Uniwersytet Jagielloński w Krakowie

Wydział Fizyki, Astronomii i Informatyki Stosowanej

Wojciech Ozimek

Nr albumu: 1124802

Wykorzystanie sztucznych sieci neuronowych do analizy obrazów na przykładzie kostki do gry

Praca licencjacka
na kierunku Informatyka

Praca wykonana pod kierunkiem
prof. dr hab. Piotra Białasa
FAIS UJ

Kraków 2018

Spis treści

1. Cel pracy

Celem pracy jest zastosowanie sieci neuronowych do rozpoznawania obrazów. Zadanie polega na rozpoznaniu ilość oczek wyrzuconych na kostce do gry. Sieć powinna rozpoznawać kostki o dowolnym zestawieniu kolorystycznym ścian i oczek oraz działać na rzeczywistym obrazie przesyłanym z kamery.
Dodatkowym aspektem poruszonym w pracy jest porównanie modeli w zależności od architektury sieci oraz zbiorów danych.

2. Wstęp

Sieci neuronowe składają się ze sztucznych neuronów, które są uproszczonymi modelami ich biologicznych odpowiedników, a pierwsze prace nad nimi zaczęły sie już w latach 50. XX wieku.

2.1 Biologiczny neuron

Neuron to komórka nerwowa zdolna do przewodzenia i przetwarzania sygnału elektrycznego, w którym zawarta jest informacja. Jest on podstawowym elementem układu nerwowego wszystkich zwierząt. Każdy neuron składa się z ciała komórki (soma, neurocyt) otaczającego jądro komórkowe, neurytu (akson) odpowiedzialnego za przekazywanie informacji z ciała komórki do kolejnych neuronów oraz dendrytów służących do odbierania sygnałów i przesyłaniu ich do ciała komórkowego [1, 4]. Impuls elektryczny między neuronami przekazywany jest w synapsie, miejscu komunikacji neuronu z następnym. Synapsa składa się z części presynaptycznej (aksonu) i postsynaptycznej (dendrytu). Neuron przewodzi sygnał tylko w sytuacji, kiedy suma potencjałów wejściowych od innych neuronów na jego dendrytach przekroczy określony poziom. W przeciwnym wypadku neuron nie przewodzi sygnału. Co więcej, zwiększenie potencjału na wejściach nie powoduje wzmocnienia potencjału na wyjściu neuronu.
Neurony połączone i działające w ten sposób tworzą sieci neuronowe. Najdoskonalszą znaną siecią neuronową jest mózg człowieka. Przeciętnie posiada on około 100 miliardów neuronów, każdy z nich połączony jest z około 10 tysiącami innych neuronów przez połączenia synaptyczne. Liczba połączeń synaptycznych szacowana jest na około 10^{15}.

2.2 Sztuczny neuron

Podstawowym modelem neuronu jest tzw. neuron McCullocha-Pittsa, nazywany również neuronem binarnym. Jest to prosta koncepcja zakładająca, że każdy neuron posiada wiele wejść, z których każde ma przypisaną wagę w postaci liczby rzeczywistej oraz jedno wyjście. Wyjściem neuronu jest wartość funkcji dla argumentu będącego sumą wszystkich wag pomnożonych przez odpowiednie wartości wejściowe, określana jako funkcja aktywacji, wyjaśniona w późniejszych rozdziałach. Wyjście danego neuronu połączone jest z wejściami innych neuronów, tak jak ma to miejsce w przypadku biologicznego neuronu [1].
Tak zbudowany, sztuczny model neuronu jest podstawowym składnikiem sztucznych sieci neuronowych z racji prostoty działania i łatwości implementacji.

Wykorzystanie pojedynczego neuronu nie pozwala na rozwiązywanie skomplikowanych zadań. Przykładowo próba realizacji prostych funkcji logicznych ANR, OR, NOT i XOR okazuje się możliwa jedynie dla trzech pierwszych podanych funkcji [14]. Problem wiąże się z brakiem możliwości uzyskania poprawnych rezultatów dla zbiorów które nie są liniowo separowalne, czego przykładem jest właśnie funkcja XOR. W takich przypadkach konieczne jest użycie większej ilości neuronów. Tworzenie rozbudowanych struktur z pojedynczych neuronów określanych mianem sieci neuronowych, znacząco zwiększa możliwości adaptacyjne dla poszczególnych problemów, niejednokrotnie zaskakując osiąganymi wynikami.

2.3 Historia

Rozpoczęcie prac nad sztucznymi neuronami można datować na rok 1943 kiedy to Warren McCulloch i Walter Pitts przedstawili wspomniany już wcześniej model [4]. Pierwsze sieci neuronowe używane były do operacji bitowych oraz przewidywania kolejnych wystąpień bitów w ciągu. Mimo licznych prób zastosowania ich do realnych problemów nie zyskały one popularności. W początkowym okresie rozwoju sieci neuronowych przyjmowano także wiele błędnych założeń, które wraz z ograniczonymi możliwościami obliczeniowymi komputerów skutecznie zniechęcały naukowców do prac nad tym zagadnieniem.

3. Technologie

3.1 Język programowania i środowisko

Python

Język programowania Python obecnie jest bardzo popularnym narzędziem wykorzystywanym w pracy naukowej. Jest to spowodowane bardzo czytelną i zwięzłą składnią, która w pełni pozwala skupić się na danym problemie. Python jest on także domyślnym językiem używany w bibliotekach wykorzystywanych do tworzenia modeli sieci neuronowych.

Jupyter Notebook

Aplikacja Jupyter Notebook pozwala uruchamiać w przeglądarce pliki, nazywane notebookami, które składają się z wielu bloków. W blokach może znajdować się wykonywalny kod programu lub jego fragment, a w pozostałych można prezentować m.in. teksty, wykresy bądź tabele, co zdecydowanie ułatwia pracę.
W tej pracy wykorzystywana jest to uruchamiania programów w języku Python.

3.2 Biblioteki

OpenCV

Biblioteka funkcji do obróbki obrazów, najczęsciej wykorzystywana w językach C++ oraz Python. W projekcie została użyta do uzyskiwania zbiorów obrazów kości do gry ze zdjęć wykonanych kamerą.

TensorFlow

Biblioteka do uczenia maszynowego oraz tworzenia sieci neuronowych. Jej ogromną zaletą jest implementacja w języku C++ umożliwiająca wykorzystywanie procesorów oraz kart graficznych. Z uwagi na charakter wykonywanych obliczeń praca przy użyciu kart graficznych jest kilkukrotnie szybsza niż na procesorze, co znacząco przyśpiesza proces uczenia. Biblioteka najlepiej wspiera języki programowania do C++ oraz Python.

Keras

Biblioteka do tworzenia modeli sieci neuronowych, wykorzystująca bardziej niskopoziomowe biblioteki jak TensorFlow, Theano lub CNTK. Zaletami Kerasa są zarówno przystępny interfejs pozwalający w krótkim czasie stworzyć model sieci oraz możliwość tworzenia zaawansowanych

modeli przy średnim pogorszeniu czasu uczenia się sieci o 3-4% w stosunku do bibliotek, na których bazuje.
W sytuacji kiedy interfejs oraz dokumentacja do TensorFlow mogą być dla początkującej osoby niezrozumiałe, jest to świetna alternatywa do wdrożenia się w to zagadnienie.

Numpy
Moduł języka Python umożliwiający wykonywanie zaawansowanych operacji na macierzach oraz wektorach, wspierający liczne funkcje matematyczne. Jest bardzo rozpowszechniony i wykorzystywany w wielu, głównie naukowych zastosowaniach. Numpy wprowadza własne typy danych oraz funkcje niedostępne w standardowej instalacji Pythona.

Matplotlib
Narzędzie do tworzenia wykresów dla języka Python oraz modułu Numpy. Z uwagi na bardzo duże możliwości jest bardzo popularny, pozostając jednocześnie prostym w użyciu. Zawiera moduł pyplot, który w założeniu ma maksymalnie przypominać interfejs w programie MATLAB.

LaTeX
Oprogramowanie do składu tekstu, nie będące edytorem tekstowy typu WYSIWYG. LaTeX bazuje na TeX, który jest systemem składu drukarskiego do prezentacji w formie graficznej. Ogromną zaletą jest możliwość tworzenia w tekście zaawansowanych wzorów matematycznych. Tekst niniejszej pracy napisany został przy pomocy tego narzędzia.

3.3 Technologie wspomagające

NVIDIA CUDA
Równoległa architektura obliczeniowa firmy NVIDIA pozwala na wielokrotne przyśpieszneie obliczeń podczas uczenia się sieci neuronowych. Dzięki bibliotekom takim jak TensorFlow lub Keras, które wspierają obliczenia na kartach graficznych czas precesu uczenia drastycznie maleje. Podczas tej pracy wykorzystana została karta NVIDIA TESLA K80 12GB GPU dostępna na Amazon AWS oraz Google Compute Engine.

Amazon AWS EC2
Platforma z wirtualnymi maszynami zwanymi instancjami, które można dostosować zależnie od potrzeb klienta. W tej pracy zostały wykorzystane instancje zoptymalizowane do obliczeń na kartach graficznych i uczenia maszynowego. Dzięki jej wykorzystaniu, czas uczenia sieci zmniejszył się średnio 6-10 krotnie w stosunku do pracy na komputerze wykorzystującym procesor.

4. Słownik pojęć

4.1 Zbiór danych

Do realizacji pracy konieczny było stworzenie zbioru danych składającego się z obrazów z kośćmi do gry. Zdjęcia zostały zrobione kamerą o rozdzielczości 1600x1200 pikseli, a następnie zmniejszone w celu zaoszczędzenia pamięci i osiągnięcia żądanego rozmiaru, który ma kluczowe znaczenie w przypadku zastosowań sieci neuronowych. Oprócz zmniejszenia obrazy były również obracane, przekrzywiane(deformowane? wypaczane? *warping*) oraz kadrowane. Tak powstałe, liczne zbiory były wykorzystywane w trybie RGB oraz w odcieniach skali szarości, co pozwoliło na oszczędzenie pamięci przez zmniejszenie liczby kanałów do jednego.
W pracy używano zbiorów z obrazami o rozmiarach 64x64 oraz 106x79 pikseli. Każde zdjęcie w zbiorze miało przypisaną wartość liczbową informującą o faktycznej ilości oczek wyrzuconych na przedstawionej kostce. Wartość ta zwana jest odpowiedzią i jest wykorzystywana w procesie uczenia sieci jako docelowa informacja, którą ma zwrócić sieć po weryfikacji danego obrazu.

Zbiór treningowy
Część zbioru danych, która wykorzystywana jest w procesie uczenia sieci określana jest jako zbiór treningowy lub zbioru uczący. Jego liczebność to zazwyczaj 60-80% całego zbioru danych. Praktycznie zawsze dane znajdujące się w tym zbiorze, przed rozpoczęciem uczenia, poddawane są losowej permutacji.

Zbiór testowy
Do oceny zdolności sieci neuronowej do rozpoznawania danych służy zbiór testowy, zwany też walidacyjnym. Celem rozdzielenia tego zbioru od danych testowych jest weryfikacja sieci na danych, które nie zostały przez nią przetworzone podczas treningu.

4.2 Budowa sieci neuronowej

Sieć neuronowa
Sieć neuronowa *(ang. ANN Artificial Neural Network)* to struktura matematyczna, składająca się z neuronów połączonych w warstwy, mająca odzwierciedlać działanie biologicznych sieci neuronowych, a w szczególności mózgu [10, 16]. Sieci mają szerokie zastosowanie w bardzo wielu dziedzinach, co przejawia się ich popularyzacją w ostatnich latach. Wymagają kosztownego obliczeniowo i czasowo procesu uczenia, podczas którego dostosowują się do danego problemu.

Użycie wytrenowanej sieci, nie wymaga powtarzania uczenia, co pozwala na jej natychmiastowe wykorzystanie.
W dziedzinie rozpoznawania obrazów sieci są w stanie zidentyfikować elementy na obrazach, bez wcześniejszej znajomości lub wiedzy na temat przedmiotu który mają rozpoznać.

Neuron

Neuron jest najmniejszym elementem sieci neuronowej, posiadającym wiele wejść i jedno wyjście [3, 4, 5]. Z neuronów w poprzednich warstwach, na każde z wejść docieraja sygnały, które mają przypisaną wagę. W neuronie obliczana jest suma ważona wejść, od której odejmowana jest wartość progowa. Jeśli suma przekroczy wartość progową, neuron jest uaktywniany, a suma przekazywana jest jako argument funkcji aktywacji neuronu na jego wyjście.
Poniższy wzór 4.1 przedstawia sumę ważoną wejść neuronu wraz z dodatkową wagą.

$$f(x_i) = f\Big(\sum_i w_i x_i + b\Big) \tag{4.1}$$

Wzór 4.1: Suma ważona w neuronie wraz z dodatkową wagą b

Wagi neuronu

Jak opisano już wcześniej, neurony połączone są między sobą dzięki licznym wejściom. Każde takie połączenie ma przypisaną wagę, która zmienia się podczas treningu, dostosowując sieć neuronową do danych treningowych. Efektem tego jest osiąganie coraz lepszych wyników w miarę trwania procesu uczenia.

Bias

Bias, użyty już we wzorze 4.1 na sumę wag w neuronie jest dodatkową wagą wejściową, umożliwiająca jego lepsze dopasowanie neuronu do danych treningowych [6].

Warstwa

Neurony w sieci zorganizowana są w warstwach. Komunikacja odbywa się tylko między kolejnymi warstwami, neurony w danej warstwie nie są ze sobą połączone [1, 7]. Istnieje wiele rodzajów warstw, a sygnał przechodzący przez całą sieć zaczyna się w tzw. warstwie wejściowej oraz kończy w tzw. warstwie wyjściowej.

Funkcja kosztu

Do porównywania wyników otrzymywanych przez sieć neuronową wykorzystywana jest funkcja kosztu, funkcja oceniajaca lub funkcja błędu. Jest ona niezbędna do prawidłowego przeprowadzenie procesu uczenia. Funkcja dostarcza informacje o różnicy między obecnym stanem sieci, a optymalnym rozwiązaniem dla danych treningowych. Algorytm uczenia analizuje wartość funkcji kosztu w kolejnych krokach w celu jej zminimalizowania.

Najczęściej wykorzystywaną funkcją kosztu jest błąd średniokwadratowy 4.2. W tej pracy z uwagi na posiadanie 6 możliwych do uzyskania wyników na wyjściu sieci, zastosowano wielowymiarową entropię krzyżową (*ang. Categorical cross-entropy*) 4.3, która jest preferowana ponieważ uwypukla aż tak bardzo nieprawidłowych wartości [32]. Jest również sugerowana przez bibliotekę Keras oraz wykorzystywana w wielu przykładowych modelach sieci neuronowych.

$$J(\eta) = \frac{1}{2}\sum_{i}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 \tag{4.2}$$

Wzór 4.2: Błąd średniokwadratowy

$$L_i = -\sum_{j} t_{i,j} log(p_{i,j}) \tag{4.3}$$

gdzie: p - predykcje, t - cele, i - wartość, j - klasa

Wzór 4.3: Wielowymiarowa entropia krzyżowa

Współczynnik uczenia

Aby dostosować stopień w jakim sieć neuronowa będzie dostosowywać się do danych wykorzystujemy współczynnik lub wskaźnik uczenia się *(ang. lerning rate)*. Często jest uzależniony od wyboru rodzaju optymalizatora. Wybranie zbyt małego współczynnika wydłuży proces uczenia, a w skrajnych przypadkach sieć nie zdąży dotrzeć do minimum. Jeśli zostanie wybrany za duży współczynnik, może spowodować problemy ze znalezieniem optymalnego rozwiązania. W niektórych algorytmach współczynnik uczenia zmniejsza się w czasie, by lepiej dostosować się do danych i zniwelować oba wspomniane problemy.

4.3 Propagacja wsteczna

Propagacja wsteczna lub wsteczna propagacja błędów *(ang. Backpropagation)* jest algorytmem uczenia sieci neuronowych [2, 8]. Służy do wyliczenia gradientu funkcji kosztu, który informuje o szybkości spadku wartości tej funkcji w danym kierunku z uwzględnieniem wag neuronów oraz biasu. Obliczenie gradientu w sieci propagowane jest od warstwy wyjściowej do wejściowe, czemu algorytm zawdzięcza swoją nazwę.

4.4 Konwolucyjna sieć neuronowa

Przy wykorzystywaniu sieci neuronowych do operowania na zdjęciach, pojawia się problem dużą ilością parametrów odpowiadających wartościom każdego z pikseli. W celu ich zmniejszenia, stosuje się konwolucyjne sieci neuronowe *(ang. CNN - Convolutuinal Neural Network)*. Do

kolejnych warstw zamiast przekazywania informacje o wszystkich pikselach znajdujących się na obrazach, sieć analizuje obraz przy użyciu filtrów konwolucyjnych i przesyła dalej informacje o zaobserwowanych cechach.
Obecnie ten typ sieci odnosi największe osiągnięcia w dziedzinie rozpoznawania obrazów, w wielu przypadkach dorównując lub nawet pokonując ludzkie wyniki.

Konwolucja
Wymieniona wyżej konwolucja, inaczej splot polega na złożeniu dwóch funkcji. W przypadku obrazów jedna z tych funkcji to obraz, który ma rozmiary większe niż druga funkcja określana mianem filtra konwolucyjnego. Zastosowanie splotu, w zależności od przypadku, pozwala na rozmycie, wyostrzanie lub wydobycie głębi z danego obrazu [31]. Poniższy wzór 4.4 przedstawia sposób obliczenia splotu dwóch funkcji.

$$h[m,n] = (f * g)[m,n] = \sum_{j}^{m} \sum_{k}^{n} f[j,k] * g[m-j, n-k] \tag{4.4}$$

Wzór 4.4: Splot funkcji. Funkcja f to dwuwymiarowa macierz z pikselami obrazu. Funkcja g to filtr. Funkcja h to nowootrzymany obraz.

4.5 Warstwy sieci neuronowej

W sieciach neuronowych wyróżniamy kilka rodzajów warstw, które poza wejściową i wyjściową, można dopierać zależnie od sposobu rozwiązania danego problemu i typu posidanych danych.

Wejściowa
W sytuacji gdy danymi jest zbiór obrazów, warstwa wejściowa ma rozmiar identyczny z wymiarami obrazu. Pierwsza warstwa charakteryzuje się brakiem wejść oraz biasu.

Wyjściowa
Rozmiar warstwy wyjściowej odpowiada ilości klas do jakiej dane były klasyfikowne. W tej pracy, gdzie czekiwanym wyjściem była liczba oczek możliwych do wyrzucenia na kostce, odpowiada to 6 klasom. Wyjściem takiej sieci jest więc wektor o wymiarach 6x1.

Konwolucyjna
Warstwa konwolucyjna służy do przetworzenia danych z poprzedniej warstwy przy użyciu filtrów konwolucyjnych [1]. Filtry mają określone wymiary i służą do znajdowania cech na obrazach lub ich fragmentach. Zastosowanie wielu warstw konwolucyjnych umożliwia filtrom analizowanie bardziej złożonych zależności na obrazach.

W pełni połączona

Najbardziej rozpowszechnionymi typami warstw są w pełni połączone *(ang. Fully Connected, Dense)*. Każdy neuron łączy się ze wszystkimi neuronami następnej warstwy. Skutkuje to dużą ilością potrzebnych do wykonania obliczeń, choć są stosunkowo proste do zaapikowania. W sieciach konwolucyjnych umieszczane są po lub między warstwami konwolucyjnymi i służą do powiązania nieliniowych kombinacji, które zostały wygenerowane przez warstwy konwolucyjne oraz ich sklasyfikowania.

Flatten

Warstwa spłaszczająca *(ang. Flatten)* stosowana jest w celu połączenia warstw konwolucyjnych z warstwami w pełni połączonymi. Realizowane jest to poprzez przekształcenie warstwy wejściowej do jednowymiarowego wektora, który następnie służy za wejście do kolejnych warstw.

Odrzucająca

Użycie warstwy odrzucającej *(ang. Dropout)* jest jednym z najlepszych sposobów na poradzenie sobie ze zjawiskiem przetrenowania, opisanym na końcu tego rozdziału [26]. Warstwa odrzucająca nie wykorzystuje wyjść pewnych neuronów, co zapobiega rozległym zależnościom między neuronami. W warstwie tej określamy prawdopodobieństwo z jakim neurony zostaną zachowane. Najczęściej stosuje się ją po warstwach w pełni połączonych, zarówno przy przechodzeniu w przód jak i tył [25].

Pooling

Warstwa ta wykorzystywana jest do zmniejszenia rozmiaru pamięci oraz ilości obliczeń w sieci neuronowej. Operacja polega na wybraniu jednego piksela z danego obszaru i przekazaniu go do następnej warstwy. Najczęściej wykorzystywaną warstwą poolingową jest MaxPooling, wybierający piksel o największej wartości. Pooling jest krytykowany za brak zachowywania informacji o położeniu piksela przekazanego na wyjście warstwy, co może objawiać się błędnymi interpretacjami danych przez sieci.

4.6 Funkcje aktywacji

Jeśli wartość progowa danego neuronu zostanie przekroczona, to suma wag zostaje przekazana jako argument do funkcji aktywacji, a obliczona wartości staje się wyjściem neuronu. Wybór funkcji jest jednym z kluczowych parametrów danej sieci, ponieważ niewłaściwy wybór może prowadzić do problemów podczas uczenia sieci.

Sigmoid

Sigmoid popularną funkcją aktywacji 4.5. Problemem z nią związanym jest ryzyko zaniknięcia gradientu, co może prowadzić do problemu tzw. umierającego neuronu [12]. Wystepuje

ono, gdy dla danej funkcji aktywacji, gradient staję się bardzo mały, co jest równoznaczne z zaprzestaniem procesu uczenia. W sigmoid gradient może zanikać obustronnie.

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{4.5}$$

Wzór 4.5: Funkcja sigmoidalna

Tangens hiperboliczny

Tangens hiperboliczny lub tanh 4.6 jest w istocie przekształconą funkcją simgodalną [12, 13]. Wykorzystanie jej powoduje większe wahania gradientu.

$$f(x) = tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})} = \frac{2}{1 + e^{-2x}} - 1 = 2sigmoid(2x) - 1 \tag{4.6}$$

Wzór 4.6: Tangens hiperboliczny

ReLU

ReLU *(and. Rectified linear unit)* 4.7 jest najpopularniejszą funkcją aktywacji wykorzystywaną w sieciach neuronowych [3, 17]. Zasługą tego jest szybki czas uczenia sieci bez znaczącego kosztu w postaci generalizacji dokładności. Problem z zanikającym gradientem jest mniejszy niż w przypadku funkcji sigmoidalnej, ponieważ występuje on tylko z jednej strony.

$$f(x) = max(0, x) \tag{4.7}$$

Wzór 4.7: ReLU - Rectified linear unit

LeakyReLU

LeakyReLU jest ulepszeniem funkcji ReLU [3] dzięki zastosowaniu niewielkiego gradientu w sytuacji, dla której ReLU jest nieaktywne. Zmiana ta pozwala na uniknięcie problemu znikającego gradientu. Jej wzór 4.8 przedstawiony jest poniżej.

$$f(x) = \begin{cases} x & \text{if } x \geqslant 0 \\ 0.01x & \text{if } x < 0 \end{cases} \tag{4.8}$$

Wzór 4.8: LeakyReLU

Softmax

Softmax jest funkcją obliczającą rozkład prawdopodobieństwa wystąpienia danego zdarzenia ze wszystkich możliwych. Wykorzystywana jest w zadagnieniach wymagających przyporządkowania elementu do jednej z wielu do wielu klas. Wzór funkcji 4.9 zaprezentowany jest poniżej.

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=0}^{k}(e^{x_j}} \tag{4.9}$$

Wzór 4.9: Funkcja softmax

4.7 Optymalizator

Optymalizator to określenie algorytmu optymalizacyjnego wykorzystywanego do obliczania wag neuronów i biasu w sieci neuronowej. Posiada kluczowe znaczenie podczas procesu uczenia sieci, zarówno w kwestii czasu oraz skuteczności. Z tego powodu to zagadnienie jest obiektem wielu obecnych badań i rozwoju sieci neuronowych [9].

Metoda największego spadku

Metoda największego spadku *(ang. Gradient Descent)* jest podstawowym algorytmem służacym do zmiany wartości wag oraz biasu podczas procesu uczenia sieci 4.10. Wadą tej metody jest przeprowadzanie jednorazowej aktualizacji po wyliczeniu gradientu dla całego zestawu danych. Spowalnia to uczenie i może powodować problem z ilością zajmowanego miejsca w pamięci. Największą wadą jest możliwość doprowadzenia do stagnacji w jednym z lokalnych minimów funkcji.

$$\theta = \theta - \eta * \nabla J(\theta) \tag{4.10}$$

Wzór 4.10: Metoda największego spadku

Stochastic Gradient Descent

Stochastic Gradient Descent *(skrót. GDA)* jest rozwinięciem metody gradientu prostego, bardzo często wykorzystywana w praktyce *Opis SGD* [28]. Ulepszenie polega na obliczaniu gradientu dla jednego lub niewielkiej ilości przykładów treningowych. Najczęściej korzysta się z więcej niż jednego przykładu, co zapewnia lepszą stabilność oraz wykorzystuje zrównoleglenie obliczeń. SGD zapewnia większą rozbieżność niż metoda gradientu prostego, co umożliwia znajdowanie nowych lokalnych minimów, ale wiąże się z koniecznością zastosowania mniejszego współczynnika uczenia.

$$\theta = \theta - \eta * \nabla J(\theta; x_i; y_i) \tag{4.11}$$

RMSprop

RMSprop umożliwia obliczanie gradientu dla każdego parametru z osobna i zapobiega zmniejszaniu się współczynnika uczenia *Opis RMSprop* [29, 28]. Algorytm dostosowuje współczynnik uczenia dla każdej wagi, bazując na wielkości jej gradientu.

Adam

Adam to skrót od angielskiej nazwy *Adaptive Moment Estimation* i jest rozwinięciem metody Stochastic Gradient Descent *Opis metody Adam* [30, 28]. Metoda ta pozwala na obliczanie z osobna gradientu dla każdego parametru oraz każdej zmiany momentum. Zapobiega dodatkowo zmniejszającemu się wskaźnikowi uczenia, a co najważniejsze jest bardzo szybka i pozwala na sprawne uczenie się sieci. W tej metodzie oblicza się dwa momenty m oraz v.

$$\begin{aligned} \hat{m_t} &= \frac{m_t}{1-\beta_1^t}, \\ \hat{v_t} &= \frac{v_t}{1-\beta_2^t}, \end{aligned} \tag{4.12}$$

Obliczone momenty podstawiane są do wzoru

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v_t}} + \epsilon}\hat{m_t} \tag{4.13}$$

gdzie najczęściej $\beta^t{}_1 = 0.9$ oraz $\beta^t{}_2 = 0.99$ a $\epsilon = 10^{-8}$

4.8 Procesy

Uczenie

Proces uczenia bądź treningu sieci służy zmianie wartości wag, najczęściej zainicjowanych pseudolosowymi wartościami oraz biasu. Uczenie sieci neuronowej jest bardzo kosztowne obliczeniowo, co wręcz uniemożliwiało trenowanie modeli w przeszłości, a obecnie jest jednym z powodów dużego zainteresowania rozwojem technologicznym kart graficznych. Operacje dodawania oraz mnożenia wektorów i macierzy wykonywane są miliony razy, mogą być przyśpieszone dzięki możliwościom zrównoleglenia obliczeń.

Epoka

Proces uczenia sieci podzielony jest na epoki. Każda epoka odpowiada przejściu wszystkich elementów z treningowego zbioru danych przez sieć. Ilość epok, podczas których sieć będzie się uczyć ustala się na co najmniej kilkanaście. W przypadku większych zbiorów danych lub większych modeli ilość epok jest zwiększana.
Często spotykaną praktyką w wielu pracach naukowych jest przedstawianie wyników dla sieci po 100 epokach treningu.

Testowanie

Model sieci neuronowej poddawany ocenie, dzięki której można określić, w jakim stopniu prawidłowo rozpoznaje obrazy. W przypadku wytrenowanych modeli istotne jest, aby zbiór służący

do testowania nie był wcześniej użyty do treningu sieci. Nauczony model powinien być w stanie rozpoznawać nowe, nieużyte podczas procesu uczenia dane i poprawnie je klasyfikować.

Predykcja

Wartości zwrócone przez sieć po umieszczeniu w niej określonych danych są określane mianem predykcji. Pozwala to na wykorzystanie nauczonego modelu w praktycznym zastosowaniu.

Przetrenowanie

Zjawisko przetrenowania *(ang. Overfitting)*, wspomniane przy opisywaniu warstwy odrzucającej, jest jednym z największych problemów podczas pracy z sieciami neuronowymi. Podczas trenowania sieci, możemy obserwować oczekiwany wzrost dokładności i spadek wartości funkcji kosztu. Przetrenowanie wystepuje kiedy dla testowych danych sprawność sieci drastycznie spada. Istnieje duże ryzyko, że autor sieci nie będzie świadomy występowania tego problemu, który w ostateczności może doprowadzić do konieczności tworzenia sieci od nowa. sieci [26]. Proces ten polega na nie wykorzystywaniu wyjść pewnych neuronów, zarówno w przypadku przechodzenia w przód oraz w tył [25]. Stosuje się ją po warstwach w pełni połączonych, w celu zapobiegania rozległym zależnościom między neuronami. W warstwie tej określone jest prawdopodobieństwo *p* z jakim neuron zostanie zachowany w warstwie oraz *p - 1* z jakim zostanie odrzucony. Najczęstsza wartość jest z zakresu 0,5-0,8.

4.9 Inne

ILSVRC

ImageNet Large Scale Visual Recognition Competition [21] to coroczny konkurs organizowany od 2010 roku, w którym naukowcy walczą o najlepszy wynik w dziedzinie rozpoznawania obrazów przez skonstruowane przez siebie algorytmy. Sieć AlexNet, na której bazowało kilka stworzonych sieci neuronowych, została stworzona na potrzeby tego konkursu w 2012 roku.

MNIST

Baza danych MNIST [19] to zbiór 60000 treningowych i 10000 testowych czarno-białych obrazów w rozmiarze 28x28x1, zawierających ręcznie napisane cyfry 0-9. Jest jednym z najpopularniejszych zbiorów służących do rozpoczęcia nauki sztucznych sieci neuronowych i uczenia maszynowego.

CIFAR-10

Zbiór CIFAR-10 [20] składa się z 50000 treningowych i 10000 testowych kolorowych obrazów w rozmiarze 32x32x2. Jest podzielony na 10 klas: samolot, samochód, ptak, kot, jeleń, pies, żaba, koń, statek, ciężarówka; gdzie każdej z nich przypada po 6000 obrazów. Jest to podobnie jak MNIST jeden z najbardziej popularnych zbiorów do uczenia maszynowego i sieci neuronowych.

Tematy do uwzględnienia: minibatch, momentum, one-hot encoding,

5. Tworzenie zbiorów danych

Priorytetem było stworzenie zestawu danych umożliwiającego rozwój różnych modeli sieci neuronowych bez konieczności każdorazowego ingerowania w zbiór lub dopasowywania go specjalnie do konkretnej sieci.
Wszystkie zbiory dostępne są pod linkami w części Dodatek A [8]

5.1 Zbiór obrazów kwadratowych

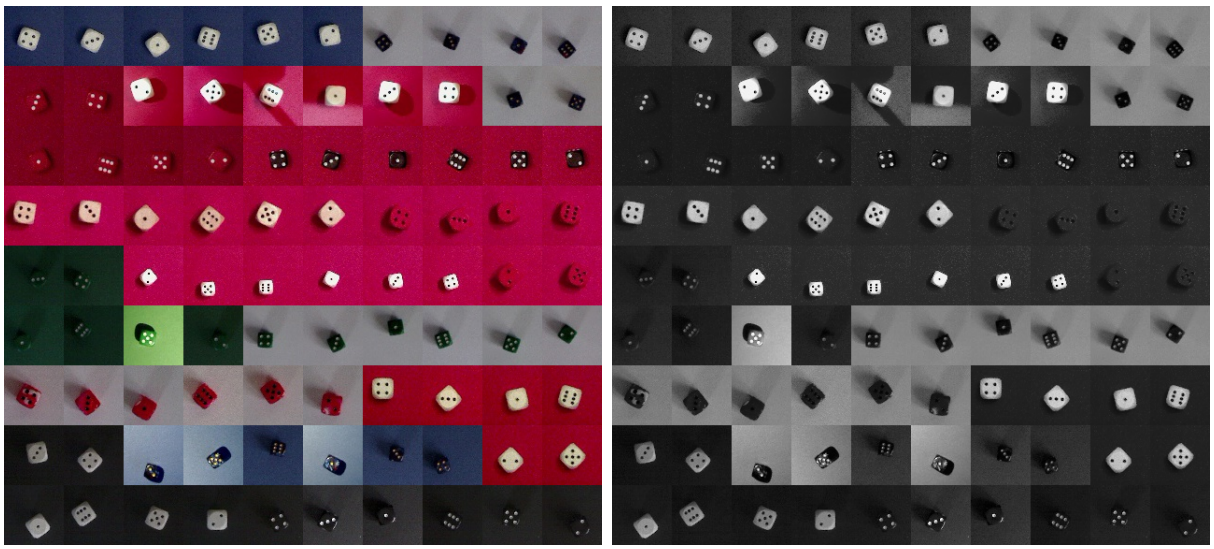
Pierwszym założeniem było zrobienie określonej ilości zdjęć kości położonej na środku obrazu otrzymywanego z kamery, na obszarze kwadratu o boku około 3 krotnie większym niż długość ściany samej kości. Miało to umożliwić kadrowanie obrazów do niewielkich rozmiarów z kośćmi położonymi w różnych jego miejscach.
Kolejną kwestią była konieczność przystosowania sieci do rozpoznawania kości o różnych kolorach ścian, oczek i samego tła. Ponieważ docelowo sieć miała operować na rzeczywistym obrazie z kamery, ważne było także zapewnienie poprawnego działania w przypadku niewielkich zniekształceń obrazu bądź szumów.
Powyższe wymagania narzuciły konkretną formę zestawów danych. Każdy zestaw ma ustaloną barwę tła, kości oraz oczek. Każda ścianka sfotografowana jest od 3 do 30 razy, określając liczebność każdego ze zbiorów miedzy 18 a 180 zdjęć. Zdjęcia w niektórych zestawach poddawane były działaniu lub ostrego punktowego światła w celu wygenerowania trudniejszych do rozpoznania obrazów. Poszczególne zestawienia kolorystyczne w tych zbiorach wypisane są w tabeli poniżej.

Kolor			Ilość zdjęć	
kości	oczek	tła	ściany	kostki
biały	czarny	czerwony	30	180
biały	czarny	granatowy	3	18
biały	czarny	czarny	8	48
beżowy	czarny	czerwony	3	18
czarny	biały	czarny	10	60
czarny	biały	czerwony	10	60
czerwony	biały	czerwony	5	30
czerwony	czarny	czerwony	3	18
granatowy	złoty	biały	4	24
granatowy	złoty	niebieski	6	36
zielony	biały	zielony	8	48
zielony	biały	biały	5	30
różowoczerwony	czarny	biały	5	30
Łączna ilość zdjęć:			*100*	*600*

Tablica 5.1: Zestawienie kolorystyczne obrazów 1

Każdy z obrazów został następnie poddany przekształceniom, by zwiększyć ich ilość, konieczną do prawidłowego uczenia się sieci. Miało to także zapewnić zniekształcone w niewielkim stopniu zdjęcia, które również powinny być rozpoznawane przez sieć. W tym przypadku zastosowano sześć takich przekształceń.
Następnym krokiem było zastosowanie obrotów o kąty 5^o, 15^o, 30^o lub 45^o, w celu dobrania najbardziej adekwatnej ilości obrazów do rozmiaru sieci oraz ilości parametrów do nauczenia się. Po wielu próbach ostatecznie wybrano kąt 15^o, zapewniający wystarczająco liczny zbiór treningowy do możliwości uczenia sieci oraz nie wydłużało znacząco czasu potrzebnego do przetworzenia wszystkich obrazów.
Powyższe procesy spowodowały uzyskanie 168 zdjęć o rozmiarze 64x64 piksele z każdego z początkowych obrazów o rozmiarze 1600x1200 pikseli. Cały zbiór danych liczył 100800 obrazów, jednakowo w wersji kolorowej RGB oraz w odcieniach skali szarości. Części treningowa i testowa zbioru danych zostały rozdzielone w stosunku 4:1, dając odpowiednio 80640 oraz 20160 zdjęć w każdym z nich.

Nieocenioną pomocą w początkowej fazie prac nad tworzeniem i uczeniem sieci neuronowych z obrazami w kształcie kwadratów był fakt, że praktycznie wszystkie przykłady dostępne w opracowaniach naukowych korzystały ze zdjęć w tym kształcie. Również wszystkie przykłady opisane w różnych poradnikach czy najpopularniejsze zbiory jak MNIST oraz CIFAR10 zawierają kwadratowe obrazy. Przyjęcie takiego kształtu ułatwiło dobór parametrów sieci, nie narzucając osobnych wartości w poziomie i pionie.



Rysunek 5.1: Obrazy o rozdzielczościach 64x64

5.2 Zbiór obrazów prostokątnych

Po nauczeniu i weryfikacji kilkunastu różnych modeli sieci zdecydowano się podjąć próbę z wykorzystaniem większych obrazów oraz kości rozmieszczonych w miejscach bardziej zróżnicowanych niż jedynie pewien, niewielki obszar w centrum obrazu. Kolejne zdjęcia miały również podnieść trudność uczenia poprzez zmniejszenie rozmiaru kości w stosunku do rozmiaru całego obrazu. Ostatnim czynnikiem decydującym o rozwinięciu zbiorów danych była trudność w rozpoznawaniu kości w czasie rzeczywistym w sytuacji, kiedy rozmiar obrazu lub jego wycinka to jedynie 64x64 piksele. Zwiększenie rozmiarów ułatwiłoby identyfikacje kości oraz umożliwiłoby detekcję ilości oczek wyrzuconych na kostce o ile tylko kość znalazłaby się w obszarze odejmowanym przez kamerę.
Plan zakładał wykorzystanie poprzednio wykonanych zdjęć oraz dodanie nowych z kośćmi rozmieszczonymi poza obszarem w centrum obrazu w celu rozwinięcia możliwości sieci. Jednocześnie zrodził się pomysł wykorzystania prostokątnych obrazów, które lepiej oddawałyby rzeczywistość, gdzie prawie wszystkie kamery, niezależnie od zastosowań, dostarczają prostokątny obraz. W tym celu analogicznie jak w przypadku kwadratowych obrazów, zostały wykonane zdjęcia o określonych zestawieniach kolorystycznych. W tabeli poniżej wypisane jest ich zestawienie:

Ilość zdjęć prostokątnych jest mniejsza niż kwadratowych z powodu czasu, jaki zajmuje wykonanie takiej ilości zdjęć oraz chęć wykorzystania obu rodzajów zbiorów do uczenia sieci. Powstała w ten sposób liczba 984 unikalnych zdjęć jest wystarczająca do realizacji zamierzonego zadania i pozwala na odpowiednie uczenie sieci.
Poprzednio wykorzystywane obrazy, miały wymiary 64x64 piksele co łącznie odpowiadało 4096 lub 12288 wartościom pikseli odpowiednio dla obrazów w skali szarości oraz kolorowych RGB. Nowo utworzony zbiór obrazów prostokątnych w pierwszym założeniu miał składać się z obrazów o rozmiarach 320x240 pikseli w skali szarości, co oznaczało 76800 wartości na jedno

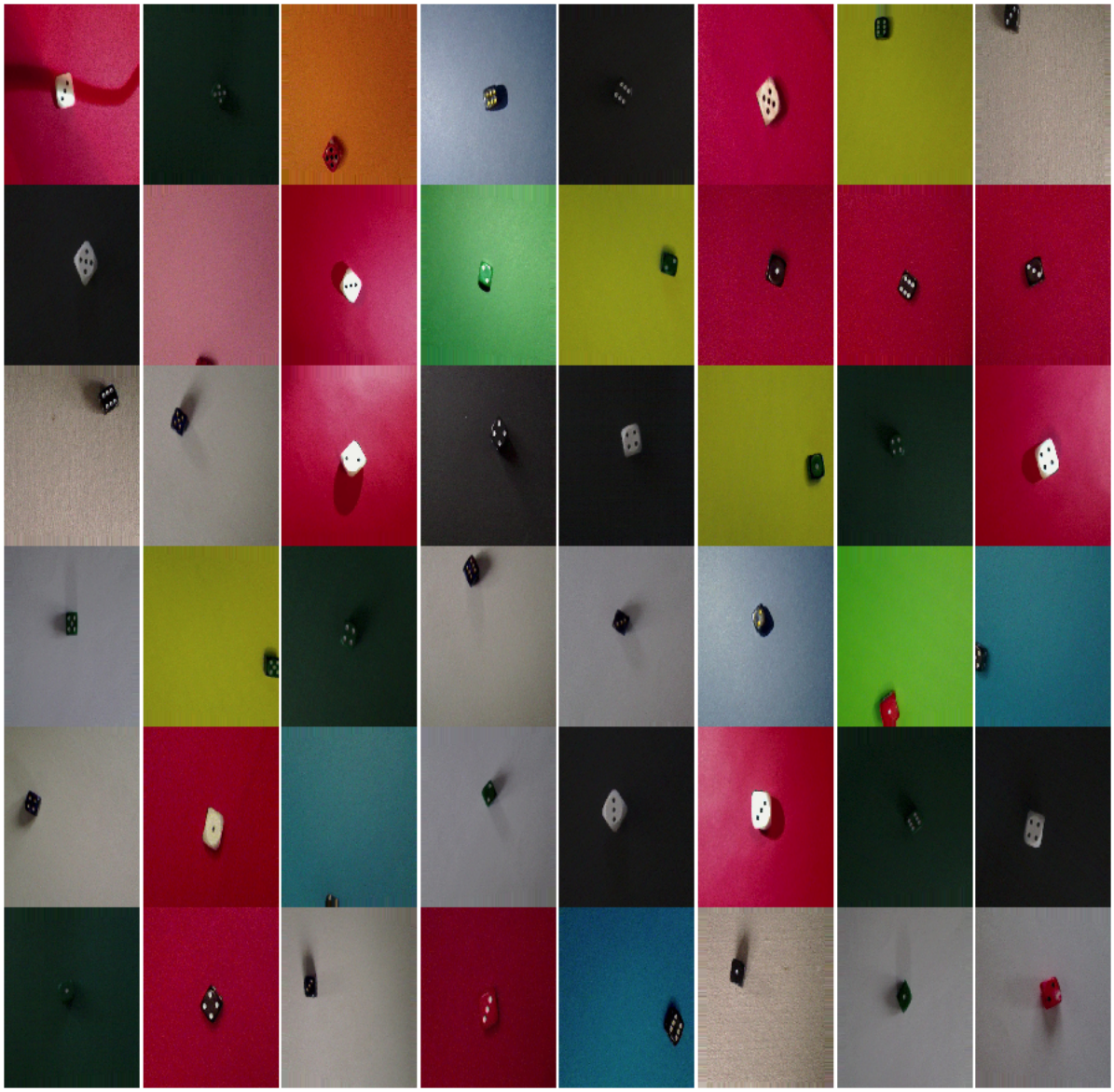
Kolor			Ilość zdjęć	
kości	oczek	tła	ściany	kostki
biały	czarny	zielony	6	36
czerwony	biały	zielony	6	36
czerwony	czarny	różowy	7	42
czarny	biały	szary	6	36
czarny	biały	niebieski	6	36
beżowy	czarny	szary	6	36
beżowy	czarny	niebieski	6	36
granatowy	złoty	biały	8	48
zielony	biały	żółty	7	42
różowoczerwony	czarny	pomarańczowy	6	36
Łączna ilość zdjęć:			*64*	*384*

Tablica 5.2: Zestawienie kolorystyczne obrazów 2

zdjęcie. W wyniku niepowodzenia procesu uczenia po kilku godzinach podjęto decyzję o dwukrotnym zmniejszeniu obrazów do 160x120 pikseli. Wartość ta została wybrana, ponieważ ilość parametrów obrazu, wynosząca 19200, była jedynie o 56% większa od ich liczby dla kolorowych obrazów 64x64. Próby uczenia się sieci pokazały jednak, że lepszym rozwiązaniem będzie zastosowanie mniejszych o 50% obrazów o wymiarach 106x79 pikseli co skutkuje liczbą 8374 wartości na jeden obraz.

Wraz z problemami związanymi z rozmiarami obrazów, zdecydowano się na zmniejszenie ilości przekształceń z sześciu do wybranych czterech, najbardziej efektywnych. Inne przedstawiały zmieniony w niewielkim stopniu obraz, niepotrzebnie wydłużając proces uczenia. Również kąt obrotu zdjęć został zwiększony z 15° do 30°, co w założeniu nie powinno powodować problemów z rozpoznawaniem kości przy różnych ustawieniach.

Wszystkie operacje umożliwiły osiągnięcie 60 obrazów z każdego początkowego zdjęcia, w rozmiarze 106x79 pikseli w odcieniach skali szarości. Całkowita ilość obrazów w pełnym zbiorze danych wynosiła 59040, co przełożyło się na 47232 obrazów treningowych i 11808 testowych, korzystając z identycznego jak wcześniej stosunku 4:1.



Rysunek 5.2: Obrazy o rozdzielczościach 106x79

6. Sieć rozpoznająca ilość oczek

W momencie zrobienia zbioru danych składających się z kwadratowych obrazów, przystąpiono do próby stworzenia i nauczenia modelu sieci neuronowej rozpoznającego ilość oczek wyrzuconych na kostce do gry.
Odnośniki do wszystkich modeli dostępne są pod linkami w części Dodatek B [9]

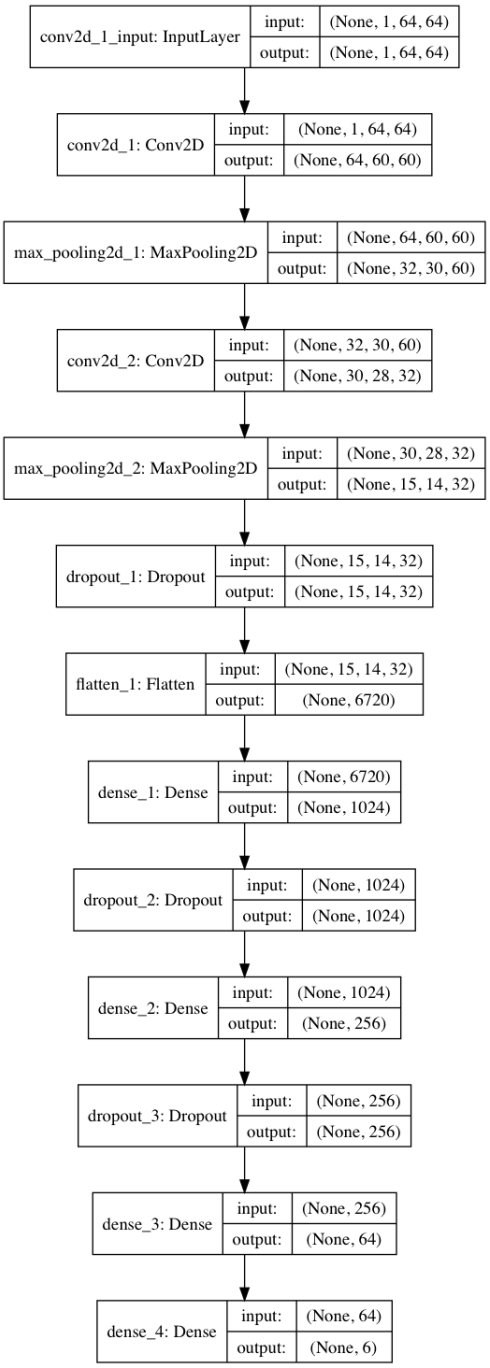
6.1 Hipotezy

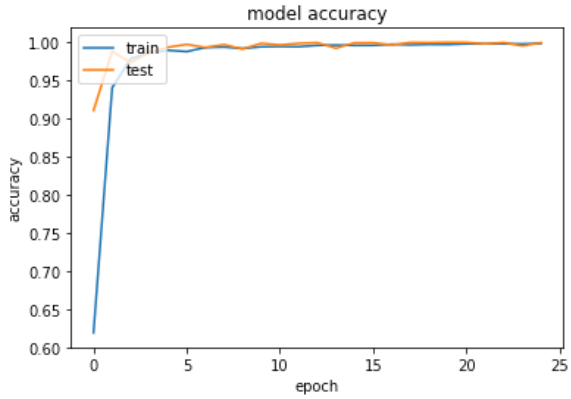
Przed przystąpieniem do tworzenia zbioru danych pojawiła się spora ilość wątpliwości. Większość z nich dotyczyła kwestii możliwości jakiegokolwiek rozpoznania ilości oczek przez sieć. Obawy te wiązały się z faktem, że dla przykładu w zbiorze MNIST, położenie cyfr na obrazie było stałe. W przypadku kiedy na obrazach kolor biały był obszarem w kształcie przypominającym pionową kreskę, z dużym prawdopodobieństwem można było założyć, że jest to cyfra 1 lub 7, a analogiczna sytuacja zachodziła dla innych cyfr. W rozpoznawaniu oczek na kostce, zarówno sama kostka mogła być umieszczona w różnych miejscach na obszarze całego obrazu, jak i dopuszczalny był jej obrót o dowolny kąt.
Następną kwestią był niewielki rozmiar oczka w stosunku do powierzchnii całego ekranu. W przypadku zbioru MNIST średnio około 12-15% obrazu stanowiła barwa biała, która decydowała o wartości zwróconej przez sieć. W przypadku kwadratowych zdjęć z kostkami rozmiar jednego oczka to jedynie 0,21% powierzchni rozmiaru, a dla obrazów prostokątnych wartość ta maleje do około 0,08%. Oznaczało to, że nawet niewielki szum lub zniekształcenia mogą utrudnić prawidłowe rozpoznanie kości.

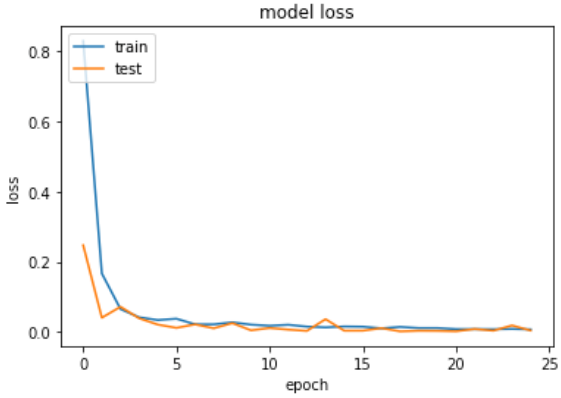
6.2 Pierwsze eksperymenty z sieciami neuronowymi

Pierwszy model
Pierwsza próba stworzenia sieci, mając na uwadze wyżej wspomniane wątpliwości, miała na celu wytrenowanie możliwie prostego zbioru obrazów. W tym celu wykorzystano jedynie najbardziej liczny zbiór obrazów z czerwonym tłem, białą kością o czarnymi oczkami. Dla zwiększenia liczby zdjęć w zbiorze zmniejszono kąt obrotu każdego z nich do 5^o, uzyskując 60480 zdjęć ze 120 oryginalnych.
Architektura tej sieci, była dobierana bez większego wdrażania się w szczegóły i bazowała na modelach sieci udostępnionych na stronach Keras oraz TensorFlow, wykorzystywanych do analizy zbiorów MNIST oraz CIFAR10. Za optymalizator został wybrany Adam, a sam trening

został ustalony na 25 epok w pierwszej turze oraz 10 epok w drugiej turze. Nigdzie wcześniej nie zauważono tego typu praktyki, aby rozdzielać epoki uczenia. Zrobiono to dlatego, aby umożliwić wcześniejsze zakończenie całego procesu w sytuacji, gdyby nie zauważono żadnych postępów. Dodatkowo pozwoliło to na zachowanie modelu po 25 epokach, który, mimo że mógłby nie osiągać dobrych rezultatów, pozwoliłby na wyciągnięcie wniosków lub dalszą naukę. Model sieci prezentował się następująco:







Rezulataty osiągnięte przez ten model były zdumiewające. Po pierwszej epoce sieć uzyskała 61,98% skuteczności ostatecznie osiągając wynik 99,88% po 25 epokach. Kolejne 10 epok praktycznie nie poprawiło już tego rezultatu.

Po analizie tego dokonania, okazało się, że tak duża ilość zdjęć otrzymanych z każdego ze zdjęć początkowych stworzyła zbiór, w którym wiele zdjęć praktycznie się powtarzało. Sieć nie

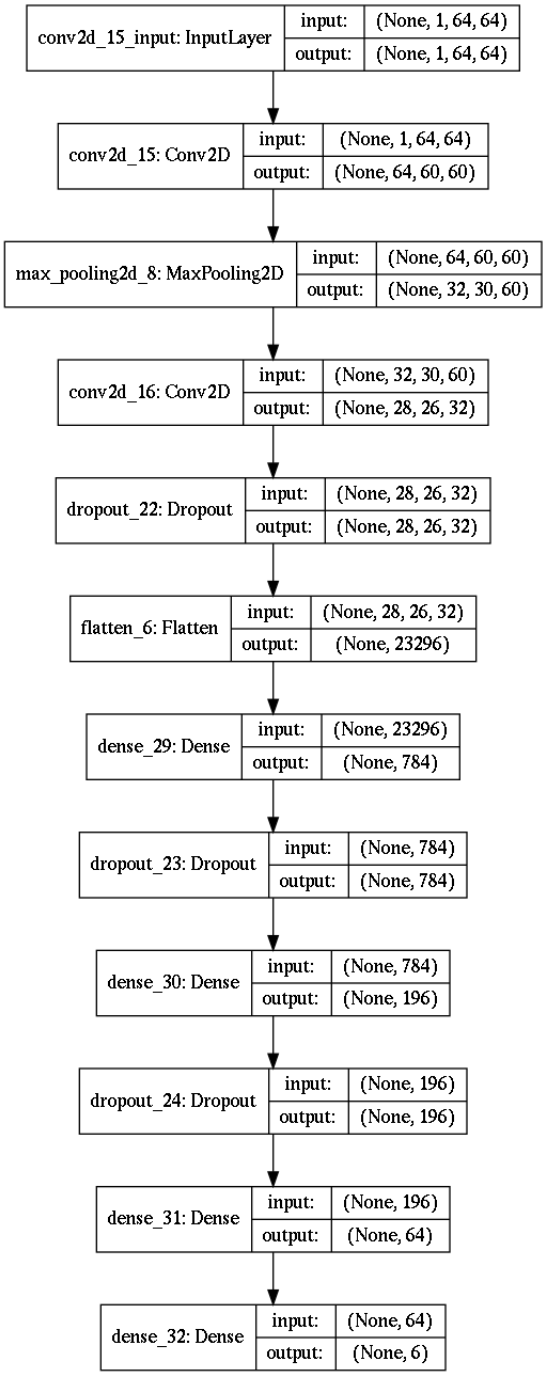
miała żadnych problemów podczas testowania na zbiorze, do którego w teorii nie miała dostępu podczas treningu. Ten fakt spowodował konieczność trenowania sieci na zbiorach zróżnicowanych pod kątem doboru różnych kolorów oraz ilości zdjęć otrzymanych z jednego początkowego zdjęcia.

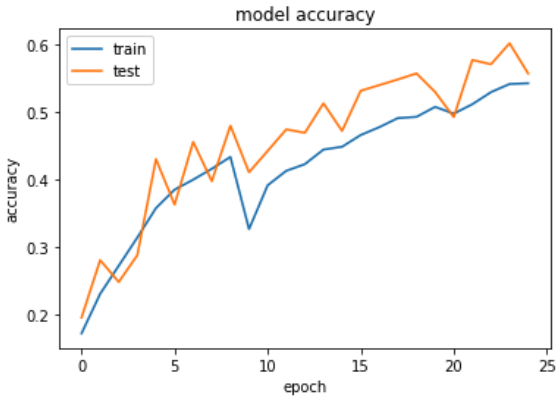
Model ze zróżnicowanymi zdjęciami

Po stworzeniu modelu, który wykazał, że zadanie stworzenia dobrze działającej sieci dla tego problemu jest wykonalne, zabrano się do kolejnego etapu prac. W tym celu wykorzystano zbiór złożony z 13 różnych zestawów kolorystycznych kości, oczek oraz tła, liczący po 80640 i 20160 zdjęć na części treningową i testową.

W tym modelu wykorzystano architekturę nieznacznie zmienioną w stosunku pierwszego modelu.

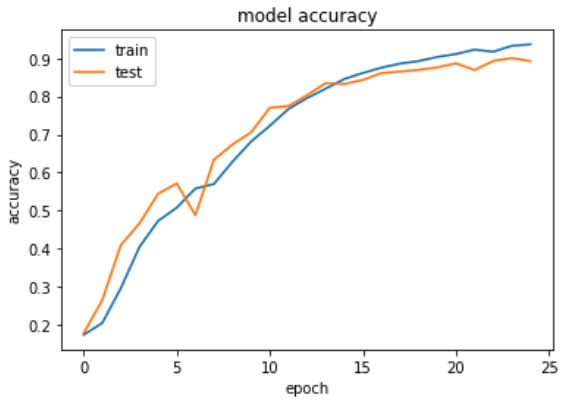
Sieć uczona była przez 25, a następnie przez 10 epok. Po pierwszych 25 epokach uzyskano dokładność 55,64% co potwierdziło przypuszczenie z przedniej sieci o możliwości pokrycia się zdjęć w zbiorach treningowym i testowym. Kolejne 10 epok poprawiło wynik sieci do 65,30%, pozwalając przy okazji zaobserwować istotny fakt. W pierwszej sesji 25 epok dokładność sieci przez ostatnie 5 epok oscylowała w okolicach 52%. W drugiej sesji niemalże od razu wartość podniosła się do 56%. Prawdopodobnie miało to związek ze sposobem, w jaki dostarczane są dane do modelu. Dane były ustalane losowo, ale dla każdej epoki w jednej sesji, układ ten się nie zmieniał. Istniała szansa, że kiedy w następnej sesji zdjęcia były przetwarzane w innej kolejności, umożliwiło to lepsze dopasowanie sieci i efekt skoku jej dokładności.

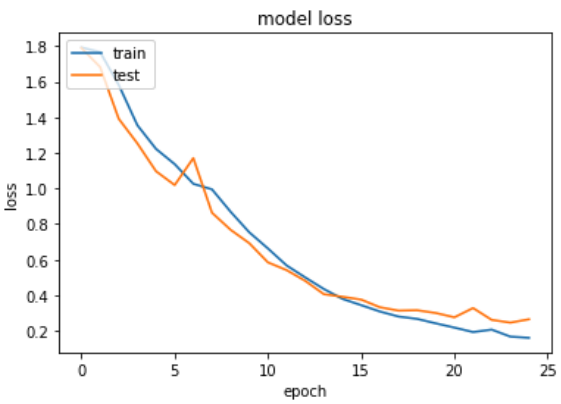




Model z wybranymi zdjęciami

Znacząca różnica w dokładności między modelem z jednolitymi oraz zróżnicowanymi zdjęciami doprowadziła do stworzenia sieci uczonej na zbiorze różnorodnych obrazów, ale dobranymi tak, aby kontrast między tłem, a kością był wyraźny. Architektura sieci jest identyczna jak w powyższych przykładach, jedyną różnicą jest przetwarzanie wyselekcjonowanych obrazów. Zastosowanie 25 epok wystarczyło aby uzyskać dokładność na poziomie 89,23% co jest rezultatem zdecydowanie lepszym niż uzyskane wcześniej 55,64%.





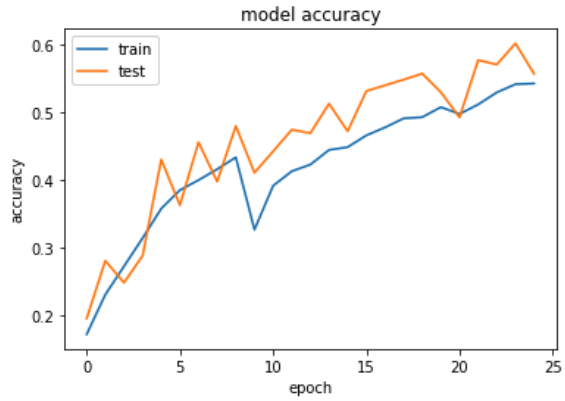
6.3 Analiza różnych optymalizatorów

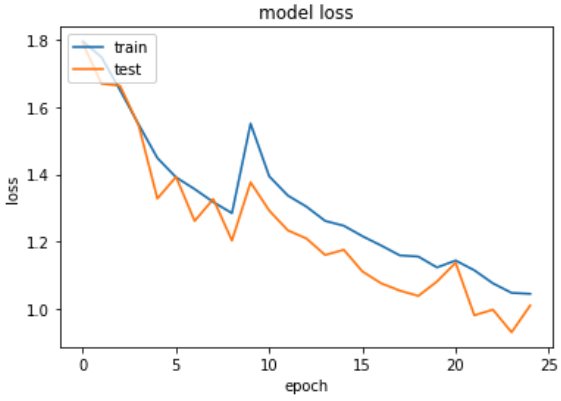
Jeżeli sama różnica w doborze konkretnych obrazów potrafi w tak dużym stopniu zmienić wyniki otrzymane przez sieć, podjęto próbę przetestowania kilku z dostępnych w bibliotece Keras optymalizatorów dla identycznych sieci oraz zbiorów danych. W tym celu postanowiono użyć optymalizatorów RMSprop oraz SGD.

Oba modele z optymalizatorami RMSprop oraz SGD, zostały podobnie jak wcześniej opisany model z optymalizatorem Adam poddane uczeniu przez 25 epok różnorodnych zbiorów obrazów. Wynik RMSprop był zdecydowanie najlepszy i wyniósł 84,34% skuteczności. Najgorszy rezultat w zestawieniu uzyskał amodel korzystający z SGD, zdobywając jedynie 36,92% poprawności. Pośrednim okazał się Adam, który jak opisane wyżej, uzyskał 55,64% po sesji uczenia przez 25 epok.

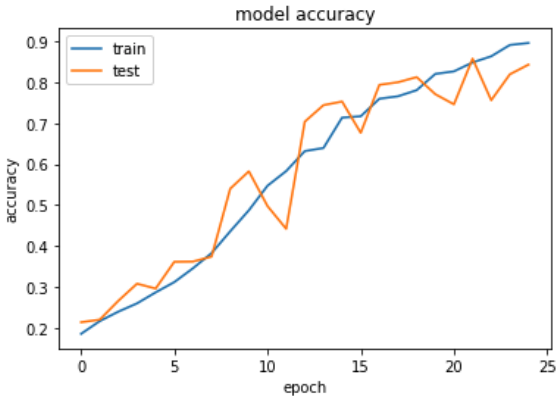
Wyniki te dowiodły, że optymalizator jest kluczowym parametrem *(ang. hyperparameter)* dla odpowiednio skutecznego uczenia. Uzyskany wynik dla RMSprop jest sporym zaskoczeniem, ponieważ w licznych tekstach naukowych to Adam uznawany jest za jeden z najlepszych optymalizatorów, ciesząc się ogromną popularnością. Również z tego powodu, pomimo gorszego niż RMSprop wyniku, Adam będzie wykorzystywany w następnych modelach.

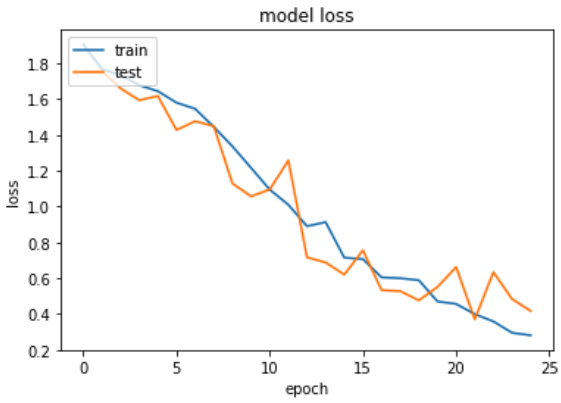
Adam:



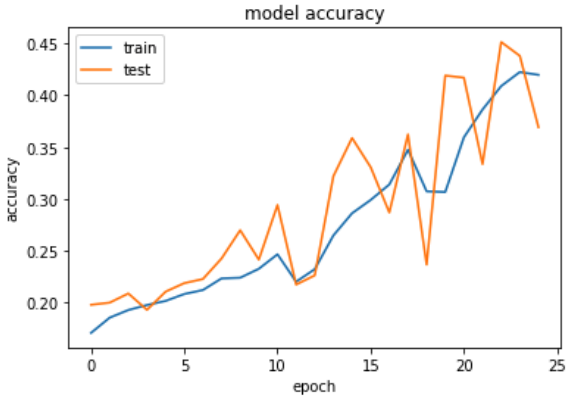


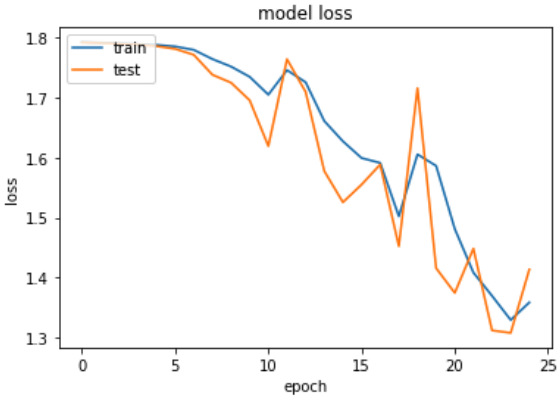
RMSprop:





SGD:

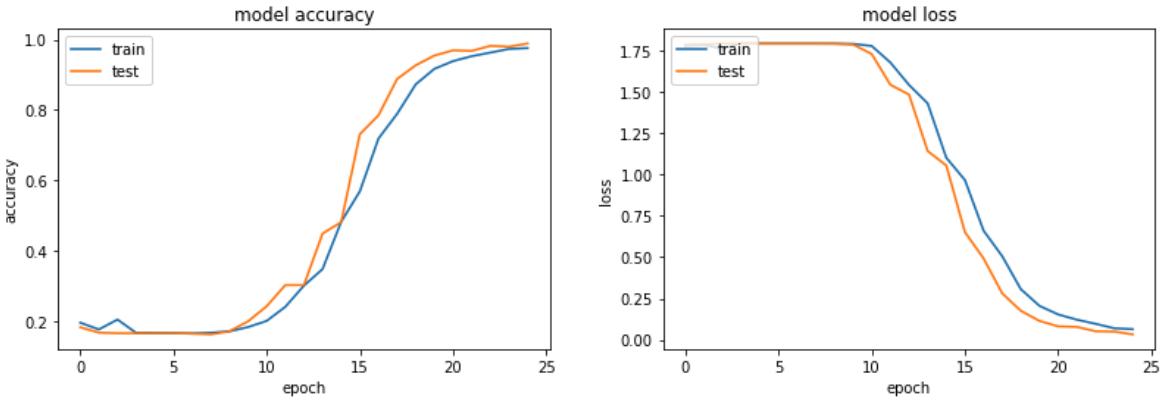




6.4 Wykorzystanie sieci AlexNet

Sieć o nazwie AlexNet została zaprezentowana przez Alexa Krizhevskyego, Geoffreya Hintona oraz Ilya Sutskevera w 2012 roku. Jej ideą jest zastosowanie większej ilośći warstw konwolucyjnych, gdzie początkowe dwie warstwy mają filtry rozmiarów odpowiednio 11x11 oraz 5x5. Następnie umieszczone są trzy warstwy konwolucyjne z filtrami 3x3, które w przeciwieństwie do pierwszych dwóch warstw nie są rozdzielone warstwami z maxpoolingiem. AlexNet zdeklasowała rywali w konkursie ILSVRC 2012 i swoimi rezultatami zszokowała wielu naukowców. Z racji tak świetnych rekomendacji, zdecydowano się na realizację modelu przypominającego jej budowę.

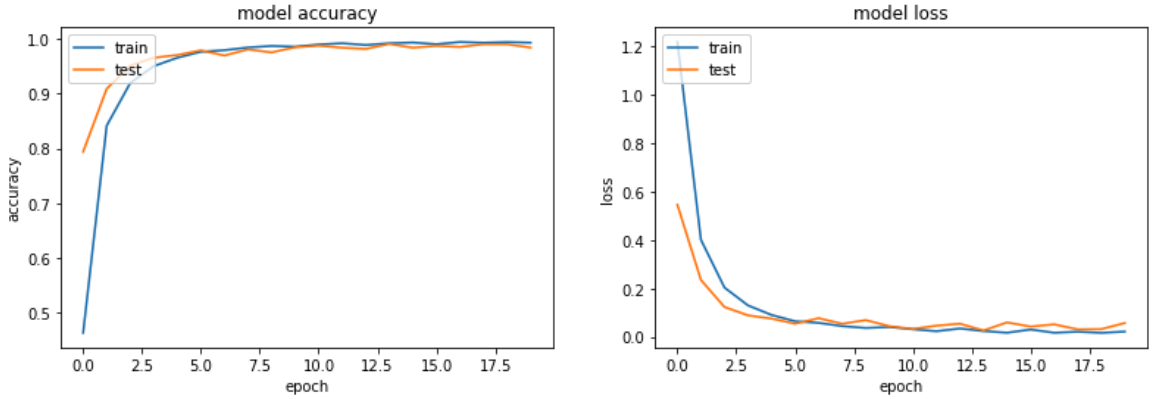
Zastosowanie uproszczonej architektury, bazującej na idei sieci AlexNet pozwoliło na osiągnięcie poprawności wynoszącej 98.88% po 25 epokach. Na wykresie uczenia się można zaobserwować, że przez pierwsze 10 epok precyzja przewidywań sieci w ogólnie nie rosła. Obserwacja sugeruje, że sieci głębsze mogą potrzebować większej ilości epok do rozpoczęcia procesu prawidłowego rozpoznawania obrazów.



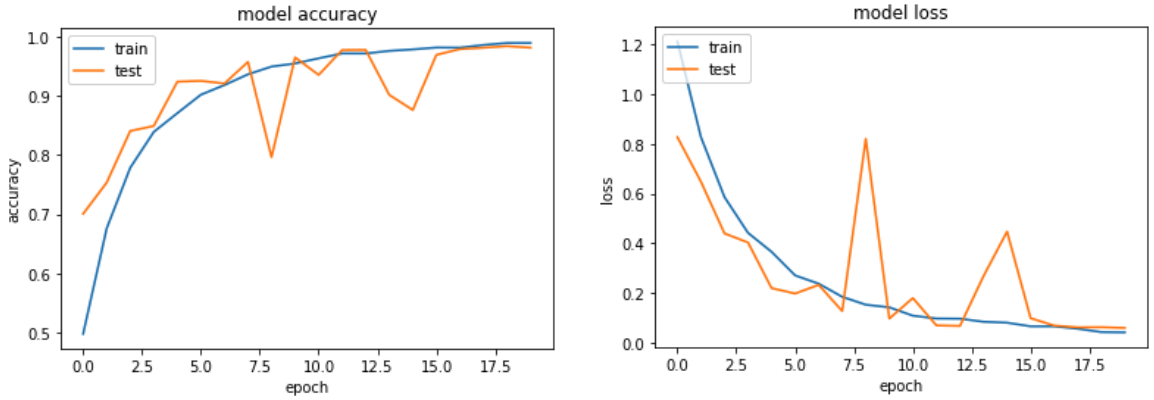
6.5 Porównanie dla obrazów kolorowych i czarno-białych

Obraz w skali szarości posiada jedynie jeden kanał odpowiadający jasności. Obrazy kolorowe RGB posiadają trzy kanały informujące o nasyceniu odpowiednio czerwonego, zielonego i niebieskiego koloru. Wzrost ilości informacji wiąże się z większym obciążeniem pamięci i większą ilością parametrów. W celu weryfikacji różnicy między uczeniem obu rodzajów obrazów podjęto próbę porównania wyników uczenia dla dwóch jednakowych sieci.

Kolorowe RGB:



Skala szarości:



Do tego eksperymentu wykorzystano narzędzie generatora dostępne w Keras, które umożliwia przekazywanie obrazów do sieci bez konieczności wcześniejszego ładowania całego zbioru do pamięci, umożliwiając pracę na bardzo dużych zestawach danych. Architektura obu sieci była uproszczona, by zniwelować długi czas uczenia. Modele obu sieci znajdują się poniżej:

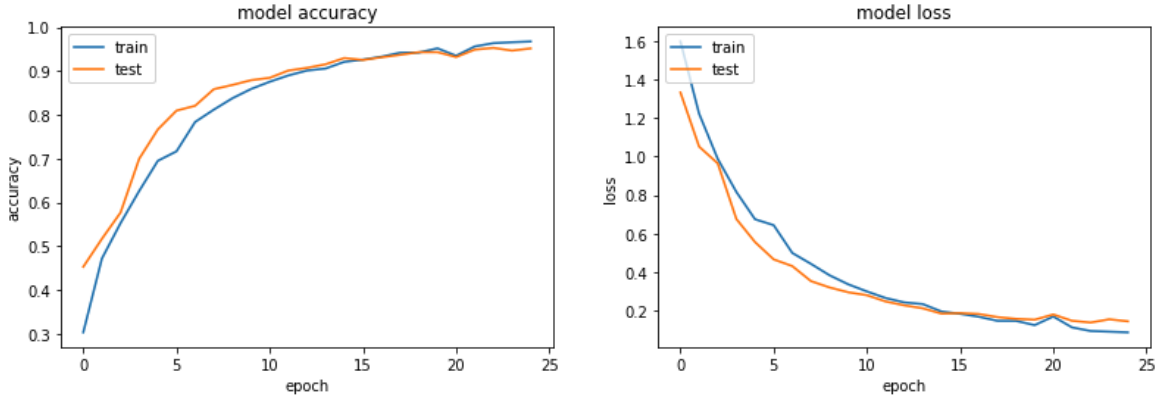
Oba modele po 20 epokach wykazały się bardzo zbliżonymi wynikami 98,47% oraz 98,12% na korzyść sieci z kolorowymi obrazami. Lepiej radzący sobie model, dodatkowo nauczył się bardzo szybko do poziomu 95%, osiągając go już po 4 epoce, gdzie operujący na obrazach w skali szarości osiągnął ten wynik po 10 epokach. Obserwacja ta może sugerować, że większa różnica w wartościach dla kolorowych obrazów może przyśpieszać uczenie, ale wymaga większej ilości pamięci i spowalnia każdą epokę o 15%.

6.6 Porównanie modelu sekwencyjnego i funkcyjnego biblioteki Keras

Pierwsze użycie modelu funkcyjnego

Standardowym sposobem na stworzenie sieci neuronowej w bibliotece Keras jest wykorzystanie sekwencyjnego modelu, opierającego się na zasadzie umieszczania warstw na stosie. Bardziej zaawansowaną możliwością jest wybranie modelu funkcyjnego API, który pozwala na budowę złożonych sieci neuronowy m.in. z wieloma wyjściami, współdzielonymi warstwami i acyklicznymi grafami.

Zdecydowanie większe możliwości modelu funkcyjnego spowodowały chęć realizacji bardziej złożonych sieci neuronowych. Pierwsza próba miała za zadanie odtworzyć sieć o identycznej budowie jak w pierwszy model operujący na różnokolorowych zbiorach i porównać wyniki po uczeniu przez 25 epok. Model został przedstawiony poniżej:



Zaskoczeniem była spora rozbieżność w ilości parametrów do nauczenia. W modelu sekwencyjnym ich ilość wynosiła 18 milionów, a w modelu funkcyjnym API aż 25 milionów parametrów, co oznacza 40% wzrost ich ilości oraz 50% wzrost czasu potrzebnego na jedną epokę przy identycznie zdefiniowanych warstwach. Ciężko uzasadnić tę różnicę, z grafu można wywnioskować, że prawdopodobnie model ten inaczej uwzględniał filtry konwolucyjne, ponieważ rozmiar warstw po ich zastosowaniu nie był jednakowy.

Niezależnie od tego, model osiągnął zdumiewająco dobry rezultat 95,17% dokładności. Wadą tego rozwiązania była, przez zwiększoną ilość parametrów i konieczność zastosowania mniejszego rozmiaru partii obrazów jednocześnie dostarczanych sieci z powodów problemów z pa-

mięcią.

Pełne porównanie modeli

Brak sugestii dostępnych w internecie oraz dokumentacji biblioteki Keras wyjaśniających różnice w ilość parametrów w modelach sekwencyjnym, oraz API spowodował chęć stworzenia dwóch dużych, identycznych modeli, bezpośrednio porównujących ilość parametrów. Modele zostały jedynie skompilowane w celu wyciągnięcia informacji o ich rozmiarach, ich uczenie zajęłoby zbyt dużo czasu.

Po kompilacji, model sekwencyjny uzyskał 47 milionów parametrów, a model funkcyjny API aż 500 milionów parametrów do nauczenia. Ta ogromna różnica spowodowała zaniechanie kolejnych prac nad modelami funkcyjnymi z uwagi na wielokrotne zwiększenie czasu potrzebnego na proces uczenia.

6.7 Próby wykorzystania prostokątnych obrazów

Doświadczenia zakończone niepowodzeniem

Dotychczasowe modele korzystały ze zbiorów o obrazach w kształcie kwadratów z kośćmi umieszczonymi w ich środkowej części. Drugi rodzaj przygotowanych zbiorów danych zwiększał trudność zadania, dzięki zastosowaniu obrazów prostokątnych z mniejszym obszarem, na którym znajdowała się kostka w stosunku do pierwsze rodzaju obrazów.

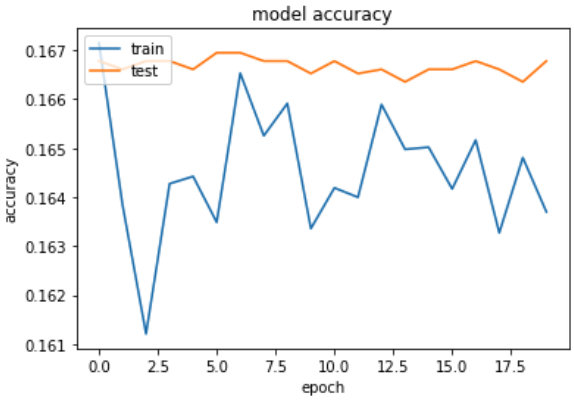
Po uzyskaniu wielu bardzo dobrych wyników powyżej 90% na obrazach o rozmiarach 64x64 przystapiono do prób znacznego zwiększenia obrazów prostokątnych.

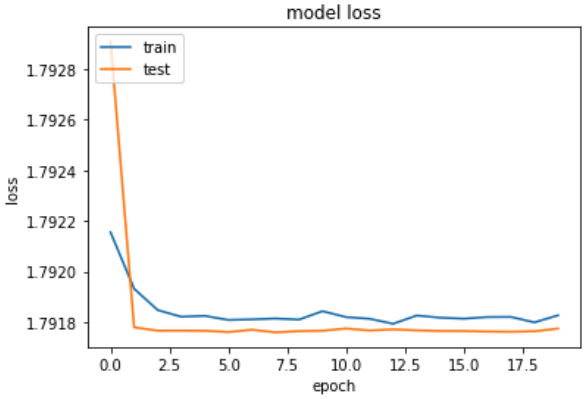
Pierwszą próbą było wykorzystanie obrazów 320x240 zarówno w wersjach kolorowych, jak i w skali szarości. Jeden z modeli bazował na architekturze AlexNet, drugi bezpośrednio ją kopiował, ale pomimo świetnego wyniku na mniejszych obrazach, w obu przypadkach uczenie zakończyło się całkowitą klęską. Warto wspomnieć, że czas potrzebny na jedną epokę był około 15-krotnie większy niż w przypadku obrazów 64x64.

Ostania próba z obrazami w tym rozmiarze, zakładała użycie uproszczonej architektury podobnie jak przy porównaniu uczenia obrazów RGB i w skali szarości. Finalnie, tak jak wcześniej, pomimo 20 epok sieć nie wykazała żadnego postępu w rozpoznawaniu kości.

Niepowodzenia spowodowały konieczność zmniejszenia zbiorów przez ograniczenie rozmiarów obrazów. Problemem podczas zmniejszania była chęć uniknięcia problemów z brakiem ostrości oczek na kostce co mogłoby uniemożliwić skuteczną naukę. Zdecydowano się na dwukrotne zmniejszenie rozmiarów obrazów, licząc, że pozwoli to na zaobserwowanie, chociaż niewielkich postępów.

Sieć bazująca na zdjęciach 160x120 była pierwszą próbą, gdzie zamiast jak dotychczas kwadratowych, użyto prostokątnych filtrów konwolucyjnych. Proces uczenia po 20 epokach niestety również zakończył się niepowodzeniem.

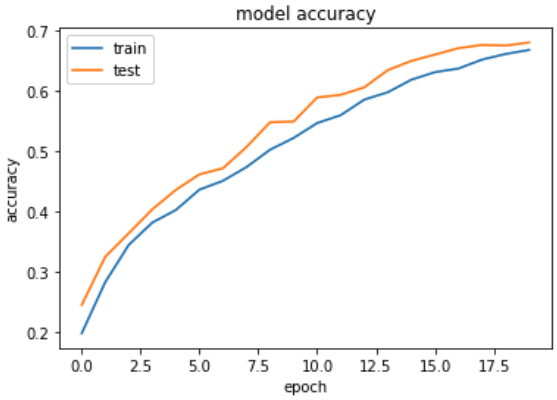


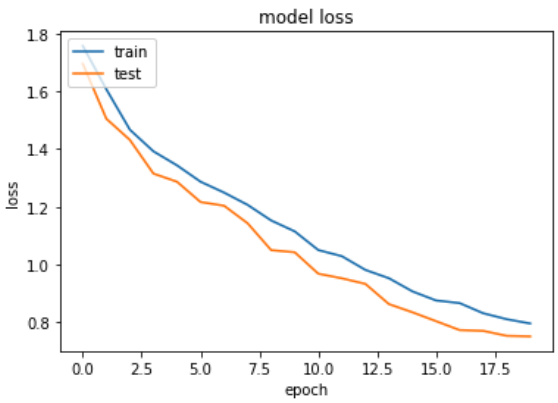


Wytrenowany prostokątny model

Powyższe porażki oraz wcześniejsze sukcesy na kwadratowych obrazach sugerowały, że prawdopodobnie modele nie są błędne, jedynie obrazy mogą mieć za duży rozmiar. Ta hipoteza rozpoczeła proces dobierania odpowiedniej rozdzielczości zdjęć tak by były małe, jednocześnie unikając rozmycia oczek na kostce nie. Najlepszym wyborem okazały się zdjęcia w rozmiarze 106x79, zachowujące proporcje jak obrazy 160x120, ale posiadające o 50% mniejszą liczbę parametrów.

Pierwsza próba przeprowadzona przez 20 epok z filtrami konwolucjnymi o prostokątnych kształtach wreszcie zakończyła się sukcesem. Sieć osiągnęła wynik 68,03% co nie było świetnym rezultatem, ale dawało informację, że dalsza nauka jest możliwa.





6.8 Doskonalenie modelu z prostokątnymi obrazami

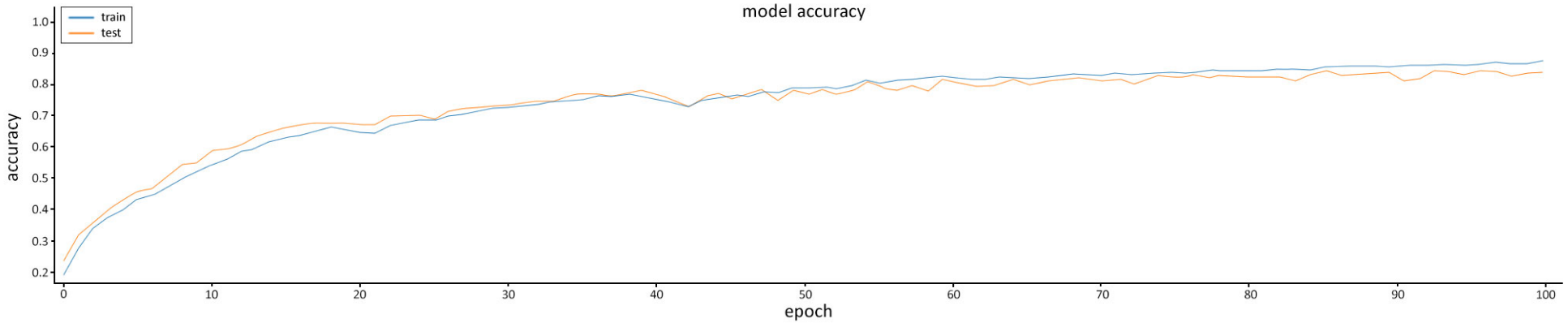
Po pierwszym sukcesie sieci z obrazami prostokątnymi, podjęto decyzję o jej ulepszeniu. Zaczęto od próby zmniejszenia ilości parametrów przez zamianę większych filtrów konwolucyjnych, większą ilością mniejszych, co znacząco zmniejszało ilość parametrów sieci potrzebną do nauczenia *Substituting Big convolution* [7]. Po zaaplikowaniu ulepszeń i nauce sieci okazało się, że sieć nie poprawiła się w żadnym stopniu. Prawdopodobnym powodem była zmiana architektury sieci wraz z powtórnym zastosowaniem kwadratowych filtrów.

Drugim pomysłem na ulepszenie sieci było zastosowanie zastępowania większych filtrów kilkoma mniejszymi połączonego ze zamianą funkcji aktywacji ReLU na LeakyReLU w celu uniknięcia

możliwych do wystąpienia problemów z zanikającym neuronem. Ulepszenie jednak podobnie jak wcześniejsze nie sprawdziło się w ogóle i nie umożliwiło poprawy dokładności sieci.

6.9 Najskuteczniejszy model z prostokątnymi obrazami

Wykres procesu uczenia sieci z obrazami 106x79 kształtem przypominał funkcję logarytmiczną co nasunęło pomysł ze zwiększeniem ilości epok. Podjęto decyzję o kontynuacji uczenia do kolejno 40, 60, 80 i 100 epok.
Podejście to okazało się bardzo skuteczne, ponieważ po każdych 20 epokach sieć odnosiła lepsze rezultaty, które prezentowały się następująco:
20 epok: 68,03%
40 epok: 78,21%
60 epok: 81,59%
80 epok: 82,39%
100 epok: 84,67%
Wyniki uświadamiają, że prawdopodobnie nawet nieudane próby z większymi zdjęciami mogłyby się powieść, konieczne byłoby jedynie zwiększenie ilości epok. Wiąże się to jednak z ogromnym nakładem czasu, ponieważ prawdopodobnie sensowne rezultaty można by osiągnąć dopiero po 100 epokach.



Rysunek 6.1: Wykres uczenia przez 100 epok

7. Podsumowanie

Zwieńczeniem całej pracy i stworzonych modeli są dwa skrypty rozpoznające ilość oczek wyrzuconych na kości w czasie rzeczywistym. Programy korzystają z wytrenowanych modeli załadowanych do pamięci, które analizują obraz dostarczany z kamery podłączonej do komputera. Modele są przystosowane do rozpoznawania kości na obrazach o wielkościach 64x64 oraz 106x79 pikseli.

Obraz dostarczany z kamery jest wielokrotnie większy, co wymusiło wyznaczenie jego części, która jest analizowana przez zaaplikowane sieci neuronowe. Ta część obrazu oznaczona jest białymi krawędziami dla łatwiejszego zlokalizowania miejsca, w którym powinna być umieszczona kość pod kamerą. Dla obrazów 64x64 wykonano kilka modeli sieci, które osiągnęły powyżej 90% skuteczności. Program umożliwia porównanie wyników dzięki zaaplikowaniu ich do analizy tego samego obszaru na zdjęciu. Pozwala to zauważyć między innymi lepsze rozpoznawanie pewnych kombinacji kolorystycznych niektórych modeli lub trudności z rozróżnieniem wyrzuconych czterech oczek od sześciu, z uwagi na efekt zlewania się ich podczas skalowania.

Dla obrazów 106x79 zaaplikowany jest jedynie jeden model, wytrenowany przez 100 epok.

Wyniki zaobserwowane podczas prac związanych z tworzeniem sieci neuronowych przyniosły wiele niespodziewanych sytuacji. Często otrzymywane wyniki były zdumiewające, biorąc pod uwagę, jak drobne różnice w budowie sieci wpływały na końcowe rezultaty. Największym zaskoczeniem były bardzo duże problemy z uczeniem obrazów prostokątnych w porównaniu do obrazów kwadratowych. Za przykład może posłużyć mniejsza o jedną trzecią ilości danych w obrazach w skali szarości i rozmiarze 106x79 pikseli w stostunku do obrazów kolorowych i rozmiaru 64x64 piksele. Proces uczenia się przebiegał trzykrotnie dłużej na jedną epokę i zastosowanie pięciokrotnie większą ilości epok. Mimo to, wynik dla obrazów kwadratowych wyniósł 98,47%, a dla prostokątnych 84,67%.

Wszystkie eksperymenty pokazały, że nawet proste zadanie rozpoznawania oczek na kostce wymaga dużej ilości danych i mocy obliczeniowej, by w uzyskać satysfakcjonujące wyniki. Przykład ten pokazuje także jak wiele zastosowań może mieć zyskujące na popularności uczenie maszynowe, które wraz z rozwojem technologii będzie mieć coraz więcej możliwości na dostosowywania algorytmów do określonych danych.

Bibliografia

[1] Stanford University course CS231n
http://cs231n.github.io/convolutional-networks/

[2] Stanford University course CS231n
http://cs231n.github.io/optimization-2/

[3] Stanford University course CS231n
http://cs231n.github.io/neural-networks-1/

[4] Eric Roberts, Stanford University
https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/

[5] Tuples Edu
https://becominghuman.ai/what-is-an-artificial-neuron-8b2e421ce42e

[6] Jaron Collis
https://medium.com/deeper-learning/glossary-of-deep-learning-bias-cf49d9c895e2

[7] Leonardo Araujo dos Santos
https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/convolutional_neural_networks.html

[8] Leonardo Araujo dos Santos
https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/backpropagation.html

[9] Anish Singh Walia
https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f

[10] Ujjesl Karn
https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

[11] Eugenio Culurciello
https://towardsdatascience.com/neural-network-architectures-156e5bad51ba

[12] Avinash Sharma V
https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

[13] Sagar Sharma
https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[14] Aditya V. D
https://becominghuman.ai/neural-network-xor-application-and-fundamentals-6b1d539941ed

[15] https://en.wikipedia.org/wiki/Feedforward_neural_network

[16] https://en.wikipedia.org/wiki/Convolutional_neural_network

[17] https://en.wikipedia.org/wiki/Rectifier_(neural_networks)

[18] https://en.wikipedia.org/wiki/Computer_vision#Recognition

[19] Yann LeCun, Corinna Cortes, Christoper J.C. Burges
http://yann.lecun.com/exdb/mnist/

[20] Alex Krizhevsky
https://www.cs.toronto.edu/~kriz/cifar.html

[21] ImageNet Large Scale Visual Recognition Competition
http://www.image-net.org/challenges/LSVRC/

[22] Adit Deshpande
https://adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html

[23] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton
https://www.nvidia.cn/content/tesla/pdf/machine-learning/imagenet-classification-with-deep-convolutional-nn.pdf

[24] Tugce Tasci, Kyunghee Kim
http://vision.stanford.edu/teaching/cs231b_spring1415/slides/alexnet_tugce_kyunghee.pdf

[25] Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, Rob Fergus
https://cs.nyu.edu/~wanli/dropc/

[26] Jason Morrison
https://medium.com/paper-club/paper-review-dropout-a-simple-way-to-prevent-neural-networks-from-overfitting-4f25e8f2283a

[27] Matthew D. Zeiler, Rob Fergus
https://arxiv.org/abs/1311.2901

[28] Sebatian Ruder
http://ruder.io/optimizing-gradient-descent/

[29] Geoffrey Hinton, Nitish Srivastava, Kevin Swersky
https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

[30] Diederik P.Kingma, Jimmy Ba
https://arxiv.org/abs/1412.6980

[31] Krzysztof Sopyła
https://ksopyla.com/python/operacja-splotu-przetwarzanie-obrazow/

[32] James D. McCaffrey
https://jamesmccaffrey.wordpress.com/2013/11/05/why-you-should-use-cross-entropy-error-instead-of-classification-error-or-mean-squared-error-for-neural-network-classifier-training/

8. Dodatek A - zbiory danych

Zbiory danych wykorzystane do uczenia przedstawionych modeli sieci neuronowych znajdują się w linkach dostępnych poniżej:

Zbiór 64x64x1, obrazy w skali szarości, opisany w częsci 5.1
`https://drive.google.com/open?id=1-qCVGTql6A3QHLxS0aQvhFxSz5wUY7uG`

Zbiór 64x64x3, obrazy kolorowe, opisany w częsci 5.1
`https://drive.google.com/open?id=1oWa8t5pCWCGyC3Mbdn_XJ2hVdf0yqrMu`

Zbiór 106x79x1, obrazy w skali szarości, opisany w części 5.2
`https://drive.google.com/open?id=1JS_C_pJqU2TmBEVvMTULTXue7VZJKONH`

9. Dodatek B - modele sieci neuronowych

Wszystkie modele sieci neuronowych przedstawione w pracy dostępne są pod linkami podanymi poniżej:

9.1 Obrazy kwadratowe 64x64

Pierwszy model z obrazami w jednej kolorystyce
`https://github.com/oziomek1/NNLearning/blob/master/dices_cnn/dice_cnn.ipynb`

Pierwszy model ze zróżnicowanymi zdjęciami
`https://github.com/oziomek1/neural_network_dice/blob/master/dice_cnn_adam.ipynb`

Model z wybranymi zdjęciami
`https://github.com/oziomek1/neural_network_dice/blob/master/dice_cnn_adam_wybrane.ipynb`

Modele do analizy optymalizatorów
Optymalizator Adam:
`https://github.com/oziomek1/neural_network_dice/blob/master/dice_cnn_adam.ipynb`
Optymalizator RMSprop:
`https://github.com/oziomek1/neural_network_dice/blob/master/dice_cnn_rmsprop.ipynb` OptymalizatorSGD:
`https://github.com/oziomek1/neural_network_dice/blob/master/dice_cnn_sgd.ipynb`

Model oparty o AlexNet
`https://github.com/oziomek1/neural_network_dice/blob/master/generator_AlexNet.ipynb`

Porównanie modeli dla obrazów kolorowych i w skali szarości
Obrazy kolorowe:
`https://github.com/oziomek1/neural_network_dice/blob/master/generator_comparison.ipynb`
Obrazy w skali szarości:

https://github.com/oziomek1/neural_network_dice/blob/master/generator_comparison_gray.ipynb

Pierwszy model funkcyjny
https://github.com/oziomek1/neural_network_dice/blob/master/dice_cnn_API.ipynb

Pełne porównanie modelu funkcyjnego i sekwencyjnego
https://github.com/oziomek1/neural_network_dice/blob/master/API_Sequential_®Keras2.ipynb

9.2 Obrazy prostokątne

Trzy nienauczone modele z prostokątnymi obrazami 320x240
https://github.com/oziomek1/neural_network_dice/blob/master/generator_320x240_AlexNet.ipynb
https://github.com/oziomek1/neural_network_dice/blob/master/generator2_320x240_AlexNet.ipynb
https://github.com/oziomek1/neural_network_dice/blob/master/generator3_320x240.ipynb

Model ze protokątnymi obrazami 160x120
https://github.com/oziomek1/neural_network_dice/blob/master/simple_NN_160x120.ipynb

Nauczony model z prostokątnymi obrazami 106x79
https://github.com/oziomek1/neural_network_dice/blob/master/simple_NN_106x79.ipynb

Ulepszanie modelu przez redukcje dużych konwolucji
https://github.com/oziomek1/neural_network_dice/blob/master/substituting_106x79.ipynb

Ulepszanie modelu przez LeakyReLU
https://github.com/oziomek1/neural_network_dice/blob/master/subst_LReLU_106x79.ipynb

Kontynuacja uczenia jedynego nauczonego modelu 20-60 epok
https://github.com/oziomek1/neural_network_dice/blob/master/simple_NN_106x79_continue.ipynb

Kontynuacja uczenia jedynego nauczonego modelu 60-100 epok
https://github.com/oziomek1/neural_network_dice/blob/master/simple_NN_106x79_continue_80_100.ipynb

9.3 Wytrenowane modele

Wytrenowane modele do weryfikacji działania na obrazie z kamery
https://drive.google.com/drive/folders/1H1JJ22AM9Jps96LIBpt-Mwf9tkqnvwtY

9.4 Programy weryfikujące modele

Program z obrazami 64x64x1
https://github.com/oziomek1/neural_network_dice/blob/master/camera_with_neural_network_test.ipynb

Program z obrazami 106x79x1
https://github.com/oziomek1/neural_network_dice/blob/master/camera_CNN_106x79.py