UNIWERSYTET EKONOMICZNY W KATOWICACH

KIERUNEK INFORMATYKA

BARTOSZ SKOLIMOWSKI
135321

Detekcja dzikich zwierząt na torze ruchu pojazdów za pomocą algorytmów sztucznej inteligencji

Detection of wild animals on the vehicle path using artificial intelligence algorithms

Praca magisterska
napisana w Katedrze Demografii i Statystyki Ekonomicznej
pod kierunkiem dr hab. Alicji Ganczarek-Gamrot

Oświadczam, że niniejsza praca została przygotowana pod moim kierunkiem
i stwierdzam, że spełnia wymogi stawiane pracom dyplomowym
Pracę akceptuję

…… ……
(data) (podpis promotora)

KATOWICE 2022

BARTOSZ SKOLIMOWSKI

Imię i nazwisko

INFORMATYKA

Kierunek

135321

Nr albumu

Katowice, dnia .....

O Ś W I A D C Z E N I E

Świadom odpowiedzialności prawnej oświadczam, że złożona praca magisterska pt.: „ Detekcja dzikich zwierząt na torze ruchu pojazdów za pomocą algorytmów sztucznej inteligencji” została napisana przeze mnie samodzielnie.

Równocześnie oświadczam, że praca ta nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (tj. Dz. U. z 2018 r., poz. 1191, z późn. zm.) oraz dóbr osobistych chronionych prawem.

Ponadto praca nie zawiera informacji i danych uzyskanych w sposób niedozwolony i nie była wcześniej przedmiotem innych procedur związanych z uzyskaniem dyplomów lub tytułów zawodowych uczelni wyższej.

Wyrażam zgodę na nieodpłatne udostępnienie mojej pracy w celu oceny jej oryginalności przez Jednolity System Antyplagiatowy prowadzony przez Ministra Nauki i Szkolnictwa Wyższego oraz przechowywania jej w Ogólnopolskim Repozytorium Prac Dyplomowych oraz wewnętrznej bazie prac dyplomowych Uniwersytetu Ekonomicznego w Katowicach. Zostałem poinformowany o zasadach dotyczących oceny oryginalności pracy dyplomowej przez Jednolity System Antyplagiatowy.

Oświadczam także, że ostateczna wersja pracy przesłana przeze mnie drogą elektroniczną jest zgodna z plikiem poddanym ocenie w Jednolitym Systemie Antyplagiatowym.

Jednocześnie oświadczam, że jest mi znany przepis art. 233 § 1 Kodeksu karnego określający odpowiedzialność za składanie fałszywych zeznań

.....

(podpis składającego oświadczenie)

Spis Treści

Wstęp

W pracy skupiono się na problemie dużej ilości wypadków samochodowych z udziałem dzikiej zwierzyny. Wiedza, którą przedstawiono w pracy została pozyskana między innymi z raportów organizacji „World Wide Fund for Nature”, w skrócie WWF. Celem badania było opracowanie algorytmu sztucznej inteligencji, który można wykorzystać do detekcji zagrożeń na drodze z udziałem dzikich zwierząt. Aby osiągnąć ten cel zebrano informacje dotyczące skali i specyfiki problemu. Zgromadzono i przedstawiono informacje dotyczące metod i narzędzi przetwarzania i identyfikacji obrazu. Nawiązując do genezy algorytmów sztucznej inteligencji omówiono podstawowe narzędzie głębokiego uczenia za pomocą sieci neuronowych. W pracy zostały wykorzystane wykresy utworzone przy pomocy programów oraz języków programowania takich jak Python oraz PowerBI. Za pomocą tego oprogramowania zostało przedstawiona graficznie skala problemu, na który składa się ilość wypadków oraz śmiertelność dzikich zwierząt. Praca zawiera również algorytm opracowany za pomocą języka programowania Python oraz bibliotek TensorFlow wraz z modułem Keras.

W rozdziale pierwszym skupiono się na szczegółowym omówieniu zdarzeń drogowych z udziałem dzikich zwierząt. Omówiono najczęstsze miejsca występowania takich kolizji, ich czas, najbardziej poszkodowane gatunki zwierząt oraz oddziaływanie wypadków na przyrodę. Podano, jakie są sposoby zapobiegania wypadkom drogowym oraz przedstawia ich skuteczność.

W rozdziale drugim omówiono zbiór danych, na którym przeprowadzono trening modelu. Zbiór danych pochodzi z portalu Kaggle[1]. Kaggle jest portalem udostępniającym dane do analizy. W tym rozdziale został również opisany proces czytania obrazu poprzez maszynę oraz techniki generalizacji zdjęć. Generalizacja zdjęć za pomocą filtrów przeprowadzana jest, aby zwiększyć szybkość algorytmów.

W rozdziale trzecim omówiona została teoria tworzenia algorytmów rozpoznawania obrazu. Wytłumaczona w nim została koncepcja konwolucyjnych sieci neuronowych. W trzeciej części pracy przedstawiona również została teoria bibliotek użytych podczas pisania pracy, mowa tu o bibliotekach języka programowania Python takich jak Keras oraz TensorFlow. Zostały opisane terminy takie jak funkcje aktywacji, metody zapobiegające przetrenowaniu modelu oraz pojęcia optymalizacji algorytmu.

Ostatni rozdział pracy jest rozdziałem badawczym. W tej części pracy został opracowany autorski algorytm klasyfikujący zdjęcia zwierząt, które najczęściej padają ofiarami kolizji drogowych. Algorytm jako wejście przyjmuje zdjęcie, a następnie klasyfikuje je i w wyjściu podaje rasę zwierzęcia oraz procenty, w których jest pewien swojej decyzji. Rozdział czwarty również zawiera koncepcje, w jakich mógłby zostać użyty taki model oraz wytłumaczenie jak mógłby się on przyczynić do ochrony przyrody, gdyby został wprowadzony na drogach.

[1] https://www.kaggle.com/ (dostęp: 07.07.2022)

Rozdział I
Statystyki wypadków z udziałem dzikich zwierząt

1.1 Kolizje drogowe

Zapewne większość osób żyjących w Polsce przeżywała sytuację, kiedy to jechała samochodem, wieczorem lub nocą przez mało uczęszczaną drogę nieopodal lasu i widziała błąkające się nieopodal drogi dzikie zwierzęta. Problem ten nie byłby tak istotny, gdyby na samym błąkaniu się kończyło, niestety wiele zwierząt oraz ludzi traci życie w takich kolizjach. Przy prędkości przekraczającej 50 km/h zwierzę w zderzeniu z samochodem nie ma najmniejszych szans na przeżycie, przy prędkościach powyżej 70 km/h zagrożone jest również życie ludzkie. Według statystyk aż 98% kolizji z udziałem jeleniowatych kończy się śmiercią zwierzęcia, pozostałe 2% ponosi takie obrażenia, które wykluczają je z normalnego funkcjonowania oraz przyczyniają się do wcześniejszej śmierci zwierzęcia w ogromnych męczarniach. Szczególnie bolesnym ciosem dla przyrody jest kolizja z udziałem przedstawicieli gatunków zagrożonych takich jak wilk, niedźwiedź czy występujący niezwykle rzadko ryś. Poza samą śmiercią zwierząt zagrożenie dla wszelakich gatunków niesie ze sobą fragmentacja. Jezdnia najczęściej przecina terytorium migrujących zwierząt. Utrudnia to procesy takie jak zdobywanie pożywienia lub poszukiwanie partnera. Jak podają statystyki Policji z 2009 roku, liczba zabitych osób w takich kolizjach wynosi 7, podczas kiedy rannych zostało prawie 250 osób. Liczby te przerażają, jednak mają się nijak do ilości liczby zdarzeń, która przekracza 17500 przypadków. Wcześniej wspominając, że mało, które zwierzę wychodzi cało z takich starć, można uświadomić sobie jak duża jest skala omawianego problemu. Mowa wciąż tylko o przypadkach zgłoszonych policji. Wiele potrąceń zwierzyny nie jest jednak zgłaszana, ponieważ ptak, zając lub inne małe zwierzę pozostawia jedynie małe wgniecenie na karoserii

samochodu, a kierowca postanawia kontynuować podróż bez większych wyrzutów sumienia. Jak podaje Borowska (2010) według przeprowadzonych ankiet, niespełna, co czwarty kierowca zgłasza kolizję na policję lub do leśniczego, także rzeczywista liczba wypadków może być dużo większa.

Problem pomimo tego, że jest duży często jest bagatelizowany. Nie podejmuje się kroków, aby jemu zapobiec, a dotacje raczej obejmują budowę nowych odcinków dróg aniżeli zwiększanie bezpieczeństwa na istniejących już trasach. W konsekwencji ciężkim zadaniem jest zaradzenie wypadkom, ponieważ skala zjawiska jest znana jedynie z domysłu oraz kalkulacji, a wpływ na populację zwierząt nie jest badany pod tym kątem. Do dziś nie prowadziło się w Polsce badań mających na celu wyłonienie konkretnego zachowania, które miałoby zwiększyć bezpieczeństwo człowieka oraz dzikich zwierząt na drogach. Aby coś się zmieniło należy zacząć pracować od podstaw, ucząc kierowców podczas kursów prawa jazdy, jaki wpływ na przyrodę ma potrącenie dzikiej zwierzyny. Kiedy kierowcy będą wyedukowani, zaczną zgłaszać kolizje, wtedy instytucje odpowiedzialne za ochronę środowiska będą mogły przeprowadzić badania na konkretnych danych oraz pokazać w nich jak bardzo źle wpływa to na gatunki zwierząt zamieszkujących polskie lasy.

Artykuł 25 Ustawy o ochronie zwierząt z dnia 21 sierpnia 1997 r. mówi wyraźnie, że *„Prowadzący pojazd mechaniczny, który potrącił zwierzę, obowiązany jest, w miarę możliwości, do zapewnienia mu stosownej pomocy lub zawiadomienia jednej ze służb (…)”,* w większości przypadków kierowcy zwyczajnie nie wiedzą, jak się zachować. Boją się informować policję, ponieważ są przekonani, że zostaną ukarani mandatem, w konsekwencji zwierzę zostaje pozostawione na pastwę losu. Wymieniony przykład również mówi o tym, dlaczego tak trudno znaleźć jakiekolwiek dane na temat ilości kolizji z udziałem zwierząt na drogach. Największa ilość danych prawdopodobnie posiadałby firmy zajmujące się ubezpieczeniami, gdyż kierowcy w przypadku kolizji zgłaszają się najczęściej do nich. Niestety te nie są skłonne do udostępniania tych liczb. Ministerstwo środowiska, Ministerstwo infrastruktury oraz Główny Urząd Statystyczny również nie posiada takich

danych, jedyne polska policja oraz niezależny projekt „Zwierzęta na Drodze” prowadzą takie statystyki.

1.2 Statystyki wypadków z udziałem zwierząt

Obserwacje, które użyte zostały w pracy pochodzą z ewidencji prowadzonej przez polską policję oraz projekt mgr. K. Kustuscha i dr hab. Inż. Andrzeja Wuczyńskiego, o nazwie: „Zwierzęta na drodze”. Jak autorzy podają na swojej stronie internetowej, celem niniejszego serwisu jest dokumentowanie i analiza śmiertelności dzikich zwierząt w wyniku kolizji drogowych w Polsce. Swym zakresem serwis obejmuje skalę ogólnokrajową, dotyczy różnych rodzajów dróg oraz zapewnia możliwość rejestrowania kolizji z udziałem wszystkich grup dziko żyjących zwierząt w naszym kraju. Ponadto, jest adresowany do wszystkich zainteresowanych osób: przyrodników, naukowców, przedstawicieli administracji i służb drogowych, a także najczęstszych użytkowników dróg – kierowców. Tak duża wszechstronność serwisu powinna umożliwić pełniejsze zrozumienie skali oraz struktury zjawiska śmiertelności zwierząt na polskich drogach, a także upowszechnić wiedzę o tym zaniedbywanym dotąd problemie.

Dogłębnie przyglądając się problemowi należy odpowiedzieć na pytania, gdzie, kiedy oraz jaki gatunek najczęściej ginie na polskich drogach. Jak podaje Borowska (2010) na podstawie ogólnopolskiej ankiety z uczestnikami kolizji z udziałem dzikich zwierząt, najwięcej osób wskazało, że do wypadku dochodziło, kiedy otoczeniem drogi był las.

Otoczenie drogi	
Las	50%
Pole	27%
Łąka	11%
Teren zabudowany	10%
Teren podmokły	2%

Rysunek 1 Otoczenie drogi w przypadku kolizji z udziałem dzikiej zwierzyny

Źródło: http://siskom.waw.pl/siskom/zwierzaki-wyniki-ankiety.pdf

Według ankietowanych, co drugi wypadek miał miejsce w otoczeniu lasu, z kolei tereny zabudowane odpowiadają za zaledwie, co dziesiąty wypadek.

Kolejnym kryterium poddawanym analizie był typ drogi z podziałem na wypadki z dużym oraz średnim i małym zwierzęciem. W przypadku tego pytania głosy ankietowanych osób były bardziej rozłożone.

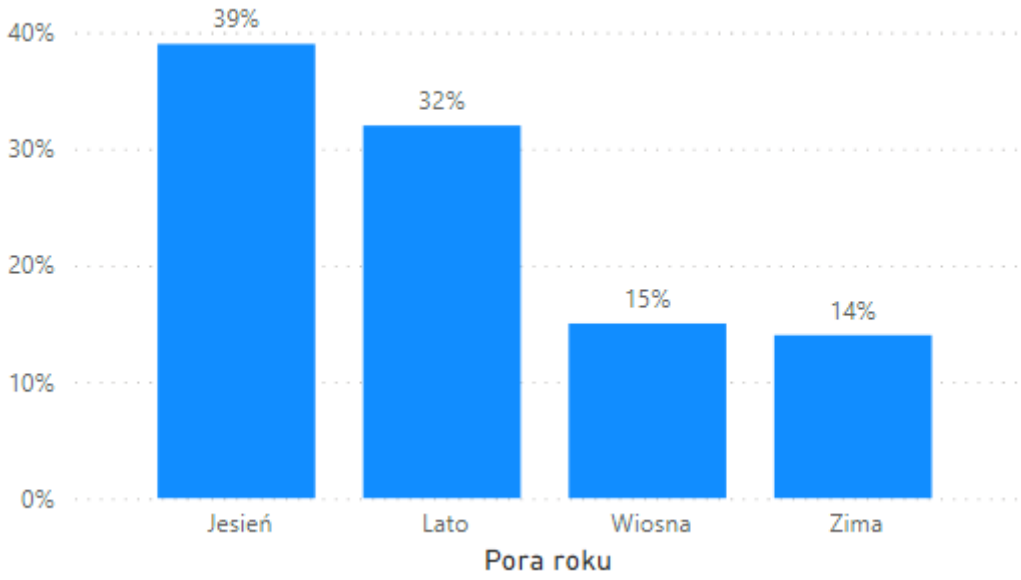
KATEGORIA DROGI	Duży ssak	Średni i mały ssak
Krajowa - autostrada	2%	4%
Krajowa - pozostałe	31%	28%
Wojewódzka	33%	20%
Powiatowa	24%	18%
Gminna	10%	30%

Rysunek 2 Udział dużych oraz małych ssaków w wypadkach w rozdzieleniu na kategorię drogi

Źródło: http://siskom.waw.pl/siskom/zwierzaki-wyniki-ankiety.pdf

Wyjątek dla obydwu typów zwierząt stanowią autostrady. Zaledwie średnio 3% kolizji miało na nich miejsce. Mały procent kolizji na autostradach wyraźnie wskazuje na wyjątkowe bezpieczeństwo tych dróg. W większości przypadków są one w dobry sposób oznakowane, posiadają specjalne siatki oraz płoty, które w prawidłowy sposób zatrzymują dziką zwierzynę uniemożliwiając jej wbiegnięcie pod koła rozpędzonego samochodu. Autostrady również są drogami, na których samochody mogą rozwijać największe prędkości, w związku z tym każdy wypadek bezpośrednio zagraża ludzkiemu życiu. Z tej zależności można również wywnioskować jak dużą uwagę człowiek zwraca na swoje bezpieczeństwo, za razem bagatelizując bezpieczeństwo dzikich zwierząt.

Kolejna analiza skupia się na wyłonieniu pory roku, w której najczęściej dochodzi do wypadków. Ankieta Borowskiej (2010) mówi, że porą roku, która przeważała w liczbie zdarzeń drogowych była jesień.

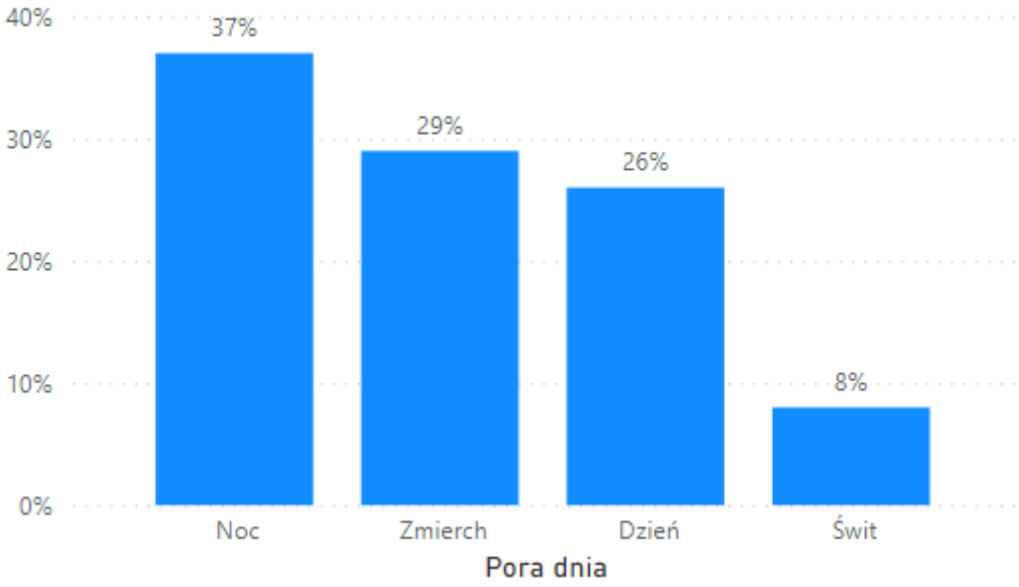


Rysunek 3 Wypadki samochodowe z udziałem dzikich zwierząt z podziałem na porę roku.

Źródło: Opracowanie własne na podstawie danych z http://siskom.waw.pl/siskom/zwierzaki-wyniki-ankiety.pdf

Według ankietowanych najtragiczniejsze pory roku to Jesień oraz Lato. Lato w przypadku większości zwierząt jest okresem godowym. Zwierzęta w tym czasie walczą o samice, szukają miejsca do rozrodu oraz tworzą areały osobnicze, a co za tym idzie – muszą migrować. W okresie godowym szczególnie narażone są samce, które w poszukiwaniu partnerki zwiększają znacznie swoje terytorium. Jesienią doszło do największej ilości wypadków, ponieważ jest to pora roku, kiedy zwierzęta migrują w celu poszukiwania miejsca do zimowania. Zima cechuje się tym, że częste złe warunki atmosferyczne zmuszają kierowców do ściągnięcia nogi z gazu a co za tym idzie zwiększenia bezpieczeństwa na drogach. Zwierzęta podczas zimy migrują najmniej, w przypadku przekraczania jezdni lub stania na poboczu zwierzę jest dobrze widoczne na białym, śnieżnym tle.

Analiza pór roku jasno wskazuje powiązania między zachowaniem zwierząt a ilością wypadków. Analiza pory dnia również tłumaczy, w jaki sposób ludzie powinni dbać o bezpieczeństwo na drodze. Niemalże 70% wypadków ma miejsce o zmierzchu lub w nocy.



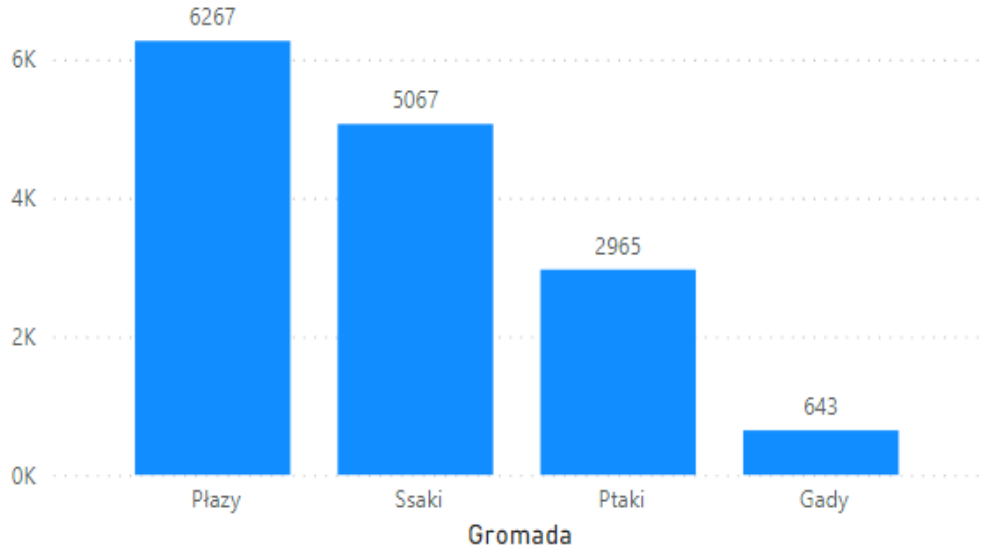
Rysunek 4 Wypadki samochodowe z udziałem dzikich zwierząt z podziałem na porę dnia.

Źródło: Opracowanie własne na podstawie danych z http://siskom.waw.pl/siskom/zwierzaki-wyniki-ankiety.pdf

Zdecydowana większość ssaków oraz ptaków żyjących w naszych lasach różni się trybem życia od człowieka. Człowiek, bowiem w ciągu dnia jest produktywny, a nocą odpoczywa. Zwierzęta leśne za dnia odpoczywają, a w nocy migrują w celu poszukiwania pożywienia. Ten właśnie czynnik powoduje tak wiele wypadków w porze nocnej, zmniejsza się wtedy ruch na ulicach, dzięki czemu zwierzyna chętniej wchodzi na jezdnie. Kolejnym ważnym aspektem są światła samochodów. Mogą one oślepić zwierzę a te w popłochu wbiegnie na jezdnie nie widząc nadjeżdżającego pojazdu. W tej sprawie waży również czynnik ludzki, mianowicie w ciągu nocy nasza czujność jest ograniczona, wzrok i pole widzenia się zmniejsza, a refleks i czas reakcji maleje o połowę.

Zarówno analiza pory roku jak i pory dnia nasuwa pewne wnioski, w jaki sposób można zacząć minimalizować wypadki z udziałem zwierząt na drogach. Przede wszystkim należałoby wprowadzić odpowiednie technologie, które w porach wieczoru oraz nocy odstraszałyby zwierzynę tak, aby ta nie miała możliwości wtargnąć kierowcy pod koła. Kolejny sposób walki o środowisko jest nieco bardziej skomplikowany od swojego poprzednika. Chodzi tu, aby przed rozpoczęciem budowy trasy sprawdzać czy nie koliduje ona z nurtami migracyjnymi zwierząt. Dzięki takiemu rozwiązaniu można by było zbudować odpowiednie przeprawy dla zwierząt w postaci mostów lub tuneli. Taka praktyka byłaby skuteczna, aczkolwiek jej minusy to przede wszystkim czasochłonność, ponieważ badanie nurtów migracyjnych zwierzyny jest operacją, która często trwa latami i wymaga sporego nakładu sił ze strony lokalnych leśniczych oraz biologów.

Praca odpowiedziała na pytania, „kiedy" oraz „gdzie" dochodzi najczęściej do wypadków drogowych z udziałem zwierząt, pora zająć się najważniejszym pytaniem, – kto jest najbardziej poszkodowany w kolizjach. Ewidencja prowadzona przez portal zwierzętanadrodze.pl prowadzona jest od roku dwutysięcznego. W portalu internetowym użytkownicy mogą w bardzo prosty sposób udokumentować kolizję ze zwierzęciem. Poniższy wykres przedstawia dane z kolizji z podziałem na gromadę zwierząt.



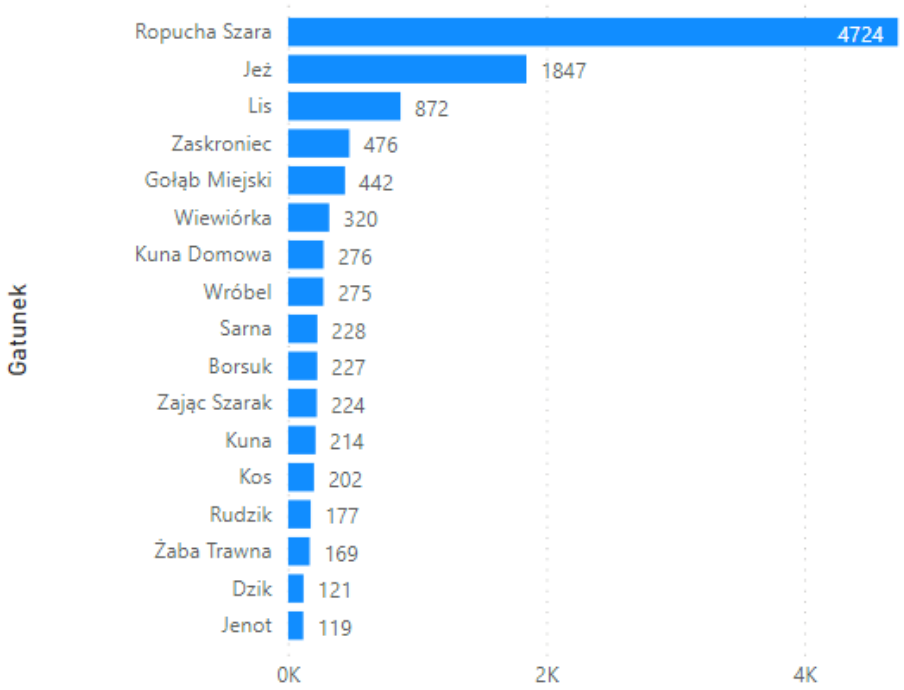
Rysunek 5 Wypadki samochodowe z udziałem dzikich zwierząt z podziałem na gromadę.

Źródło: Opracowanie własne na podstawie danych z https://zwierzetanadrodze.pl/

Na rysunku 5 można zauważyć ogromną ilość zwierząt z gromady płazów oraz ssaków. Te dwie najczęściej ginące gromady dają ponad jedenaście tysięcy wypadków samochodowych. Liczba ta jest zatrważająca, lecz warto przyjrzeć się również gromadom ptaków i gadów, które moim zdaniem są o wiele zaniżone, ponieważ często kierowca nie jest w stanie zauważyć, że miał kolizję z ptakiem poprzez jego drobną budowę. Gady z kolei są jeszcze mniejsze, co za tym idzie trudno jest zidentyfikować kolizję z gadem. Jak podaje Anna Krzysztofiak (2005) w Polsce występuje tylko dziewięć gatunków, z czego pięć z nich możemy spotkać na terenie Wigierskiego Parku Narodowego. Są to trzy jaszczurki: zwinka, żyworodna i padalec oraz dwa węże: zaskroniec zwyczajny i żmija zygzakowata. Oprócz nich na ziemiach polskich można spotkać również jaszczurkę murową, jaszczurkę zieloną, zaskrońca zwyczajnego i gniewosza plamistego. Charakterystyczne są również dwa gatunki obce, żółw błotny oraz żółw ozdobny, które pomimo występowania w Polsce nie są zaliczane do fauny, ponieważ znalazły się tu w sposób sztuczny. Ze słów Anny Krzysztofiak można wywnioskować jak rzadko

spotykane są gady w porównaniu do innych gromad zwierząt. Zatem te sześćset czterdzieści dwie kolizje z tymi rzadko występującymi, często objętymi ochroną zwierzętami to również ogromna tragedia.

Analiza gromady pozwoliła spojrzeć na rozkład poszkodowanych w nieco bardziej ogólny sposób, kiedy opracowano wizualizację z gatunkiem jako parametr podziału można wywnioskować wiele ciekawych informacji. Pierwsze miejsce zajmuje Ropucha Szara, która ma ponad cztery i pół tysiąca zgłoszeń o kolizjach z jej udziałem.

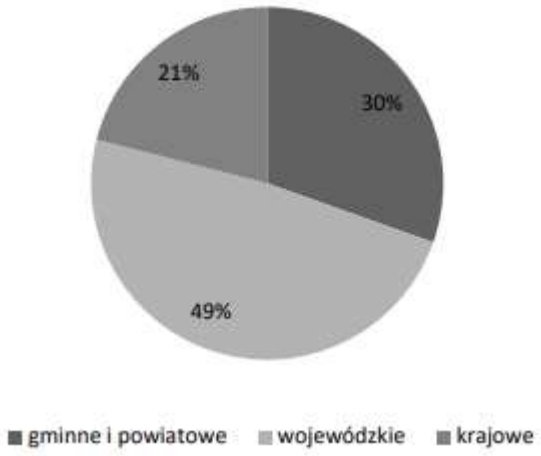


Rysunek 6 Wypadki samochodowe z udziałem dzikich zwierząt z podziałem na gatunek.

Źródło: Opracowanie własne na podstawie danych z https://zwierzetanadrodze.pl/

Warto wspomnieć, że na powyższym rysunku znajdują się aż trzy gatunki, które są objęte ścisłą ochroną gatunkową w Polsce. Jest to Zaskroniec, Kos oraz Rudzik. Z powyższych gatunków pierwsze miejsca zajmują zwierzęta, które w przypadku kolizji nie zagrażają życiu kierowcy. Dopiero Sarna, znajdująca się na dziewiątym miejscu jest zwierzęciem, które przy odpowiedniej prędkości może wyrządzić człowiekowi krzywdę samym ciężarem swojego ciała. Jako ciekawostkę niewidoczną na wykresie można dodać to, że w wypadkach brało udział dwadzieścia siedem łosi, sześć szopów praczy, pięć wilków, trzy żubry, jeden niedźwiedź brunatny, jeden szakal złocisty oraz jeden jastrząb. Zwierzęta te nie tylko są bardzo rzadkie oraz objęte ścisłą ochroną, ale również służą za wizytówkę Polski, więc ochrona ich powinna stać na bardzo wysokim poziomie.

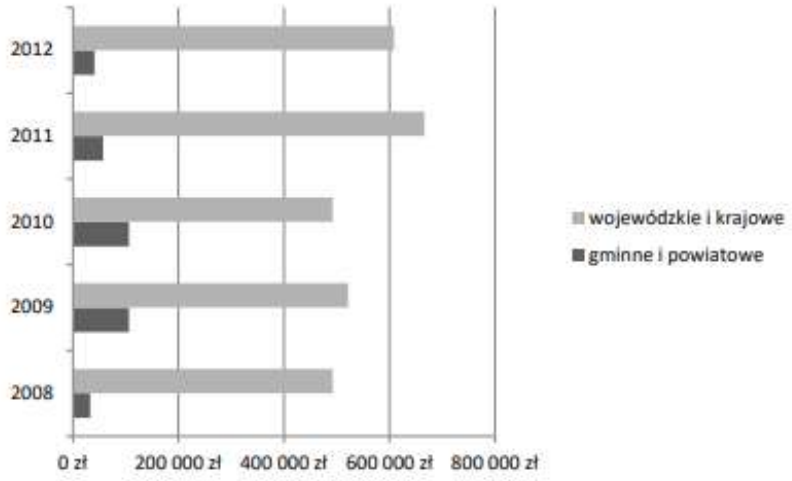
Jeżeli chodzi o analizy wypadków samochodowych z udziałem zwierząt warto również przyjrzeć się zbiorowej pracy studentów Uniwersytetu Przyrodniczego w Poznaniu o tytule: „Rozwój infrastruktury drogowej a kolizje z udziałem zwierzyny leśnej". W artykule M. Iwińskiego, A. Zydronia, M. Antkowiak oraz P. Szczepańskiego (2010) podjęto próbę analizy wpływu rozwoju inwestycji drogowych na liczbę wypadków z udziałem dziko żyjących zwierząt na przykładzie dróg przebiegających przez gminy: „Kórnik i Mosina". Na podstawie badań określono, że głównym czynnikiem wpływającym na liczbę zdarzeń ze zwierzyną leśną jest natężenie ruchu kołowego, a także fragmentaryzacja krajobrazu. Rozwijająca się infrastruktura drogowa o wysokich klasach szlaków komunikacyjnych powinna stanowić naturalną barierę dla zwierząt, a zdarzenia na tego typu szlakach mieć charakter incydentalny. Analiza zebranego materiału wykazała, że rozwój infrastruktury drogowej nie wyeliminował kolizji z udziałem zwierząt. Analiza ta różni się od poprzedniej, ponieważ dane skupiają się na zaledwie dwóch gminach, kiedy dane z poprzedniego portalu skupiały się na całej Polsce. Autorzy zaznaczyli, że dane pozyskali za pomocą Komendy Głównej Policji. Materiał udostępniony przez policję obejmował 138 zdarzeń w latach 2008-2012 na terenie Powiatu Poznańskiego, a dokładniej dwóch gmin: Kórnika oraz Mosiny. Poniższy wykres przedstawia rozkład wypadków na poszczególne typy dróg.



Rysunek 7 Wypadki samochodowe z udziałem zwierząt z rozróżnieniem na typ drogi.

Źródło: Rozwój infrastruktury drogowej a kolizje z udziałem zwierzyny leśnej. Iwiński, Zydroń, Antkowiak, Szczepański 2017 r.

Widać, że większość wypadków miało miejsce na drogach wojewódzkich. Wynik jest nieco wyższy, jednak bardzo podobny do analizy zawartej na rysunku drugim. Wniosek, zatem jest taki, że zarówno na terenach całej Polski jak i na wybranych, małych próbkach najczęściej do wypadków dochodzi na drogach wojewódzkich. Zapewne dzieje się tak, ponieważ drogi te są najczęściej uczęszczane a ich infrastruktura, łącząc ze sobą miasta w województwie krzyżuje się z drogami, którymi migrują zwierzęta. Końcowym etapem zbiorowej pracy studentów z Uniwersytetu Przyrodniczego w Poznaniu było przeprowadzenie analizy ekonomicznej skutków wypadków z udziałem zwierzyny leśnej. Jak podają autorzy Iwiński, Zydroń, Antkowiak i Szczepański (2017) ze względu na ograniczony charakter metadanych dla materiału empirycznego przyjęty został podział dla dróg: niższych klas – powiatowych i gminnych, gdzie kolizje skutkowały jedynie uszkodzeniami mienia, oraz wyższych klas – wojewódzkich i krajowych, gdzie koszty ekonomiczne zdarzeń dotyczyły również uszczerbku na zdrowiu uczestnika wypadku. Poniższy wykres przedstawia koszty zdarzeń z udziałem zwierzyny dla danych typów dróg.



Rysunek 8 Koszty zdarzeń z udziałem zwierzyny dla danych typów dróg.

Źródło: Rozwój infrastruktury drogowej a kolizje z udziałem zwierzyny leśnej. Iwiński, Zydroń, Antkowiak, Szczepański 2017 r.

W rekordowym dla dróg wojewódzkich i krajowych roku 2011 koszty zdarzeń dla tak małej próbki wynosiły blisko 700 tysięcy złotych. Z kolei najwyższe koszty dla dróg gminnych i powiatowych przedstawione zostały w latach 2009 oraz 2010, w przypadku obydwu lat oscylowały one w okolicach 100 tysięcy złotych. Kwoty te pokazują w bardzo dobry sposób, ile kosztują nas takie wypadki. Za każdym razem, kiedy potrącane jest zwierzę, kosztem nie jest tylko wgnieciona karoseria samochodu, jest nim również częsta potrzeba hospitalizacji poszkodowanego, koszty związane z interwencją weterynarza lub leśniczego oraz w przypadku większej zwierzyny koszty usuwania zwłok zwierzęcia. Cały proces odpowiedniego reagowania w przypadku kolizji w tak małym obrębie terytorialnym potrafi wygenerować wiele kosztów, niewyobrażalną różnicą byłoby wydanie tych pieniędzy na odpowiednią prewencję oraz zapobieganie wypadkom w najbardziej niebezpiecznych miejscach.

Po wstępnej analizie danych pochodzących z różnych źródeł można wywnioskować, na jakich drogach oraz w jakich okolicach powinno się zwracać największą uwagę na kolizje

ze zwierzętami. Wciąż jednak wielkim mankamentem jest brak posiadania ujednoliconego systemu, który umożliwiłby lepsze dbanie o przyrodę. W Polsce dane są gromadzone lokalnie, przez co często są w różnych formatach, są niepełne i rozproszone. Podmioty dysponujące danymi takie jak: policja, nadleśnictwo, urzędy, parki narodowe, nie kontaktują się ze sobą, przez co prawdopodobnie każda próba złączenia danych o kolizjach zakończyłaby się fiaskiem, ponieważ nie byłoby pewności czy dana kolizja nie została zarejestrowana w dwóch niezależnych od siebie zbiorach danych. Gromadzeniem oraz ujednoliceniem danych powinno się zajmować GDOŚ i Ministerstwo Infrastruktury. Dane o wypadkach powinny być łatwo dostępne tak, aby ułatwić analizy wykonywane w ich oparciu. Uzupełnieniem dla danych mogą być informacje przez podmioty związane z ochroną przyrody, np. ośrodkami rehabilitacyjnymi dla zwierząt, gdzie trafiają zwierzęta poszkodowane w trakcie kolizji, również Parki Narodowe oraz Nadleśnictwa mogą uzupełniać dane na podstawie znalezionych przy poboczu lub w lesie martwych zwierzętach, po których widać, że ucierpiały w wypadku. Obecnie bardzo popularne jest dbanie i zajmowanie się naturalnym środowiskiem, więc istnieje duże prawdopodobieństwo, że system ten zostanie zjednolicony oraz będzie mógł służyć zmniejszeniu ryzyka wypadków. Po wnikliwej analizie wraz z graficznym przedstawieniem poszczególnych aspektów kolizji z udziałem dzikich zwierząt, warto przejść do kolejnego punktu, który będzie zawierał idee zmniejszenia występowania tych przykrych zdarzeń.

1.3 Sposoby na zmniejszenie liczby kolizji drogowych

Jeszcze do niedawna metody na zapobieganie liczby kolizji drogowych dzieliły się na te działające na ludzi oraz na te działające na zwierzęta. Metody działające na człowieka to między innymi:

- Znaki drogowe,

- Sygnały świetlne,

- Elektroniczne tablice ostrzegawcze.

Z kolei metody odstraszające zwierzęta to:

- Odblaski,

- Gwizdki na zderzakach,

- Bariery chemiczne.

W tych dwóch metodach oczywiście pomijamy sposób zapobiegania, którym jest tworzenie przejść dla zwierząt i odgradzanie pobocza jak ma to miejsce na większości autostrad. Co jednak, gdyby użyć dostępnej dla ludzi sztucznej inteligencji i zamontować system kamer wyposażonych w algorytm, który będzie w stanie automatycznie klasyfikować rasę zwierzęcia? System ten po sklasyfikowaniu zwierzęcia będzie wysyłał do osoby sygnał za pomocą świateł ustawionych przy jezdni. Można również zamiast sygnału utworzyć oddzielną aplikację na telefon bądź moduł do popularnych aplikacji takich jak Apple Maps czy Google Maps, który będzie nas zawiadamiał, że blisko miejsca, przez które jedziemy znajduje się zwierzę przy drodze. System ten ma również jedną poważną zaletę,

w przypadku zwykłych czujników ruchu nie jesteśmy w stanie sklasyfikować gatunku zwierzęcia znajdującego się przy drodze. Sztuczna inteligencja może z łatwością zapisywać te dane oraz tworzyć bazę, w której znajdować się będą wszelakie informacje na temat najczęstszych prób przekroczenia dróg przez dany gatunek. Dane te w przyszłości mogą zostać wykorzystane do sporządzenia mapy dróg migracyjnych gatunków oraz na podstawie ich budowania specjalnych przejść dla zwierząt przy drogach.

Znak drogowy wyposażony w system z sygnałem świetlnym jest znany ludzkości od wielu lat. Pomimo swojej ogromnej skuteczności nie jest on wykorzystywany w wielu Europejskich krajach. Jak podaje amerykański portal zajmujący się sprawami patentowymi „Justia Patents" istnieje oficjalny system pod nazwą „Wildlife warning system" został zgłoszony do amerykańskiego biura patentowego w 2013 roku. System ten składa się z dwóch urządzeń. Jeden to „Źródło ostrzegania", które zawiadamia kierującego o niebezpieczeństwie, a drugie to „Generator Sygnału", który ma za zadanie zbierać za pomocą czujnika na podczerwień sygnały. Urządzenia są ze sobą połączone poprzez bezprzewodową sieć. Często do obydwu urządzeń stosuje się również panel fotowoltaiczny w celu zaoszczędzenia na energii potrzebnej do zasilenia aparatury. System ten był testowany w Niemczech. Liczba kolizji w miejscu, gdzie go zastosowano spadła prawie do zera.

Rysunek 9 System ostrzegania przy drodze numer 191 w parku Yellowstone.

Źródło: Fotografia Marcel Huijser.

Ważną wadą tego systemu jest to, że jest on drogi. Zastosowanie go na drodze wiąże się z kosztem poprowadzenia tamtędy instalacji elektrycznej, zamontowania lamp ostrzegających w małych odstępach oraz kamer bądź czujników, które będą w stanie wychwycić ruch zwierzęcia. Następna wada to częste niedociągnięcie systemu, jeżeli użyjemy czujników to system może uruchamiać się, kiedy wykryje spadający liść, człowieka spacerującego poboczem lub powalone drzewo, co wprawi kierowcę w błąd. Dlaczego system używający kamer ma przewagę nad systemem ze zwykłymi czujnikami? Pierwszym argumentem jest możliwość szerszego zastosowania, kamery mogą ze stu procentową pewnością poprawnie klasyfikować zwierzę i nie ostrzegać kierowcy na marne w przypadku spacerującego człowieka, spadającego liścia bądź zerwanej gałęzi. Jeżeli zastosujemy

algorytm wówczas możemy również go wykorzystać do prowadzenia ewidencji oraz zbierania danych na temat prób pokonania jezdni przez zwierzynę.

Rysunek 10 Zwierzęta przekraczające jezdnie.

Źródło: Piotr Krzyzanowski, fotografia dla Polskapresse.

W przy obecnym rozwoju technologii, człowiek zrozumiał jak ważnym oraz drogim czynnikiem są dane. Dane zebrane przez taki system mogłyby się przyczynić do przeprowadzenia badań, wyciągnięcia cennych wniosków, a następnie wprowadzenia metod prewencyjnych w celu polepszenia sytuacji. System przypominający wcześniej wspomniany „Wildfire warning system" tylko zamiast czujnika wykorzystujący kamery oraz system do zbierania danych mógłby odmienić w dużym stopniu bezpieczeństwo na drogach. Dalsza część pracy zawiera szczegółowy algorytm, który mógłby zostać użyty w takim systemie. Na podstawie dokładnych danych w postaci zdjęć zwierzęcia system potrafi poprawnie sklasyfikować rasę. System mógłby w pełni zapewnić brakującą wiedzę

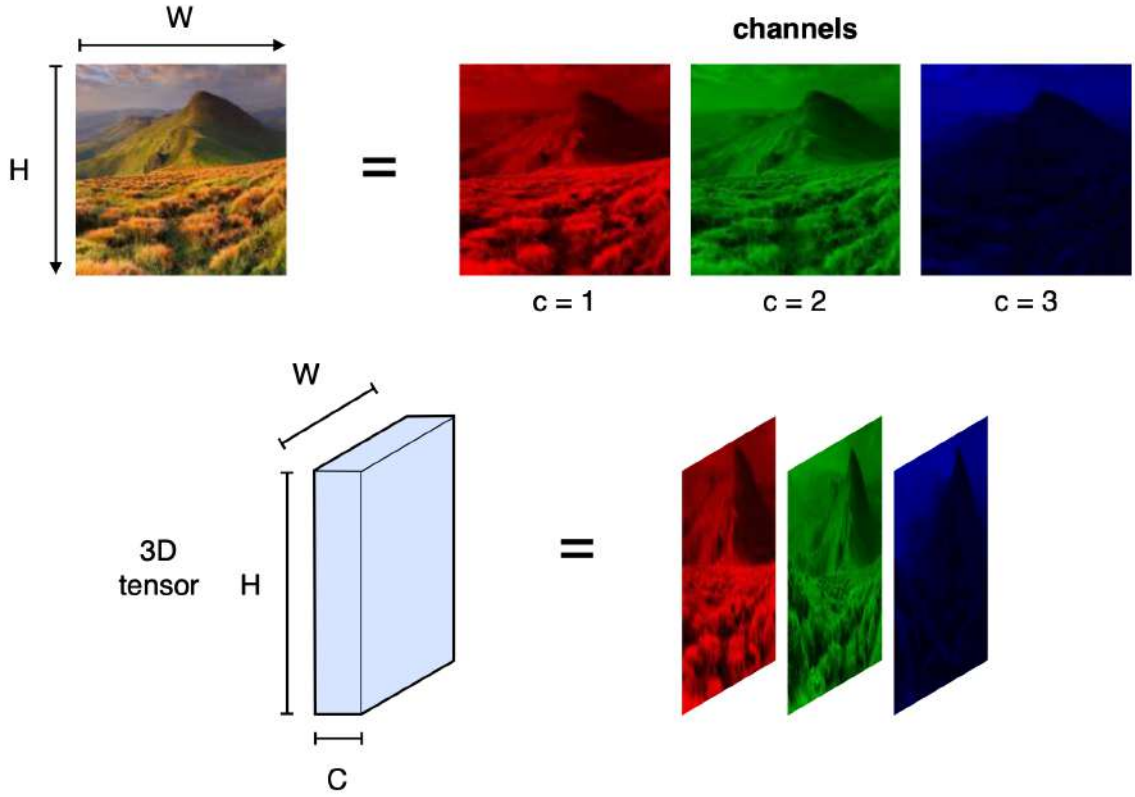
na temat dróg migracyjnych zwierząt, a co za tym idzie zapewnić odpowiednie podłoże do przeprowadzenia badań przez leśników oraz odpowiedzialne organy. Obecna tendencja na rynku sprzyjająca inwestowaniu w technologie uznawane jako bezpieczne dla środowiska oraz ratujące zwierzęta również może być postrzegana jako ogromna szansa dla rozwoju tego typu oprogramowania. Wiele ogromnych korporacji wspiera w każdy możliwy sposób projekty mające na celu opiekę nad planetą Ziemią. Infrastruktura drogowa w większości przypadków budowana oraz utrzymywana jest z pieniędzy skarbu państwa, w tym przypadku władze mogą przychylnie patrzeć na tego typu projekty, ponieważ istnieje nadciągający z zachodu trend na ekologie pod każdą postacią. Bezpieczeństwo drogowe również powoli się zmienia, lata temu jezdnie nie były wyposażone chociażby w bariery drogowe, które mają na celu minimalizacje niebezpieczeństwa w razie wypadku samochodowego. Ekrany dźwiękoszczelne blisko domów również były kiedyś czymś bardzo rzadkim, obecnie po wielkich protestach ze strony mieszkańców okolic ruchliwych dróg są one budowane praktycznie wszędzie. Obydwa wymienione ulepszenia infrastruktury drogowej kiedyś były niepotrzebnym wynalazkiem, a obecnie są standardem każdej inwestycji. Buduje się również więcej pasów zieleni – trend ten szczególnie widziany jest w miastach takich jak Warszawa, Gdańsk czy Kraków. Podsumowując świadomość człowieka się zmienia, a co za tym idzie zmienia się też jego podejście do ochrony życia oraz zwiększania jego komfortu, prawdopodobną prognozą w tym przypadku jest przeniesienie tego na zwierzęta i zaczęcie dbać o ich zdrowie i życie.

Rozdział II

Opis oraz specyfikacja zbioru danych

2.1 Sposób czytania obrazu poprzez maszynę

Baza danych, na której został przeprowadzony projekt jest bazą danych niestrukturalnych, dane te cechują się tym, że ich struktura nie jest do końca ustalona i nie może być w prosty sposób rozpoznana poprzez język programowania Python, w którym wykonany jest projekt. W skład danych niestrukturalnych wchodzą zdjęcia, zapisy głosowe, zapisy wideo, przy pracy z takimi bazami danych należy użyć specjalnych, dedykowanych pod to bibliotek języka programowania Python. Biblioteka, która umożliwia pracę z przetwarzaniem obrazu, z języka angielskiego „Computer Vision" nazywa się Keras. Zapisy strukturalne, które są przeciwieństwem danych niestrukturalnych cechują się tym, że są uporządkowane, najczęściej są to dane znajdujące się w tabeli, czyli w postaci, która ułatwia ich odczytanie oraz analizę poprzez komputer. Aby zagłębić się w to w jaki sposób działają sieci neuronowe rozpoznające obraz należy poznać system odczytu obrazu poprzez komputer. Fotografia wyświetlana poprzez algorytm jest stworzona z milionów małych pikseli, te zdefiniowane są poprzez ich dokładne koordynaty oraz trzy pozostałe parametry, które określają kolor piksela. Każda z tych liczb reprezentuje jeden kolor, system ten nazywa się RGB od angielskich pierwszych liter kolorów red, green oraz blue. Natężenie koloru zawsze jest podane w kolejności od czerwonego do niebieskiego. Każdy z kolorów ma zakres od 0 do 255. Dla przykładu (255,0,0) będzie określał kolor czerwony, a (0,255,0) będzie określał kolor zielony. Komputer odczytuje obraz jako macierz posiadającą ogromne ilości uporządkowanych po parametrach: „Wide", oznaczające w języku polskim szerokość od oraz „Height" oznaczające w języku polskim wysokość wraz z ich natężeniem kolorów



Rysunek 11 Nakładanie się na siebie warstw RGB.

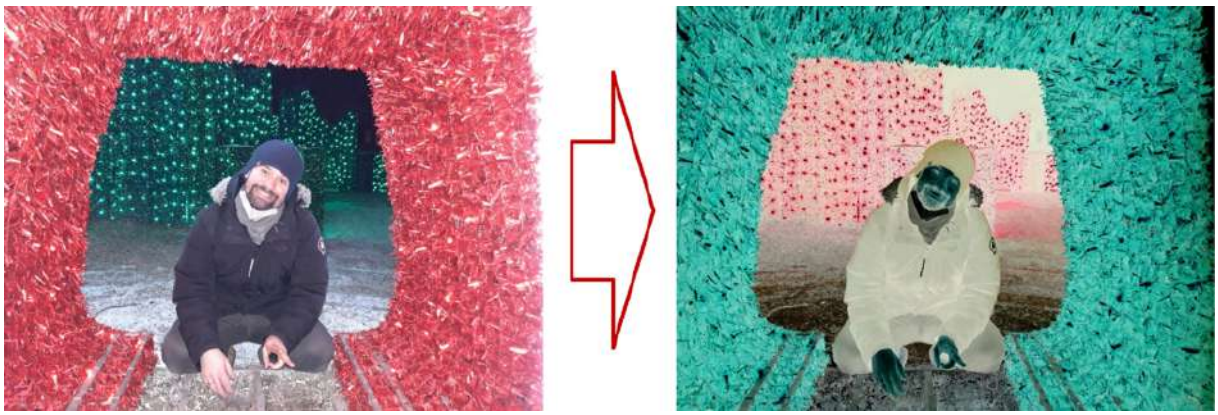
Źródło: Andrea Vedaldi – Convolutional Networks for Computer Vision Applications

Na powyższym rysunku zostało przedstawione nakładanie się na siebie trzech warstw kolorów oraz to w jaki sposób za pomocą wektorów W oraz H uporządkowane są piksele. Ilość pikseli na zdjęciu będzie określała jak bardzo po przybliżeniu zdjęcie będzie ostre, dla przykładu ilość pikseli, która jest optymalna dla fotografii wielkości kartki A4 to 3508 x 2480. Fotografia jest jednak siatką o takich wymiarach, więc pomnożone przez siebie piksele dają wynik prawie 9 milionów pozycji, z których każdy piksel ma określony poprzez system RGB kolor. Dla człowieka taki rozmiar wydaje się być bardzo wielki, jednakże dla komputera pod względem samego odczytu nie sprawia on większych problemów. Nowoczesne procesory oraz karty graficzne są zoptymalizowane tak aby przekształcać wielowymiarową macierz obrazu na wektor, który łatwiej jest dla nich

obliczyć. Obliczenia zaczynają się wydłużać, kiedy trzeba dokładnie, jak ma to miejsce przy Konwolucyjnych Sieciach Neuronowych, przeanalizować każdy piksel oraz krok po kroku, wraz z każdą ukrytą warstwą wyłaniać z niego coraz to inne szczegóły takie jak kontur, kształt, a w późniejszych warstwach cechy charakterystyczne takie jak na przykład róg, trąba, skrzydło czy puszysty ogon zwierzęcia. Działanie ukrytych warstw zostało opisane dokładnie w kolejnym rozdziale pracy. Wracając jednak do systemu odczytu obrazów poprzez komputer, aby nie spowalniać obliczeń wiele technik pozwala na operacje utraty informacji. Lepszy obraz krawędzi, będzie widoczny już, kiedy fotografia zostanie przekształcona na odwrócenie kolorów, operacje tą przeprowadza się odejmując natężenie barw RGB każdego z pikseli od 255, które jest końcem skali. Wzór tej operacji wygląda zatem następująco:

$$nowy\_pixel = 255 - stary\_pixel$$

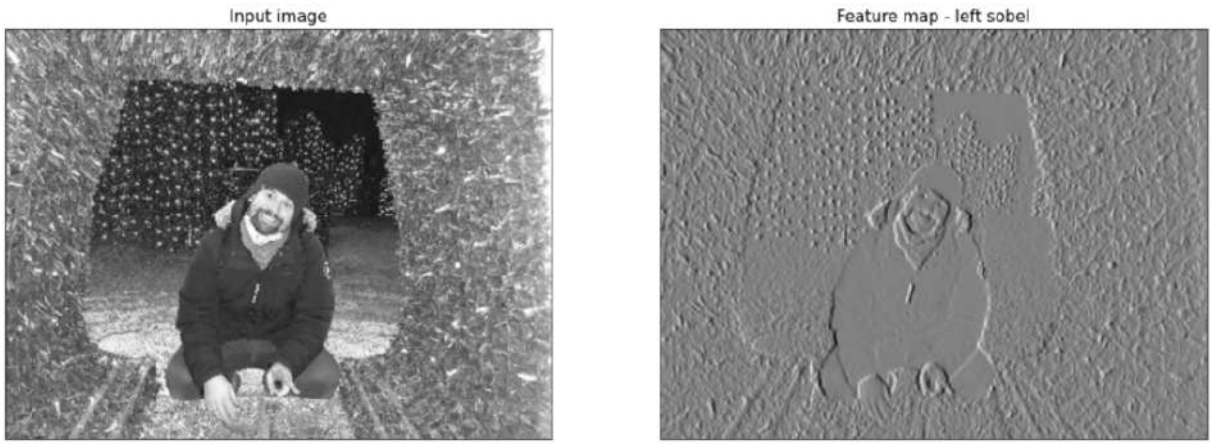
Należy oczywiście pamiętać, aby odejmowanie przeprowadzić oddzielnie dla koloru czerwonego, zielonego oraz niebieskiego.



Rysunek 12 Porównanie zdjęcia w oryginale oraz z nałożonym filtrem odwrócenia kolorów.

Źródło: https://miroslawmamczur.pl/jak-dzialaja-konwolucyjne-sieci-neuronowe-cnn/

Jak podaje Mamczur (2021) w sieciach konwolucyjnych macierz, którą nakłada się na macierz pikseli nazywa się filtrem bądź kernelem. Macierz, która zostanie nałożona na wejściowe zdjęcie opisane w kodzie parametrem „small_image" przeobrazi je w widoczny na poniższym obrazku sposób.



Rysunek 13 Porównanie zdjęcia czarno-białego oraz zdjęcia z nałożonym filtrem left sobel.

Źródło: https://miroslawmamczur.pl/jak-dzialaja-konwolucyjne-sieci-neuronowe-cnn/

Dzięki temu zabiegowi obraz stracił swe kolory, jednak widoczne są na nim kontury i pierwotne linie oddzielające od siebie różne kolory.

Kod 2.1

Źródło: https://miroslawmamczur.pl/jak-dzialaja-konwolucyjne-sieci-neuronowe-cnn/

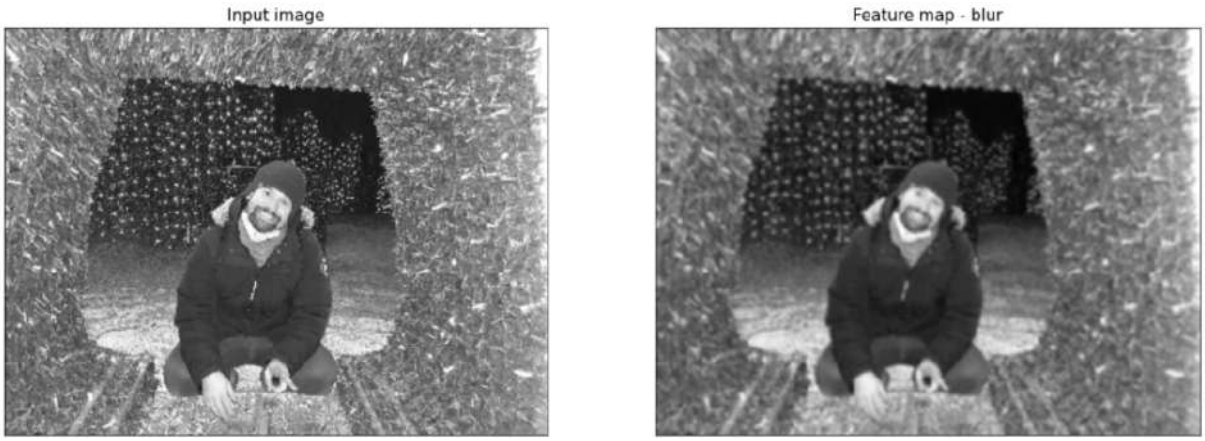
```
from scipy.signal import convolve2d

def apply_kernel_to_image(img, kernel, title=''):

    feature = convolve2d(img, kernel, boundary='symm', mode='same')
```

```
    # Plot

    fig = plt.figure(figsize=(20, 10))

    ax1 = fig.add_subplot(1, 2, 1)

    ax1.imshow(img, 'gray')

    ax1.set_title('Input image', fontsize=15)

    ax1.set_xticks([])

    ax1.set_yticks([])

    ax2 = fig.add_subplot(1, 2, 2)

    ax2.imshow(feature, 'gray')

    ax2.set_title(f'Feature map - {title}', fontsize=15)

    ax2.set_xticks([])

    ax2.set_yticks([])

    plt.show()

#left sobel

kernel = np.array([
```

```
    [1, 0, -1],

    [2, 0, -2],

    [1, 0, -1]])

apply_kernel_to_image(small_image, kernel, 'left sobel')
```

Powyższy efekt można uzyskać poprzez zastosowanie takiego kodu. Funkcja zaprezentowana w kodzie ma za zadanie wyświetlić obok siebie dwie fotografie, wejściową oraz po jej prawej stronie wyjściową. Pod parametrem „kernel" do funkcji wprowadzana jest siatka filtra, który ma być nałożony na funkcję wyjściową.

Kolejnym efektem, który może zostać nałożony na zdjęcie jest powszechnie znany w fotografice „blur". Tworzy on efekt nieostrego, zamazanego obrazu widzianego przez mgłę, jest to powszechna technika pozwalająca na utratę informacji w sposób kontrolowany.



Rysunek 14 Porównanie zdjęcia czarno-białego oraz zdjęcia z nałożonym filtrem blur.

Źródło: https://miroslawmamczur.pl/jak-dzialaja-konwolucyjne-sieci-neuronowe-cnn/

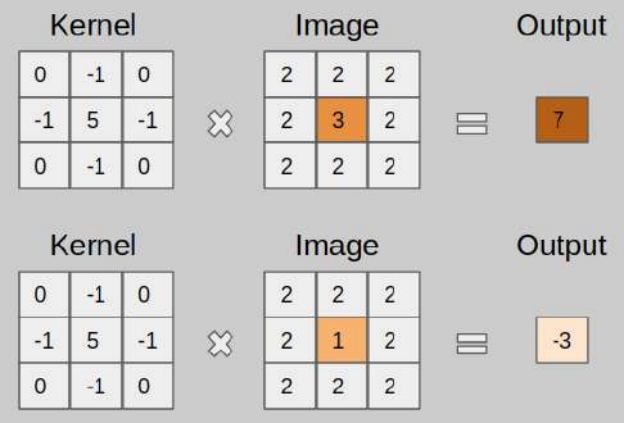
Kod 2.2

Źródło: https://miroslawmamczur.pl/jak-dzialaja-konwolucyjne-sieci-neuronowe-cnn/

```
kernel = np.array([

    [0.01, 0.01, 0.01],

    [0.01, 0.01, 0.01],

    [0.01, 0.01, 0.01]])

apply_kernel_to_image(small_image, kernel, 'blur')
```

Efekt ten uzyskuje się mnożąc macierz obrazu przez 0.01. Na podstawie powyższych rysunkach przedstawione zostało jak w fotografice stwarza się pewne efekty, te same sposoby oraz obliczenia wykorzystywane są w programach do edycji oraz retuszu zdjęć i filmów.

Algorytm utraty informacji, który został użyty w powyższych kodach jest niczym innym niż nakładaniem krok po kroku na macierz siatki wag, mnożenie ich, aby na koniec zsumować je i otrzymać z nich mniejszą siatkę. Utrata informacji następuje za pomocą zmniejszenia liczby pikseli, operacja została opisana na poniższym przykładzie:



Rysunek 15 Przykład sumowania macierzy z natężeniem pikseli oraz macierzy filtra.

Źródło: https://towardsdatascience.com/types-of-convolution-kernels-simplified-f040cb307c37

Przykłady te są uproszczone do skali, która bez problemu pozwoli na ich opisanie. W pierwszej kolumnie przedstawiona została siatka z Kernelem, druga kolumna przedstawia piksele na bardzo uproszczonym obrazie. Można zatem przystąpić do obliczeń:

$$3*5 + 2*-1 + 2*-1 + 2*-1 + 2*-1 = 7$$

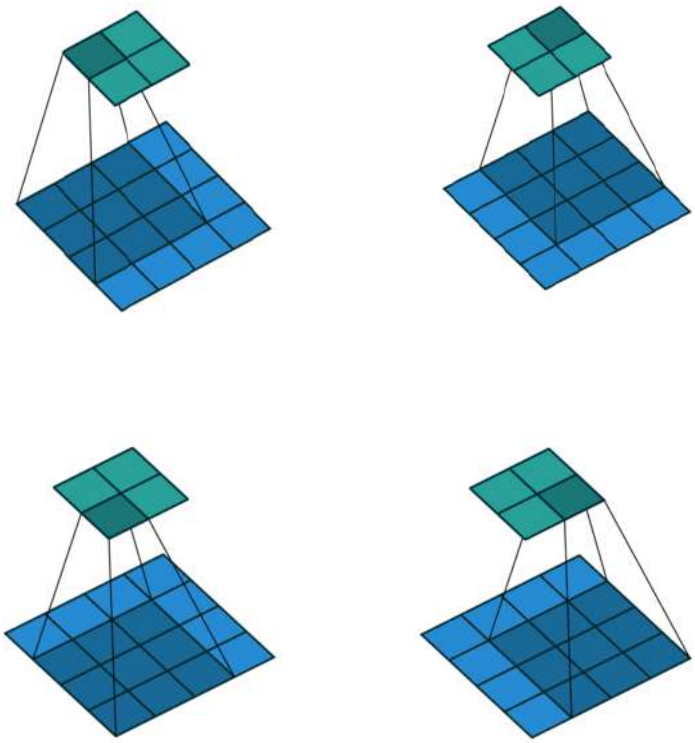
Na obrazie w środku widniał piksel o wartości 3, który został podniesiony do wartości 7. Kolejny przykład przedstawia spadek wartości piksela:

$$1*5+ 2*-1 + 2*-1 + 2*-1 + 2*-1 = -3$$

W tym przykładzie wartość spadła z 1 do -3.

Oczywiście cały przykład jest uproszczony, w prawdziwym algorytmie te obliczenia są powielane trzykrotnie dla każdej warstwy koloru RGB, a następnie nakładane na siebie w celu uzyskania finalnej wersji obrazu.

Na poniższym rysunku przedstawione jest jak z siatki 4x4, która posiada 16 pikseli można w ten sposób uzyskać siatkę 2x2, która posiada 4 piksele.



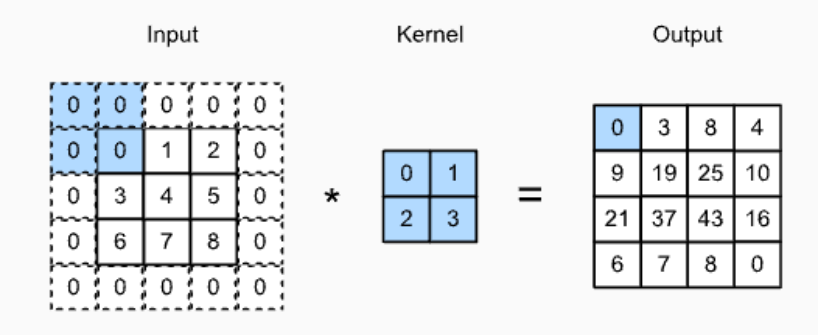
Rysunek 16 Przedstawienie generalizacji pikseli za pomocą konwolucji.

Źródło: https://towardsdatascience.com/types-of-convolution-kernels-simplified-f040cb307c37

Utrata informacji jest zauważalna na małym przykładzie, łatwo zatem wyobrazić sobie jak dużym ułatwieniem dla obliczeń jest zastosowanie tej techniki na siatce, która posiada kilka milionów pikseli.

Jak podaje Mamczur (2015) warstwy splotowe są podstawą sieci konwolucyjnych CNN, ponieważ zawierają wyuczone filtry, które wyodrębniają cechy odróżniające od siebie różne obrazy. Kiedy definiujemy warstwę CNN nie podajemy tych filtrów, filtry te są dobierane i optymalizowane podczas procesu trenowania sieci, są one dobrane tak aby minimalizować błędy przy rozwiązywaniu problemu. W przypadku tej pracy kernele będą dobrane tak, aby jak najlepiej odróżnić od siebie gatunki zwierząt. Mirosław Mamczur w swojej publikacji naukowej z 2015 roku zawarł również wytłumaczenie trzech ważnych kroków generalizacji obrazu, które pragnę przybliżyć. Kernel Size nazywany rozmiarem jądra często nazywany jest też rozmiarem filtra, odnosi się on do wymiarów przesuwanego okna nad wejściem. Wybór tego hiperparametru ma ogromny wpływ na zadanie klasyfikacji obrazu. Na przykład małe jądra są w stanie wydobyć z danych wejściowych znacznie większą ilość informacji zawierających wysoce lokalne funkcje. Mniejszy rozmiar jądra prowadzi również do mniejszego zmniejszenia wymiarów warstw, co pozwala na głębszą architekturę. Proces ten działa w dwie strony, duży rozmiar jądra wyodrębnia mniej informacji, co prowadzi do szybszego zmniejszenia wymiarów warstw, czego skutkiem często jest gorsza wydajność. W skrócie duże jądra lepiej nadają się do wyodrębnienia większych elementów. Ostatecznie wybór odpowiedniego rozmiaru jądra będzie zależał od zadania i zestawu danych, ale generalnie mniejsze rozmiary jądra prowadzą do lepszej wydajności zadania klasyfikacji obrazu.

Padding – jak zostało opisane powyżej, problemem podczas nakładania warstw splotowych jest to, że istnieje tendencja do utraty pikseli na obwodzie naszego obrazu, gdyż dąży się do wyodrębnienia piksela ze środka ramki 3x3. Jednym z prostych rozwiązań tego powszechnego problemu jest dodanie dodatkowych pikseli zwanych jako wypełniacze dookoła naszego obrazu wejściowego, zwiększając w ten sposób rozmiar obrazu, zazwyczaj ustawiamy wartość tych pikseli na zero.

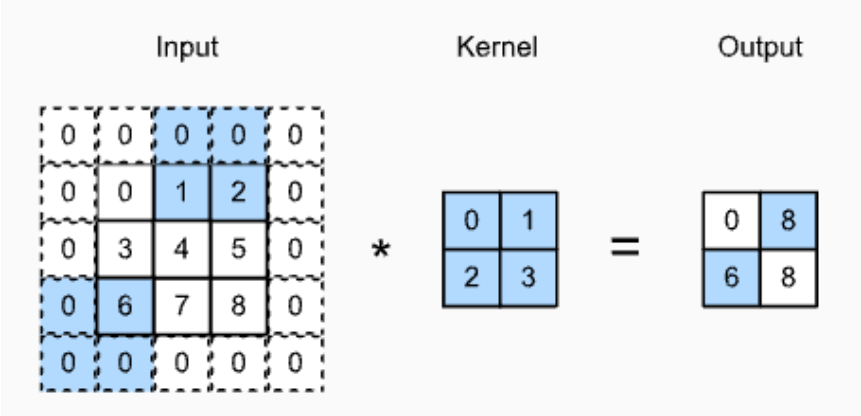


Rysunek 17 Zastosowanie paddingu.

Źródło: https://d2l.ai/chapter_convolutional-neural-networks/padding-and-strides.html

Na powyższym rysunku zwiększyliśmy rozmiar obrazu o jeden, dodając zaznaczone przerywaną linią zera, dzięki temu algorytm jest w stanie wyliczyć dokładniej znajdujące się na brzegach obrazu znaki charakterystyczne. Obliczenia wykonywane są tym samym sposobem jak w powyższych problemach.

Następnym parametrem generalizacyjnym jest parametr: „Strides”, który można tłumaczyć na kroki. Jak podaje Mamczur (2015) parametr ten wskazuje o ile pikseli jądro powinno zostać przesunięte na raz. Czyli inaczej mówiąc, oznacza krok przesunięcia okna filtra. Najczęściej używa się kroku wynoszącego 1 dla warstw splotowych. Oznacza to, że iloczyn skalarny jest wykonywany w oknie wejściowym na przykład 2x2 w celu uzyskania wartości wyjściowej, a następnie jest przesuwany o jeden piksel dla każdej kolejnej operacji.



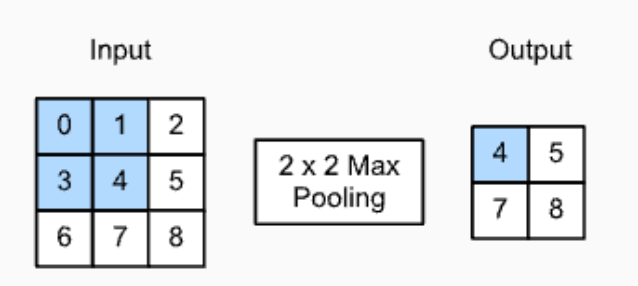
Rysunek 18 Zastosowanie strides.

Źródło: https://d2l.ai/chapter_convolutional-neural-networks/channels.html

Na powyższym obrazie widzimy krok ustawiony na 1 z siatką 2x2, proszę zwrócić uwagę, że w przypadku większego kroku zostałyby pominięte niektóre piksele co znacznie zmniejszyłoby informacje zawartą na obrazie. Wnioskiem, który można wyciągnąć z tej operacji jest to, że łatwo z jej pomocą zmieniać rozdzielczość obrazu, można w łatwy sposób zmniejszyć długość lub szerokość, na którym przeprowadzana jest generalizacja.

Kolejny sposób agregacji obrazu nazywa się: „Pooling”. Często ostatecznym zadaniem sieci neuronowych jest odpowiedzenie na pytanie czy danych obraz przedstawia przykładowo kota, tak więc zazwyczaj pojedyncze operacje wszystkich warstw przetwarzania obrazu powinny być wrażliwe na cały sygnał wejściowy. Kiedy funkcja stopniowo agreguje informacje uzyskując coraz to bardziej ogólne mapy, osiągamy cel jakim jest ostatecznie poznanie globalnej reprezentacji, czyli odpowiedzi na pytanie, przy jednoczesnym zachowaniu wszystkich zalet warstw splotowych na pośrednich warstwach przetwarzania. Co więcej, przy wykrywaniu cech niższego poziomu, takich jak krawędzie, często chcemy, aby nasze reprezentacje były nieco niezmienne względem rozpatrywania ich przez algorytm. Na przykład, jeśli weźmiemy pod uwagę obraz z ostrą granicą między czernią a bielą i przesuniemy cały obraz o jeden piksel w prawo, to wynik dla nowego

obrazu może być zupełnie inny, krawędzie przesuną się tylko o jeden piksel. W rzeczywistości przedmioty rzadko występują dokładnie w tym samym miejscu, nawet w przypadku statywu i nieruchomego obiektu, wibracje aparatu spowodowane ruchem migawki mogą przesunąć wszystko o piksel lub więcej. Ważna jest zatem funkcja Poolingu, która służy głównie do łagodzenia wrażliwości warstw splotowych na lokalizację. Podobnie jak poprzednie przykłady operator Poolingu składa się z okna w stałym kształcie, które jest przesuwane po wszystkich regionach danych wejściowych zgodnie z jego krokiem, obliczając pojedyncze dane wyjściowe dla każdej lokalizacji, przez która przechodzi okno o stałym kształcie. Na etapie obliczeń wyróżnia się dwa typy Poolingu, jest to maksymalny Pooling oraz średni Pooling. W tym pierwszym oblicza się maksymalną wartość dla piksela w zadanej siatce, a w średnim Poolingu oblicza się średnią arytmetyczną dla wszystkich elementów w siatce.



Rysunek 19 Zastosowanie Max Poolingu.

Źródło: https://d2l.ai/chapter_convolutional-neural-networks/pooling.html

Powyższy rysunek przedstawia generalizacje za pomocą Poolingu maksymalnego siatkę 3x3 do siatki 2x2. Siatka wyjściowa została obliczona w następujący sposób:

$$\mathrm{Max}(0,1,3,4)=4,$$

$$\mathrm{Max}(1,2,4,5)=5,$$

$$\mathrm{Max}(3,4,6,7)=7,$$

$$\mathrm{Max}(4,5,7,8)=8.$$

W przypadku Poolingu średniego oblicza się te parametry w analogiczny do powyższego przykładu sposób. Na poniższym rysunku przedstawione zostały zmiany po zastosowaniu omówionych rodzajów Poolingu, ciekawostką jest, że został również przedstawiony: „Min Pooling", który polega na wyciąganiu minimalnej wartości z piksela, widać jednak, że zatraca on przejścia między konturami.

Rysunek 20 Przedstawienie skutków zastosowania różnych technik poolingu.

Źródło: https://medium.com/@bdhuma/which-pooling-method-is-better-maxpooling-vs-minpooling-vs-average-pooling-95fb03f45a9

Jak zostało przedstawione na rysunku, average pooling wygładza obraz, z kolei max pooling sprawia, że obraz jest skoncentrowany na skrajnych kolorach. Niestety nie da się powiedzieć, która metoda jest najlepsza, ponieważ każda z wymienionych operacji ma swoje dobre oraz złe strony.

W powyższym rozdziale starałem się wytłumaczyć w jaki sposób komputer odbiera obraz, jakie operacje są możliwe do przeprowadzenia ze zdjęciami oraz które operacje najbardziej mogą pomóc programiście przy tworzeniu optymalnego algorytmu. Nowoczesne biblioteki takie jak TensorFlow mają wbudowane algorytmy, które pozwalają na automatyczne dobranie optymalnych operacji, niemniej jednak warto znać dane działania od podszewki, aby lepiej kontrolować przebieg tworzenia algorytmu.

Rozdział III Konwolucyjne Sieci Neuronowe

3.1 Historia Sieci Neuronowych

Jak podaje Matthew Kirk (2017) ludzkie oko ma zadziwiającą zdolność dopasowywania wzorców. Od urodzenia ludzie są w stanie zrozumieć otaczający ich chaos, aby skutecznie poruszać się po świecie. Wynika to oczywiście z naszego wychowania, naszego środowiska, ale przede wszystkim z naszego mózgu. Mózg przeciętnego człowieka zawiera około 86 miliardów neuronów, które komunikują się ze sobą poprzez sieć synaps. Neurony te za pomocą małych ładunków elektrycznych tworzą wielką siatkę, która uruchamia się przy określonej funkcji naszego ciała. Neurony są w stanie sterować naszymi kończynami, tworzyć myśli, liczyć w pamięci czy rozpoznawać całe spektrum kolorów. Pozornie neurony działają w totalnie losowy sposób, w Internecie łatwo natknąć się na filmy, które przedstawiają jak owa sieć działa w naszym mózgu, jednakże sieć ta działa w bardzo określony sposób co odkryli już w zeszłym stuleciu neurolodzy oraz kogniwistycy. Matematycy dawno temu podjęli próbę matematycznego zmapowania mózgu człowieka, aby na wzór działania mózgu opracować sieci neuronowe. Badania na ten temat trwają nieustannie od ponad 65 lat, wiele technik wymyślonych na początku było bardzo trywialnych, lecz z czasem rozwój technologiczny pozwolił na przełom, dzięki któremu w dzisiejszych czasach mówi się o prawdziwej sztucznej inteligencji, która w podejmowaniu decyzji jest często szybsza i lepsza od człowieka.

Rozwój tej dziedziny nastąpił w roku 1964 od badań Davida Hubela – kanadyjskiego naukowca, który w swoim eksperymencie podłączył mikro elektrody do mózgu kota i pokazywał na ekranie urządzenia pionowe oraz poziome kreski. Badacz zauważył, że na różne ułożenia kreski neurony w mózgu kota reagowały inaczej. Dzięki temu eksperymentowi udało się odkryć dwa typy komórek w pierwotnej korze wzrokowej zwane

komórkami prostymi i komórkami złożonymi. Następnym przełomowym momentem była praca Kunihiko Fukushimy, który w latach 80', zainspirowany pracą Hubela pokazał światu sztuczną sieć neuronową używaną do rozpoznawania znaków pisanych odręcznie w języku japońskim. Ten moment uważa się za powstanie pojęcia Konwolucyjnych Sieci Neuronowych. W roku 1989 Yann LeCun wykorzystał tak zwaną propagację wsteczną, aby modyfikować współczynniki jądra splotu, zrobił on to dla popularnego zbioru obrazków z ręcznie zapisanymi cyframi (MNIST). Uczenie od tamtego czasu stało się w pełni automatyczne, działało lepiej niż te prowadzone przez człowieka, a projektowanie współczynników było dostosowane do szerszego zakresu problemów z rozpoznawaniem obrazu i typów obrazów. Tamte czasy jednak zmagały się z wielkim problemem jakim była ograniczona moc obliczeniowa, komputery w latach 80-90 miały bardzo mało pamięci operacyjnej przez co procesowanie jakichkolwiek algorytmów było za wolne, aby opracować coś na miarę sztucznej inteligencji. Rozkwit sieci neuronowych nastąpił w 2012 roku, kiedy to ukraiński naukowiec Alex Krizhevsky wraz ze swoim zespołem opracował w ramach konkursu ImageNew algorytm, który zdeklasował inne zespoły. Ten wynik porównuje się z tym jakby ktoś w sprincie na 100 metrów był w stanie przebiec ten dystans w ciągu 6 sekund, kiedy reszta zawodników potrzebuje na niego 9 sekund.

Obecnie Konwolucyjne Sieci Neuronowe są jedną z najszybciej rozwijających się dziedzin nauki, pokłada się w nich wielkie nadzieje w sektorach takich jak medycyna, transport, czyli autonomiczne samochody oraz choćby cały sektor biznesu, w którym sztuczna inteligencja mogłaby podejmować za człowieka trafne decyzje biznesowe zwiększając dzięki temu maksymalizacje zysków.

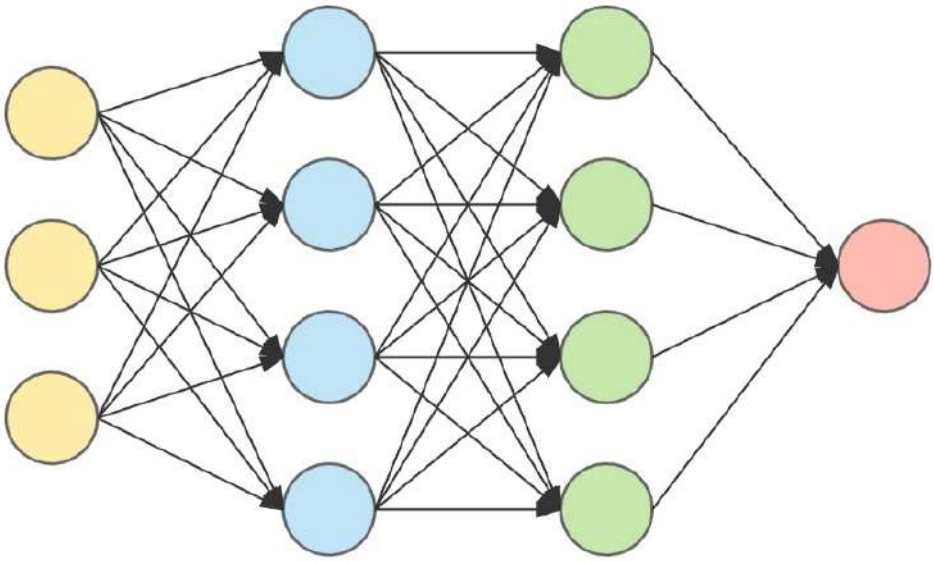
Rysunek 21 Klasyfikacja obiektów poprzez algorytm sztucznej inteligencji.

Źródło: https://towardsdatascience.com/everything-you-ever-wanted-to-know-about-computer-vision-heres-a-look-why-it-s-so-awesome-e8a58dfb641e

Na powyższym rysunku przedstawione zostało przykładowe klasyfikowanie obiektów przez algorytm wykorzystywany w nowoczesnych modelach samochodów autonomicznych. Funkcje w zaawansowanych algorytmach potrafią w bardzo szybkim czasie klasyfikować wiele obiektów naraz, dzięki czemu autopilot jest w stanie sam wyhamować samochód, gdy wykryje wbiegającą na jezdnię osobę szybciej niż zrobiłby to człowiek. W obecnych czasach samochody autonomiczne istnieją jednak ich strona prawna zostaje nie do końca określona, więc jedynym miejscem, w którym są one legalne jest stan California w Stanach Zjednoczonych Ameryki.

3.2 Działanie Konwolucyjnych Sieci Neuronowych

Konwolucyjne Sieci Neuronowe nazywa się również uczeniem głębokim, z języka angielskiego „Deep Learning”, nazwa ta wzięła się od metaforycznej głębokości obliczeń. W większości przypadków, kiedy omawia się sieci neuronowe, użytkownicy mogą natknąć się na podobny rysunek



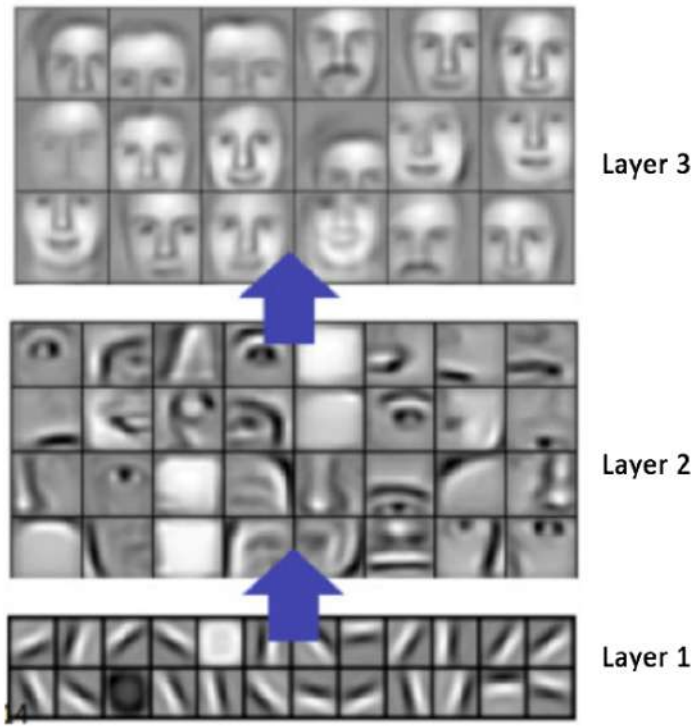
Rysunek 22 Uproszczona ilustracja perceptronu.

Źródło: https://www.w3schools.com/ai/ai_neural_networks.asp

Na rysunku został przedstawiony perceptron, na pierwszy rzut oka może on wydawać się zbiorem połączonych ze sobą kropek, jednak jest w tym reguła. W tym przypadku na rysunku znajduje się perceptron z czteroma warstwami, kolejno warstwą wejściową zaznaczoną kolorem żółtym, dwoma warstwami ukrytymi oznaczonymi kolorami niebieskim oraz zielonym i ostatnią warstwą wyjściową w kolorze czerwonym. Jest to rysunek poglądowy, którego używa się jedynie dla przykładu bowiem przeważnie w konwolucyjnych sieciach neuronowych warstw ukrytych jest mnóstwo, a samych połączonych ze sobą kropek, które nazywają się „biasami" jest często tysiące lub nawet miliony.

Wyobrażenie sobie całej sieci neuronowej jest skomplikowane, każdy pojedynczy bias zwany również neuronem zawiera liczbę, która za pomocą funkcji aktywacji jest zamieniana na nieliniową tak aby model nauczył wyłapywać się jak najmniejsze intuicyjne zależności. W pierwszych warstwach algorytm uczy się na małych próbkach pikseli jak rozpoznawać krawędzie, małe okręgi, linie proste, przerywane oraz ciągle.

Następne warstwy za pomocą poolingu opisanego w poprzednim rozdziale łączą zbiory z poprzednich warstw w większe grupy, które są w stanie rozpoznać małe cechy charakterystyczne takie jak oko, brew, ogon lub róg. Ostatnie warstwy sieci neuronowej potrafią zgromadzić wiele tych informacji i przesłać do ostatniej warstwy wyjściowej, z jakim prawdopodobieństwem wskazują, że obiektem, który został zidentyfikowany jest dane zwierzę. Algorytm klasyfikuje to za pomocą treningu, w którym nauczył się rozpoznawania cech charakterystycznych u zwierząt, w przypadku wiewiórki omawianej w projekcie, te cechy to rude ubarwienie, puszysty ogon, charakterystyczna postawa, szpiczaste uszy oraz wąsy.



Rysunek 23 Sposób klasyfikacji szczegółów dla kolejnych warstw neuronów.

Źródło: https://sarathpanat.medium.com/all-about-convolutional-neural-network-cnn-6ccce6738958

Powyższy rysunek poglądowy pokazuje w jaki sposób poszczególne warstwy klasyfikują coraz to bardziej skomplikowane cechy charakterystyczne. Widoczne w pierwszej warstwie kreski, kropki oraz kółka przekształcają się w nosy, oczy, brwi oraz wąsy po to, aby w ostatniej warstwie zbudować z nich twarze i sklasyfikować co one przypominają. Jest to oczywiście rysunek poglądowy bowiem algorytm używa więcej warstw oraz obliczeń, które ciężko by było przedstawić na zrozumiałym dla ludzkiego oka rysunku.

Jak podaje Mamczur (2021) jednym z powodów, dlaczego sieci neuronowe są w stanie osiągnąć tak olbrzymią dokładność, jest ich nieliniowość. Nieliniowość jest niezbędna do wytworzenia nieliniowych granic decyzyjnych, tak aby wynik nie mógł być zapisany jako liniowa kombinacja danych wejściowych. Gdyby nie było nieliniowej funkcji aktywacji, głębokie sieci neuronowe przekształciłyby się w pojedynczą, równoważną warstwę splotową, która nie działałaby tak dobrze. W prostszych słowach funkcja aktywacji działa w ten sposób, że jeżeli obliczony ładunek sygnału wejściowego jest dostatecznie wielki to reprezentujący go neuron się aktywuje, jest to operacja zero-jedynkowa. Algorytm w ten sposób sprawdza czy dany zbiór pikseli reprezentuje coś co może mu się przydać czy nie, przez co potrafi on w bardzo szybki sposób klasyfikować obiekt wraz z kolejną warstwą neuronów. Jak podaje Pelic (2011) sztuczny neuron można rozpatrywać jako specyficzny przetwornik sygnałów działający według następującej zasady: na wejście przetwornika doprowadzone są sygnały wejściowe, które następnie są mnożone przez odpowiednie współczynniki wag, ważone sygnały wejściowe są następnie sumowane i na tej podstawie wyznacza się aktywność neuronu. W bloku sumowania wykonane jest algebraiczne sumowanie ważonych sygnałów wejściowych oraz generowany jest sygnał wyjściowy φ:

$$\varphi = \sum_{i=1}^{R} w_i u_i + b = w^T u + b$$

gdzie: w – wektor współczynników wag w_i, u – wektor sygnałów wejściowych u_i, R – liczba wejść neuronu, b – próg (bias).

Sygnał φ poddawany jest przetwarzaniu przez blok aktywacji f(φ) realizujący zależność y = f(φ). Ostatecznie sygnał wyjściowy ma postać, którą opisał Andrew Ng:

$$y = f(\varphi) = f(\sum_{i=1}^{R} w_i u_1 + b\ ) = f(w^T u + b)$$

Funkcja aktywacji, w zależności od konkretnego celu, jakiemu służy neuron, może przyjmować różne postacie, najczęściej wykorzystywane w obecnych czasach to:

- Funkcja sigmoidalna unipolarna,

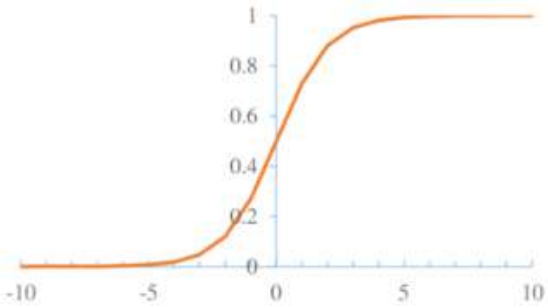
- Funkcja ReLU,

- Funkcja SoftMax

Poniższe rysunki przedstawiają funkcje na wykresach. Wszystkie funkcje wraz ze wzorami zostały przedstawione w kursie „Deep Learning Specialization" opracowanego przez Andrew Ng (2012).

Funkcja sigmoidalna unipolarna:

$$f(x) = \frac{1}{1 + e^{-\beta x}}$$



Rysunek 24 Funkcja sigmoidalna.

Źródło: https://www.researchgate.net/figure/Nonlinear-function-a-Sigmoid-function-b-Tanh-function-c-ReLU-function-d-Leaky_fig3_323617663

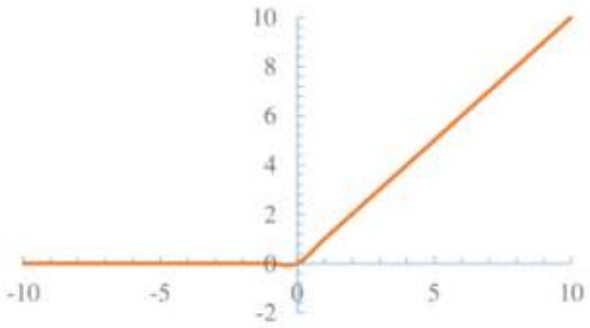
Nowoczesne sieci neuronowe maja wiele warstw, a jeśli istnieje w nich kilka warstw aktywowanych przez funkcje sigmoidalne to istnieje wysokie prawdopodobieństwo uzyskania zerowej szybkości uczenia się. Błąd ten występuje, ponieważ nachylenie funkcji sigmoidalnej jest bardzo płytkie, gdy wejście jest dalekie od zera, co spowalnia w znacznym tempie proces uczenia się gradientu.

Funkcja ReLU

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

Pochodna funkcji ReLU

$$\frac{df(x)}{dx} = \begin{cases} 1 & dla\ x > 0 \\ 0 & dla\ x < 0 \end{cases}$$



Rysunek 25 Funkcja ReLU

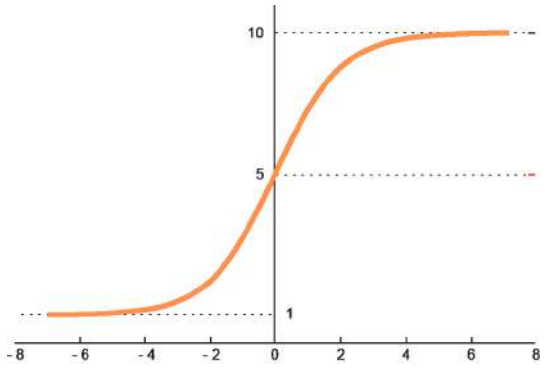
Źródło: https://www.researchgate.net/figure/Nonlinear-function-a-Sigmoid-function-b-Tanh-function-c-ReLU-function-d-Leaky_fig3_323617663

Jak podaje Andrew Ng (2012) funkcja ReLU rozwiązuje wiele problemów sigmoidów. Obliczanie dzięki niej jest łatwe i szybsze. Za każdym razem, gdy wejście jest dodanie, ReLU ma nachylenie -1, które zapewnia silny gradient w dół. ReLU nie jest jednak ograniczone do zakresu 0-1, więc jeśli zostanie użyte w warstwie wyjściowej, nie ma gwarancji, że będzie w stanie reprezentować prawdopodobieństwo.

Funkcja SoftMax

$$\sigma(z)i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Gdzie: $\sum_{j=1}^{K} \sigma(z)_j = 1$ - prawdopodobieństwo zawsze sumuje się do jedności, e – liczba Eulera, K – szerokość wektorów wejściowego i wyjściowego, jak podają Tadeusiewiecz oraz Szaleniec (2015) znormalizowana funkcja wykładnicza stosowana głównie w najwyższej warstwie klasyfikatorów, w celu obliczenia prawdopodobieństwa przynależności wektora wejściowego z do każdej z K klas wyjściowych.



Rysunek 26 Funkcja SoftMax.

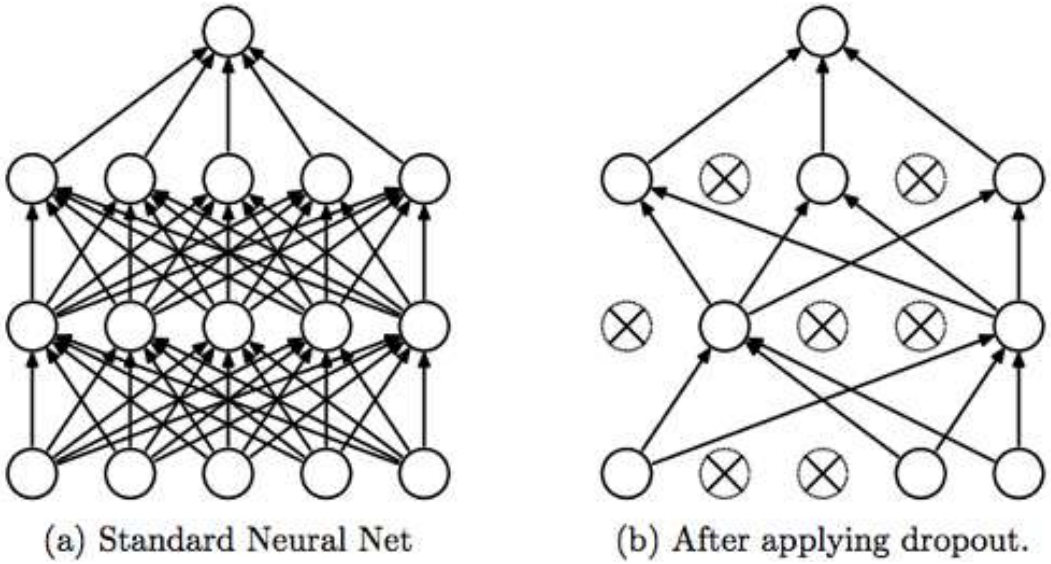
Źródło: https://www.researchgate.net/figure/Graphic-representation-of-the-softmax-activation-function_fig5_348703101

Operacja SoftMax służy kluczowemu celowi, mianowicie upewnieniu się, że wyniki algorytmu sumują się do 1. Z tego powodu operacje SoftMax są przydatne do skalowania wyników modelu na prawdopodobieństwa. Po przejściu przez aktywacje za pomocą funkcji SoftMax każda klasa odpowiada teraz odpowiedniemu prawdopodobieństwu.

3.3 Warstwa porzucenia oraz przetrenowanie modelu.

Przetrenowanie modelu to efekt towarzyszący projektom, które nie zawierają odpowiednich zbiorów danych. Dobrą praktyką przy konwolucyjnych sieciach neuronowych jest zastosowanie przynajmniej 200 zdjęć na jeden podmiot do wyuczenia algorytmu, w przypadku, kiedy ilość zdjęć jest niewystarczająca są metody, które pozwalają na zwiększenie efektywności oraz precyzji modelu.

Warstwa porzucenia z języka angielskiego „Dropout Layer" to kolejny krok stosowany w modelach, jest on niezwykle ważny, ponieważ bez niego model mógłby poddać się zjawisku zwanemu przetrenowaniem. Przetrenowanie polega na zbyt dokładnym oraz powierzchownym klasyfikowaniu danych cech podczas wczesnych faz kojarzeń. Jako przykład można podać przetrenowany algorytm, który w jednej z pierwszych warstw sklasyfikowałby róg, algorytm nauczony na pamięć skojarzy róg z bykiem oraz nie skupi się na przeanalizowaniu innych cech charakterystycznych. Oczywiście podejście to jest złe, ponieważ cechę taką jak róg posiada wiele gatunków zwierząt. Dropout bądź porzucenie jak podaje Mamczur (2021) polega na losowym ustawieniu wychodzących krawędzi ukrytych jednostek na 0 przy każdej aktualizacji fazy treningu. Metoda ta jest bardzo efektywna, ponieważ co każde przejście losowo wyłączane są pewne połączenia. Dzięki tej technice sieć neuronowa nie nauczy się „na pamięć" zbyt szybko, ponieważ architektura co przeliczenie odrobinę się zmienia poprzez zerowanie losowych połączeń neuronów.



Rysunek 27 Perceptron klasyczny oraz perceptron z zastosowaniem techniki dropout.

Źródło: https://towardsdatascience.com/machine-learning-part-20-dropout-keras-layers-explained-8c9f6dc4c9ab

Na powyższym rysunku przedstawione zostały dwa perceptrony z warstwą wejścia, dwoma warstwami ukrytymi oraz warstwą wyjścia. Perceptron po lewej stronie pokazuje jak działa standardowa operacja łączenia ze sobą neuronów bez aktywowania techniki porzucenia. Widać, że każdy neuron z danej warstwy łączy się z każdym następnym z warstwy kolejnej. Ilustracja po prawej stronie pokazuje neurony, które zostały wykluczone poprzez technikę dropoutu. Neurony te są ukazane jako kółka z krzyżykiem w środku. Operacja ta ma za zadanie wykluczyć pewne zależności w modelu i wymusić na neuronach, aby zawarte w nich detale również wpływały na klasyfikację obrazu. Dzięki temu zabiegowi zwiększa się dokładność modelu.

Kolejna technika nazwana z języka angielskiego „Early Stopping" jest bardzo prostą metodą, którą opisał Mamczur (2021) polega ona na tym, aby zakończyć uczenie, gdy strata na zbiorze testowym nie rośnie. Czyli trenując dalej sieć nie poprawiamy mocy modelu na zbiorze testowym. Biblioteka TensorFlow została użyta w projekcie, posiada ona funkcję

EarlyStopping, która monitoruję parametr straty na zbiorze testowym po zakończeniu każdej epoki. Jeśli strata nie maleje, wówczas trening sieci zostaje zatrzymany. Należy zdefiniować 3 podstawowe parametry ustawiając EarlyStopping:

- Monitor – definiujemy, co chcemy monitorować na podstawie czego zaczynamy proces uczenia,
- Patience – tym parametrem definiuje się liczbę epok po ilu zatrzyma się model, jeśli funkcja nie zaobserwuje zmniejszania się funkcji straty
- Verbose – w jaki sposób będzie wyświetlana informacja o EarlyStoppingu.

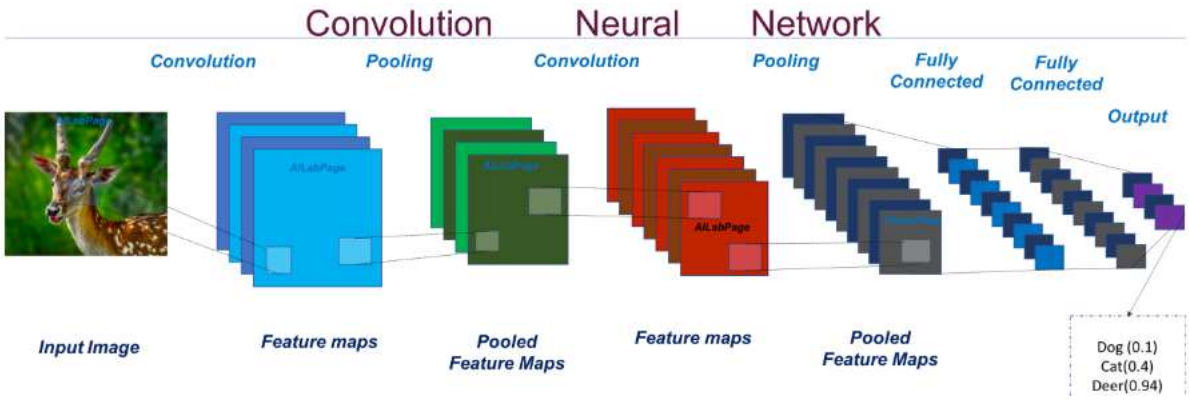
Dzięki tej technice w prosty sposób można zaoszczędzić sporo czasu. Jeżeli zatrzymamy trening danej sieci bardzo wcześnie, wówczas model skupi się na trenowaniu innych sieci w bardziej wydajny sposób, albowiem każda sieć wymaga pamięci operacyjnej do wykonywania obliczeń.

Kolejna, ostatnia metoda polega na regularyzacji wcześniej wspomnianych wag, które w sposób losowy dobierane są do modelu. Regularyzacja jest swoistym procesem nakładania na wagi kar, w postaci przemnażania ich poprzez bardzo małe wartości. Celem tego zabiegu jest uzyskanie małych liczb, przez co niektóre warstwy, dla których wygenerowane zostały losowo większe numery nie będą w żaden sposób faworyzowane przy aktywacji. Pragnę tylko przypomnieć, że dany neuron aktywuje się, gdy jego suma ważonych elementów przekroczy zero, w wypadku wartości ujemnych funkcja aktywacji w większości przypadków zwraca zero.

Wszystkie powyżej wskazane techniki mogą pomóc w uzyskaniu optymalnego modelu, który w jak najszybszym czasie będzie w stanie sklasyfikować dany obraz. Należy jednak pamiętać, że żadna, nawet najbardziej zaawansowana technika nie jest w stanie zastąpić braków danych.

3.4 Warstwa Wyjściowa

Reasumując informacje opisane w poprzednich rozdziałach każda z warstw ma swoje funkcje oraz cele w sieciach neuronowych. Pierwsza warstwa to obraz, następne warstwy przerabiają go w różny sposób tak, aby jak najlepiej rozpoznać kolor, krawędzie oraz cechy charakterystyczne. Następne warstwy „ściskają" wektor K-wymiarowy dowolnych wartości rzeczywistych do wartości umożliwiających klasyfikację obiektu w wartościach probabilistycznych od 0 do 1. To jest powód, dla którego każdy obraz w sieciach neuronowych jest reprezentowany jako macierz wartości pikseli. W momencie, kiedy warstwy konwolucji oraz poolingu działają jako ekstraktory cech z obrazu wejściowego, ostatnie warstwy w pełni połączonych neuronów zbierają te informacje oraz przekształcają w prawdopodobieństwo, do jakich kategorii udało się dany obraz zakwalifikować.



Rysunek 28 Proces klasyfikowania fotografii.

Źródło: https://vinodsblog.com/2018/10/15/everything-you-need-to-know-about-convolutional-neural-networks/

Powyższy rysunek przedstawia proces obrazu jako danych wejściowych, który przechodzi poprzez warstwy ukryte i w warstwie wyjściowej przypisane jest mu prawdopodobieństwo. W tym przypadku najwyższe prawdopodobieństwo 0.94 wskazuje, że na obrazie znajduje się sarna. Sama funkcja zwraca uporządkowane wyniki w N-wymiarowym wektorze, gdzie N jest liczbą klas, które program ma do wyboru. W przypadku powyższego rysunku klasy są 3 zatem algorytm zwróci 3 wymiarowy wektor.

3.5 Kompilacja modelu

W momencie, kiedy model ma już ustalone kroki, jego warstwy, operacje poolingu, poszczególne funkcje aktywacji oraz parametry są zdefiniowane należy go skompilować. Kompilowanie modelu to nic innego niż tłumaczenie kodu napisanego przez człowieka na język, który będzie zrozumiały przez komputer, który policzy wcześniej wymienione parametry. Krok ten jest podejmowany na zbiorze danych przeznaczonych do treningu, postawione są przed nim dwa zadania.

Pierwszym zadaniem jest określenie funkcji straty, z języka angielskiego „Loss Function". Funkcja ta ma za zadanie sprawdzić jak prognoza wypracowana przez algorytm myli się od prawdziwej wartości. Najpopularniejszą operacją matematyczną jest użycie entropii krzyżowej (ang. Cross-entropy). Jak podają Ramsundar oraz Bosagh Zadeh (2018) entropia jest matematyczną metodą pomiaru odległości pomiędzy dwoma rozkładami prawdopodobieństwa:

$$H\ (p, q) = -\sum_{x} p(x) \log q(x)$$

Tutaj p i q są dwoma rozkładami prawdopodobieństwa. Zapis p(x) oznacza prawdopodobieństwo p względem zdarzenia x. Ta definicja jest warta dokładnego omówienia. H zapewnia pojęcie odległości

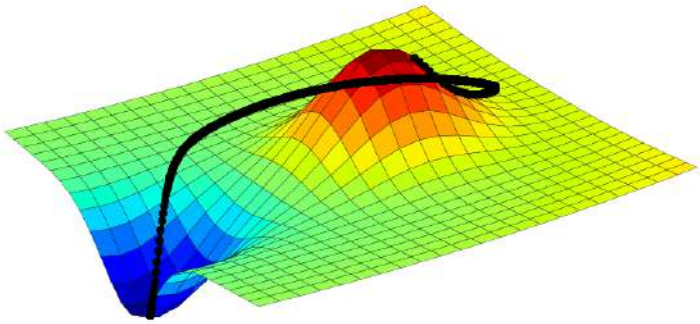
$$H(p,p) = -\sum_{x} p(x) \log p(x)$$

Wielkość ta jest entropią p i zazwyczaj jest zapisywana po prostu jako H(p). Jest to miara tego, jak nieuporządkowany jest rozkład; entropia jest zmaksymalizowana, gdy wszystkie zdarzenia są równie prawdopodobne. Funkcja straty będzie mierzyła różnice dla każdej epoki treningu. Wartości funkcji zazwyczaj przedstawia się na wykresie pokazujący jej wartość w poszczególnych epokach, obecnie modele są na tyle zaawansowane oraz zoptymalizowane, że w pierwszej epoce zazwyczaj funkcja straty jest wielka, a w następnych gwałtownie spada. Ze spadku oraz stabilizacji funkcji straty można wywnioskować, na której epoce powinno się zatrzymać model, gdyż nie jest w stanie wyciągnąć więcej szczegółów z danego zbioru danych.

Jak podaje Andrew Ng (2012) kolejnym zadaniem jest określenie algorytmu optymalizacji. Algorytm optymalizacji ma na celu pokierowanie funkcji straty do znalezienia jak najmniejszego błędu. Do dyspozycji jest wiele algorytmów takich jak propagacja RMS(ang. Root mean square) czy stochastyczny spadek wzdłuż gradientu i metoda pędu zwana SGD(ang. Stochastic gradient descent), najczęściej używanym algorytmem jednak jest propagacja ADAM (ang. Adaptive moment estimation). Aby zrozumieć, jak działają wymienione wyżej algorytmy należy najpierw zrozumieć koncepcję gradientu prostego.

Jak podają Jain, Fandango oraz Kapoor (2018) metoda gradientu prostego jest iteracyjnym algorytmem optymalizacji w celu znalezienia minimum funkcji. Autorzy dla zobrazowania działania algorytmu posłużyli się następującym przykładem. Gdy człowiek utknie na szczycie góry i chce w jak najszybszy sposób zejść w dół pierwszym krokiem będzie obserwacja zbocza góry we wszystkich kierunkach wokół i podjęcie decyzji o podążaniu w kierunku największego nachylenia zbocza w dół. Po każdym kroku ponownie należy ocenić wybór kierunku. Długość spaceru będzie zależała od stopnia nachylenia zbocza w dół. Jeśli nachylenie jest bardzo strome, robimy większe kroki,

ponieważ może to pomóc szybciej dotrzeć w dół. W ten sposób po pokonaniu mniejszej lub większej liczby kroków można zejść na dół. Analogiczna sytuacja ma miejsce w uczeniu maszynowym, gdzie celem jest zminimalizowanie błędów i kosztu poprzez aktualizację wag algorytmu. Aby znaleźć minimum funkcji kosztu przy użyciu gradientu, aktualizuje się wagi algorytmu proporcjonalnie do gradientu w kierunku najbardziej stromych zejść. W literaturze dotyczącej sieci neuronowych stała proporcjonalności znana jest również jako współczynnik uczenia. W uczeniu maszynowym na dużą skalę optymalizacja metodą gradientu prostego jest jednak dość kosztowna, ponieważ robimy tylko jeden krok po pojedynczym przejściu na całym zbiorze danych treningowych. Tak więc przy kilku tysiącach kroków czas potrzebny do osiągnięcia minimum funkcji kosztu jest ogromny.



Rysunek 29 Droga z punktu startowego do punktu minimum lokalnego.

https://ozzieliu.com/2016/02/09/gradient-descent-tutorial/

Na powyższym rysunku przedstawione zostało iteracyjne osiągnięcie przejścia z punktu startowego do minimum lokalnego. Jak podaje Andrew Ng (2012) optymalne rozwiązanie problemu to stochastyczne spadki wzdłuż gradientów, jest to technika, w którym aktualizacja wag algorytmu następuje po każdym przykładzie treningowym, bez czekania,

aż cały zestaw danych przejdzie przez algorytm. Stąd nazwa stochastyczny, aby zaznaczyć przybliżony charakter gradientu. Optymalizacja ADAM jest wariantem SGD (ang. Stochastic Gradient Descent), w którym utrzymuje się wskaźnik uczenia na parametr (wagę) i aktualizuje się go na podstawie wariancji i średniej poprzednich gradientów tego parametru. Optymalizacja za pomocą techniki ADAM okazała się najbardziej trafna w przypadkach rozwiązywania problemów uczenia głębokiego.

3.6 Podsumowanie modelu oraz jego trening

W momencie tworzenia własnych kroków w kodzie lub korzystania z gotowej architektury pomocą staje się funkcja biblioteki TensorFlow „model.summary()". Dzięki tej funkcji język Python wyświetla po kolei wszystkie kroki, które model przeprowadził wraz z opisem warstw, wielkością macierzy oraz ilością parametrów. Funkcja ta jest przydatna, aby sprawdzić po kolei jak nasz model przekształcał dane, ile wykorzystał przy tym parametrów oraz gdzie popełnił ewentualne błędy.

Kolejny krok to trenowanie modelu za pomocą funkcji: „model.fit_generartor()". Funkcja ta w języku Python przyjmuje kilka parametrów:

- Object – w tym miejscu umieszczamy nazwę zbioru, który chcemy trenować,
- Epoch – odpowiada za liczbę epok użytych do trenowania modelu – jedna epoka oznacza przejście całego zbioru przez sieć oraz powrót,
- Steps per epoch – określa całkowitą liczbę kroków pobranych z generatora jak tylko skończy się jedna epoka i rozpocznie się następna,
- Callbacks – lista funkcji zwrotnych zastosowanych podczas uczenia modelu. W przypadku tego projektu jako Callbacks zostały podane dwie funkcje: EarlyStopping oraz ModelCheckpoint.

Krok ten jest niezwykle ważny, ponieważ zajmuje od najwięcej czasu, modele, które w bardzo szczegółowy sposób muszą klasyfikować obiekty trenuje się nawet do kilkunastu godzin. Oczywiście zmienną decyzyjną jest w tym przypadku pamięć operacyjna komputera. Nowoczesne sieci neuronowe wykorzystują specjalne maszyny wirtualne, które posiadają duże ilości RAMu dla zwiększenia prędkości obliczeń.

3.7 Wizualizacja treningu

Ludzkie oko z większą łatwością potrafi zauważyć różnice, kiedy przedstawiona jest ona w sposób graficzny. Dobrą praktyką po wytrenowaniu modelu jest sporządzenie dodatkowego kodu, którego wynikiem jest ilustracja funkcji straty oraz dokładności z rozróżnieniem na poszczególne epoki. Na rysunku 30 przedstawione zostały dwie miary, funkcji straty oraz dokładności. Obliczanie funkcji straty zostało dokładnie omówione w rozdziale 3.5. „Accuracy" oznaczające z języka angielskiego dokładność wylicza się ze wzoru:

$$\text{Accuracy} = \frac{Liczba\ poprawnych\ prognoz}{Liczba\ wszystkich\ prognoz}$$

W projekcie, który został opisany w rozdziale czwartym kod do sporządzenia wspomnianej wizualizacji prezentuje się w następujący sposób:

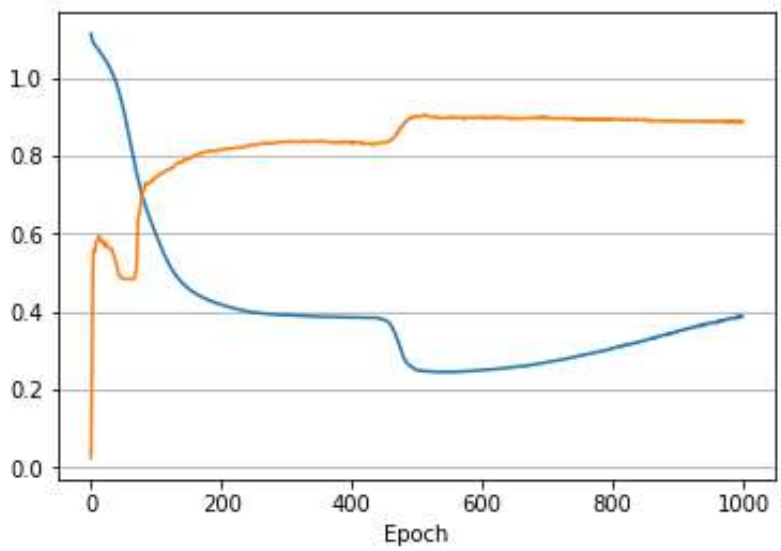
Kod 3.1

```
h = his.history

h.keys()

plt.plot(h['loss'], 'go--', c='green')

plt.plot(h['accuracy'], 'go--', c='red')
```

```
plt.title('Loss vs Acc')


plt.show()
```

W kodzie wstępnie wyciągnięto historię z modelu zapisanego pod zmienną „his", następnie na jednym wykresie nałożone zostały dwie linie:

- Loss – oznaczająca funkcję straty, zaznaczona niebieską linią
- Acc – oznaczająca Accuracy, czyli dokładność, zaznaczona pomarańczową linią.



Rysunek 30 Wykres przedstawiający zmienne funkcji straty oraz dokładności.

Źródło: http://www.jussihuotari.com/2018/01/17/why-loss-and-accuracy-metrics-conflict/

Na powyższym rysunku przedstawiony został wykres z dwoma zmiennymi. Na osi X znajduje się ilość epok. Widoczny jest gwałtowny spadek funkcji straty pomiędzy pięćdziesiątą, a setną epoką.

W ten oto sposób wynikiem będzie ułamek poprawnych prognoz w każdej epoce. Wynik dokładności zatrzymuje się na poziomie 90%, jest to znak, że pomiar jest zadowalający, ponieważ dziewięćdziesiąt na sto fotografii zostało poprawnie sklasyfikowane.

Rozdział IV Opis aplikacji

4.1 Architektura rozwiązania

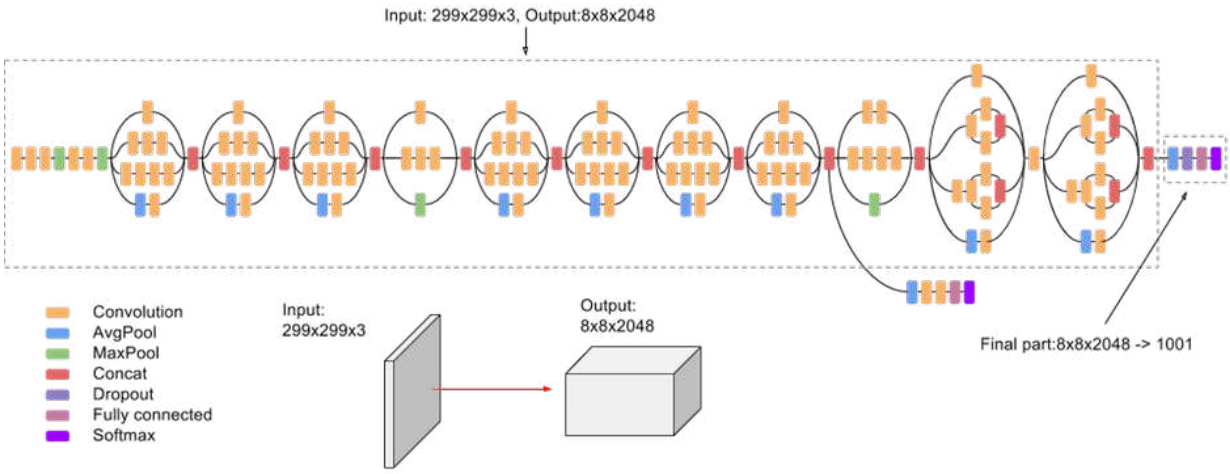
Projekt został napisany w całości w języku Python. Do przechowywania danych w postaci zdjęć została użyta chmura Dropbox (www.dropbox.com). Usługa Dropbox używana jest głownie do przechowywania kopii zapasowych i synchronizacji plików między komputerami. Portal ten istnieje od 2008 i jest bardzo wygodny poprzez łatwy interface i przyjazność wobec nowych użytkowników. Zdjęcia użyte do wytrenowania modelu zostały ściągnięte bezpośrednio z portalu Kaggle i umieszczone w Dropboxie.

Do pisania kodu w języku Python potrzebny jest edytor kodu, zwany też jako IDE (ang. Integrated Development Environment). Początkowo projekt był realizowany w środowisku PyCharm, lecz później całe repozytorium zostało przeniesione do nowoczesnej usługi w chmurze Google Colab. Google Colab to propozycja ułatwienia życia programistom za pomocą wygodnego kompilatora kodu, który może działać w przeglądarce. Rozwiązanie jest o tyle wygodne, że pozwala na dostęp z dowolnego komputera. Projekty w łatwy sposób można ze sobą współdzielić, a sam interface oferuje wiele przydatnych funkcji takich jak na przykład podpięcie maszyny wirtualnej w celu wykonania większych obliczeń. Architektura ta została wybrana ze względu na swoją kompatybilność, Google Colab posiada wbudowane funkcje, które pozwalają na ściąganie bezpośrednio plików z chmury Dropboxa. Edytor kodu ponadto jest darmowy i automatycznie podpina najnowszą, stabilną wersję Pythona tak aby wszystkie biblioteki działały. W języku Python zostały zainstalowane następujące biblioteki:

- Numpy – biblioteka do obsługi wielowymiarowych tabel i macierzy,
- Pandas – biblioteka do analizy i manipulowania zbiorami danych,
- Matplotlib – biblioteka do tworzenia wykresów oraz wizualizacji,
- Keras – biblioteka ułatwiająca tworzenie modeli i ich ocenę,

- TensorFlow – biblioteka zawierająca silnik wykonawczy, który jest odpowiedzialny za wszystkie obliczenia w głębokich sieciach neuronowych.

Model, który został zaimportowany do przeprowadzenia obliczeń oraz klasyfikowania obrazów nosi nazwę: „InceptionV3”. Silnik ten skupia się głównie na wykorzystywaniu mniejszej mocy obliczeniowej poprzez odpowiednie modyfikacje w architekturze. Algorytm został opracowany przez Christiana Szegedy, Vincenta Vanhoucke, Sergeya Loffe oraz Jonathana Shlens w 2015 roku i opublikowany w artykule o nazwie: „Rethinking the Inception Architecture for Computer Vision”. W porównaniu z innymi dostępnymi modelami, silnik ten okazał się bardzo wydajny obliczeniowo, zarówno pod względem liczby parametrów generowanych przez sieć, jak i ponoszonych kosztów ekonomicznych w postaci pamięci. Techniki uwzględnione w modelu „InceptionV3” obejmują faktoryzowane sploty, regularyzację, redukcję wymiarów oraz obliczenia równoległe, które pozwalają zaoszczędzić czas. Faktoryzowane sploty polegają na zmniejszeniu zapotrzebowania na wydajność obliczeniową, poprzez zmniejszenie liczby parametrów występujących w sieci. Faktoryzowane sploty również stale monitorują wydajność sieci. Kolejna technika to większa generalizacja danych wejściowych za pomocą tworzenia mniejszych zwojów. Model zastępuje większe zwoje mniejszymi zwojami, co prowadzi do szybszego treningu. W publikacji Vihara Kuramy (2020) podany został przykład zmiany filtra z siatką 5x5, który posiada 25 parametrów na dwa filtry o siatce 3x3, które mają tylko 18 parametrów więc prowadzą do większej generalizacji danych wejściowych. Sploty asymetryczne są kolejną z funkcji zawartych w tym modelu. Splot o siatce 3x3 można zastąpić splotem 1x3, po którym następuje splot 3x1. Jeśli splot o siatce 3x3 zostanie zastąpiony splotem 2x2, liczba parametrów będzie nieco wyższa niż proponowany splot asymetryczny. Operacja ta również dąży do zwiększenia wydajności. Kolejnym rozwiązaniem są klasyfikatory pomocnicze. Są to małe sieci neuronowe wstawiane między warstwy podczas uczenia, a poniesiona strata jest dodawana do straty sieci głównej. Jako pierwsza technika ta została wykorzystana w modelu GoogLeNet, jednakże tam pełniła rolę głębszej sieci, w modelu InceptionV3 pełni ona rolę regulatora.



Rysunek 31 Wizualizacja modelu InceptionV3

Źródło: https://blog.paperspace.com/popular-deep-learning-architectures-resnet-inceptionv3-squeezenet/

Na powyższym rysunku przedstawiony został cały model InceptionV3. Jak łatwo zauważyć jest on bardzo rozbudowany, a co za tym idzie jego dokładność jest bardzo wysoka. W projekcie został on użyty w celu sprawdzenia wydajności tego modelu oraz zgłębienia wiedzy na ten temat przez autora. InceptionV3 jest obecnie uznawany jako jeden z najszybszych oraz najwydajniejszych modelów na rynku.

4.2 Pochodzenie zbioru danych

Zbiór danych, na którym został nauczony oraz przetestowany algorytm opracowany w Rozdziale III pochodzi ze strony: „Kaggle.com”. Kaggle jest platformą internetową, która od 2010 roku służy analitykom danych za pole do pokazywania swoich umiejętności, wykazania się podczas licznych konkursów oraz miejsca, w którym można nauczyć się podstaw. Platforma działa na zasadzie wolnego dostępu, po założeniu konta można publikować na niej treści oraz ściągać z niej dane. Dane, które zostały wykorzystane do projektu opisanego w pracy noszą nazwę „Recognize Animals”. Zbiór danych został zebrany oraz opublikowany przez użytkownika o nazwie: „Prateek”, w roku 2020.

Plik z danymi ma rozmiar 235 MB, w większości są to zdjęcia o formacie JPEG w ilości 7901, JPG - 1157 oraz 48 zdjęć PNG, sam folder zawiera trzy główne pliki:

- Train (89% danych)
- Test (10% danych)
- Testingsetanimals.csv (1% danych)

Jest to standardowy podział zbiorów danych na zbiór treningowy oraz testowy, trzeci plik w postaci CSV zawiera instrukcje dla kolejności wczytywania danych i nie będzie odgrywał znaczącej roli w późniejszych krokach.

Zbiór treningowy (ang. Train) jest podzielony na pięć gatunków, na których algorytm będzie uczył się rozpoznawania gatunków zwierząt. W przypadku uczenia maszynowego dobrą praktyką jest używanie do nauczania ponad dwustu zdjęć, oczywiście ilość zależy od poziomu trudności i szczegółowości zagadnienia. W przypadku algorytmu wykorzystanego w tym projekcie szczegółowość nie jest wielka, lecz ze względu na małą liczbę przypadków, na których będzie trenowany algorytm na jeden gatunek zwierzęcia przypada od tysiąc trzystu do tysiąca dziewięciuset zdjęć. Dzięki tak dużej bazie danych algorytm może z bardzo wielką dokładnością nauczyć się rozpoznawać poszczególne gatunki.

Samych gatunków zwierząt, na których przeprowadzone zostało badanie jest pięć. Poniższa tabela przedstawia nazwę gatunku, oryginalną nazwę w zbiorze danych oraz ilość zdjęć jakie posiada baza danych.

Tabela 1.1 Nazwy gatunków oraz ilość zdjęć w zbiorze treningowym.

Nazwa gatunku	Oryginalna nazwa w bazie danych	Ilość zdjęć w zbiorze treningowym
Słoń	Elefante_train	1302
Motyl	Farfalla_train	1901
Krowa	Mucca_train	1680
Owca	Pecora_train	1639
Wiewiórka	Scoiattolo_train	1676

Źródło: opracowanie własne na podstawie informacji z www.kaggle.com

Specjalnie do projektu został użyty zbiór danych ze zróżnicowanymi wielkościowo zwierzętami, aby ułatwić później pracę nad wysyłaniem sygnału ostrzegawczego opisanego w poprzednim rozdziale. Dla przykładu motyl jest za małym zwierzęciem, aby ostrzegać kierowcę o jego obecności w pobliżu jezdni. Baza danych jest na tyle różnorodna, że dla algorytmu skomplikowane będzie odróżnienie od siebie takich gatunków jak owca, krowa oraz słoń; dla ludzkiego oka takie operacje są banalnie proste jednak algorytmy uczą się głównie koloru, kształtu oraz punktów charakterystycznych takich jak rogi, ogony, kły u zwierząt. W dalszym ciągu rozwoju projektu wytrenowany algorytm może być na tyle dokładny, aby rozpoznawać dokładną rasę zwierzęcia, jego płeć czy nawet przybliżony

wiek. W obecnej fazie projekt jest za mało zaawansowany, aby prowadzić samemu takową bazę danych oraz możliwości obliczeniowe są ograniczone ze względu, że są one w pełni wykonywane na prywatnym komputerze. Dla tak dokładnych detali jak wiek, płeć czy rasa należy przewidzieć odpowiednio opisaną bazę danych minimum dziesięcioma tysiącami zdjęć. Takie zasoby w większości przypadków są płatne, a na ich wykorzystywanie należy poprosić o zgodę administratorów, którzy są odpowiedzialni za opisywanie oraz uzupełnianie informacji o swojej bazie.

Rysunek 32 Przykładowe zwierzęta ze zbioru danych.

Źródło: https://www.kaggle.com/datasets/pratik2901/animal-dataset

Zbiór testowy (ang. Test) jest to zbiór dziewięćset dziesięciu zdjęć, które nie są w żaden sposób uporządkowane oraz opisane. Folder ten zawiera zdjęcia w celu testowania poprawności przewidywania algorytmu. W późniejszej fazie pracy znajduje się opis procesu losowego wyboru zdjęcia z pliku testowego oraz automatyczne

zakwalifikowanie go do jednego z gatunków, w ramach testu poprawności algorytmu. Jest to popularny sposób sprawdzania dokładności wytrenowanej sieci neuronowej, dobrą praktyką jest ustalanie zbioru treningowego oraz zbioru testowego w proporcjach 80% zbioru treningowego do 20% zbioru testowego.

4.3 Kod oraz opis aplikacji

W owym rozdziale wklejony zostanie kod z aplikacji oraz dodane zostaną komentarze objaśniające.

Kod 4.1

```
!wget
https://www.dropbox.com/s/6sprpz5e6snyppg/Recognize_animals_data
set.zip?dl=0


!unzip /content/Recognize_animals_dataset.zip?dl=0
```

Za pomocą komendy „wget” oraz „unzip” najpierw został pobrany z chmury plik ze zdjęciami, a następnie został on rozpakowany.

Kod 4.2

```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

```

```
import keras

from keras.layers import Dense, Flatten

from keras.models import Model

from keras.applications.inception_v3 import InceptionV3,
preprocess_input

from keras.preprocessing.image import ImageDataGenerator,
load_img, img_to_array
```

Ściągnięte zostały wszystkie wymienione biblioteki oraz zaimportowane zostały ich poszczególne funkcje. Operację importu poszczególnych funkcji wykonuje się w celu optymalizacji kodu wedle zasady o braku niepotrzebnych wywołań. Biblioteki oraz ich przeznaczenie zostały opisane w rozdziale 4.1.

Kod 4.3

```
base_model = InceptionV3(input_shape=(256,256,3),
                         include_top=False)
```

Jako model bazowy zaimportowany został model InceptionV3, jako parametr: „input_shape” został mu podany wymiar zdjęcia, które będzie miało rozmiar 256 na 256 pikseli oraz 3 warstwy kolorów: czerwony, zielony oraz niebieski.

Kod 4.4

```
for layer in base_model.layers:

```

```
    layer.trainable = False
```

W tym kroku za pomocą pętli, dla każdej warstwy w modelu został wyłączony parametr: „trainable”. Ustawienie tego parametru na fałsz oznacza jego wyłączenie. Wszystkie wagi warstwy z możliwych do trenowania stają się niemożliwe do trenowania. Operację tą nazywa się „zamrażaniem” warstwy. Stan zamrożonej warstwy nie zostanie zaktualizowany podczas treningu. W momencie wyłączenia tej funkcji algorytm staje się szybszy bez większej utraty wydajności.

Kod 4.5

```
X = Flatten()(base_model.output)

X = Dense(units=5, activation='sigmoid')(X)

model = Model(base_model.input, X)

model.compile(optimizer = 'adam', loss =
keras.losses.binary_crossentropy, metrics=['accuracy'])
```

W powyższym kroku za pomocą funkcji „Flatten” zostało wydane polecenie na spłaszczenie danych wyjściowych, następnie w funkcji „Dense” przekazano za pomocą parametru „units” jak wiele klas ma rozpoznać algorytm w momencie klasyfikacji. Został on ustawiony na 5, ponieważ tyle jest gatunków zwierząt w zbiorze danych. Algorytmowi została przypisana funkcji aktywacji „Sigmoid”. Kolejna linijka kodu to przyłączenie zmiennej „X” do modelu, a następnie jego kompilacja za pomocą optymalizacji „ADAM” oraz wyliczeniem funkcji straty za pomocą binarnej entropii krzyżowej. Metryką, która

ma opisywać model została wybrana „Accuracy”, którą można tłumaczyć na język polski jako dokładność.

Kod 4.6

```
model.summary()
```

Powyższy kod wykonuje podsumowanie modelu. Dzięki wywołaniu tej komendy ukaże się opis każdej warstwy w modelu. Celowo nie zostanie to umieszczone w pracy, ponieważ warstw jest bardzo dużo. To jakie kroki posiada model jest przedstawione na rysunku 32.

```
Total params: 22,171,429

Trainable params: 368,645

Non-trainable params: 21,802,784
```

Pod koniec komendy wyświetlony jest następujący komunikat. W modelu jest ponad 22 miliony parametrów, z tego tylko nieco ponad 350 tysięcy bierze udział w treningu. Stało się tak poprzez wyłączenie parametru trenowalności w każdej warstwie.

Kod 4.7

```
train_datagen = ImageDataGenerator(featurewise_center=True,

                                   rotation_range=0.4,

                                   width_shift_range=0.3,
```

```
                                    horizontal_flip=True,
                                    preprocessing_function=
                                    preprocess_input,
                                    zoom_range=0.4,
                                    shear_range=0.4)
```

Powyższy kod generuje partie danych z rozszerzeniem w czasie rzeczywistym. Nakłada on filtry na zdjęcia tak aby były one lepiej przystosowane do czytania przez model.

Parametr „featurewise_center" ustawiony na „True" sprawia, że zdjęcia ma średnią 0. Oznacza to, że kolory na fotografii zostaną ujednolicone.

Parametr „rotation_range" ustawiony na wartość „0.4" będzie odpowiadał za zakres stopni dla obrotów losowych. Parametr „width_shirt_range" ustawiony na wartość „0.3" odpowiada za losowe przesunięcia o ułamek całkowitej szerokości w celu nauczenia modelu, że obrazy nie zawsze znajdują się w centrum zdjęcia. Parametr „horizontal_flip" ustawiony na wartość „True" sprawia, że obrazy będą losowo obracane horyzontalnie. Parametr „zoom_range" ustawiony na wartość „0.4" odpowiada za ułamek losowego zbliżenia obrazu. Parametr „shear_range" ustawiony na wartość „0.4" odpowiada za ułamek intensywności ścinania pod kątem kierunku przeciwnym do ruchu wskazówek zegara w stopniach.

Kod 4.8

```
train_data =
train_datagen.flow_from_directory(directory='/content/Recognize
animals dataset/animal_dataset_intermediate/train',

target_size=(256,256),

batch_size=64)
```

Powyższy kawałek kodu przypisuje do zmiennej „train_data” wszystkie obrazy znajdujące się w pliku „train” znajdującego się w repozytorium. Obrazy są formatowane tak aby miały na wejściu 256x256 pikseli. Parametr „batch_size” odpowiada za liczbę próbek, która będzie propagowana przez sieć. Oznacza to, że w każdym kroku model będzie pobierał po 64 próbki na jeden krok tak długo aż algorytm nie wytrenuje się wystarczająco.

Kod 4.9

```
train_data.class_indices
```

Powyższa komenda wygeneruje następujący komunikat:

Kod 4.10

```
{'elefante_train': 0,

 'farfalla_train': 1,

 'mucca_train': 2,
```

```
 'pecora_train': 3,

 'scoiattolo_train': 4}
```

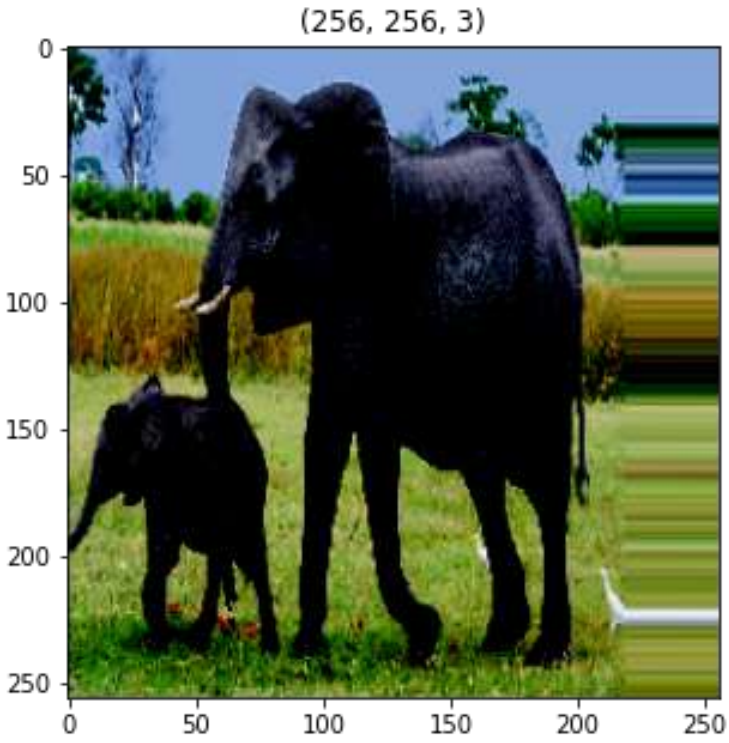
Jest to słownik z nazwami gatunków oraz przypisanymi do nich indeksami. Dla słonia został przypisany indeks 0, ponieważ indeksowanie w języku Python zaczyna się o 0.

Kod 4.11

```
t_img, label = train_data.next()

def plotImages(img_arr, label):

  '''

  input : image array

  output: plot images

  '''

  for idx, img in enumerate( img_arr ):

    if idx <= 10 :

      plt.figure(figsize=(5,5))

      plt.imshow(img)

      plt.title(img.shape)
```

```
        plt.axis = False

        plt.show()

plotImages(t_img, label)
```

Powyższy kod zawiera funkcje, do której przypisane są dwa parametry wejściowe. Funkcja ma na celu wyświetlać przekształcony obraz ze zbioru treningowego.



Rysunek 33 Losowo przekształcony obraz

Źródło: Opracowanie własne dzięki danym z www.kaggle.com

Na powyższym rysunku znajduje się przekształcony obraz. Zdjęcie zostało przesunięte w lewą stronę o czym wskazują rozciągnięte piksele po prawej stronie zdjęcia. Widoczne są również ujednolicone kolory w celu lepszego uwidocznienia konturów fotografii. Algorytm przekształca zdjęcia w losowy sposób tak, aby utrudnić modelowi uczenie się na pamięć pewnych zależności.

Kod 4.12

```
mc = ModelCheckpoint(filepath="./best_model.h5",
                     monitor="accuracy",
                     verbose = 1,
                     save_best_only=True)
es = EarlyStopping(monitor= "accuracy",
                   min_delta = 0.01,
                   patience = 5,
                   verbose = 1)
cb = [mc,es]
his = model.fit_generator(train_data,
                          steps_per_epoch=10,
                          epochs=30,
                          callbacks=cb)
```

Powyższy kod w pierwszym kroku za pomocą funkcji „ModelCheckpoint” tworzy wywołanie zwrotne w celu zapisania modelu Keras lub wag modelu. Technika ta jest dobrą praktyką, ponieważ model lub wagi można później załadować, aby kontynuować uczenie z zapisanego stanu. Model będzie zapisany w repozytorium pod nazwą „best_model.h5”, zostanie zapisany tylko najlepszy egzemplarz z wszystkich wytrenowanych algorytmów dzięki ustawieniu parametru „save_best_only” na wartość True, a wartość jaka będzie o tym decydować to „accuracy”. Kolejny krok to opisana w rozdziale 3.3 metoda „EarlyStoppingu”. Dzięki podanym parametrom metoda będzie nadzorowała wartość „accuracy”. Parametr „min_delta” ustawiony na „0.01” odpowiada za minimalną zmianę monitorowanej ilości, która ma zostać zakwalifikowana jako poprawa, oznacza to, że bezwzględna zmiana mniejsza niż ilość „0.01” będzie liczona jako brak poprawy, a dany model będzie zatrzymywany w celu przyśpieszenia innych, bardziej efektywnych modeli. Parametr „patience” ustawiony na 5 oznacza, że po 5 epokach bez poprawy trening zostanie przerwany. W następnym kroku generowany jest model.

```
Epoch 1/30
10/10 [==============================] - ETA: 0s - loss: 1.3051 - accuracy: 0.6276
Epoch 1: accuracy improved from -inf to 0.62759, saving model to ./best_model.h5
10/10 [==============================] - 28s 1s/step - loss: 1.3051 - accuracy: 0.6276
Epoch 2/30
10/10 [==============================] - ETA: 0s - loss: 0.2922 - accuracy: 0.8859
Epoch 2: accuracy improved from 0.62759 to 0.88594, saving model to ./best_model.h5
10/10 [==============================] - 12s 1s/step - loss: 0.2922 - accuracy: 0.8859
Epoch 3/30
10/10 [==============================] - ETA: 0s - loss: 0.2167 - accuracy: 0.9406
Epoch 3: accuracy improved from 0.88594 to 0.94063, saving model to ./best_model.h5
10/10 [==============================] - 13s 1s/step - loss: 0.2167 - accuracy: 0.9406
Epoch 4/30
10/10 [==============================] - ETA: 0s - loss: 0.1211 - accuracy: 0.9469
Epoch 4: accuracy improved from 0.94063 to 0.94687, saving model to ./best_model.h5
10/10 [==============================] - 12s 1s/step - loss: 0.1211 - accuracy: 0.9469
Epoch 5/30
10/10 [==============================] - ETA: 0s - loss: 0.1891 - accuracy: 0.9344
Epoch 5: accuracy did not improve from 0.94687
10/10 [==============================] - 11s 1s/step - loss: 0.1891 - accuracy: 0.9344
Epoch 6/30
10/10 [==============================] - ETA: 0s - loss: 0.1454 - accuracy: 0.9500
Epoch 6: accuracy improved from 0.94687 to 0.95000, saving model to ./best_model.h5
10/10 [==============================] - 12s 1s/step - loss: 0.1454 - accuracy: 0.9500
Epoch 7/30
10/10 [==============================] - ETA: 0s - loss: 0.1540 - accuracy: 0.9516
Epoch 7: accuracy improved from 0.95000 to 0.95156, saving model to ./best_model.h5
10/10 [==============================] - 12s 1s/step - loss: 0.1540 - accuracy: 0.9516
Epoch 8/30
10/10 [==============================] - ETA: 0s - loss: 0.1413 - accuracy: 0.9453
Epoch 8: accuracy did not improve from 0.95156
10/10 [==============================] - 11s 1s/step - loss: 0.1413 - accuracy: 0.9453
Epoch 9/30
10/10 [==============================] - ETA: 0s - loss: 0.1853 - accuracy: 0.9281
Epoch 9: accuracy did not improve from 0.95156
10/10 [==============================] - 11s 1s/step - loss: 0.1853 - accuracy: 0.9281
Epoch 10/30
10/10 [==============================] - ETA: 0s - loss: 0.1704 - accuracy: 0.9484
Epoch 10: accuracy did not improve from 0.95156
10/10 [==============================] - 12s 1s/step - loss: 0.1704 - accuracy: 0.9484
Epoch 11/30
10/10 [==============================] - ETA: 0s - loss: 0.1678 - accuracy: 0.9531
Epoch 11: accuracy improved from 0.95156 to 0.95312, saving model to ./best_model.h5
10/10 [==============================] - 13s 1s/step - loss: 0.1678 - accuracy: 0.9531
Epoch 12/30
10/10 [==============================] - ETA: 0s - loss: 0.1346 - accuracy: 0.9578
Epoch 12: accuracy improved from 0.95312 to 0.95781, saving model to ./best_model.h5
10/10 [==============================] - 12s 1s/step - loss: 0.1346 - accuracy: 0.9578
Epoch 12: early stopping
```

Rysunek 34 Tekstowa wizualizacja trenowania modelu

Źródło: Opracowanie własne

Powyższy rysunek przedstawia postępy w trenowaniu modelu w kolejnych epokach. W 12 epoce trening został przerwany poprzez funkcje EarlyStopping. Samo trenowanie modelu zajęło około 10 minut.

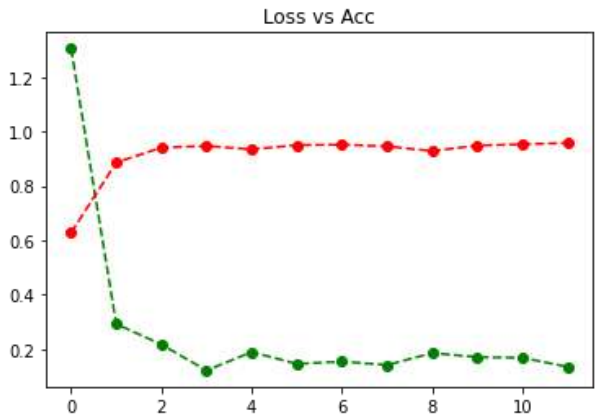
Generowany model jako pierwszy parametr przyjmuje dane do trenowania, w celu omówienia następnych parametrów należy przybliżyć pojęcie epoki. Jak podają Tadeusiewicz oraz Szaleniec (2015) podczas uczenia sieci neuronowej trzeba wykonać bardzo wiele kroków algorytmów uczenia zanim błąd stanie się akceptowalnie mały. Tymczasem zbiór uczący zawiera zwykle ograniczoną liczbę przypadków uczących, w typowych przypadkach setki lub nawet tysiące razy mniej liczną niż liczba koniecznych kroków algorytmu uczenia. Z tego zestawienia wynika, że zbiór uczący musi być wykorzystywany w procesie uczenia wielokrotnie. Dla zaznaczenia tego faktu wprowadzono do sieci neuronowych pojęcie epoki, rozumiejąc pod tym pojęciem jednorazowe użycie w procesie uczenia wszystkich przypadków uczących zawartych w zbiorze uczącym. Po wykonaniu wszystkich kroków należących do jednej epoki algorytm uczący dokonuje oceny zdolności sieci do generalizacji wyników uczenia przy wykorzystaniu zbioru testowego. Po stwierdzeniu, że zarówno błąd obliczany na zbiorze uczącym, jak i błąd wyznaczony dla zbioru walidacyjnego nadal jeszcze obiecująco maleją – algorytm uczący wykonuje następną epokę. W przeciwnym przypadku proces uczenia zostaje zatrzymany. Zatem parametr „steps_per_epoch” ustawiony na 10 oznacza partie próbek do przeszkolenia. Służy on do określenia, ile partii próbek ma być użytych w jednej epoce. Parametr określa również zakończenie jednej epoki rozpoczęcie drugiej. Parametr „epoch” ustawiony na 30 oznacza, że docelowo model będzie 30 razy powtarzał obliczenia na wszystkich fotografiach ze zbioru. Model wykonał tylko 12 epok po czym się zatrzymał, ponieważ wartość dokładności przestała rosnąć i zatrzymała się na satysfakcjonującym poziomie 95.78%. Oznacza to, że algorytm porównując wyniki przewidziane do stanu rzeczywistego w prawie 96 przypadkach na 100 uzyskał poprawny rezultat.

Kod 4.13

```
h = his.history

h.keys()
```

```
plt.plot(h['loss'], 'go--', c='green')

plt.plot(h['accuracy'], 'go--', c='red')

plt.title('Loss vs Acc')

plt.show()
```

Powyższy kod przedstawia zwizualizowanie każdej epoki oraz wartości funkcji straty i dokładności modelu. Dzięki tym komendom tworzy się wykres.



Rysunek 35 Wykres przedstawiający zmienne funkcji straty oraz dokładności.

Źródło: Opracowanie własne na podstawie danych z www.kaggle.com

Na powyższym rysunku przedstawione zostały dwie zmienne, zieloną linią wskazana została funkcja straty, a czerwoną linią przedstawiona została dokładność wzrastająca z każdą epoką. Na rysunku doskonale widać działanie funkcji EarlyStoppingu, kiedy wartość

„accuracy” zatrzymała się na mniej więcej stałej wartości przez 5 epok – algorytm przestał obliczać dalsze epoki. Ostatnią epoką jest epoka 12.

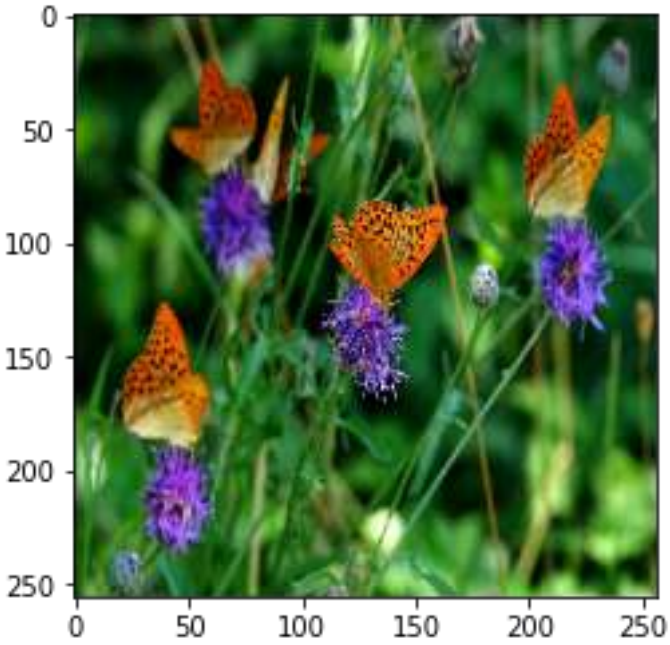
Walidacja modelu polega na wybraniu losowego obrazu ze zbioru testowego i sprawdzenie jak algorytm radzi sobie z jego zakwalifikowaniem. Po podsumowaniu modelu można się spodziewać, że będą się zdarzać pewne błędy, jednak będą one występować niezwykle rzadko. Jedynie około 4 przypadki na 100 zostaną źle sklasyfikowane.

Kod 4.14

```
path = '/content/Recognize animals dataset/animal_dataset_intermediate/test/ea35b2062bf5033ed1584d05fb1d4e9fe777ead218ac104497f5c97faeebb5bb_640.jpg'

img = load_img(path, target_size=(256,256))

imgplot = plt.imshow(img)

plt.show()

i = img_to_array(img)

i = preprocess_input(i)

input_arr = np.array([i])

input_arr.shape

pred = np.argmax(model.predict(input_arr))

prediction = model.predict(input_arr)
```

```
plt.show()

if(pred == 0):

  print('Elephant')

elif(pred == 1):

  print('Butterfly')

elif(pred == 2):

  print('Cow')

elif(pred == 3):

  print('Sheep')

else:

  print('Squirrel')
print(prediction)
```

Powyższy kod w pierwszej linijce ustala ścieżkę do losowego zdjęcia zwierzęcia, następnie ładuje je ustawiając jego rozmiar na 256x256 pikseli. Następne polecenia wyświetlają zdjęcie i za pomocą funkcji „model.predict" klasyfikują zdjęcie, które wcześniej zostało skompresowane do postaci tablicy pikseli w celu umożliwieniu obliczeń. Funkcja „np.argmax" wyciągnie z predykcji najwyższe prawdopodobieństwo, pod którym kryje się wcześniej ustalona w słowniku klasa. Kolejne polecenia to instrukcja warunkowa wypisująca nazwę zwierzęcia, w zależności od wartości parametru „pred". Dla przykładu, jeżeli parametr będzie miał wartość 4 to instrukcja warunkowa wygeneruje napis „Squirrel"



Rysunek 36 Zdjęcie motyli wykorzystane do testowania algorytmu.

Źródło: Opracowanie własne na podstawie danych z www.kaggle.com

Na powyższym rysunku znajduje się oryginalne zdjęcie ze zbioru danych, na którym został wykonany test. W następnej linijce kod zwrócił wypisaną przez instrukcję warunkową nazwę „Butterfly", a w jeszcze następnej linijce program zwrócił listę z prawdopodobieństwami:

```
[[4.2921985e-14 1.00000e+00 1.0431680e-09 4.1667092e-10 1.8407889e-17]]
```

W liście widoczne jest 5 numerów, przypisane są one kolejno do każdej z klas od 0 do 4, które zostały nadane gatunkom w poprzednich krokach. Najwyższy numer przypada dla klasy numer 3, która została nadana motylowi, zatem algorytm poprawnie rozpoznał i wskazał gatunek zwierzęcia.

Autorski kod umożliwia w łatwy sposób rozpoznawanie gatunków zwierząt. Cały mechanizm jest bardzo dobrze zoptymalizowany, ponieważ w przypadku chęci ponownego uruchomienia nie trzeba od początku szkolić modelu. Pomijając najbardziej czasochłonny etap, wystarczy jedynie wczytać wytrenowany już model, a ten będzie mógł pełnić swoją funkcję. Sam model zajmuje pamięć około 5 MB więc jest względnie „lekki". Ta lekkość pozwala na uruchomienie go nawet na bardzo trywialnych maszynach z małą pamięcią operacyjną.

4.4 Możliwości rozwoju aplikacji.

W poprzednim rozdziale opisano skomplikowany etap budowy rozwiązania rozpoznającego gatunki zwierząt. Cała aplikacja ma na celu, za pomocą zainstalowanego przy drodze systemu kamer śledzić występowanie dzikiej zwierzyny oraz wysyłać sygnał w celu ostrzeżenia. Sygnał z takiej maszyny może być wysyłany w różny sposób. Pierwszym sposobem jest zamontowanie w słupkach drogowych sygnału świetlnego, który po otrzymaniu od systemu informacji o zwierzęciu w okolicach jezdni zapalałby się na czerwono – ostrzegając tym samym kierowcę przed niebezpieczeństwem.

Rysunek 37 Słupek drogowy z wbudowanym sygnalizatorem świetlnym.

Źródło: https://tioman.pl/slupki-przy-drodze/

Zdjęcie przedstawia słupek z wbudowanym sygnalizatorem świetlnym w kolorze czerwonym utożsamianym na drodze z ostrzeżeniem. W połączeniu ze znakiem, objaśniającym kierowcy, że w danym miejscu istnieje zagrożenie wtargnięcia dzikiej zwierzyny na jezdnie, system ten mógłby prowadzić do zdjęcia nogi z gazu przez kierowców, a co za tym idzie do ocalenia wielu stworzeń. Kolejny sposób, dzięki któremu system mógłby ostrzec kierowcę przed zagrożeniem jest nieco bardziej skomplikowany. Zamiast informacji wysyłanej do słupka przydrożnego, system wysyłałby informację poprzez sieć do modułu aplikacji nawigacyjnych takich jak GoogleMaps, AppleMaps lub bezpośrednio do samochodu. Aplikacje nawigacyjne z każdym rokiem stają się coraz bardziej powszechne, a co za tym idzie rozbudowane. Systemy w nich zawarte pokazują kierowcy najszybszą drogę, polecają trasy pozwalające ominąć korek więc mógłby zostać dodany do tych aplikacji moduł, który zaczytywałby z pokonywanej trasy informacje o zagrożeniu wtargnięcia zwierzyny na jezdnię. Sposób ten jest bardziej kosztowny, ponieważ wymusza zbudowanie odpowiedniego modułu aplikacji oraz wyposażenie samego systemu w sieć internetową pozwalającą synchronizację oraz wysyłanie sygnałów.

Sam system, oprócz kodu musiałby składać się z procesora, który mógłby w czasie rzeczywistym obliczać dane oraz decydować o tym czy wysłać sygnał. Rynek dysponuje wieloma mikro procesorami, na których taka operacja byłaby możliwa. Jednym z wielu dostępnych rozwiązań jest wyposażenie systemu w pulpit zdalny Raspberry Pi.

Rysunek 38 Mikrokontroler Raspberry Pi.

Źródło: https://anydesk.com/pl/downloads/raspberry-pi

Rysunek przedstawiający Raspberry Pi. Ten mikrokontroler wyposażony jest w kilka wtyczek HDMI, gniazdo sieciowe, USB oraz zależnie od modelu posiada on procesor od 1 do 8 GB RAM, który z pewnością obsłużyłby wszystkie zadania od zbierania danych, analizowania ich oraz wysyłania sygnałów. Kolejnym elementem, potrzebnym do obsłużenia tego procesu jest aparat wraz z kartą pamięci. Pamięci nie potrzeba zbyt wiele, ponieważ aparat mógłby robić jedno zdjęcie na sekundę, wysyłać je do analizy i w przypadku braku potrzeby wysłania sygnału kasować je, aby nie zajmować pamięci. W przypadku dostrzeżeniu przez system zwierzęcia zapisywałby on jego zdjęcie w bazie danych wraz z godziną oraz lokalizacja, tak aby umożliwić później leśnikom oraz organizacjom statystycznym prowadzenie pomiarów. Kolejny sprzęt potrzebny do uruchomienia takiego systemu to bateria. System potrzebuje źródła energii, stałego lub ładowalnego w postaci baterii. Oczywiście najlepszym wyjściem byłoby zastosowanie paneli fotowoltaicznych lub wiatraków prądotwórczych w celu zapewnienia autonomiczności energetycznej. Cały taki system może na pierwszy rzut oka wydać się kosztowny, lecz po przeanalizowaniu, ile kosztuje budowa kilometra drogi,

nie jest to aż tak wielka kwota. Rozwiązanie to mogłoby zredukować liczbę wypadków oraz przyłożyć sporą cegiełkę do aktywnej ochrony natury. Wiele krajów w obecnych czasach decyduje się na bardziej kosztowne, lecz ekologiczne rozwiązania więc automatycznie nasuwa się wniosek o szansie dla wdrożenia takiego systemu. Kolejną mocną stroną jest dostarczenie danych na temat migracji zwierząt dla leśników oraz organizacji zajmujących się badaniem ścieżek migracyjnych. Dzięki tym danym mógłby rozwinąć się sektor badań nad dziką zwierzyną.

Zakończenie

W ramach zrealizowania celu napisano obszerną pracę skupiającą się na wypadkach drogowych z udziałem zwierząt poprzez dogłębną analizę problemu oraz napisanie autorskiego algorytmu, który z wysoką skutecznością jest w stanie rozpoznawać gatunki zwierząt i ostrzegać przed nimi kierowców. Aplikację opracowano korzystając z najnowszych publikacji oraz badań z renomowanych polskich jak i zagranicznych uniwersytetów. Realizacja projektu została oparta o nowoczesną technologię Computer Vision zajmującą się wysokopoziomowym rozumieniem przez komputer obrazów oraz filmów. Po przedstawionych w rozdziale pierwszym faktach można wnioskować, że najczęściej do wypadków dochodzi na drogach krajowych oraz wojewódzkich, które są drogami pozwalającymi rozwijać duże prędkości, a nie posiadają wielu systemów mogących zapobiec kolizji z udziałem zwierzęcia. Do zdarzeń na drogach najczęściej dochodzi nocą, a najbardziej poszkodowaną przez kierowców gromadą są płazy. Potrącone przez samochód zostają również zwierzęta, które w Polsce znajdują się pod ochroną. Z rozdziału pierwszego nasuwa się wniosek, że zwiększenie bezpieczeństwa niesie za sobą obopólne korzyści, ponieważ ludzie jak i zwierzęta są narażeni na utratę zdrowia lub życia. Kolejne rozdziały opisują teorie wdrożenia systemu. System opisany w projekcie został opracowany w sposób optymalizujący jego wszelakie procesy. Wprowadzono nowoczesne rozwiązania takie jak przechowywanie danych w chmurze Dropbox, aby umożliwić do nich dostęp wielu użytkownikom. Kolejnym nowoczesnym oraz optymalnym rozwiązaniem jest zastosowanie kompilatora języka Python, Google Colab. System ten umożliwił przechowywanie całego kodu w chmurze, co pozwala programiście na łączenie się z dowolnego miejsca na Ziemi. Drugi rozdział zawiera obszerną teorię pozwalającą czytelnikowi zrozumieć w jaki sposób komputer odbiera i czyta obrazy. Zostały w nim omówione techniki utraty informacji na zdjęciach, które wykorzystuje się w celu usprawnienia obliczeń. Trzeci rozdział zawiera opisy technik skomplikowanych

kalkulacji analizujących zdjęcia. Przedstawione w rozdziale trzecim autorskie algorytmy są w stanie zaklasyfikować obiekt do jednej z pięciu testowych grup, jakimi są gatunki zwierząt. Sam projekt został napisany w języku Python, który jest jednym z najszybciej rozwijających się języków. Postawiony cel na opracowanie autorskiego oprogramowania do klasyfikacji zwierząt został osiągnięty z 96% skutecznością, oznaczającą, że algorytm jest w stanie poprawnie sklasyfikować 96 na 100 fotografii. Sama aplikacja może zostać opatentowana oraz użyta do ostrzegania kierowców przed dzikimi zwierzętami, chroniąc tym samym zdrowie i życie kierowców, pasażerów jak i zwierzyny. Postęp technologiczny oraz coraz większy dostęp do nowoczesnych rozwiązań otwiera nowe ścieżki, które, póki co wciąż są kosztowne i nie patrzy się na nie przychylnie. Z każdym rokiem jednak cena nowych technologii maleje, a ludzkość nabiera większej inteligencji emocjonalnej oraz idącej za nią empatii. Jako zachowanie empatyczne można rozumieć opiekę oraz troskę nad zwierzętami. Same inwestycje budowy dróg to wielka ingerencja w środowisko naturalne, wiele tras przechodzi przez lasy, parki narodowe lub rezerwaty przyrody, które skupiają ogromny urodzaj gatunków zwierzęcych. W ślad za inwestycjami komunikacyjnymi powinny rozwijać się również inwestycje związane z zapewnieniem bezpieczeństwa na zetknięciu dróg ze szlakami dzikich zwierząt. Autorska propozycja jest przykładem zwiększenia bezpieczeństwa tych miejsc. Dzięki takim technologiom jak sztuczna inteligencja, która nazwana jest: „Elektrycznością XXI wieku” można rozwiązać wiele problemów dzisiejszego świata.

Bibliografia:

1. Bertsekas Dimitri: Conver Optimization Algorithms. Belmont 2015
2. Borowska Sylwia: Śmiertelność zwierząt na drogach w Polsce, 2010, https://zwolnij.wwf.pl/dokumenty/raport.pdf. data dostępu 03.08.2022
3. Borowska Sylwia: Zderzenia drogowe z udziałem dzikich zwierząt, 2009, http://siskom.waw.pl/siskom/zwierzaki-wyniki-ankiety.pdf. data dostępu 03.08.2022
4. Cheng Samuel: Convolutional Neural Networks. Norman, Oklahoma 2017
5. Cholles Francois: dokumentacja biblioteki Keras, 2015, https://keras.io/guides/, data dostępu 03.08.2022
6. Cormen Thomas, Leiserson Charles, Rivest Ronald, Stein Clifford: Introduction to Algorithms. New York 2022
7. Dean Jeff, Hinton Geoffrey, Alphabet Inc. engineers: TensorFlow Library. Mountain View 2015
8. Derat Arden, blog: Applied Deep Learning – Part 4: Convolution Neural Networks, 2017, https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2, data dostępu 03.08.2022
9. Esposito Dino – Wprowadzenie do uczenia maszynowego. Warszawa 2020
10. Fukushima Kunihiko: Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Visual Pattern Recognition. Berlin 1982
11. Ganesh Prakhar, blog: Types of Convolution Kernels: Simplified, 2019, https://towardsdatascience.com/types-of-convolution-kernels-simplified-f040cb307c37, data dostępu 03.08.2022
12. Hubel David, Wiesel Torsten: Brain and Visual Perception: The Story of a 25-Year Collaboration. New York 2005
13. Huotari Jussi, blog: Why Loss and Accuracy Metrics Conflict, 2019, http://www.jussihuotari.com/2018/01/17/why-loss-and-accuracy-metrics-conflict/, Data dostępu 03.08.2022
14. Jain Ankit, Fandango Armando, Kapoor Amita: TensorFlow. Warszawa 2019

15. Karumanchi Narashima: Data Scructure and Algorithmic Thinking with Python Data Structure and Algorithmic Puzzles. Telangana 2016
16. Kirk Matthew: Python w Uczeniu Maszynowym. Warszawa 2018
17. Krizhevsky Alex: OverFeat – ImageNet. New York 2012
18. Krzysztofiak Anna, blog: Wigry, 2005, https://www.wigry.org.pl/kwartalnik/nr18_gady.htm, data dostępu: 03.08.2022
19. Kurama Vihar, Blog: A review of popular Deep Learning architectures: ResNet, InceptionV3 and SqueezeNet, https://blog.paperspace.com/popular-deep-learning-architectures-resnet-inceptionv3-squeezenet/, data dostępu 03.08.2022
20. Kustusch Karol, Wuczyński Andrzej, Raport: Zwierzęta na drodze, 2017, https://zwierzetanadrodze.pl/files/Zwierzeta_na_Drodze_-_Raport_2017_1.pdf, data dostępu 03.08.2022
21. Kustusch Karol, Wuczyński Andrzej, Blog: Zwierzęta na drodze, 2022, https://zwierzetanadrodze.pl/menus/home, data dostępu 03.08.2022
22. Le Cun Yann, Dehaene Stanislas, Girardon Jacques: La Plus Belle Histoire de l'intelligence: Des origines aux neurones artificiels: vers une nouvelle étape de l'évolution. Paris 2018
23. Mamczur Mirosław: blog: Uczenie maszynowe: Jak działają konwolucyjne sieci neuronowe, 2021, https://miroslawmamczur.pl/jak-dzialaja-konwolucyjne-sieci-neuronowe-cnn/, Data dostępu 03.08.2022
24. Moolayil Jojo: Learn Keras for Deep Neural Networks. Vancouver 2019
25. Ng Andrew, blog: Cursera – Deep Learning Specialization, 2012, https://www.coursera.org/specializations/deep-learning, data dostępu: 01.08.2022
26. Pelic Marcin - Metody Sztucznej Inteligencji w Sterowaniu. Poznań 2011
27. Polska Policja: Statystyki wypadków drogowych, 2022, https://statystyka.policja.pl/st/ruch-drogowy/76562,wypadki-drogowe-raporty-roczne.html, data dostępu 03.08.2022
28. Ramsundar Bharath, Bosagh Zadeh Reza: Głębokie uczenie z TensorFlow. Warszawa 2019

29. Raspberry Pi Foundation, blog: Research of publications, 2008, https://www.raspberrypi.org/research/publications/, data dostępu 03.08.2022
30. Schalk Andreas, Schalk Rainer, biuro patentowe, 2015: https://patents.justia.com/patent/20150070164, data dostępu 03.08.2022
31. Seb, blog: An introduction to neural network loss functions, 2021, https://programmathically.com/an-introduction-to-neural-network-loss-functions/, data dostępu 03.08.2022
32. Shafkat Irhum, blog: Intuitively Understanding Convolutions for Deep Learning, 2018, https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee, data dostępu 03.08.2022
33. Tadeusiewicz Ryszard, Szaleniec Maciej – Leksykon Sieci Neuronowych. Wrocław 2015
34. Trothe Christian, Herzog Sven, publikacja naukowa: https://tu-dresden.de/bu/umwelt/forst/wb/wildoekologie/ressourcen/dateien/publikationen/Monitoring-der-elektronischen-Wildwarnanlage?lang=en, 2014, data dostępu 03.08.2022
35. Venkatesan Ragav, Li Baoxin – Convolutional Neural Networks in Visual Computing. Londyn 2017
36. Wheeler Tim, Kochenderfer Mykel – Algorithms for Optimization. London 2019
37. Zhang Aston, Li Mu, blog: Dive Into Deep Learning, 2019, https://d2l.ai/chapter_convolutional-neural-networks/padding-and-strides.html, data dostępu 03.08.2022

Spis Rysunków: