Detekcja obcych przedmiotów na taśmie produkcyjnej

Aleksandra Veselý
Wydział Elektroniki, Politechnika Wrocławska,
Wybrzeże Stanisława Wyspiańskiego 27, 50-370, Wrocław

Dawid Zalewski
Wydział Elektroniki, Politechnika Wrocławska,
Wybrzeże Stanisława Wyspiańskiego 27, 50-370, Wrocław

Styczeń 19, 2021

Streszczenie: Praca zawiera informacje na temat realizacji projektu z przedmiotu "Diagnostyka Systemów". Temat pracy zrealizowany został w oparciu o głębokie sieci neuronowe. Przedstawiony jest sposób działania sieci i poszczególnych elementów składowych. Omówiona jest struktura sieci, ilość warstw oraz wykorzystane funkcje. W pracy został opisany sposób przygotowania środowiska programistycznego, a także wykorzystane biblioteki. Omówiony został sposób przygotowania bazy danych, podział na materiały uczące i testujące. Przedstawiony jest proces uczenia sieci, jak również przykłady odpowiedzi jakich udziela sieć podczas testowania. Część badawcza zawiera eksperyment ukazujący wpływ ważnych czynników na poprawne działanie sieci.

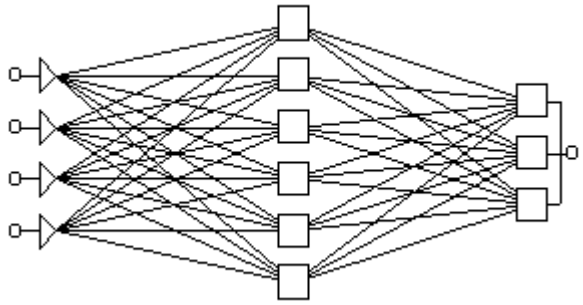
1. WSTĘP

Temat sztucznej inteligencji pojawia się w naszym życiu coraz częściej. Słyszymy o niej w mediach, czytamy coraz więcej artykułów na jej temat, nie tylko w prasie branżowej. Takie terminy jak uczenie maszynowe, uczenie głębokie lub sztuczna inteligencja stają się terminami znanymi szerszej liczbie osób. Na temat sztucznej inteligencji powstało wiele futurystycznych wizji kreowanych przez media. Ważne jest, aby znać faktyczny zakres możliwości, jaki daje nam ta technologia.
Wiedza, którą człowiek zdobył obserwując ludzki mózg pozwoliła na wprowadzenie sztucznych sieci neuronowych. Są one wykorzystywane do zadań rozpoznawania, kojarzenia, przewidywania i klasyfikacji[1]. Dzięki temu znajdują zastosowanie w wielu, bardzo różnych dziedzinach, takich jak: finanse, ekonomia, medycyna, technika, fizyka, geologia. Sieci neuronowe mogą być wykorzystane tam, gdzie analizowany jest problem klasyfikacji, predykcji czy sterowania. Celem pracy jest wykorzystanie głębokiej sieci neuronowej do identyfikacji i klasyfikacji obiektów.

2. SZTUCZNE SIECI NEURONOWE

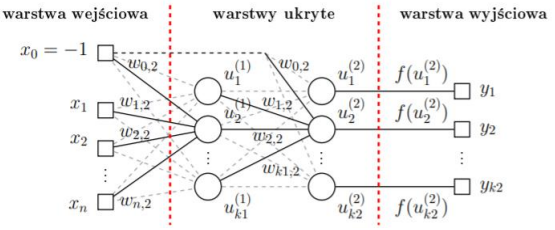
W przypadku uczenia głębokiego wielowarstwowe reprezentacje danych są prawie zawsze tworzone za pomocą modeli określanych jako sieci neuronowe[2]. Sztuczna sieć neuronowa to struktura zbudowana z pojedynczych, ułożonych w warstwy jednostek obliczeniowych nazywanych neuronami. Sygnały wejściowe są przesyłane kolejno przez wszystkie warstwy neuronów w sieci. Sieci składają się z wielu warstw neuronów. Zadaniem neuronów jest wypracowanie rozwiązania na podstawie podanych danych wejściowych.
Niektóre koncepcje uczenia głębokiego wywodzą się z inspiracji działaniem naszego mózgu. Sieci oparte są na bardzo prostym modelu, który przedstawia jedynie podstawową istotę działania biologicznego systemu nerwowego. Na biologii oparte są schematy sztucznych sieci neuronów wchodzących w skład sieci oraz w pewnym stopniu jej struktura. Schematy połączeń neuronów w sieci neuronowej są wybierane

arbitralnie i nie stanowią modelu rzeczywistych połączeń. Jednak badanie i rozwój sieci neuronowych jest jedną z prób odgadnięcia istoty działania ludzkiego mózgu. Sztuczne sieci neuronowe składają się z trzech typów warstw: wejściowej (która zbiera dane i przekazuje je dalej), ukrytej (w której szukane są powiązania między neuronami, zachodzi proces uczenia) i wyjściowej (podającej wnioski i wyniki).



Rysunek 1: Model sieci neuronowej [3]

Wielowarstwowe sieci neuronowe są jedną z najczęściej wykorzystywanych architektur neuronowych. Neurony pogrupowane są w warstwy. Wyróżniane są warstwa wejściowa, kilka warstw ukrytych i warstwa wyjściowa. W tej strukturze wszystkie wejścia poprzedniej warstwy są połączone z wejściami neuronów kolejnej warstwy[4].



Rysunek 2: Model sieci wielowarstwowej[5]

2.1. SIECI KONWOLUCYJNE

Najpopularniejszymi sieciami wśród modeli przetwarzania obrazu są sieci konwolucyjne CNN. Konwolucja w analizie obrazu interpretowana jest jako filtr. Można powiedzieć, że ideą CNN jest wyuczenie parametrów filtru bądź zestawów filtrów, które najlepiej wydobędą informacje z obrazu w ramach zadanego problemu [6]. Sieci konwolucyjne analizują dane wejściowe rozkładając je na cechy. Cechy wyszukiwane są wśród danych w sąsiedztwie. Sieci konwolucyjne mogą uczyć się przestrzennej hierarchii wzorców. Oznacza to, że pierwsza warstwa uczy się małych lokalnych wzorców (np. krawędzie), druga większych struktur składających się z elementów rozpoznanych przez pierwszą warstwę itd.[7] Rozwiązanie to pozwala na coraz głębsze uczenie coraz bardziej złożonych elementów, cech. Głębokie sieci konwolucyjne potrafią stopniowo filtrować różne części danych uczących i wyostrzać ważne cechy w zadaniu rozpoznawania i klasyfikacji obrazów.

2.2. MAXPOOLING

Operacja max-pooling to operacja skalowania, pozwalająca na zmianę rozdzielczości danych wchodzących do kolejnych warstw. Maxpooling powoduje zmniejszenie rozdzielczości. Celem stosowania tej operacji jest stopniowe zmniejszanie rozmiaru reprezentacji danych w celu zmniejszenia ilości parametrów i obliczeń w sieci.

2.3. DROPOUT

Podczas uczenia sieci może dojść do nadmiernego dopasowania sieci. Oznacza to, że modele zaczynają za bardzo dopasowywać się do danych treningowych i tracą zdolność generalizacji. Aby temu zapobiec, można wykorzystać funkcję porzucania Dropout. Technika ta polega na losowym wyborze wyjść warstwy podczas uczenia i wyzerowanie jej. Ułamek określający część wyzerowanych wyjść nazywa się współczynnikiem porzucania i zazwyczaj jest liczbą z przedziału 0.2-0.5 (co odpowiada 20%-50%). Po nauczeniu sieci dropout jest wyłączony. Podczas testowania żadne jednostki nie są porzucane. Wartości wyjściowe warstwy mnożone są przez współczynnik porzucania, aby zrekompensować aktywność większej ilości jednostek podczas testowania niż trenowania [8].

3. STACJA ROBOCZA

Ważnym elementem przystąpienia do realizacji projektu było odpowiednie przygotowanie stacji roboczej. W tym celu należało zainstalować i odpowiednio skonfigurować programy, sterowniki i biblioteki, aby wszystko było ze sobą kompatybilne. Projekt wykonany został na systemie operacyjnym

Windows 10. Całość została napisana w języku Python. Początkowo należało pobrać odpowiednią wersję interpretera Pythona. Odpowiednią wersją okazała się wersja 3.8 (przy wersji nowszej 3.9 pojawiał się problem w instalacji i użyciu biblioteki TensorFlow).

3.1. PYCHARM

Kolejnym ważnym elementem była instalacja środowiska PyCharm. PyCharm to zintegrowane środowisko programistyczne dla języka programowania Python firmy JetBrains [9]. Umożliwia edycję i analizę kodu, debugger, testowanie jednostkowe. Posiada szeroki zakres bibliotek. Wersja użyta przy realizacji projektu to 2020.2.3.

3.2. NVIDIA

Zalecane jest, aby wykonywać kod uczenia głębokiego na nowoczesnym procesorze graficznym NVIDIA. Do realizacji projektu wykorzystano kartę graficzną NVIDIA GTX1080 TI ze sterownikiem graficznym NVIDIA 461.09.

3.3. CUDA

Następnie pobrana została CUDA[10], czyli opracowana przez firmę NVIDIA uniwersalna architektura procesorów wielordzeniowych, umożliwiająca wykorzystanie mocy obliczeniowej karty graficznej do rozwiązywania zadań w wydajniejszy sposób. Wersja, która uznawana jest za stabilną to 10.1.

4. BIBLIOTEKI

W programowaniu sieci neuronowej niezbędne było wykorzystanie bibliotek.

4.1. CUDNN

Aby wszystko poprawnie działało, należało pobrać odpowiednią wersję biblioteki cuDNN, która musiała być kompatybilna z wcześniej pobraną wersją CUDA. Biblioteka cuDNN to biblioteka firmy NVIDIA służąca do obsługi sprzętowej przy programowaniu głębokich sieci neuronowych. Zapewnia wysoką wydajność akceleracji GPU. Wersją użytą przy projekcie była 7.6.4.38.

4.2. KERAS

Keras to interfejs API do głębokiego uczenia się napisany w języku Python, działający na platformie uczenia maszynowego TensorFlow. Jest bardzo popularny i umożliwia łatwe tworzenie różnorodnych architektur sieci neuronowych. Użytą wersją była 2.4.3.

4.3. TENSORFLOW

TensorFlow to biblioteka o otwartym kodzie źródłowym dostępna w serwisie GitHub. Jest to jedna z bardziej znanych bibliotek, jeśli chodzi o obsługę głębokich sieci neuronowych. Głównym powodem popularności TensorFlow jest łatwość tworzenia i wdrażania aplikacji przy jego użyciu. Zapewnia interfejsy API w większości głównych języków i środowisk potrzebnych do projektów uczenia głębokiego. Wersja, z której korzystaliśmy to 2.2.4.

5. BAZA DANYCH

Niezbędnym elementem realizacji sieci neuronowych jest zbiór danych. Ważne jest, aby taka baza posiadała dobrze dobrane zdjęcia, które pozwolą zrealizować poprawną naukę sieci. Początkowo wykorzystywaliśmy zbiór danych pobrany ze strony kaggle.com. Był to zbiór zawierający kilka folderów ze zdjęciami narzędzi. Nas interesowały tylko młotki i śrubokręty. W początkowej implementacji sztucznej sieci neuronowej uzyskiwaliśmy niskie wyniki procesu uczenia i walidacji rzędu 65%. Okazało się, że powodem takiego działania sieci był niejednoznaczny zbiór danych. Zbiór danych z kaggle.com zawierał dane, które:

- zawierały inne narzędzia,
- zawierały wiele narzędzi jednego typu na jednym zdjęciu,
- zawierały zdjęcia innych przedmiotów, zupełnie niezwiązanych z młotkami i śrubokrętami.

Rozwiązaniem problemu było utworzenie nowej bazy danych. Niestety w Internecie nie znaleźliśmy żadnej bazy danych zawierającej zdjęcia interesujących nas przedmiotów. Z tego powodu utworzyliśmy taką bazę samodzielnie, poprzez pobieranie grafik z Google grafika (ze strony polskiej, czeskiej, niemieckiej, japońskiej, ukraińskiej, rosyjskiej i chińskiej), a także innych stron ze zdjęciami. Ważne było, aby grafiki się nie powtarzały. W ten sposób udało nam się utworzyć foldery z danymi:

- 1700 danych dla młotków,
- 1700 danych dla śrubokrętów.

Utworzyliśmy także trzeci folder z różnymi zdjęciami. Utworzenie takiego folderu pozwoliło na klasyfikowanie obcych przedmiotów jako ‘innych’. W sytuacji, gdy na taśmie pojawiał się obcy przedmiot, zostawał on klasyfikowany do trzeciej kategorii – kategorii „obce przedmioty”. Takich przedmiotów również umieściliśmy 1700. Ze względu na małą ilość danych musieliśmy wykorzystać augmentację danych.

5.1. AUGMENTACJA DANYCH

Gdy mamy niewielką ilość danych, to sieć neuronowa raczej nie stworzy dobrych uogólnień, które sprawdzą się na nowych danych[11]. W przypadku, gdy mamy mały zbiór danych, warto jest skorzystać z techniki augmentacji danych. Polega ona na generowaniu nowych danych poprzez wykonywanie odpowiednich losowych przekształceń na posiadanych danych takich jak: obrót, zmiana szerokości i wysokości, przycinanie, przybliżanie, odbicie. Za każdym razem parametry przekształceń są dobierane losowo (losowy kąt obrotu, losowe powiększenie itp.). Dobór parametrów zależy od charakteru bazy danych wejściowych. W naszym przypadku staraliśmy się nie augmentować danych zbyt mocno, ponieważ zdjęcia były docięte praktycznie idealnie na wymiar. Dzięki augmentacji możemy zwiększyć ilość danych z jaką nasza sieć może pracować, bez dostępu do większej bazy danych zdjęć. Za pomocą tej techniki udało nam się powiększyć zbiór danych około pięciokrotnie (do 23100 danych).

5.2. PODZIAŁ NA ZBIORY

Przygotowaną bazę danych należało podzielić na dwa zbiory: zbiór uczący i zbiór testowy. Ze zbioru uczącego dodatkowo wyznaczyliśmy pewien procent (25%) służący za zbiór walidacyjny. Zbiór uczący to zbiór zawierający zestaw danych przeznaczonych do uczenia sieci. Na jego podstawie model uczy się odpowiednio klasyfikować obiekty. Zbiór walidacyjny to zbiór, który używany jest do testu modelu oraz optymalizacji parametrów. Służy do przeprowadzania w trakcie uczenia okresowej walidacji mającej na celu zapobieganie przeuczeniu. Moment początku wzrostu błędu dla zbioru walidacyjnego stanowi sygnał do zakończenia nauki sieci i pobrania najlepszych jej wag z epoki poprzedzającej początek wzrostu tego błędu[12]. Zbiór testowy to zbiór zawierający dane przeznaczone do przeprowadzania po zakończeniu uczenia jednorazowej kontroli (czy sieć poprawnie się wyuczyła).

5.3. WCZYTYWANIE DANYCH

Program został napisany w taki sposób, aby wszelkie zmiany w parametrach wejściowych wymagały jak najmniejszej obsługi w kodzie. W programie podajemy katalog bazowy, w którym znajdują się trzy foldery. Każdy z folderów, który został umieszczony w katalogu bazowym jest wczytywany do programu jako osobna kategoria przedmiotów, których sieć neuronowa będzie starała się nauczyć. Zdjęcia są przeskalowywane do wymiarów 150 x 150 pikseli, zapisywane do wektorów *numpy array*. Każdy z kanałów koloru oryginalnie ma wartości w zakresie 0 – 255. Sieć osiąga lepsze rezultaty, gdy pracuje w zakresach wartości 0 – 1 (skala szarości), dlatego każda z wartości podzielona została przez współczynnik skalujący 255. Analogicznie jest tworzony *numpy array*, który przechowuje informacje o etykietach. Wektor przechowujący tę informację ma długość ilości zdjęć w katalogu. Zawartość wektora to *string’i* mające taką samą nazwę co folder (młotek, śrubokręt, inne).

5.4. ETYKIETOWANIE

Po wczytaniu danych i ich zapisie na wektor *numpy array*, dane są dzielone na zbiór treningowy i walidacyjny za pomocą funkcji *train_test_split*. Ostatnim krokiem jest poprawny zapis etykiet. Aby to zrobić zamieniliśmy łańcuch stringów w wektorze *numpy array* na kodowanie „gorącą jedynką”. Oznacza to, że tworzymy wektor o długości ilości kategorii. Wszystkie miejsca w tym wektorze są wypełnione „0”, oprócz miejsca, w którym chcemy zaznaczyć przynależność do danej kategorii (w tym miejscu wpisujemy „1”). Przykładowo, jeśli chcemy zakodować „gorącą jedynką” kategorię młotek (jedną z trzech), zamienimy *numpy array* z [młotek] na [1,0,0]. Aby w praktyce wykorzystać wyżej wymienione funkcje, należy zastosować funkcję*: preprocessing.LabelBinarizer()*.

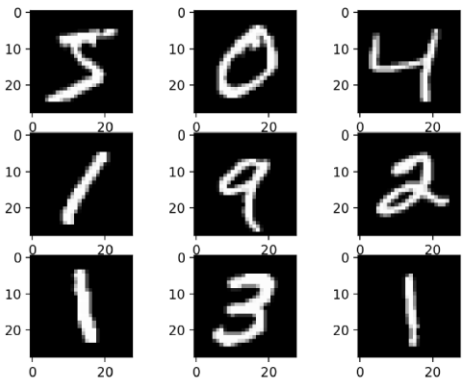
6. OPIS PRACY

Celem pracy była implementacja głębokich sieci neuronowych do klasyfikacji obiektów na taśmie produkcyjnej. Na taśmie produkcyjnej znajdują się dwa przedmioty – młotki i śrubokręty. Uczymy sieć, aby rozpoznawała obiekty z bazy danych zdjęć. Sieć potrafi rozpoznawać obydwa z nich. Może zdarzyć się sytuacja, że na taśmie pojawi się obcy obiekt. Wtedy powinniśmy mieć informację zwrotną o takim zdarzeniu, a taki obiekt nie powinien zostać rozpoznany (powinien zostać przypisany do kategorii „inne przedmioty"). Celem pracy jest klasyfikacja obiektów oraz otrzymanie informacji zwrotnej na temat rozpoznanego przedmiotu.

6.1. PROGRAM MNIST

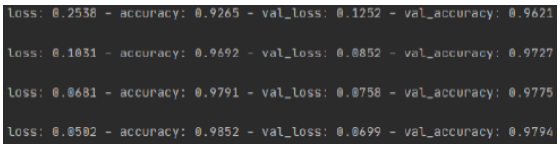
Pierwszym wykonanym przez nas projektem był program rozpoznawania odręcznie napisanych cyfr. Problem dotyczył klasyfikacji obrazów w skali szarości. Dane podzielone były na 10 kategorii (od 0 do 9). Korzystaliśmy ze zbioru MNIST. Zbiór ten zawiera 60000 danych uczących i 10000 danych testowych. Został utworzony przez Narodowy Instytut Standaryzacji i Technologii w latach 80. ubiegłego wieku. Realizacja tego zadania pozwoliła nam poznać funkcje takie jak: poznanie rozmiaru danych – shape, dodawanie kolejnych warstw, kompilacja, przygotowywanie danych – reshape, przygotowywanie etykiet, poznanie atrybutów tensorów, wyświetlanie danych. Dodatkowo przeprowadziliśmy pierwsze testowanie sieci, które wyświetlało nam wartości straty sieci i jej dokładność.

6.1.1. WYNIKI



Rysunek 3: Podgląd zawartości bazy danych MNIST

Uczenie sieci zbudowanej w oparciu o sieci gęste przyniosła bardzo zadowalające wyniki. Na Rysunku 4. widać, że są to wyniki około 97%.



Rysunek 4: Wyniki w kosoli

6.2. PROGRAM PROJEKTU: „TAŚMA PRODUKCYJNA"

Po przeanalizowaniu programu MNIST przeszliśmy do rozpoczęcia pracy nad docelowym projektem „Taśmy produkcyjnej". Aby wykrywać przedmioty na taśmie, musieliśmy zastosować inną architekturę. Wada sieci gęstych, które zostały zastosowane w pierwszym projekcie MNIST, to bardzo wiele połączeń w przypadkach obrazów o większej rozdzielczości (150 x 150) oraz restrykcyjne wycinanie zdjęć. Jeśli nasz obiekt byłby w innej części obrazu lub byłby obrócony, to taka sieć nie rozpozna obiektu. Po przejrzeniu literatury, zdecydowaliśmy się na wykorzystanie sieci konwolucyjnych. Są one odporne na rotację i położenie obiektu. Ponadto ilość połączeń nie musi być uzależniona liniowo z wielkością obrazu wejściowego. W projekcie zaimplementowaliśmy sieć zbudowaną z trzech warstw sieci konwolucyjnych. Początkowo opracowaliśmy model sieci neuronowej uczącej się dwóch kategorii – młotków i śrubokrętów. Po wyuczeniu sieci i sprawdzeniu rezultatów na pojedynczym zdjęciu za pomocą funkcji *model.predict* okazało się, że sieć ocenia, jaki obiekt jest na obrazie praktycznie 0 / 1. W takich przypadkach, gdy obce przedmioty wykrywane byłyby jako 50% młotek i 50% śrubokręt, mielibyśmy informację zwrotną, że klasyfikowany przedmiot to najprawdopodobniej przedmiot obcy. Niestety w naszym przypadku, po podaniu obcego przedmiotu na zdjęciu testowym, sieć klasyfikowała go do jednej z dwóch kategorii, ze 100% pewnością stwierdzając, że jest to młotek lub śrubokręt. Po konsultacji z Panem dr inż. Krzysztofem Halawą uznaliśmy, że najlepszym rozwiązaniem będzie dodanie

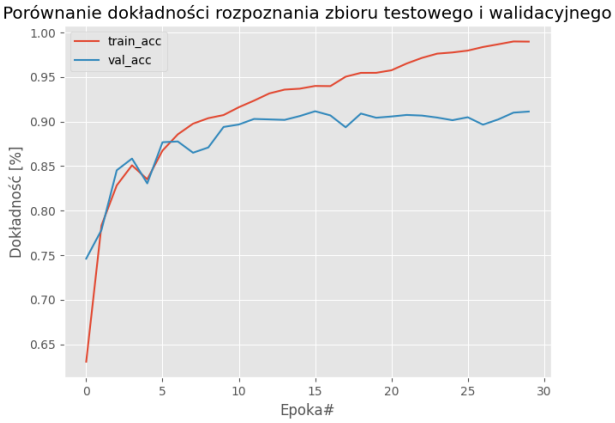
trzeciej kategorii, która będzie zawierała bardzo zróżnicowane zdjęcia. Na początku chcieliśmy skorzystać z bazy danych CIFAR100, która posiada 100 różnych kategorii zdjęć. Niestety zdjęcia są w niskiej rozdzielczości, zaledwie 32x32 pikseli (gdzie zdjęcia młotków i śrubokrętów to 150x150). Takie zdjęcia zawierały mało szczegółów, aby uczyć na nich sieć konwolucyjną. Dodatkowo obawialiśmy się, że dokonanie up-scallingu do 150x150 pikseli spowoduje, że zdjęcia będą na tyle rozpikselowane, że sieć nauczy się wykrywać to rozpikselowanie jako główną cechę trzeciej kategorii. Z tego względu zdecydowaliśmy się na utworzenie własnej bazy danych, dzięki wykorzystaniu serwisu internetowego umożliwiającego generowanie i pobieranie przypadkowych zdjęć (bez powtórzeń) z Google grafika. Po zakończeniu treningu sieci okazało się, że zastosowanie trzeciej kategorii było bardzo dobrym rozwiązaniem do naszego problemu. Sieć wciąż podejmowała decyzje z niemal 100% pewnością, ale przedmioty obce były rozpoznawane i przypisywane do trzeciej kategorii.

7. BADANIA

Ważnym elementem projektu było przeprowadzenie badań.

7.1. UCZENIE SIECI

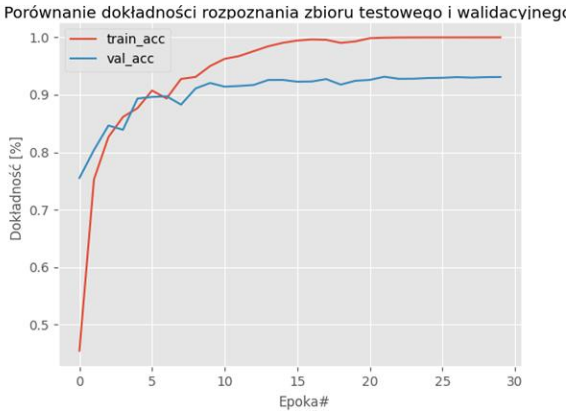
Pierwsze badanie na sieci wykazało poziom rozpoznawania obrazów ze zbioru walidacyjnego na poziomie 91%. Zauważamy, że od około 7 epoki dochodzi do przeuczenia. Architektura sieci składała się z samych sieci konwolucyjnych. Batch size wynosił 500 (była to maksymalna wartość przy wykorzystanej karcie graficznej), a wejście miało rozmiar (150, 150, 3).



Rysunek 5: Porównanie dokładności rozpoznania zbioru testowego i walidacyjnego, pierwsza udana próba.

7.2. DODANIE WARSTW

Drugie badanie dotyczyło zmiany architektury sieci, poprzez dodanie trzech warstw sieci gęstych po warstwach sieci konwolucyjnych. Oprócz wpływu na jakość walidacji chcieliśmy sprawdzić, czy wyniki prawdopodobieństw rozpoznania danych kategorii będą bardziej zróżnicowane. Po badaniach okazało się, że średnia wartość walidacji była odrobinę wyższa niż w poprzednim badaniu. Wyniki prawdopodobieństw rozpoznania kategorii były wciąż w okolicach wartości 0/1.

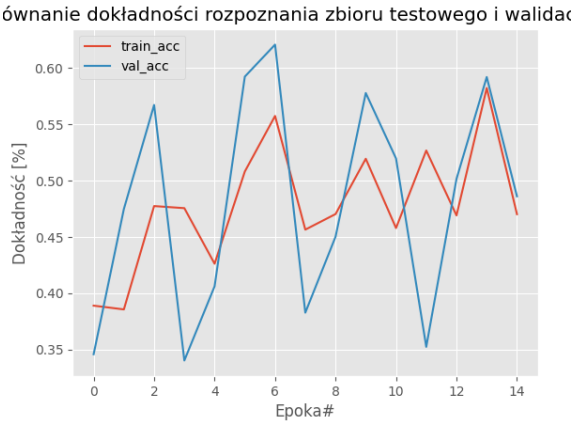


Rysunek 6: Porównanie dokładności rozpoznania zbioru testowego i walidacyjnego, przy zmienionej architekturze.

7.3. ZMIANA ARCHITEKTURY

Trzecie badanie dotyczyło zmiany architektury sieci, poprzez zmianę sieci konwolucyjnych na

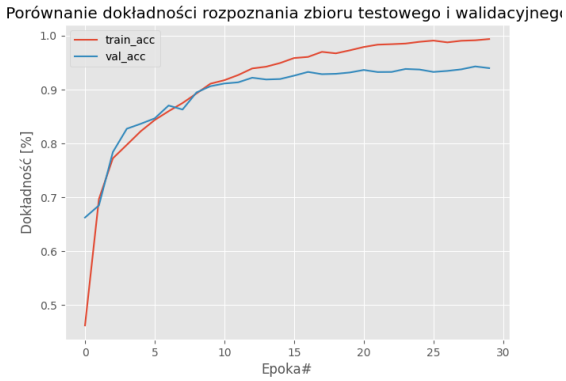
trzy warstwy sieci gęstych. Chcieliśmy sprawdzić wpływ zmiany architektury na jakość walidacji. Po badaniach okazało się, że średnia wartość walidacji była znacznie niższa niż w poprzednich badaniach. Widać, że sieci gęste nie nadają się do naszego zadania – klasyfikacji obiektów.



Rysunek 7: Porównanie dokładności rozpoznania zbioru testowego i walidacyjnego, przy w pełni zmienionej architekturze.

7.4. WYKORZYSTANIE DROPOUT'U

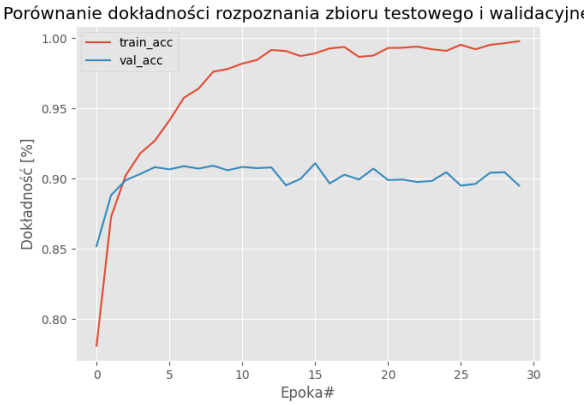
W poprzednich badaniach zauważyliśmy przeuczenie sieci, dlatego przy czwartym teście zastosowaliśmy dropout. Średnie wyniki dokładności walidacji były lepsze (Rysunek 8).



Rysunek 8: Porównanie dokładności rozpoznania zbioru testowego i walidacyjnego, przy dodaniu dropout'u.

7.5. ZMIANA ROZMIARU BATCH

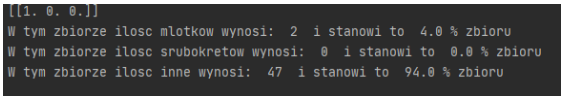
Piątym testem, który przeprowadziliśmy, było sprawdzenie wpływu wartości *batch_size* (ilość danych wsadowych wczytywanych przy każdej iteracji dla jednej epoki) na jakość walidacji. Zwykle im większa wartość batch_size, tym lepsze wyniki. Na rysunku 5. pokazane są wyniki uzyskane przy zastosowaniu maksymalnego rozmiaru *batch_size* (na wykorzystanej karcie graficznej). Chcieliśmy sprawdzić, jak przebiegałoby uczenie, jeśli mielibyśmy do dyspozycji słabszą kartę graficzną. *Batch_size* został ustawiony na wartość 50. Na rysunku 9. widać, że batch_size ma znaczny wpływ na uzyskane wyniki. Przeuczenie zachodzi znacznie szybciej, średnie wyniki jakości walidacji są gorsze, a dodatkowo wzrósł czas obliczeń.



Rysunek 9: Porównanie dokładności rozpoznania zbioru testowego i walidacyjnego, przy zmianie batch_size.

8. TESTOWANIE

Po nauczeniu sieci neuronowej rozpoznawania młotków, śrubokrętów, innych przedmiotów i zapisaniu modelu, przeszliśmy do testowania zaprojektowanej sieci. Napisaliśmy program, którego zadaniem było uruchomienie sieci i sprawdzenie na wybranym przez nas strumieniu nowych danych wejściowych (innych niż te, na których sieć się uczyła), czy sieć poprawnie rozpoznaje przedmioty. Największym problemem dla sieci powinno być wykrywanie obcych przedmiotów. Testowanie wykonaliśmy na strumieniu danych zawierających w pełni obce przedmioty (nie młotki i nie śrubokręty). Po wykonaniu testów okazało się, że sieć rozpoznaje obce przedmioty osiągając dokładność rzędu 94%.



Rysunek 10: Wyniki testowania z informacją w kosoli

Pierwszy wektor na rysunku 10. obrazuje, w jaki sposób zapisane są prawdopodobieństwa. Każda z kolumn oznacza jedną kategorię. Suma wszystkich kolumn (w jednym wierszu, dla jednego zdjęcia) oznacza prawdopodobieństwa, czyli powinna sumować się do 1. Tak napisany program oraz nauczona sieć jest gotowa, aby wykorzystywać ją w prawdziwych zadaniach klasyfikacji obiektów.

9. WNIOSKI

- Dobór i poprawność bazy danych ma kluczowe znaczenie. Niektóre zadania mogą być nawet niemożliwe do wykonania, jeśli nie posiadamy odpowiednio przygotowanych danych, na których nasza sieć mogłaby się poprawnie nauczyć klasyfikacji obiektów.

- Bardzo ważne jest instalowanie odpowiednich wersji poszczególnych elementów oprogramowania, tak aby wszystkie były ze sobą kompatybilne.

- Jeśli nie możemy znaleźć interesującej nas bazy danych, dobrym rozwiązaniem jest wykorzystanie skryptu do pobierania danych z Google Grafika.

- Sieci konwolucyjne znacznie lepiej nadają się do wykrywania obiektów na obrazie niż sieci gęste.

- Drop'out może pomóc w walce z przeuczeniem sieci, lecz nie zawsze gwarantuje on dobre wyniki.

- Posiadanie dobrego hardware (dobrej karty graficznej NVIDIA) do uczenia sieci jest bardzo ważne. Bez odpowiedniej karty graficznej nie można nawet przystąpić do uczenia. Jednym z kluczowych parametrów jest wielkość pamięci, aby móc załadować dużo danych jednorazowo podczas uczenia.

- Jeśli jednorazowo przetwarzamy duże ilości danych wejściowych ważne jest, aby posiadać wolną pamięć RAM. Jeśli stacja robocza posiada małą ilość możliwe jest, że będzie wymagane sztuczne zwiększenie RAM'u, poprzez oddelegowanie części pamięci z dysku systemowego.

- Przy dobrze dobranych parametrach, czas nauki nieskomplikowanego zbioru danych, przy wykorzystaniu odpowiedniego sprzętu może wynosić kilka minut.

- Po przekroczeniu pewnej granicy uczenie może się zatrzymać. To czy wyniki rzędu 95% są zadowalające zależy od konkretnego zadania, do którego wykorzystujemy sieć neuronową (w parku rozrywki dokładność działania pasów rzędu 95% byłaby zdecydowanie za niska, z kolei w sortowni odpadów dokładność rzędu 95% byłaby na pewno wystarczająca).

LITERATURA

[1] W. Dobrosielski, "Zastosowanie sztucznych sieci neuronowych do rozpoznawania obrazów"
[2] F. Chollet, „Deep Learning – Praca z językiem Python i biblioteką Keras", 2018, s.26
[3] Sieci neuronowe, rysunek sieci neuronowej: https://www.statsoft.pl/textbook/stathome_stat.html?https%3A%2F%2Fwww.statsoft.pl%2Ftextbook%2Fstneunet.html
[4] A. Górniak, Sieci neuronowe, instrukcja laboratoryjna z przedmiotu „Programowanie sieciowe", instrukcja do lab.3.
[5] A. Górniak, rysunek wielowarstwowej sieci neuronowej, instrukcja laboratoryjna z przedmiotu „Programowanie sieciowe", instrukcja do lab.3.
[6] L.Podlodowski, „Sieci neuronowe i uczenie głębokie", 2016, opis sieci konwolucyjnej:http://pages.mini.pw.edu.pl/~podlodowskil/resources/uczenie_glebokie.pdf
[7] F. Chollet, „Deep Learning – Praca z językiem Python i biblioteką Keras", 2018, s.137
[8] K. Halawa, „Obliczenia neuronowe", opis funkcji Dropout: http://krzysztof.halawa.

staff.iiar.pwr.wroc.pl/MLP_CNN_Keras_2020_16_04.pdf
[9] Środowisko PyCharm: https://pl.wikipedia.org/wiki/PyCharm
[10] CUDA, opis: https://pl.wikipedia.org/wiki/CUDA
[11] K. Halawa, „Obliczenia neuronowe", opis techniki augmentacji: http://krzysztof.halawa.staff.iiar.pwr.wroc.pl/MLP_CNN_Keras_2020_16_04.pdf
[12] R. Tadeusiewicz, M. Szaleniec, „Leksykon sieci neuronowych", 2015, str.131