Uniwersytet Jagielloński

Wydział Fizyki, Astronomii i Informatyki Stosowanej

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

Wojciech Ozimek

Nr albumu: 1124802

Praca wykonana pod kierunkiem

prof. dr hab. Piotra Białasa

kierownika Zakładu Technologii Gier

FAIS UJ

kwiecień 2018

Spis treści

1. Wstęp

1.1 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) odpowiedzialny za przekazywanie informacji z ciała komórki do kolejnych neuronów oraz dendrytów służących do odbierania sygnałów i przesyłaniu ich do ciała komórkowego. Impuls elektryczny z jednego neuronu do drugiego przekazywany jest w synapsie, miejscu komunikacji danego neuronu z poprzednim. Synapsa składa się z części presynaptycznej (aksonu) i postsynaptycznej (dendrytu). Neuron przewodzi sygnał tylko w sytuacji kiedy suma potencjałów na wejściach od innych neuronów na jego dendrytach przekroczy określony poziom. W przeciwnym wypadku neuron nie przewodzi sygnału. Dodatkowo, zwiększenie potencjału na wejściach nie powoduje wzmocnienia potencjału na wyjściu neuronu.
Neurony połączone i działające w ten sposób tworzą sieci neuronowe, których dobrym przykładem może być 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}.

1.2 Sztuczny neuron

Matematycznym 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 aktywacji dla argumentu którym jest suma wszystkich wag pomnożonych przez odpowiednie wartości wejściowe. Wyjście danego neuronu połączone jest z wejściami innych neuronów, tak jak ma to miejsce w przypadku biologicznego neuronu.

Powyższy, sztuczny model neuronu jest podstawowym budulcem sztucznych sieci neuronowych z racji prostoty działania i łatwości implementacji. W rzeczywistych zastosowaniach używane są pojecia takie jak wektor wejściowy oraz wektor wag, które oznaczają odpowiednio wszystkie wartości wejścioweoraz wszystkie wagi dla danego neuronu.
Pojedynczyn neuron pozwala na uzyskanie mocno ograniczonych rozwizań. Przykładowo korzystając z prostych funkcji logicznych ANR, OR, NOT, XOR okazuje się że pojedynczy neuron jest w stanie poprawnie rozwiązywać jedynie pierwsze trzy z podanych funkcji. Problem wiąże się z brakiem możliwości uzyskania poprawnych rezultatów dla zbiorów które nie są liniowo separowalne. W takich przypadkach konieczne jest użycie większej ilości neuronów, a więc tworzenie sieci neuronowych których możliwości adaptacyjnego do poszczególnych problemów są niejednokrotnie bardzo zaskakujące.

1.3 Historia

Rozpoczęciem prac nad sztucznymi neuronami można datować na rok 1943 kiedy to Warren McCulloch i Walter Pitts przedstawili wspomniany już wcześniej model neuronu. Pierwsze sieci neurone używane były np do operacji bitowych i przewidywania kolejnych wystąpień bitów w ciągu. Mimo licznych prób zastosowania ich do realnych problemów nie zyskały one popularności. Powodem tego były prace naukowe sugerujące ograniczenia sieci takie jak brak możliwości rozszerzenia sieci do więcej niż jednej warstwy oraz jedynie jednokierunkowe połączenie między neuronami. 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.
Przełomem okazał sie rok 1982 kiedy to John Hopfield przedstawił sieć asosjacyjną (zwaną siecią Hopfielda). Nowością było dwukierunkowe połączenie neuronów co zapewniało możliwość uczenia się danych wzorców. Kolejnymi przełomowymi odkryciami były zarówno wprowadzenie wielowarstwowych sieci neuronowych oraz wstecznej propagacji. Nowe odkrycia pozwoliły na zastosowanie sieci w wielu różnych dziedzinach, wymagając jednak dużych ilości obliczeń, obszernych zbiorów treningowych, wielu tysięcy iteracji i długiego czasu nauki. Korzyścią jest jednak fakt, że wytrenowany model można wykorzystać w praktycznie każdych warunkach i natychmiastowo bez konieczności uczenia.

2. Cel pracy

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

3. Technologie

Poniższy spis przedstawia technologie użyte podczas tworzenia tej pracy licenckiej. Każda technologia jest krótko opisana wraz z powodem dla którego została użyta.

3.1 Język programowania i środowisko

Python

Język programowania Python obecnie jest bardzo popularnym narzędziem wykorzystywanym w pracach naukowych. Jest to spowodowane bardzo czytelną i zwięzłą składnią, która w pełni pozwala skupić się na danym problemie. Jest on także używany w bibliotekach które wykorzystywane były do tworzenia modeli sieci neuronowych.

Jupyter Notebook

Aplikacja Jupyter Notebook pozwala uruchamiać w przeglądarce pliki tzw. notebooki które składają się z wielu bloków. W blokach może znajdować się kod programu lub jego fragment, a w pozostałych można prezentować m.in. tekst, wykresy bądź tabele. Korzystanie z Jupyter Notebooka zdecydownaie ułatwia pracę, umożliwiając tworzenie kodu wraz z podglądem wykresów bądź danych prezentowanych w innej formie bez konieczności przełączania między oknami bądź kartami.

3.2 Biblioteki

OpenCV

Biblioteka funkcji do obróbki obrazów, najczęsciej wykorzystywana w języki C++ oraz Python. W projekcie została użyta do uzyskiwania zbiorów obrazów kości do gry ze zdjęć uzyskanych kamerą. Uzyskane zbiory charakteryzowały się ściśle określonymi wymiarami każdego z obrazów, obrotami obrazów o ściśle określony kąt, trybami RGB oraz monochromatycznym jak również

kadrowaniem w celu osiągnięcia zakładanych proporcji między wielkością kostki na zdjęciu a rozmiarem obrazu.

TensorFlow

Biblioteka do uczenia maszynowego oraz tworzenia sieci neuronowych. Jej ogromną zaletą jest możliwość wykorzystywania zarówno procesorów jak i procesorów graficznych. Z uwagi na charakter wykonywanych obliczeń praca przy użyciu kart graficznych jest kilka razy szybsza niż na procesorze, co znacząco przyśpiesza proces uczenia. Najlepiej wspierane języki programowania do C++ oraz Python.

Keras

Biblioteka do tworzenia modeli sieci neuronowych wykorzystująca inne bardziej profesjonalne biblioteki jak TensorFlow, Theano lub CNTK. Zaletami Kerasa są zarówno przystępny interfejs pozwalający w krótkim czasie stworzyć model sieci oraz możliwość stworzenia zaawansowanych modeli przy średnim pogorszeniu czasu uczenia się sieci o 3-4% w stosunku do TensoFlow.
W sytuacji kiedy interfejs oraz dokumentacja do TensorFlow są 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, wykorzystywany w wielu, głównie naukowych zastosowaniach. Numpy wprowadza własne typy danych oraz funkcje którę są niedostępne w instalacji Pythona. Może być rozbudowany o moduł Scipy, który nie był jednak wykorzystany przy tworzeniu tej pracy.

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 ogranizacji tekstu wraz z językiem odpowiednich znaczników. Praca w LaTeX jest przeciwieństwem edytorów tekstowych typu WYSIWYG jak MSWord. LaTeX bazuje na TeX który jest systemem składu drukarskiego do prezentacji w formie graficznej. Ogromną zaletą jest możliwość tworzenia w tekście zaawansowanych wzorów matematycznych.

3.3 Technologie poza programistyczne

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. Usługa działa na zasadzie rozliczenia godzinowego podczas korzystania z niej. W tej pracy zostały wykorzystane instancje zoptymalizowane do obliczeń na kartach graficznych i do uczenia maszynowego, wyposażone w wcześniej wspomnianą kartę NVIDIA TESLA K80.

Google Compute Engine

Platforma analogiczna do wyżej wspomnianej, dysponująca tymi samymi modelami kart. W pracy wykorzystywana mniej niż Amazon AWS, głownie z powodu przyzwyczajenia do tamtejszej strony. Dodatkowym celem było sprawdzenia czy różnica w oprogramowaniu serwerów będzie miała wpływ na szybkość nauki sieci neuronowych, co jednak nie znalazło potwierdzenia w praktyce. Wartym wspomnienia jest fakt, że na obu platformach czas trwania uczenia sieci zmniejszył się średnio 6-10 krotnie w stosunku do pracy na komputerze wykorzystującym procesor.

4. Słownik pojęć

4.1 Zbiór danych

Zbiór danych, obrazów lub zestaw danych to zbiór wszystkich zdjęć kości wykonanych na poczet pracy z wykorzystaniem kamery. Każde zdjęcie w zbiorze jest przetworzone w celu zmniejszenia czasu uczenia się sieci oraz potrzebnej pamięci. Zdjęcia były wykonywane kamerą o rozdzielczości 1600x1200 pikseli. Każde ze zdjeć w zbiorze zostało poddane procesowi skalowania oraz kadrowania w celu osiągnięcia odpowiedniego rozmiaru. W celu uzyskania większej liczebności zbioru wszystkie obrazy zostały dodatkowo poddane operacji obrotu o dany kąt. Każdy ze zbiorów został zduplikowany i poddany konwersji z trybu RGB na skalę szarości przez usunięcie informacji o barwie oraz nasyceniu kolorów, pozostawiając jedynie informację o jasności piksela.
Po przeprowadzeniu całego procesu, każdy obraz miał wymiary 64x64, zarówno w wersji kolorowej i czarno białej, co skutkowało rzeczywistymi rozmiarami odpowiednio 64x64x3 oraz 64x64x1.
Każdy element w zbiorze ma przypisaną wartość liczbową informującą o faktycznej ilości oczek wyrzuconych na kostce przedstawianej na zdjęciu. Wartość ta zwana jest także odpowiedzią i jest wykorzystywana w procesie uczenia sieci jako docelowa informacja, którą ma zwrócić sieć po weryfikacji danego obrazu.

Zbiór treningowy

Część zbioru danych który wykorzystywany jest w procesie uczenia sieci określany jest mianem zbioru treningowego lub zbioru uczącego. Jego liczebność to zazwyczaj 60-80% całego zbioru danych. Praktycznie we wszystkich zastosowaniach dane w tym zbiorze przed rozpoczęciem uczenia poddawane są losowej permutacji.

Zbiór testowy

Zbiór testowy lub zbiór walidacyjny służy do oceny zdolności sieci do rozpoznawania danych.

Celem rozdzielenia tego zbioru od danych testowych jest weryfikacja sieci na danych które wcześniej nie zostały przetworzone przez sieć.

4.2 Budowa sieci neuronowej

Sieć neuronowa

Sieć neuronowa bądź sztuczna sieć neuronowa *(ang. ANN Artificial Neural Network)* jest strukturą matematyczną, która ma odzwierciedlić działanie biologicznych sieci neuronowych. Posiada możliwość uczenia się poprzez obliczenia i przetwarzanie sygnałów w elementach określanych neuronami. W rozpoznawaniu obrazów są w stanie zidentyfikować elementy na obrazach na przykład człowieka, bez wcześniejszej znajomości lub wiedzy na temat przedmiotu podlegającego rozpoznaniu. Realizują to poprzez analizę przykładowych obrazów z informacją czy znajduje się na nich pożądany obiekt, a następnie zmianę swoich własnych parametrów w celu poprawnej identyfikacji kolejnych zdjęć. Ten rodzaj uczenia się okreslany jest mianem uczenia nadzorowanego, gdzie każdy obraz ma przypisaną wartość.
Ich zastosowanie jest bardzo szerokie i wykraczające poza rozpoznawanie obrazów, jednak w związku z powiązaniem tego zagadnienia z tematyką pracy, podany powyżej przykład ma jak najlepiej oddać istotę działania sieci neuronowych.

Neuron

Najmniejszy element sieci neuronowej, przyjmuje wiele wejść i jedno wyjście. Może zawierać także próg (ang. threshold), który może być zmieniony przez funkcje uczącą. Neuron wyposazony jest także w funkcję aktywacji, odpowiednio modyfikującą jego wyjście. Każde wyjście neuronu z poprzedniej warstwy połączone jest z wejściami innych neuronów w warstwie następnej.

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

Wagi neuronu

Połączenia w sieci realizowane są między wyjściem poprzedniego neuronu i oraz wejściem następnego neuronu j. Każde takie połączenie ma przypisaną wartość wagi w_{ij}. Podczas procesu uczenia wagi zmieniają się, dostosowując sieć neuronową do otrzymywanych danych, co skutkuje zmniejszeniem wartości błędu.

Bias

Bias to dodatkowa waga wejściowa do neuronu umożliwiająca jego lepsze dopasowanie do da-

nych treningowych. W sytuacji kiedy wszystkie wagi neuronu mają zerowe wartości, unikamy problemów podczas procesu wstecznej propagacji.

Warstwa

Sieć neuronowa zorganizowana jest w warstwach. Neurony w danej warstwie nie są ze sobą w żaden sposób połączone, komunikacja odbywa się tylko między kolejnymi warstwami. Istnieje wiele rodzajów warstw, a sygnał który przechodzi przez całą sieć zaczyna się w tzw. warstwie wejściowej oraz kończy w tzw. warstwie wyjściowej. Istnieją sieci neuronowe (rekurencyjne sieci neuronowe, *(ang. RNN - Recurrent Neural Network)*) w ktorych sygnał może przechodzić przez warstwy kilkukrotnie w trakcie jednej epoki.

4.3 Konwolucyjna sieć neuronowa

Konwolucja

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.

$$h[m,n] = (f * g)[m,n] = \sum_{j}^{m} \sum_{k}^{n} f[j,k] * g[m-j, n-k] \tag{4.2}$$

Konwolucyjna sieć neuronowa

Konwolucyjna lub splotowa sieć neuronowa *(ang. CNN - Convolutuinal Neural Network)* to typ sieci odnoszących największe osiągnięcia w dziedzinie rozpoznawania obrazów, w wielu przypadkach dorównując lub nawet pokonując ludzkie wyniki. Zawdzięczają to swojej budowie, która różni się od zwykłych sieci wykorzystaniem warstw konwolucyjnych i poolingowych, poprzedzających warstwy w pełni połączone. Sieć taka analizuje obraz przy użyciu filtrów konwolucyjnych, dzięki którym jest w stanie rozpoznawać cechy obrazów co znacząco poprawia ich klasyfikacje.

4.4 Propagacja wsteczna

Propagacja wsteczna lub wsteczna propagacja błędów *(ang. Backpropagation)* jest jednym z najskuteczniejszych algorytmów uczenia sieci neuronowych. Polega na minimalizacji funkcji

kosztu korzystając z metody najszybszego spadku lub bardziej zoptymalizowanych sposobów, o których napisane jest w sekcji *Optymalizator*. Swoją nazwę zawdzięcza sposobowi w jaki propagowane są te błędy, od warstwy wyjściowej do wejściowej.

4.5 Warstwy sieci neuronowej

Rodzaje warstw

Poniższe rodzaje warstw zostały użyte w modelach przedstawionych w tej pracy.

Wejściowa

W pracy, gdzie zbiorami danych są zbiory obrazów, każdy pojedynczy piksel obrazu odpowiada jednej wartości liczbowej. W związku z tym rozmiar pierwszej warstwy wejściowej jest identyczny z wymiarami obrazu. Warstwa wejściowa charakteryzuje się brakiem wejść oraz biasu.

Wyjściowa

Rozmiar warstwy wyjściowej odpowiada ilości klas do jakiej wejściowe dane miały zostać sklasyfikowne. Oczekiwanym wyjściem sieci w pracy była liczba oczek możliwych do wyrzucenia na kostce, co odpowiada 6 klasom, po jednej na każdą wartość na boku kostki. Wyjściem wszystkkich przedstawianych w tej pracy sieci był wektor o wymiarach 6x1.

Konwolucyjna

Warstwa konwolucyjna służy do przetworzenia danych z poprzedniej warstwy przy użyciu filtrów konwolucyjnych. Filtry mają określone wymiary i służą do znajdowania cech na obrazach lub ich fragmentach. Najczęściej spotykanymi przykładami filtrów są kwadraty o wymiarach 3x3 piksele, które przetwarzają informacje zawarte w 9 pikselach na jeden piksel wyjściowy. W nauczonym modelu sieci filtry potrafią przekazywać inforamcje o obecności na przykład konkretnego rodzaju łuku lub innej cechy, niezbędnej przy kwalifikacji danego obrazu do danej kategorii.

Zastosowanie wielu warstw konwolucyjnych umożliwia sieci analizowanie bardziej złożonych zależności na obrazach i jest określane jako głęboka sieć. Szeroki model sieci posiada większą liczbę neuronów w każdej z warstw co umożliwia prezycyjniejszą obserwację danych. Ograniczeniem w przypadku sieci głębokiej i szerokiej jest ilość i czas obliczeń, co wymusza wybranie kompromisu miedzy ilością warstw i neuronów w stosunku do danego problemu.

Aktywacyjna

Jest to wydzielenie funkcji aktywacji do osobnej warstwy, które jest realizowane w niektórych bibliotekach. Celem takiego zabiegu jest możliwości podglądu danych na wyjściu neuronu, tuż przed zaaplikowaniem funkcji aktywacji.

W pełni połączona

Sieć neuronowa składa się z w pełni połączonych warstw *(ang. Fully Connected, Dense)*. W konwolucyjnych sieciach neuronowych warstwy te występują po warstwach konwolucyjnych i służą do powiązania nieliniowych kombinacji które miały zostały wygenerowane przez warstwy konwolucyjne oraz ich klasyfikowania. Dodatkowo nie wymagają dużych nakładów obliczeniowych i są proste do zaaplikowania. Swoją nazwę biorą od sposobu w jaki realizowane sa połączenia między warstwami. Neurony z poprzedniej warstwy łączą się ze wszystkimi neuronami następnej warstwy.

Flatten

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

Odrzucająca

Warstwa odrzucająca *(ang. Dropout)* zapobiega przetrenowaniu *(ang. Overfitting)* sieci. Proces ten polega na nie braniu pod uwagę wyjść pewnych neuronów, zarówno w przypadku przechodzenia w przeód oraz w tył. 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.

Pooling

Warstwa tzw Poolingu wykorzystywana jest do zmniejszenia rozmiaru pamięci oraz ilości obliczeń wymaganych przez sieć neuronową, jak również może zapobiegać przetrenowaniu. Operacja zmniejszenia polega na wybraniu jednego piksela z danego obszaru przekazaniu go dalej. Najczęściej wykorzystywaną warstwą poolingową jest MaxPooling, który wybiera piksel o największej wartości. Obszar z jakiego wybieramy dany piksel jest zależy od danej warstwy, ale najczęściej jest to kwadrat o wymiarach 2x2 co oznacza znaczące ograniczenie zużycia pamięci

i koniecznych obliczeń. Pooling jest krytykowany ponieważ nie zachowuje informacji o położeniu piksela przekazanego na wyjście warstwy co może objawiać się błednymi interpretacjami podczas testowania sieci.

4.6 Funkcje aktywacji

Funkcja aktywacji Przy pomocy funkcji aktywacji obliczana jest wartość wyjściowa neuronów w sieci neuronowej. Argumentem dostarczanym do funkcji aktywacji jest suma wejść neuronu pomnożonych przez przypisane im wartości wag. Zależnie od konkretnego rodzaju funkcji aktywacji, neuron po przekroczeniu danego progu wysyła sygnał wyjściowy, odbierany przez neurony znajdujące się w następnej warstwie.

$$f\Big(\sum_i w_i x_i + b\Big) \tag{4.3}$$

Rodzaje funkcji aktywacji

Przedstawione poniżej funkcje aktywacji zostały użyte w modelach zaprezentowanych w tej pracy.

Liniowa

Funkcja ta jest praktycznie nie wykorzystywana w sieciach neuronowych. Połączenie wielu warstw których neurony posiadają liniową funkcję aktywacji można przedstawić za pomocą jednej warstwy, ponieważ złożenie wielu funkcji liniowych również będzie funkcją liniową. Nieliniowość funkcji pozwala na klasyfikacje danych przechodzących przez sieć.

$$f(x) = x \tag{4.4}$$

Sigmoid

Największym problemem funkcji sigmoidalnej jest duże ryzyko zaniknięcia gradientu, co może prowadzić do problemu tzw umierającego neuronu. Zjawisko to ma miejsce gdy dla danej funkcji aktywacji, gradient staję się bardzo mały. Jest równoznaczne z zaprzestaniem procesu uczenia, ponieważ gradient zera również wynosi zero. W przypadku tej funkcji gradient może zanikać obustronnie.

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

Tanh

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

ReLU

ReLU *(and. Rectified linear unit)* jest najpopularniejszą funkcją aktywacji wykorzystywaną w sieciach neuronowych. 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.

$$f(x) = max(0, x) = \begin{cases} x & \text{if } x \geqslant 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{4.7}$$

LeakyReLU

LeakyReLU jest ulepszeniem ReLU dzięki zastosowaniu niewielkiego gradientu w sytuacji dla której ReLU jest nieaktywne. Zmiana ta pozwala na uniknięcie problemu tzw umierającego neuronu.

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

Funkcja kosztu

Funkcja kosztu lub funkcja błędu jest niezbędna do prawidłowego przeprowadzenia procesu uczenia. Daje ona informacje o różnicy między obecnym staniem sieci o optymalnym rozwiązaniem. Algorytm uczenia sieci analizuje wartość funkcji kosztu w kolejnych krokach w celu zminimalizowania jej.

4.7 Optymalizatory

Optymalizator

Inne określenie algorytmu optymalizacyjnego wykorzystywanego do obliczania wag neuronów i biasu sieci neuronowej. Posiada kluczowe znaczenie podczas procesu uczenia sieci zarówno w kwestii czasu oraz skuteczności. Z tego powodu jest to jeden z kluczowych obszarów obecnych badań i rozwoju sieci neuronowych.

Metoda gradientu prostego

Metoda gradientu prostego *(ang. Gradient Descent)* jest podstawowym algorytmem służacym do uaktualniania wartości wag oraz biasu podczas uczenia sieci. Wadą tej metody jest przeprowadzanie jednorazowej aktualizacji po wyliczeniu gradientu dla całego zestawu danych. Jest to bardzo wolne, w niektórych przypadkach może powodować problem z ilością zajmowanego miejsca w pamięci a przede wszystkim może prowadzić do pozostania w jednym z lokalnych minimów funkcji.

$$\theta = \theta - \eta * \nabla J(\theta) \tag{4.9}$$

Stochastic Gradient Descent

(nazwa w języku angielskim z powodu braku znalezienia polskiego odpowiednika) Stochastic Gradient Descent *(skrót. GDA)* jest rozwinięciem metody gradientu prostego, bardzo często wykorzystywana w praktyce. Ulepszenie polega na obliczaniu gradientu dla jednego lub niewielkiej ilości przykładów treningowych. Najczęsciej korzysta się z więcej niż jednego przykładu co zapewnia lepszą stabilność oraz wykorzystuje zrównoleglanie obliczeń. SGD zapewnia to większą rozbieżność niż metoda gradiento prostego, co umożliwia znajdowanie nowych lokalnych minimów ale wiąże się z koniecznością zastosowania mniejszego stopnia uczenia.

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

RMSprop

RMSprop umożliwia obliczanie gradientu dla każdego parametru z osobna i zapobiega zmniejszaniu się stopnia uczenia. Algorytm dostosowuje stopień 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. Metoda ta pozwala na obliczanie z osobna gradientu dla każdego parametru oraz każdej zmiany momentum. Zapobiega dodatkowo zmniejszającemu się stopniowi uczenia, a co najważniejsze jest bardzo szybka i pozwala na sprawne uczenie się sieci. W tej metodzie obliczamy dwa momenty m oraz v.

$$\begin{aligned} \hat{m_t} &= \frac{m_t}{1 - \beta_1^t}, \\ \hat{v_t} &= \frac{v_t}{1 - \beta_2^t}, \end{aligned} \tag{4.11}$$

Powyżej obliczone momenty podstawiamy do wzoru

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t \tag{4.12}$$

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 się sieci neuronowej jest bardzo kosztowne obliczeniowo, co wręcz uniemożliwiało trenowanie modeli w przeszłości, a obecnie jest jednym z powodów dużego technologicznego przeskoku w produkcji kart graficznych. Operacje dodawania i mnożenia wektór i macierzy wykonywane są setki milionów razy, które mogą być przyśpieszone dzięki możliwościom zrównoleglania 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 sie uczyć ustala się na co najmniej kilkanaście. W przypadku większych zbiorów danych lub większych modeli sieci zwiększamy ilość epok.
Często spotykaną praktyką w wielu pracach naukowych jest przedstawianie wyników dla sieci po 100 epokach uczenia się.

Testowanie

Model sieci neuronowej może zostać poddany ocenie, dzięki której można określić w jakim stopniu nauczyła się sieci. 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.

Tematy do uwzględnienia: minibatch, nesterov, momentum, softmax, learning rate, categorical crossentropy, Liczebność zbioru (konkretne uzasadnienie konieczności duzej liczby danych), one-hot encoding, hyperparameters: ilosc filtrow, rozmiar filtrow, rozmiar maxpoolingu, mnist, cifar10, ILSVRC

5. Tworzenie zbiorów danych

Wraz z postępem w tworzeniu pracy zmieniały się założenia oraz cele. Głownymi motywatorami zmian były postępy w tworzeniu zbiorów danych oraz samych modeli sieci. Priorytetem było stworzenie zestawu danych umożliwiającego rozwój różnych modeli sieci neuronowych bez konieczności każdorazowego ingerowania w zbiór lub dopasowywania go specjalnie do konkretnej sieci.

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 choć z kośćmi położonymi w różnych jego miejscach.
Kolejną kwestią była konieczność przystosowania sieci do rozpoznawania kości o różnych kolorach ścian, oczek jak i samej barwy 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 więc konkretną formę zestawów danych. Każdy zestaw ma ustaloną barwę tła, kości oraz samego oczka. Każda ścianka sfotografowana od 3 do 30 razy, określając liczebność każdego ze zbiorów miedzy 18 a 180 zdjęć. Zdjęcia w niektórych zestawach poddawane były działaniu zmiennego oświetlenia naturalnego 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

Ilość zdjęć wykonywanych dla danych zestawień kolorystycznych była jak widać mocno zróżnicowana. Największa ilość, zdjęcia z białą kostką, czarnymi oczkami i czerwonym tłem, była związana z testowaniem różnego rodzaju oświetlenia kości w wyniku uczenia różnych modeli sieci. Przeważnie ilość zdjęć wahała się między 4 a 8 na jedną ścianę.

Każdy z obrazów został następnie poddany przekształceniom by zwiększyć ich ilość, konieczną do umożliwienia prawidłowego uczenia się sieci oraz przede wszystkim 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 w stosunku do rozmiaru sieci oraz ilości parametrów do nauczenia się. Po wielu próbach ostatecznie wybrano kąt 15^o, ponieważ dawało to 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, zarówno 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 korzystają 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 kwestie doboru parametrów sieci, nie narzucając osobnych wartości w poziomie i pionie.

5.2 Zbiór obrazów prostokątnych

Po nauczeniu i weryfikacji kilkunastu różnych modeli sieci, zdecydowano się podjąć próbę działania z wykorzystaniem większych obrazów oraz kości rozmieszczonych w miejscach bardziej zróżnicowanych niż jedynie pewien, niewielki obszar w centralnym punkcie obrazu. Kolejne zdjęcia miały również podnieść trudność etapu 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 ułatwiło by identyfikacje kości oraz umożliwiło by detekcję ilości oczek wyrzuconych na kostce o ile tylko kość znalazła by się w obszarze odejmowanym przez kamerę.
Plan zakładał wykorzystanie poprzednio wykonanych zdjęć oraz dodanie nowych zbiorów 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 chęci 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 tworzony zbiór obrazów prostokątnych w pierwszym założeniu miał składać się z obrazów o rozmiarach 320x240 pikseli w skali szarości, co oznaczało 76800 wartości na jedno zdjęcie. W wyniku niepowodzenia procesu uczenia po kilku godzinach, podjęto decyzję o dwukrotnym zmniejszeniu obrazów do 160x120 pikseli. Wartość ta została wybrana ponieważ ilość parame-

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

trów obrazu, wynosząca 19200, była jedynie o 56% większa od ich liczby dla kolorwych 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 efektowych. Inne w dużym uproszczeniu przedstawiały praktycznie niezmieniony 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 w różnych ustawieniach.

Wszystkie operacje umożliwiły osiągnięcie 60 obrazów z każdego z nich 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.

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.

Hipotezy Przed przystąpieniem do tworzenia zbioru danych zrodziła się bardzo duża ilość pytań oraz wątpliwości. Większość z nich dotyczyła kwestii możliwości, że sieć będzie w stanie jakkolwiek rozpoznać ilość oczek. 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 może być umieszczona w różnych miejscach na obszarze całego obrazu, jak również dopuszczalny jest 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 stanowi barwa biała, która decyduje o wartości zwróconej przez sieć. W przypadku kwadratowych zdjęć z kostkami, rozmiar jednego oczka to jedynie 0,21% powierzchnii rozmiaru, a dla obrazów prostokątnych wartość ta maleje do około 0.08%. Oznacza to, że nawet niewielki szum lub zniekształcenia mogą utrudnić prawidłowe rozpoznanie kości.

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 obrazy z czerwonym tłem, białą kością o czarnymi oczkami. Dla zwiększenia liczby zdjęć w zbiorze, zmniejszono kąt obrotu każdego z nich do 5o, 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 co najmniej zdumiewające. Po pierwszej epoce sieć uzyskała 61,98% skuteczności ostatecznie osiągając wynik 99,88% po 25 epokach. Kolejne 10 epok praktycznie nie zmieniło 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 kolorów oraz rozmiarów kości na zdjęciach.

Pierwszy model ze zróżnicowanymi zdjęciami Po stworzeniu modelu który udowodnił że da się zrealizować zadanie stworzenia dobrze działającej sieci dla tego problemu, zabrano się do kolejnego etapu prac. W tym celu wykorzystano opisany w części SŁOWNIK/ZBIORY$_DANYCH zbi$
$W tym model u wykorzystano architekturę jedyni enie znacznie zmienioną w stosunku do użytej w pierwszym$

Sieć uczona była w 25, a następnie na 10 epokach. Po pierwszych 25 epokach uzyskano dokladność 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 pewien dość 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 własnie przetwarzanie wyselekcjonowanych zdjęć.
Zastosowanie 25 epok wystarczyło aby uzyskać dokładność na poziomie 89,23% co jest rezultatem zdecydowanie lepszym niż uzyskane wcześniej 55,64%.

Analiza różnych optymalizatorów Jeśli sama różnica w doborze konkretnych obrazów potrafi tak bardzo zmienić wyniki otrzymane przez sieć, podjęto próbę przetestowania kilku z optymalizatorów dla identycznych sieci oraz zbiorów danych. W tym celu postanowiono użyć optymalizatorów RMSprop oraz SGD, opisanych dokładniej w sekcji SŁOWNIK/OPTYMALIZATORY.

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 dla RMSprop był zdecydowanie najlepszy i wyniósł 84,34% skuteczności. Najgorszy w zestawieniu model korzystający z SGD uzyskał jedynie 36,92% poprawności. Pośrodku obu, okazał się Adam który jak opisane jest wyżej, uzyskał 55,64% po sesji uczenia przez 25 epok. Wyniki te dowiodły, że optymalizator jesty 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 przez co cieszy się ogromną popularnością. Również z tego powodu, pomimo gorszego niż RMSprop wyniku, Adam będzie wykorzystywany w praktycznie wszystkich następnych modelach.

Wypis stworzonych modeli sieci $\mathrm{dice}_c nn 64x64x1 pierwszymodelever, 99.88\% accuracy, 25i10epok(pra$

$\mathrm{dice}_c nn_c ontinue 64x64x1 pierwszymodel, tylkodouczaniedistance51\% accuracy, 10epok$

dice$_c$nn$_a$dam64x64x1pierwszymodelzróżnymizdjęciami, 55%accuracy, wcontinue65%, 25epok, 2048ba

dice$_c$nn$_a$dam$_w$ybrane64x64x1architekturajakdice$_c$nn$_a$dam, zdjeciajedynietezdużymkontrastem(npbe

dice$_c$nn$_r$msprop64x64x1architekturajakdice$_c$nn$_a$dam, RMSpropzamiastAdam, 84%, podouczeniu10ep

dice$_c$nn$_s$gd64x64x1architekturaprawiejakdice$_c$nn$_a$dam, 256zamiast196w2−giejDense, 37%, 25epok, 2

− −

dice$_c$nn$_A$PI64x64x1architekturadocelowojakdice$_c$nn$_a$dam, pierwszymodelAPIipierwszeodkryciero znic

dice$_A$lexNet64x64x1, głębokasiećopartanaAlexNetzILSVRC, 98.88%25epok, 2048batch

generator$_c$omparison64x64x3porównanieuczenia3kanałówRGBzBW, 98.4%20epok, 64batch, 18snaepo

generator$_c$omparison$_g$ray64x64x1porównanie, większybatchsizedlaBWniżRGB, 98.1%20epok, 256bat

API$_S$equential®$_{Keras640x480x1różnicamiędzyAPIaSequential,błedykompilacji}$
API$_S$equential®$_{Keras2120x120x1różnicamiedzyAPIaSequential,poprawny}$
generator$_3$20x240$_A$lexNet, 320x240x3!!, nienauczona, 20epok, 64batch

generator2$_3$20x240$_A$lexNet, 320x240x1, nienauczona, 20epok, 64batch

generator2$_3$20x240, 320x240x1, nienauczona, 20epok, 64batch

$\text{simple}_N N_1 06x79106x79x1, pierwszanauczonazprostokatnymiobrazami, filtrykonwolucyjneniesatut$

$\text{simple}_N N_1 60x120160x120x1, nienauczona, 20epok, 256batch$

$\text{substituting}_1 06x79106x79x1, próbaprzyśpieszeniapracyprzezredukcjeobliczenwsimple_1 06x79jednakz$
$nienauczona, 20epok, 256batch$

$\text{substituting}_1 60x120160x120x1, próbaprzyśpieszeniapracyprzezredukcjeobliczenwsimple_1 06x79jedna$
$nienauczona, 20epok, 256batch$

$\text{subst}_L ReLU_1 06x79106x79x1, próbaprzyśpieszeniaprzezredukcjeilosciobliczenwrazzzastosowaniemL$