowaniu losowych modyfikacji danych wejściowych na bieżąco w trakcie trwania treningu. Z kolei drugie zakłada wygenerowanie zwielokrotnionego zbioru treningowego o określonej wielkości przed przystąpieniem do treningu. Ze względu na wygodę użytkowania i przekazywania tworzonego pakietu oprogramowania pomiędzy członkami grupy zdecydowano się na wariant pierwszy.

Interfejs *tf.data* dostarcza wygodny sposób nakładania arbitralnych modyfikacji na dane przepływające przez potok. Jak opisano wcześniej iniekcja transformacji ma miejsce przed etapem podziału danych na porcje. Aby zapewnić odpowiednią elastyczność decyzji postanowiono zaimplementować szereg mechanizmów augmentacji, z których po przeprowadzeniu pobieżnego badania ich efektywności tylko część miała zostać wykorzystana. Rozwiązanie ponownie zamknięto w postaci klasy parametryzowalnej z poziomu plików konfiguracyjnych. Udostępnia ona następujące transformacje obrazów wejściowych:

- losowe korekcje jasności
- losowe korekcje kontrastu
- losowe odbicia lustrzane względem osi X i Y
- losowe przybliżenia
- losowe rotacje
- losowe przesunięcia wzdłuż osi X i Y
- losowe transformacje typu *shear* wzdłuż osi X i Y

Wszystkie wartości losuje się z rozkładów jednostajnych, których granice można skonfigurować przed treningiem. Puste piksele powstałe na skutek modyfikacji są wypełniane kolorem białym stanowiącym domyślne tło we wszystkich zdjęciach. Ramy czasowe projektu nie pozwoliły na gruntowne przebadanie wpływu transformacji na wyniki uczonej sieci. Posiłkując się drobnymi testami oraz analizą charakteru zbioru danych zdecydowano się na wykorzystanie odbić lustrzanych względem osi X i Y oraz rotacji o kąt z przedziału

$[-20^\circ; 20^\circ]$  (kolejność zgodna z porządkiem aplikowania). Należy zaznaczyć, że transformacje te były niezmiennie na przestrzeni wszystkich przeprowadzonych w czasie projektu treningów.



**Rysunek 3:** Porównanie mechanizmów zwielokrotniania danych na przykładzie uczenia części perceptronowej wraz z dwiema ostatnimi warstwami konwolucyjnymi. Widać, że w przypadku przyjętego zbioru transformacji (na niebiesko) dokładność na zbiorze walidacyjnym rośnie szybciej niż w przypadku dodania losowych korekcji jasności i kontrastu (na zielono).

### 3 Klasyfikatory

Przed przystąpieniem do przeprowadzenia pierwszych treningów pozostały do wykonania dwie czynności. Po pierwsze należało uzupełnić tworzony pakiet oprogramowania o narzędzia umożliwiające wygodne konfigurowanie parametrów uczenia. Oparcie projektu także i w tym aspekcie o pliki konfiguracyjne pozwoliło ostatecznie odseparować sferę wytwarzania modeli od detali implementacyjnych oraz ujednoliciło sposób organizacji pracy w zespole. W zaproponowanym rozwiązaniu wykorzystano dodatkowo mechanizm *Tensorboard*, który umożliwia monitorowanie szerokiego zbioru statystyk w trakcie treningu.

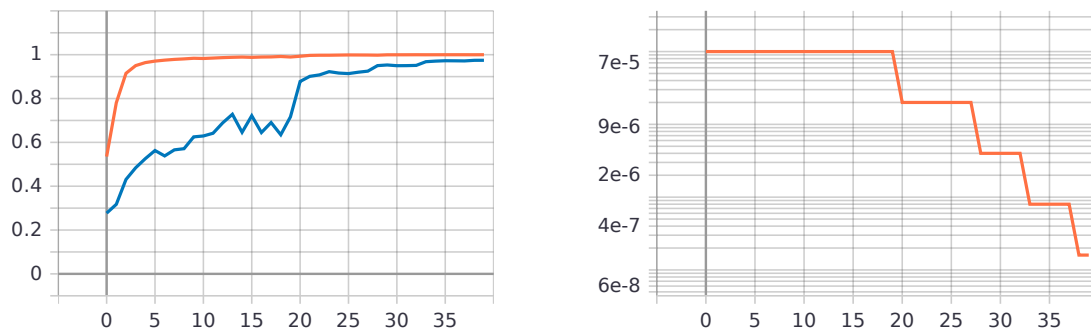
Ostatnim krokiem na drodze do rozpoczęcia uczenia było ustalenie zestawu parametrów. Aby móc porównać skuteczność działania oraz efektywność uczenia w przypadku różnych podejść opisanych w projekcie przyjęto, że parametry te powinny być stałe na przestrzeni wszystkich treningów <sup>1</sup>. W ten sposób przyjęto:

- rozmiar porcji (ang. *batch size*): 64
- inicjalizacja wag i obciążeń: inicjalizacja Glorota z rozkładem normalnym
- algorytm optymalizacji: Adam
- funkcja strat: kategoriowa entropia skrośna
- krok uczenia (ang. *learning rate*): adaptacyjny (początkowo  $10^{-4}$ )

<sup>1</sup>W ramach projektu przeprowadzono kilka pobocznych treningów modyfikujących przyjęte wartości parametrów, w celu zbadania ich wpływu na proces uczenia.

Rozmiar porcji oraz optymalizator przyjęto na podstawie obserwacji doboru tych parametrów w szeregu artykułów analizowanych w ramach przygotowań do projektu. Sugerując się rozważaniami teoretycznymi oraz wynikami zaprezentowanymi w pracy autorstwa X. Glorota oraz Y. Bengio [8] zdecydowano się na wykorzystanie zaproponowanej przez nich metody inicjalizacji z rozkładem normalny. Zapewnia ona zdaniem twórców odpowiednie zależności statystyczne pomiędzy parametrami kolejnych warstw sieci, co prowadzić ma do zmniejszenia ryzyka wystąpienia zjawisk zanikającego oraz eksplodującego gradientu. Przeprowadzona w artykule analiza zakłada wykorzystanie nieliniowości typu sigmoidalnego. Niestety artykuł rozważający analogiczne rozwiązanie dla funkcji aktywacji typu ReLU (wykorzystanej w projekcie) autorstwa Kaiminga He et al. [9] został znaleziony przez zespół dopiero po wykonaniu serii treningów. Ze względu na zadowalające rezultaty zdecydowano nie powtarzać uczenia ze zmienioną metodą inicjalizacji.

Jako funkcję strat przyjęto początkowo kategorię entropię skrośną. W późniejszym etapie zaczęto rozważać również funkcję Hinge'a. Bazując na rozważaniach teoretycznych [10] oraz analizie praktycznej [11] zdecydowano się jednak pozostać przy pierwotnym wyborze. W procesie uczenia wykorzystano adaptacyjną wielkość kroku. Jego wartość początkową ustalono na bazie pobieżnej analizy przypadków. Krok zmniejszany był 0.02—krotnie, gdy wartość straty na zbiorze walidacyjnym nie malała na przestrzeni 4 epok. Tak obrana strategia pozwoliła znacząco zwiększyć efektywność treningów.



**Rysunek 4:** Przebieg dokładności modelu oraz wielkości kroku uczeni w procesie uczenia (przypadek uczenia całej sieci).

3.1 Klasyfikator perceptronowy

3.2 Maszyna Wektorów Wspierających

3.3 Porównanie wyników

## 4 Sieci głębokie

4.1 Uczenie ostatniej warstwy splotowej

4.2 Uczenie dwóch ostatnich warstw splotowych

4.3 Uczenie pełnej sieci

4.4 Uczenie sieci o uproszczonej strukturze

4.5 Porównanie wyników

## 5 Wizualizacja

5.1 Class Activation Map

5.2 Deep Dream

## 6 Podsumowanie



## Bibliografia

- [1] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap i D. Silver, „Mastering Atari, Go, chess and shogi by planning with a learned model,” *Nature*, nr. 588, s. 604–609, 2020.
- [2] M. G. Bellemare, S. Candido, P. S. Castro, J. Gong, M. C. Machado, S. Moitra, S. S. Ponda i Z. Wang, „Autonomous navigation of stratospheric balloons using reinforcement learning,” *Nature*, nr. 588, s. 77–82, 2020.
- [3] A. W. Senior, R. Evans, J. Jumper, J. Kirkpatrick, L. Sifre, T. Green, C. Qin, A. Žídek, A. W. R. Nelson, A. Bridgland, H. Penedones, S. Petersen, K. Simonyan, S. Crossan, P. Kohli, D. T. Jones, D. Silver, K. Kavukcuoglu i D. Hassabis, „Improved protein structure prediction using potentials from deep learning,” *Nature*, nr. 577, s. 706–710, 2020.
- [4] K. Simonyan i A. Zisserman, „Very Deep Convolutional Networks for Large-Scale Image Recognition,” 2015.
- [5] „Automatic Mass Detection in Breast Using Deep Convolutional Neural Network and SVM Classifier,” 2019.
- [6] W. Luo, Y. Li, R. Urtasun i R. Zemel, „Understanding the Effective Receptive Field in Deep Convolutional Neural Networks,” 2017.
- [7] H. Muresan i M. Oltean, „Fruit recognition from images using deep learning,” *Informatica*, nr. 10, s. 26–42, 2018. adr.: <http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm>.
- [8] X. Glorot i Y. Bengio, „Understanding the difficulty of training deep feedforward neural networks,” *Journal of Machine Learning Research*, nr. 9, s. 249–256, 2019.
- [9] K. He, X. Zhang, S. Ren i J. Sun, „Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification,” 2015.
- [10] L. Rosasco, E. D. Vitodevito, A. C. Piana i A. Verr, „Are Loss Functions All the Same?,” 2004.
- [11] K. Janocha i W. M. Czarnecki, „On Loss Functions for Deep Neural Networks in Classification,” 2016.