POLITECHNIKA ŚLĄSKA

WYDZIAŁ AUTOMATYKI, ELEKTRONIKI I INFORMATYKI

KIERUNEK INFORMATYKA

Projekt inżynierski

Rozpoznawanie gestów i ruchów przy pomocy Sieci Neuronowej

Autor: Dominik Florencki

Kierujący pracą: tytuł oraz imię i nazwisko kierującego pracą (promotora), czcionka 14 pt

Konsultant: tytuł oraz imię i nazwisko konsultanta (w przypadku braku konsultanta wykasować linię)

Gliwice, miesiąc rok (np/. Gliwice, Styczeń 2019) font 12 pt,

Załącznik Nr 2 do Zarz. Nr 97/08/09

Oświadczenie

Wyrażam zgodę/nie wyrażam* zgody na udostępnienie mojej pracy dyplomowej/rozprawy doktorskiej*

……..……., dnia ……

……..……..……
(podpis)

……
(poświadczenie wiarygodności podpisu przez Dziekanat)

** właściwe podkreślić*

Oświadczenie promotora

Oświadczam, że praca „Tytuł pracy dyplomowej inżynierskiej” spełnia wymagania formalne pracy dyplomowej inżynierskiej.

Gliwice, dnia …… ……..……..……

(podpis)

Spis treści

1. Wstęp

W ostatnich latach, tematy uczenia maszynowego, sieci neuronowych oraz uczenia głębokiego zyskują coraz większą popularność, głównie ze względu na coraz bardziej złożone problemy rozpoznawania i przetwarzania obrazów i dźwięków. Jednym z trudniejszych zagadnień jest rozpoznawanie gestów, ruchów ręki człowieka z obrazu cyfrowego. Do rozpoznawania pojedynczych gestów, należy połączyć analizę poszczególnych klatek filmu z sekwencyjnością, kolejnością ich wyświetlania w czasie.
Z dostępnych modeli głębokich sieci neuronowych, duży potencjał mają sieci typu 3D CNN + LSTM. Dlatego celem niniejszej pracy była analiza dwóch architektur wpływu liczby warstw sieci na dokładność modelu oraz budowa aplikacji, pozwalającej użyć stworzonego modelu do rozpoznawania gestów w czasie rzeczywistym w łatwy i przystępny sposób dla użytkownika. W związku z tym przeprowadzono seria eksperymentów, pozwalających na wielowarstwową analizę parametru, budowę modelu oraz aplikacji w celu jego zastosowania.

2. Analiza tematu

Gesty jako język niewerbalny człowieka, odgrywa bardzo ważną rolę w życiu codziennym. System rozpoznawania gestów prowadzi do większego zrozumienia zachowań ludzkich oraz w znaczny sposób może przyczynić się do budowy coraz to lepszych, bardziej naturalnych maszyn-robotów, na wzór i podobieństwo człowieka.

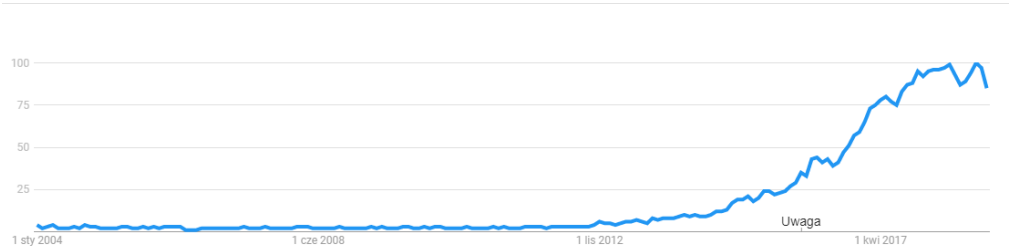
Zagadnienie te jest jednym z podstawowych wyzwań jakie stoją przed wizją komputerową. Sztuczne sieci neuronowe w swej historii były już stosowane w algorytmach klasyfikacji obrazów, jednak znaczny postęp nastąpił przez zbadanie konwolucyjnych sieci neuronowych. [3]

Sieci działają podobnie jak filtry, pozwalając wyodrębnić cechy szczególne. Cechy są wyszukiwane wśród danych o bliskim sąsiedztwie, które następnie są uogólniane. Otrzymane tak dane, są pożądanym zbiorem głównych charakterystyk, które dają ogrom informacji i możliwości w klasyfikacji. Przez to sieć ta jest często stosowana do rozpoznawania obrazów, czyli danych dwuwymiarowych (sieci konwolucyjne 2D). Autorzy artykułu [6] posługując się ideą sieci konwolucyjnej 2D, stworzyli model uczący się poprzez obrazy ręcznie pisanych cyfr, ich rozpoznawania oraz porównali precyzje zastosowanego modelu do podobnych implementacji. Dla zbioru MNIST osiągnęli dokładność rzędu 99.72%.

W przypadku zwiększenia liczby wymiarów do trzech, można użyć sieci konwolucyjnych 3D, ale też często łączy się sieci konwolucyjne 2D z sieciami rekurencyjnymi.

Sieci rekurencyjne są powszechnie stosowane w celu symulacji pamięci stanu poprzedniego. Jak każda sieć, składa się ona z neuronu, który na dodatek posiada sprzężenie zwrotne. Pozwala to na ponowne odwołanie się do danych, które zostały wyliczone chwilę wcześniej. Dzięki zastosowaniu takiego mechanizmu, następuje symulacja pamięci krótkotrwałej lub długotrwałej. W przypadku pamięci krótkotrwałej, najczęściej spotykaną implementacją jest sieć LSTM (Long-Short Term Memory) i GRU (gated recurrent unit), gdzie występuje możliwość zapomnienia stanów poprzednich. Natomiast sieci długotrwałe pamiętają wszystkie stany poprzednie. Kolejnym ważnym przełomem w dziedzinie uczenia maszynowego, było stworzenie

wielowarstwowego modelu uczenia sieci neuronowych. Już w latach 90 zostały zaproponowane odpowiednie algorytmy, lecz były one mało skuteczne. Dopiero w 2006 r. Hinton pokazał inne podejście do tworzenia i uczenia sieci bardziej złożonych. Metoda ta zakłada stworzenie hierarchicznego modelu warstw kolejnych sieci oraz uczenie go warstwa po warstwie, nadzorując cały ten proces [4]. Zwiększyło to jakość odwzorowania sieci z wieloma warstwami względem jednowarstwowych. Przyczyniło się to do znacznego wzrostu dokładności algorytmów i rozwoju nowej metodyki uczenia sieci głębokich (ang. Deep networks), uczenia głębokiego (ang. Deep learning). Na rysunku nr.3 zobrazowano, liczbę wyszukiwań słów „deep learning" na całym świecie na przestrzeni lat 2004-2018 używając najpopularniejszej wyszukiwarki internetowej Google.



Rys. 3. Wykres liczby wyszukań słowa "deep learning" na przestrzeni lat 2004-2018 na całym świecie [5]

Analiza gestów wykonywanych przez człowieka jest analizą pewnego rodzaju filmu. Film jest natomiast serią zdjęć, obrazów 2D o pewnej wysokości i szerokości w przestrzeni czasowej. Wszystko to składa się na realny obiekt trójwymiarowy. Obrazy 2D są analizowane przez wiele modeli, zgodnie z ImageNet Large-Scale Visual Recognition Challenge, od roku 2012 na topowych miejscach znajdują się sieci o architekturze CNN [7]. Zgodnie z artykułem [8], dodawanie większej ilości warstw do modelu, przyczynia się do zmiany jego dokładności. Dzięki zastosowaniu takiego zabiegu, dla danych CIFAR-100 osiągnięto dokładność 85.59% [8]. Każda warstwa sieci konwolucyjnej składa się z dwóch typów warstw pośrednich. Warstwy konwolucyjnej, odpowiedzialnej za analizę cech w obrazie oraz warstwy łączącej (pooling layer) do zredukowania rozmiaru, cech, jak i złożoności obliczeniowej. W celu zebrania danych obliczonych przez sieć tablica wielowymiarowa wyników jest spłaszczana i łączona poprzez warstwę gęstą (ang. dense layer lub fully connected layer). Tak otrzymane dane są najczęściej jednowymiarowe i można zredukować je do pożądanej liczby wyników. Liczba poszczególnych warstw, jak i ich parametryzacja jest indywidulana dla każdego modelu i wpływa w znacznym stopniu na jego

dokładność. Dla danych ImageNet [9], została przeprowadzona analiza jakości modelu z różną liczbą warstw, kolejno 18, 34, 50, 101, 152. Wpłynęło to na potrzebną moc obliczeniową od 1.8*10^9 FLOPs (dla 18 warstw) do 11.3*10^9 FLOPs (dla 152 warstw), czyli potrzebna moc zwiększyła się 6.27-krotnie. Również dokładność modelu wzrosła, najlepsze wyniki uzyskała sieć o największej liczbie warstw [9]. Nie tylko jednak liczba warstw wpływa na jakość modelu, ale również jego implementacja. Porównanie oryginalnego modelu, jego nowej implementacji oraz zastosowanie nowego pomysłu, daje różne wyniki, pod względem dokładności jak i złożoności obliczeniowej, dla tych samych danych [8]. Sieci CNN są bardzo dobrym rozwiązaniem w celu analizy cech obrazów, szczególnie dwuwymiarowych.

Sieci rekurencyjne z pamięcią krótkotrwałą typu LSTM i GRU, służą często do wykrywania sekwencyjnych cech, ale również w celu klasyfikacji. W literaturze dostępne są dane dotyczące przeprowadzonych serie eksperymentów porównujące oba modele, bazujące na tych samych danych [11]. Autorzy wykazali zbliżoną jakość modeli ze względu na trafność klasyfikacji. Widoczna jest natomiast różnica czasu realizacji uczenia się.

Wielu badaczy stworzyło odmienne wersje architektury LSTM, zmieniając jej głębokość oraz poszczególne parametry (funkcje aktywacyjne, wielkość warstw ukrytych, itp.). Na podstawie użytych danych Hutter Prize seria eksperymentów wykazała iż analizowane modele wykazały błąd testowy od 1.25 do 1.67 [12].

W celu analizy filmów, czyli obrazów w czasie, często używa się architektury 3DCNN oraz 2DCNN+RNN. Dwa wymiary odpowiadają za wysokość i szerokość obrazu, a trzeci wymiar to czas. Dla danych filmowych, nagranych z czterech kamer, stworzono model postaci szkieletowej. Użycie modelu 3DCNN daje bardzo dobre wyniki, które zostały przedstawione w artykule [13]. Zostało zastosowane wstępne przetwarzanie obrazu, obcinanie, w celu zredukowania danych oraz normalizacja kontrastowa dla każdego kanału koloru, żeby zredukować różnice w kolorach. Najlepsze wyniki uzyskał model, składający się z 5 warstw konwolucyjnych oraz trzech warstw łączących [13]. Model 3D CNN został również użyty w analizie ruchu w sporcie, ekspresji twarzy oraz gestów dłońmi [16]. W pracy wykazano pierwsze osiągnięcie dokładności tego typu modelu na ponad 90%.
Jak wykazano w publikacji [15] dane filmowe z jednej kamery mogą być też przetwarzane przez połączenie sieci konwolucyjnych oraz sieci rekurencyjnych. W celu wychwycenia punktów charakterystycznych postaci szkieletowej, została użyta architektura składająca się z dwóch warstw sieci CNN, a następnie wyniki przekazano są do sieci rekurencyjnej LSTM. Użycie

dwóch architektur pozwoliło osiągnąć wysokie rezultaty na tle innych modeli jedno-architekturowych [15].

W publikacji [14], porównano dwa modele 3DCNN i 2DCNN + LSTM, dla których została przeprowadzona seria eksperymentów. Dla tych samych danych, związanych z rozpoznawaniem emocji w głosie zastosowano różne wersje modeli różniące się architekturą, liczbą warstw oraz parametrami. Dane dźwiękowe zostały przekonwertowane na wykres widma amplitudowego sygnału (spektrogram) w skali logarytmicznej. Zostały utworzone tablice dwuwymiarowe rozłożone w czasie. Takie informacje były przekazywane do różnych wersji modeli. Następnie zebrano wyniki i przedstawione porównanie wszystkich modeli. Najtrafniejszy okazała się architektura 3DCNN, z trzema warstwami konwolucyjnymi.

Używanie sieci głębokich często wiąże się ze znacznym zużyciem mocy obliczeniowej. Dlatego też w procesie ewaluacji modeli, często są używane procesory graficzne GPU wraz z CPU z dużą liczbą pamięci RAM. Dla analizy rozpoznawania obrazów oraz rozpoznawania liczb został użyty serwer z dwoma procesorami Intel Xeon E5-2678 V3 2.5 GHz z 8 NVIDIA Tesla TiTanXp GPU, 512 GB pamięci, 240 GB SSD [6]. W celu zastosowania modelu generującego szkielet człowieka na podstawie obrazu z czterech kamer, użyto sprzętu Intel Xeon E5 CPU oraz Nvidia 1080 GPU, co pozwalało na przetwarzanie 5 klatek filmu w czasie 1ms – 13ms [13].

3. Wymagania i narzędzia

Praca została podzielona na dwa duże zadania. Pierwsze to zbudowanie modelu sieci neuronowej w oparciu o architekturę 3DCNN + LSTM, a drugie to zbudowanie aplikacji do ewaluacji tego modelu.

3.1. Model sieci neuronowej.

Wymaganie funkcjonalne:

1) Model powinien obsługiwać dane Jester [17]
2) Model powinien mieć możliwość wyświetlenia całej architektury i parametrów
3) Model powinien mieć możliwość zmiany liczby warstw, zmieniając tylko jeden parametr
4) Model powinien mieć możliwość wyświetlenia poszczególnych kształtów danych wielowymiarowych w poszczególnych architekturach, 3D CNN, LSTM, FC

Wymagania niefunkcjonalne:

1) model powinien zostać stworzony w oparciu o język python
2) model powinien zostać stworzony w oparciu o bibliotekę pytorch
3) model powinien zostać stworzony w oparciu o architekturę 3D CNN, LSTM i FC
4) model powinien zajmować się zmianą wymiarów danych, między kolejnymi warstwami

Wybrane narzędzia.

W celu realizacji modelu sieci neuronowej wybrana została biblioteka PyTorch, z uwagi na łatwość użytkowania oraz szybkość działania. W artykule [18] została przeprowadzona analiza czasowa dla architektury LSTM, gdzie biblioteka PyTorch osiągnęła wysokie wyniki na tle innych.

3.2. Aplikacja do ewaluacji modelu.

Wymagania funkcjonalne:

1) aplikacja powinna mieć jedno okno
2) aplikacja powinna mieć możliwość wybrania pliku z modelem przez eksplorator plików
3) aplikacja powinna mieć możliwość wybrania pliku zawierającego listę klas wynikowych do przewidzenia przez eksplorator plików
4) aplikacja powinna wyświetlać architekturę warstwową wraz z użytymi w niej parametrami wybranego modelu
5) aplikacja powinna pobierać obraz z domyślnej kamery internetowej komputera
6) aplikacja powinna wyświetlać obraz z kamery internetowej
7) aplikacja powinna wyświetlać aktualne wartości wyliczone przez model dla każdej klasy wynikowe osobno
8) aplikacja powinna wyświetlać opóźnienie (różnicę czasową) wyświetlanych danych w stosunku do czasu pobrania tych danych
9) aplikacja powinna ewaluować model dopiero po wybraniu zarówno pliku z modelem jak i pliku z listą klas wynikowych

Wymagania niefunkcjonalne:

1) Aplikacja powinna być napisana na system operacyjny Windows
2) Aplikacja powinna być napisana z użyciem wielowątkowości

3) Aplikacja powinna umożliwić komunikację między wątkami używając sygnałów
4) Aplikacja powinna obsługiwać pliki typu .csv dla klas wynikowych
5) Aplikacja powinna obsługiwać pliki typu .pth.tar dla modelu seici
6) Aplikacja powinna zostać skompilowana wraz ze wszystkimi bibliotekami w celu odtworzenia jej na innych urządzeniach
7) Aplikacja powinna wyświetlać obraz z kamery nie większy niż 640x480 px
8) Aplikacja powinna skalować obraz do 160x120 px dla modelu
9) Aplikacja powinna obcinać, konwertować na tensor i normalizować obraz w taki sam sposób w jaki model był tworzony

Wybrane narzędzia.

W celu implementacji aplikacji, został wykorzystany język programowania Python oraz biblioteka PyQt5, ze względu na łatwość użytkowania i obsługę wielu wątków. Zarządzaniem i obsługą modelu sieci neuronowej zajmuje się biblioteka PyTorch. W budowie aplikacji zostało użyte środowisko PyCharm w wersji 2018.2.1 firmy JetBrains.

4. Specyfikacja zewnętrzna

4.1. Model sieci neuronowej.

W celu wygenerowania modeli sieci neuronowej należy użyć gotowej architektury, zainstalować interpreter języka Python w wersji min. 3 oraz bibliotekę PyTorch w wersji min. 0.4.0. Architektura została zaprojektowana pod kątem dwóch systemów operacyjnych Windows oraz Linux. Główna konfiguracja znajduje się w plikach .json. Kolejne pliki to:

- config_quick_train.json – zawiera konfigurację treningową modeli dla danych treningowych
- config_quick_testing.json – zawiera konfigurację dla mniejszej liczby danych w celu przetestowania czy architektura jest poprawnie zbudowana

Implementacja modeli sieci neuronowej znajduje się w plikach model.py oraz modelCNNLSTM.py. Po wprowadzeniu zmian w modelach, można przetestować, czy sama sieć jest poprawnie skonstruowana, czy wszystkie parametry się zgadzają, kompilując cały plik z modelem. Zostanie wyświetlony w wierszu poleceń schemat sieci neuronowej oraz kolejne wymiary danych między modułami (rys. 11).

```
CNN in:      torch.Size([5, 3, 18, 84, 84])
CNN out:     torch.Size([5, 512, 2, 5, 5])
LSTM in:     torch.Size([5, 512, 50])
LSTM out:    torch.Size([5, 512, 50])
FC in:       torch.Size([5, 25600])
FC out:      torch.Size([5, 5])

MyNetwork(
  (CNN): CNN(
    (list): ModuleList(
      (0): Sequential(
        (0): Conv3d(3, 64, kernel_size=(3, 3, 3), stride=(1, 1, 1),
padding=(1, 1, 1))
```

```
        (1): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ELU(alpha=1.0)
        (3): MaxPool3d(kernel_size=(1, 2, 2), stride=(1, 2, 2), padding=0,
dilation=1, ceil_mode=False)
      )
      (1): Sequential(
        (0): Conv3d(64, 128, kernel_size=(3, 3, 3), stride=(1, 1, 1),
padding=(1, 1, 1))
        (1): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ELU(alpha=1.0)
        (3): MaxPool3d(kernel_size=(2, 2, 2), stride=(2, 2, 2), padding=0,
dilation=1, ceil_mode=False)
      )
      (2): Sequential(
        (0): Conv3d(128, 256, kernel_size=(3, 3, 3), stride=(1, 1, 1),
padding=(1, 1, 1))
        (1): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ELU(alpha=1.0)
        (3): MaxPool3d(kernel_size=(2, 2, 2), stride=(2, 2, 2), padding=0,
dilation=1, ceil_mode=False)
      )
      (3): Sequential(
        (0): Conv3d(256, 512, kernel_size=(3, 3, 3), stride=(1, 1, 1),
padding=(1, 1, 1))
        (1): BatchNorm3d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ELU(alpha=1.0)
        (3): MaxPool3d(kernel_size=(2, 2, 2), stride=(2, 2, 2), padding=0,
dilation=1, ceil_mode=False)
      )
      (4): Sequential(
        (0): Conv3d(512, 512, kernel_size=(3, 3, 3), stride=(1, 1, 1),
padding=(1, 1, 1))
        (1): BatchNorm3d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ELU(alpha=1.0)
        (3): MaxPool3d(kernel_size=(1, 1, 1), stride=(1, 1, 1), padding=0,
dilation=1, ceil_mode=False)
      )
    )
  )
  (LSTM): LSTM(
    (lstm_layer1): LSTM(50, 50, num_layers=2, batch_first=True)
  )
  (FC): FC(
    (list): ModuleList(
      (0): Linear(in_features=25600, out_features=512, bias=True)
    )
    (fc_layer2_act): ELU(alpha=1.0)
    (fc_layer3): Linear(in_features=512, out_features=5, bias=True)
  )
)
```

Rys. 11. Przykładowy schemat sieci neuronowej wraz z kolejnymi wymiarami danych dla poszczególnych modułów, wyświetlany w wierszu poleceń.

Program treningowy oraz testowy należy uruchomić korzystając z flagi „--config" oraz przy braku wspierania architektury CUDA w komputerze: „--use_gpu=False". Przykładowy zapis parametrów znajduje się na rys. 12.

```
train.py --config configs/config_quick_train.json –use_gpu=False
```

Rys. 12. Przykładowe wywołanie programu treningowego bez użycia GPU

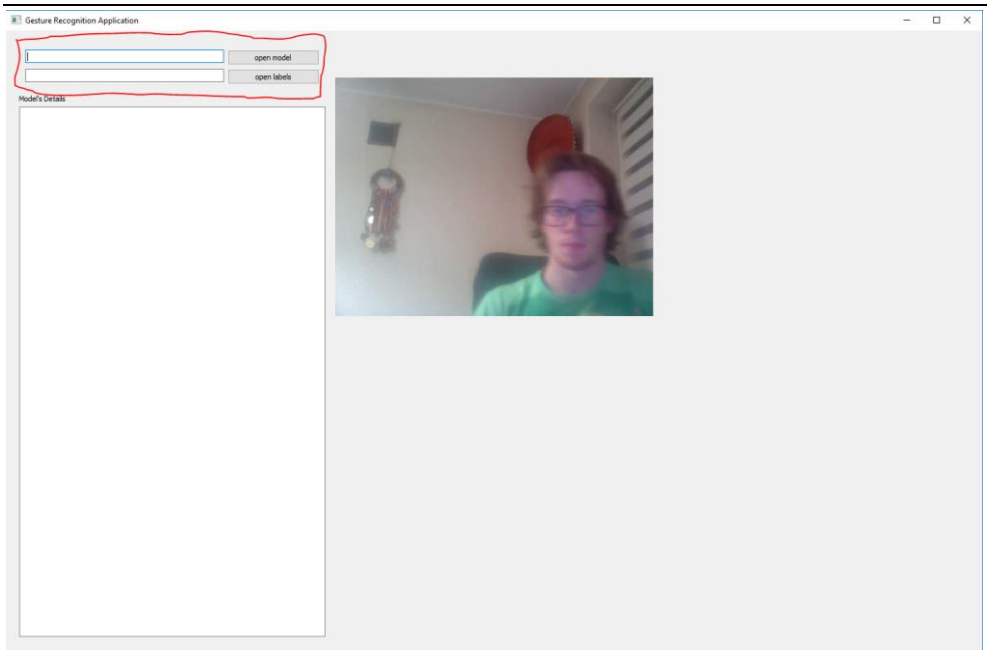
Po wywołaniu programu treningowego, zostaną wyświetlone w konsoli Python kolejne informacje, takie jak aktualny współczynnik uczenia (ang. learning rate), koszt (ang. loss), dokładność modelu dla kolejnych danych treningowych oraz na końcu każdego cyklu jakość modelu dla danych testowych. Informacje te będą wyświetlane dla każdego cyklu. Ponadto zostanie wygenerowany wykres zależności kosztu do ilości cykli w miejscu wskazanym przez plik konfiguracyjny, funkcja ta nazywana jest funkcją kosztu.

W czasie trwania treningu sieci są tworzone pliki typu .pth.tar, kolejno o nazwach checkpoint oraz best_model. Plik checkpoint.pth.tar zawiera model najmłodszy, tzn. model wygenerowany po zakończeniu ostatniego cyklu. Natomiast plik best_model.pth.tar zawiera najlepszy model, jeśli chodzi o dokładność na danych testowych spośród wszystkich cykli programu. Takie pliki można użyć bezpośrednio w aplikacji do ewaluacji modelu.

4.2. Aplikacja do ewaluacji modelu sieci neuronowej.

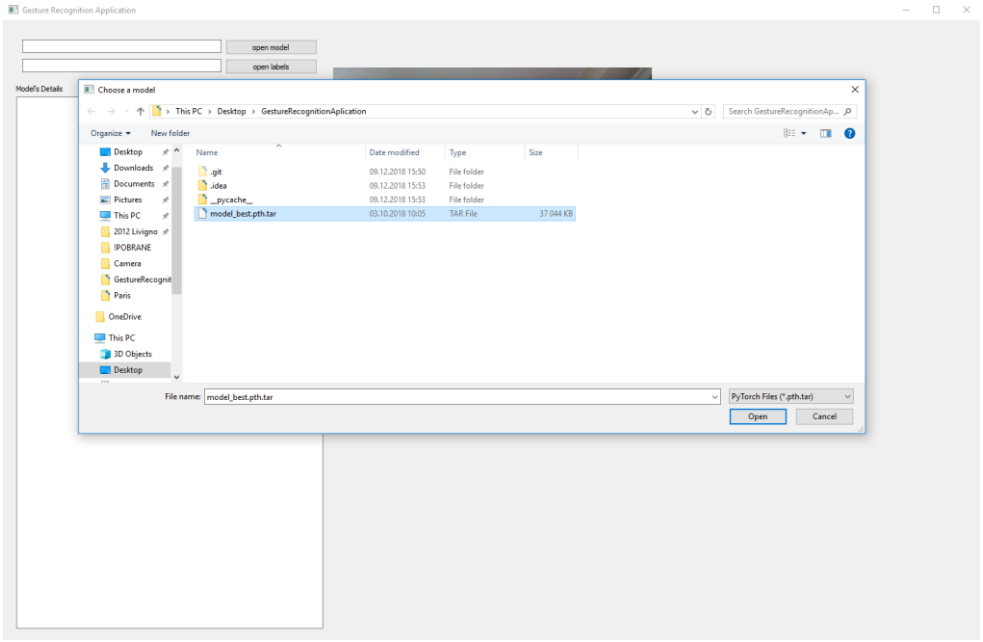
W celu włączenia aplikacji należy uruchomić plik z rozszerzeniem .exe na maszynie z systemem Windows. Ścieżka do pliku to `GestureRecognitionAplication\dist\__main__\__main__.exe`
Wszystkie potrzebne biblioteki dynamiczne (.dll) znajdują się w tym samym katalogu co plik wykonywalny (.exe).

Aplikacja pozwala wybrać model w celu ewaluacji przy pomocy kamery internetowej komputera oraz pozwala wybrać plik typu .csv w celu dostarczenia informacji jakie klasy wynikowe są pożądane. Rys.13.



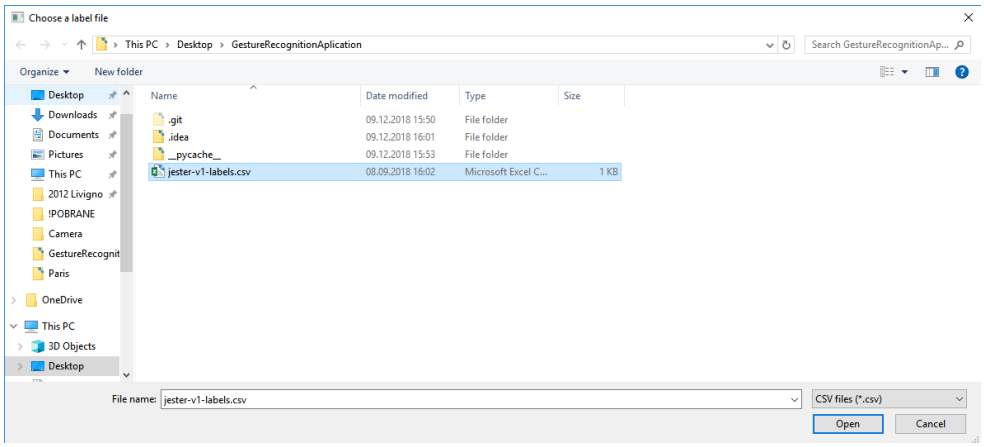
Rys. 13. Ekran startowy aplikacji do ewaluacji modelu z zaznaczonym miejscem do wybrania modelu oraz pliku z klasami wynikowymi.

Po naciśnięciu przycisku „open label” zostanie wyświetlone okno menadżera plików w celu wybrania odpowiedniego modelu. Tylko modele o rozszerzeniu .pth.tar mogą zostać wybrane. Rys. 14.

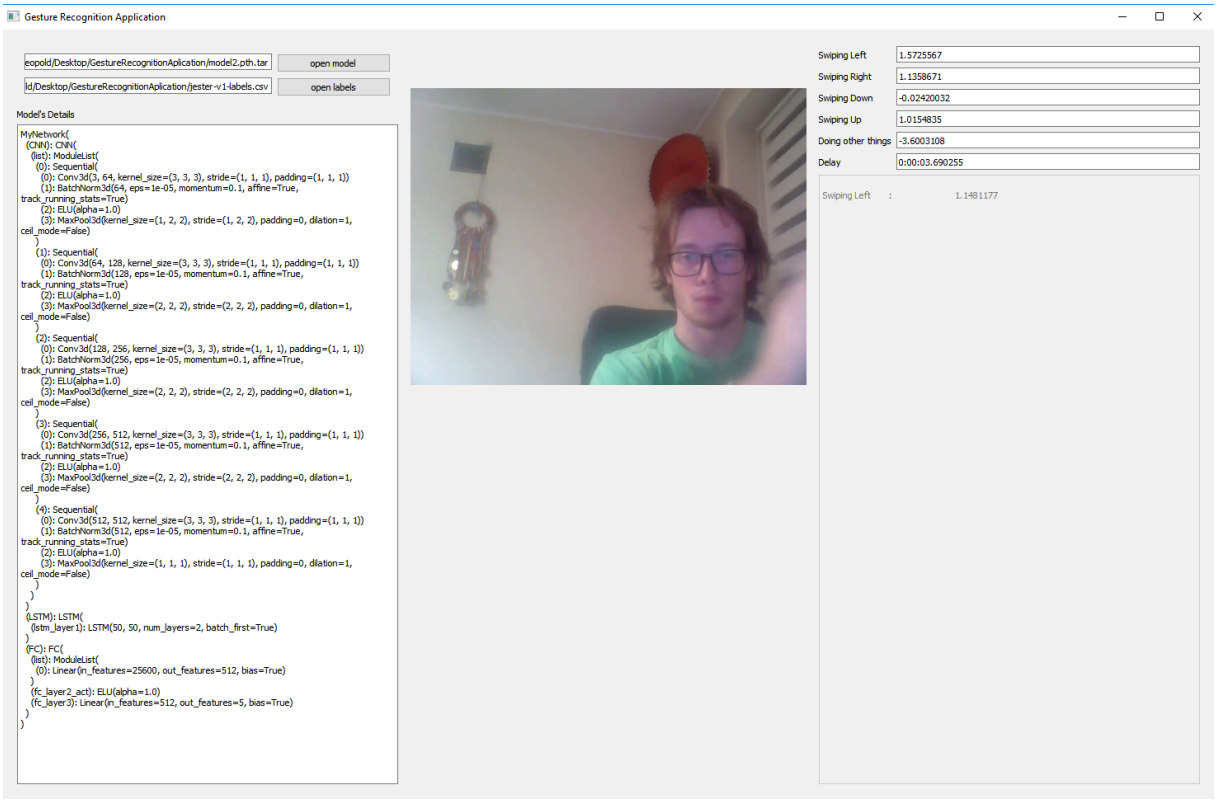


Rys. 14. Wybranie modelu w aplikacji

Po naciśnięciu przycisku „open label” zostanie wyświetlony menadżer plików w celu wybrania pliku z pożądanymi klasami wynikowymi. Tylko pliki o rozszerzeniu .csv są wspierane, możliwe do wybrania. Rys. 15.



Po wybraniu zarówno pliku z modelem, jak i pliku z klasami wynikowymi. Program generuje opis modelu w rubryce „Model’s Details” oraz z prawej strony okna klasy wynikowe, wraz z wartościami. Dodatkowo jest prezentowana informacja o opóźnieniu w rybryce „Delay”, opisująca przesunięcie czasowe, wyświetlanego obrazu z kamery do wyświetlenia wyników, po przeliczeniu przez sieć. Poniżej klas wynikowych, program prezentuje listę wykrytych gestów wraz z ich wartościami. Rys. 16.



Rys. 16. Zrzut działania programu.

5. Specyfikacja wewnętrzna

5.1. Tworzenia modelu sieci neuronowej.

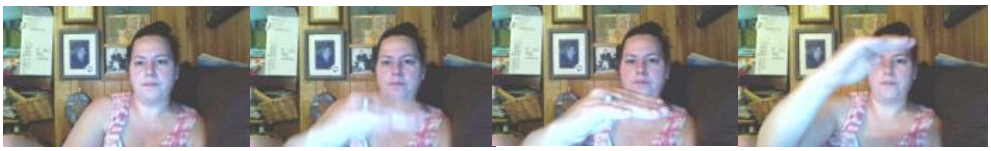
Architektura aplikacji do przetwarzania modelu została zaczerpnięta z [19] i [20] na licencji MIT. Zajmuje się pobieraniem danych, gromadzeniem i reprezentacją wyników przy użyciu wykresów oraz zapisywaniem najskuteczniejszego modelu w pamięci lokalnej.

Firma „The Twenty Billion Neurons" udostępnia w celach naukowych zbiór gestów [1]. Znajduje się w nim 148 092 filmów, podzielonych na część treningową 118 562 oraz testową 14 747, dla 27 różnych gestów. Z powodu bardzo dużej liczby danych i braku dostępu do sprzętu komputerowego o wysokiej mocy obliczeniowej, na poczet eksperymentów jak i całej pracy, dane zostały ograniczone do pięciu gestów i odpowiadającej im liczbie danych treningowych oraz testowych (tab. 1).

Gest	Liczba filmów treningowych	Liczba filmów testowych
Ruch ręką w dół (swiping down)	1401	164
Ruch ręką w lewo (swiping left)	1403	176
Ruch ręką w prawo (swiping right)	1335	172
Ruch ręką w górę (swiping up)	1450	155
Łącznie	5589	667

Tab. 1 Lista gestów wraz z odpowiadającą liczbą danych dla etapu treningowego jak i testowego.

Dane zostały podzielone na foldery, gdzie nazwa każdego folderu to kolejna liczba od 1 do 148092. Każdy folder zawiera różną liczbę obrazów formatu .jpg, wyłonionych z filmów przy prędkości 12 klatek na sekundę. Nazwy kolejnych obrazów to liczby, zaczynające się od 1. Każda klatka ma wymiary 176x100 px, rozdzielczość w poziomie jak i w pionie 96 dpi oraz głębokość bitową 24. Przykładowe obrazy zostały przedstawione na rys. 5. Ponieważ zbiór danych został ograniczony, lista użytych filmów treningowych znajduje się w pliku typu .csv.



Rys. 5.

W realizacji modelu została wybrana liczba klatek równa 18, ponieważ liczba klatek w poszczególnych folderach wahała się od 18 wzwyż.

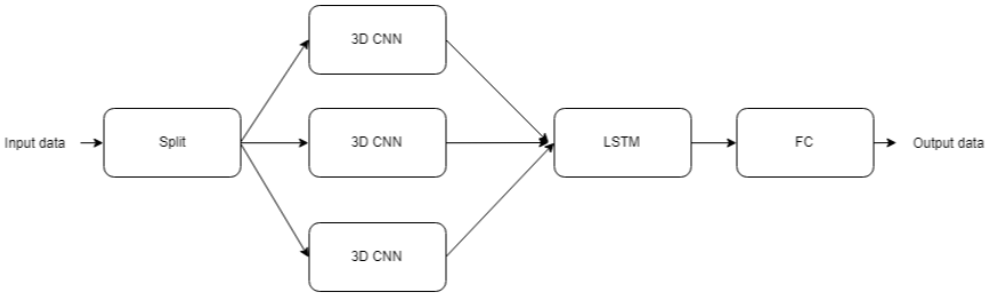
Po wczytaniu danych do pamięci programu, dane są kolejno przycinane do wymiarów 84x84 px względem środka obrazu, zamieniane na „tensor” i normalizowane. Tak otrzymane dane mają postać tabeli czterowymiarowej. Dodatkowo jako piąty wymiar dochodzi wielkość zbioru danych z jaką uczy się symultanicznie sieć neuronowa (batch size), w pracy przyjęto wartość 5. Po wczytaniu i przetworzeniu obrazów, format danych wejściowych sieci neuronowych ma postać:

B x C x N x H x W,

gdzie
B - reprezentuje batch size, czyli 5
C - liczbę kanałów kolorów, dla tych danych równą 3, ze względu na RGB
N - liczbę klatek, czyli 18
H - wysokość obrazu, czyli 84
W - szerokość obrazu, czyli 84

W pracy zostały zaimplementowane dwie architektury, pierwsza jest połączniem wielowarstwowej sieci 3D CNN z LSTM (rys. 6), druga jest połączeniem 3 równoległych wielowarstwowych sieci 3D CNN z LSTM (rys.7).

Rys 6 Schemat sieci 3D CNN + LSTM



Rys. 7 Schemat sieci równoległej 3D CNN + LSTM

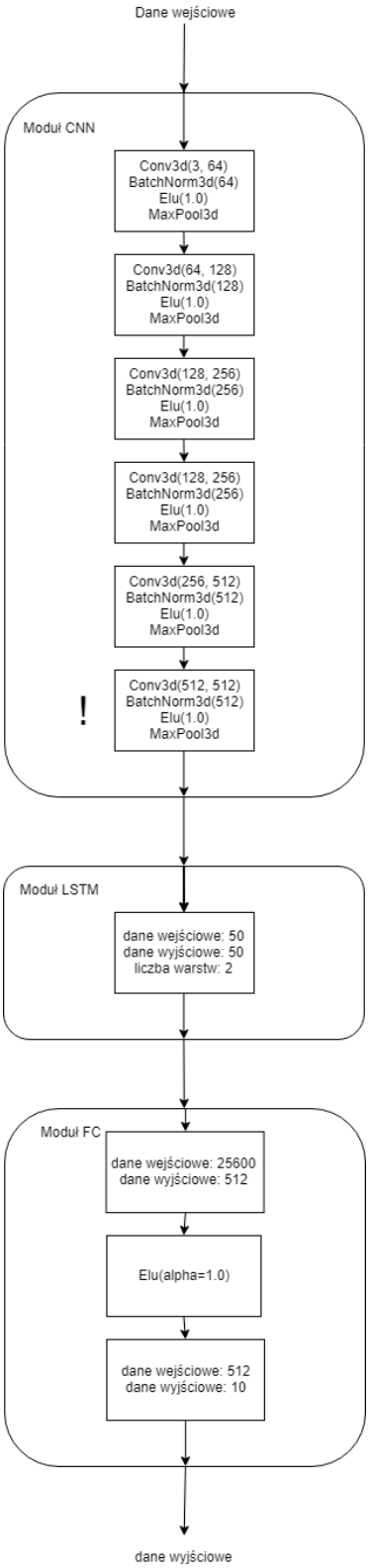
5.1.1. Sieć 3D CNN + LSTM

Dane są przetwarzane w sposób liniowy przechodząc z jednego modułu do drugiego. Kolejne wymiary danych w środku sieci neuronowej na każdym etapie modelu, zostały przedstawione na tab. 8.

```
CNN in:     [5, 3, 18, 84, 84]
CNN out:    [5, 512, 2, 5, 5]
LSTM in:    [5, 512, 50]
LSTM out:   [5, 512, 50]
FC in:      [5, 25600]
FC out:     [5, 5]
```

Tab. 8.

Pełny schemat wraz z parametrami został przedstawiony na rys. 9.


Dane wejściowe
Moduł CNN
Conv3d(3, 64)
BatchNorm3d(64)
Elu(1.0)
MaxPool3d
Conv3d(64, 128)
BatchNorm3d(128)
Elu(1.0)
MaxPool3d
Conv3d(128, 256)
BatchNorm3d(256)
Elu(1.0)
MaxPool3d
Conv3d(128, 256)
BatchNorm3d(256)
Elu(1.0)
MaxPool3d
Conv3d(256, 512)
BatchNorm3d(512)
Elu(1.0)
MaxPool3d
!
Conv3d(512, 512)
BatchNorm3d(512)
Elu(1.0)
MaxPool3d
Moduł LSTM
dane wejściowe: 50
dane wyjściowe: 50
liczba warstw: 2
Moduł FC
dane wejściowe: 25600
dane wyjściowe: 512
Elu(alpha=1.0)
dane wejściowe: 512
dane wyjściowe: 10
dane wyjściowe

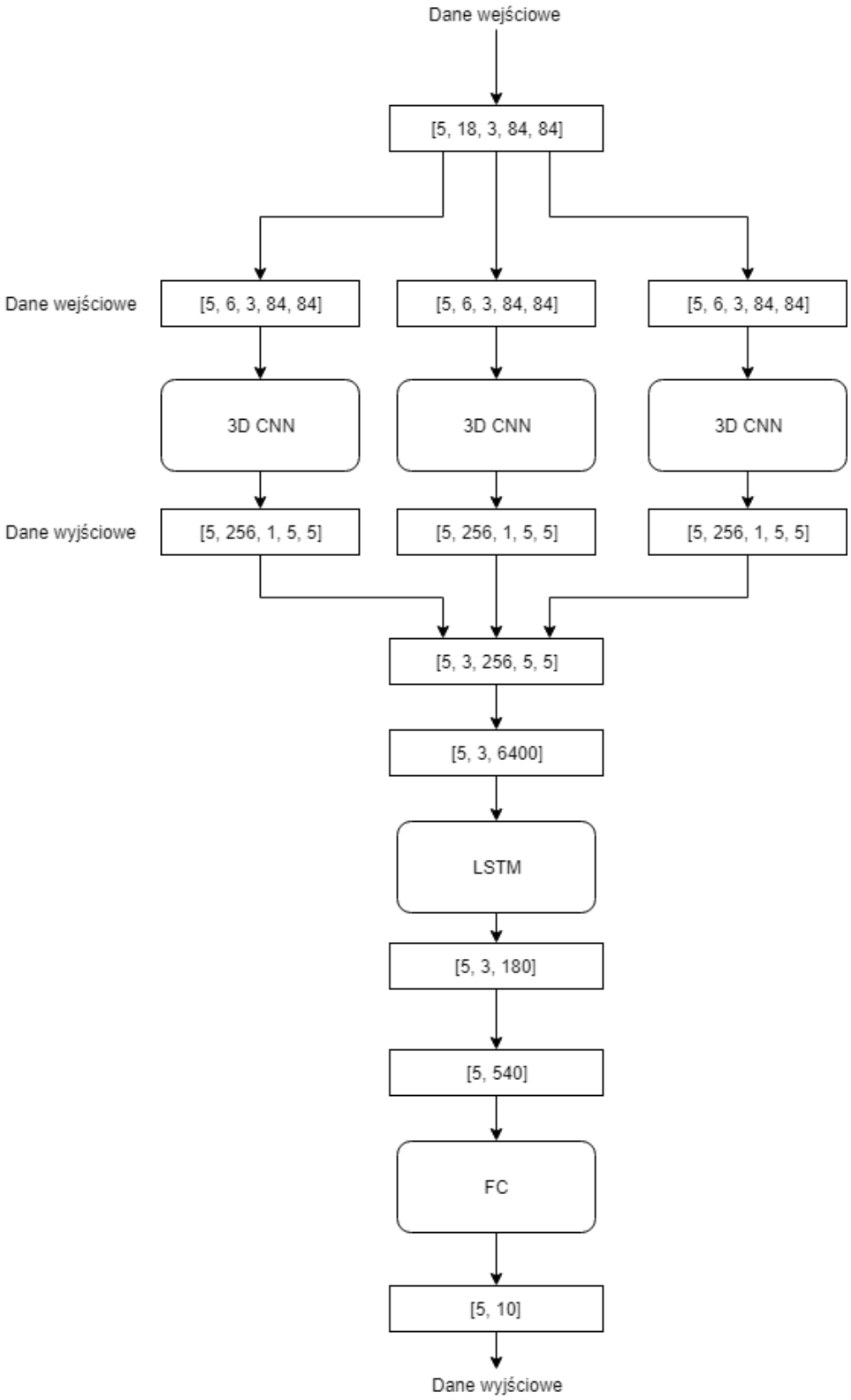
Rys. 9. Pełny schemat sieci 3D CNN + LSTM

Moduł CNN składa się z wielu warstw. Każda warstwa składa się z warstwy sieci konwolucyjnej (conv3d), warstwy normalizującej (BatchNorm3d), warstwy aktywacyjnej (jako algorytm został wybrany Elu) oraz warstwy łączącej (MaxPooling3d). Zastosowano 4 takie warstwy kolejno zwiększając liczbę kanałów od 3 do 512. Piąta warstwa w tym module (zaznaczona na rys. 9 wykrzyknikiem „!") to warstwa nie zmieniająca wymiarów danych, jedynie służy wyodrębnieniu pożądanych cech.

5.1.2. Sieć równoległa 3D CNN + LSTM

Sieć ta opiera się na podziale danych wejściowych na trzy moduły CNN. Po wyliczeniu wszystkich cech charakterystycznych następuje złączenie wszystkich wyników na powrót i przekazanie ich w celu wychwycenia kolejności do modułu LSTM, a następnie spłaszczenie wyników do pożądanej ilości klas wynikowych.

W tym rozwiązaniu przetwarzanie danych jest o wiele trudniejsze. Kolejne kroki przetwarzania tablicy pięciowymiarowej dla poszczególnych modułów zostało przedstawione na rys. 10.


Dane wejściowe
[5, 18, 3, 84, 84]
Dane wejściowe
[5, 6, 3, 84, 84]
[5, 6, 3, 84, 84]
[5, 6, 3, 84, 84]
3D CNN
3D CNN
3D CNN
Dane wyjściowe
[5, 256, 1, 5, 5]
[5, 256, 1, 5, 5]
[5, 256, 1, 5, 5]
[5, 3, 256, 5, 5]
[5, 3, 6400]
LSTM
[5, 3, 180]
[5, 540]
FC
[5, 10]
Dane wyjściowe

Rys. 10. Analiza przekształceń tablicy danych w poszczególnych krokach.

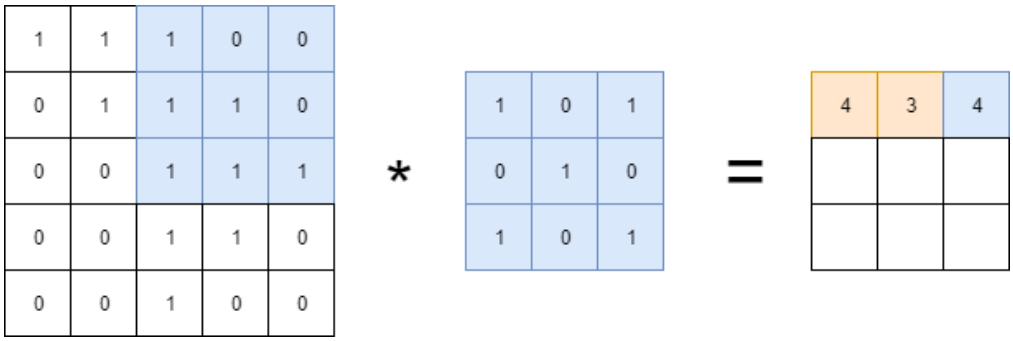
Każdy z trzech modułów 3D CNN został podzielony na cztery warstwy konwolucyjne. Oznacza to, że w skład całego modelu wchodzi dwanaście warstw konwolucyjnych. Szczegółowy schemat sieci równoległej 3D CNN + LSTM został przedstawiony na rys. 11.

Wstawić schemat sieci

Zarówno sieć 3DCNN+LSTM jak i równoległa sieć 3DCNN+LSTM zawierają podobne warstwy wewnętrzne w modułach CNN, dlatego też algorytmy działające wewnątrz będą zbliżone.

Warstwa konwolucyjna (conv3d) działa na zasadzie filtra, odpowiedzialna jest za analizę danych i znalezienie relacji pomiędzy nimi. Posiada szereg parametrów, takich jak liczba kanałów wejściowych, liczba kanałów wyjściowych, wielkość pola recepcyjnego (ang. kernel), krok (ang. stride), oraz wypełnienie (ang. padding).

Wielkość pola recepcyjnego, inaczej wielkość filtra pozwala zmniejszyć wielkość pierwotnego obrazu zachowując jego cechy szczególne. Przykład działania dla danych dwuwymiarowych 5x5 i dla filtra o wymiarach 3x3 znajduje się na rys. 11.



Rys. 11. Przykład działania warstwy konwolucyjnej dla obrazu o wymiarach 5x5 i filtra 3x3.

Kolejne tablice o wymiarach 3x3 są przesuwane horyzontalnie i wertykalnie względem obrazu pierwotnego, przez co finalny obraz jest mniejszy, lecz zawiera w sobie więcej danych. Wielkość przesunięcia we

wszystkich płaszczyznach jest definiowana przez krok (ang. stride). Może zaistnieć sytuacja, gdzie wielkość filtra oraz krok nie będzie się idealnie pokrywać z obrazem wejściowym. W celu uniknięcia tego problemu, stosowane jest tzw. zerowe wypełnienie (ang. padding). Wypełnia ona część obrazu zerami, tak by filtr zawsze pasował do danych, zgodnie ze wzorem 1, gdzie K reprezentuje rozmiar filtra.

$$wypelnienie = \frac{(K-1)}{2}$$

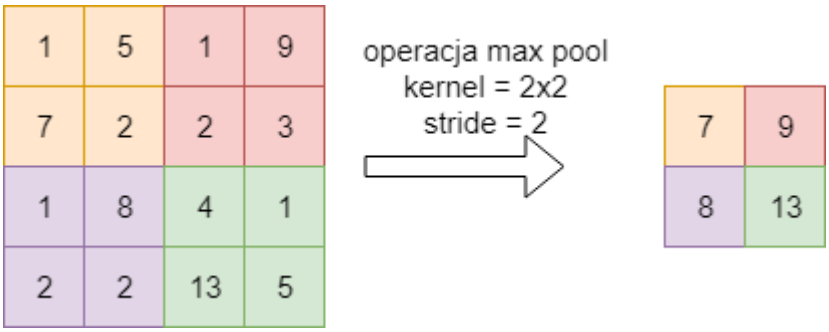
Wzór 1. Zależność wypełnienia od wielkości filtra

Następną warstwą przetwarzającą obraz jest warstwa funkcji aktywacyjnej, używająca algorytm ELU, który został przedstawiony na wzorze 2.

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}, \quad f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ f(x) + \alpha & \text{if } x \leq 0 \end{cases}$$

Wzór 2. Wzór funkcji aktywacyjnej ELU [21]

Jako ostatnia warstwa w każdym module została użyta odmiana warstwy łącznej, która oblicza maksima (ang. MaxPooling). Redukuje ona liczbę danych znajdując wartości maksymalne w zadanych przedziałach. Zasada działania została przedstawiona na rys. 12. Głównymi parametrami są wielkość filtra oraz krok, działają one analogicznie jak w warstwie konwolucyjnej.



Rys. 12. Przykład działania algorytmu warstwy łącznej (ang. MaxPooling) dla obrazu o wymiarach 4x4, dla parametrów: wielkość filtra (ang. kernel) 2x2 oraz krok (ang. stride) 2.

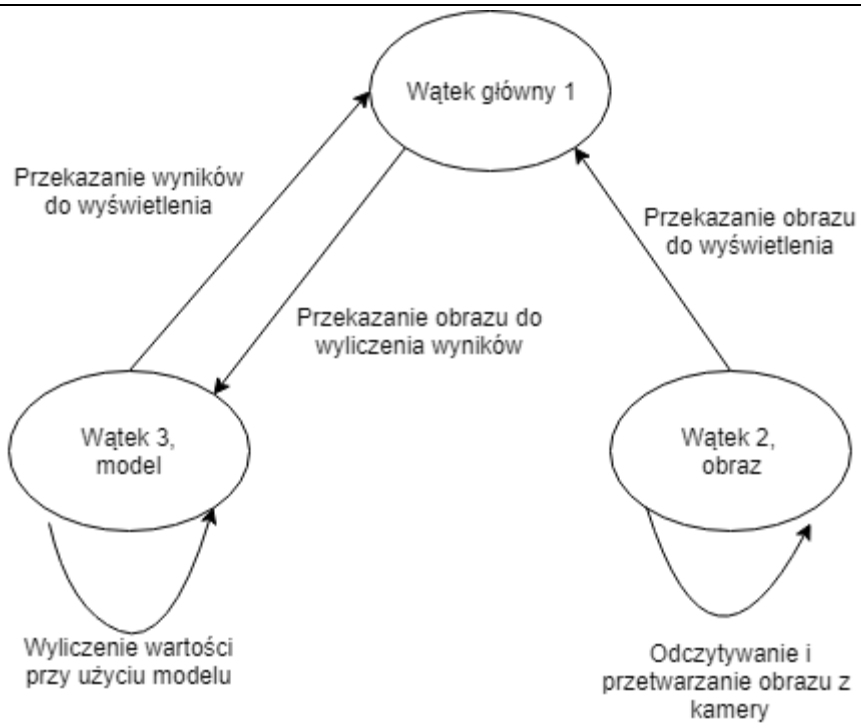
Opis moich parametrów na przykładzie czegoś tam …

Opis wybranego optimizer i ew. fukncja kosztu cross entrophy

5.2. Aplikacja do ewaluacji modelu

Ze względu na wymagające obliczenia, w przypadku ewaluacji modelu oraz dobrą praktykę programowania, działanie aplikacji zostało zrównoleglone pod względem wątków. Podczas przetwarzania danych, używa ona trzech wątków, gdzie wątek główny, uruchamiany jest wraz z inicjacją całego programu oraz dwóch dodatkowych o ściśle określonych zadaniach.

Wątek główny (wątek 1) jest odpowiedzialny za przygotowanie aplikacji do działania, zainicjowanie innych wątków oraz wyświetlania danych w oknie aplikacji. Pozostałe wątki są odpowiedzialne kolejno za przetwarzanie danych z kamery internetowej komputera (wątek 2), wykonywanie obliczeń przy użyciu wybranego modelu sieci neuronowej (wątek 3). Cała komunikacja pomiędzy wątkami odbywa się przy użyciu sygnałów. Mając na uwadze enkapsulacje pamięci dla poszczególnych elementów, wraz z sygnałem przekazywane też są potrzebne dane, przez co jest eliminowany hazard danych. Wątek 2, odczytuje obraz z kamery wideo z prędkością 10 klatek na sekundę. Każda klatka jest skalowana do rozdzielczości 640x480 px i wysyłana do wątku 1 w celu jego wyświetlenia. Wątek 1 po wyświetleniu obrazu na ekranie, wysyła ten obraz do wątka 3, gdzie jest interpretowany przez model sieci neuronowej. Po wyliczeniu wartości dla poszczególnych klas wynikowych, tablica z obliczeniami jest wysyłana do wątka 1, gdzie ostatecznie jest wyświetlana informacja. Kolejność wysyłania sygnałów między wątkami jest pokazana na rys. 11.



Rys. 11. Schemat komunikacji między wątkami w aplikacji

Dzięki takiemu podziałowi zadań, każdy wątek ma jasno sprecyzowane zadanie, co wpływa na lepsze wykorzystanie zasobów procesora, jak i na przyśpieszenie działania samej aplikacji.

Aplikacja została skompilowana pod kątem systemu Windows, przy użyciu narzędzia PyInstaller. Mieści się ona w katalogu GestureRecognitionAplication\dist__main__ wraz ze wszystkimi potrzebnymi bibliotekami w formatach .dll.

6. Weryfikacja i walidacja

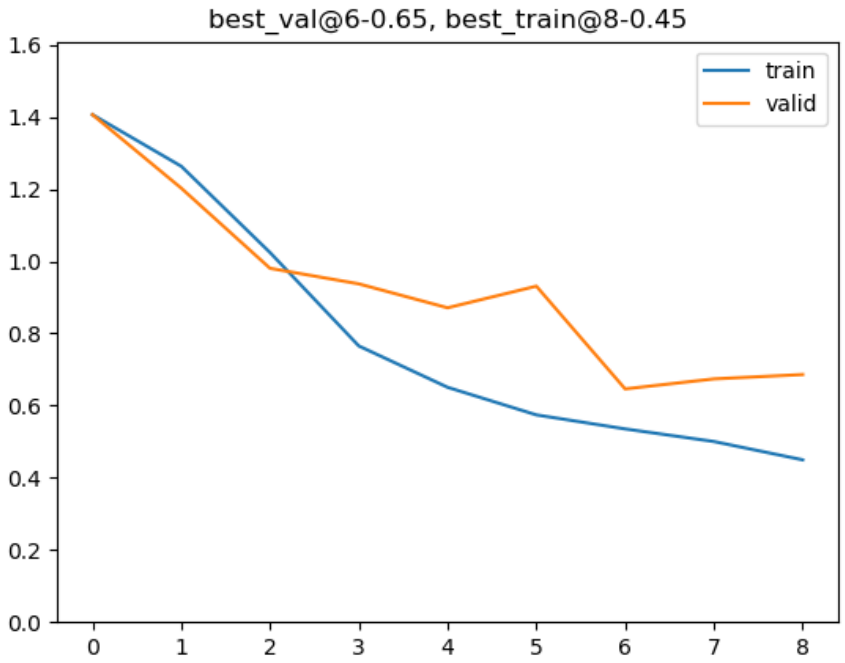
Sieci były uczona i testowana używając maszyny z systemem operacyjnym Windows 10, procesorem Intel Core i7-3537u 2.00 GHz-2.5Ghz, 8 GB pamięci RAM, dyskiem 512 GB SSD Crucial mx300 oraz kartą graficzną NVIDIA GeForce GT-740m.

W ramach analizy modeli, została wykonana seria eksperymentów dla dwóch rodzajów sieci neuronowych dla architektury typu 3D CNN + LSTM oraz dla architektury równoległej 3D CNN + LSTM. W celu wykazania obiektywnych różnic uczenia się między tymi architekturami, większość parametrów pokrywa się ze sobą, uczone są na tej samej jednostce komputerowej oraz używając tych samych algorytmów optymalizacyjnych. Dla architektury 3D CNN + LSTM zostały stworzone dwa modele. Jeden z nich posiada 6 warstw konwolucyjnych, w drugim liczba ta została zwiększona do 10 warstw. Dla architektury równoległej 3D CNN + LSTM został stworzony jeden model, z liczbą 4 warstw konwolucyjnych w każdym module CNN.

W celu analizy skuteczności powyższych modeli, zostały wzięte pod uwagę dwa parametry współczynnik kosztu oraz średnia precyzja modelu w kolejnych epokach.

6.1. Wyniki architektury 3D CNN + LSTM.

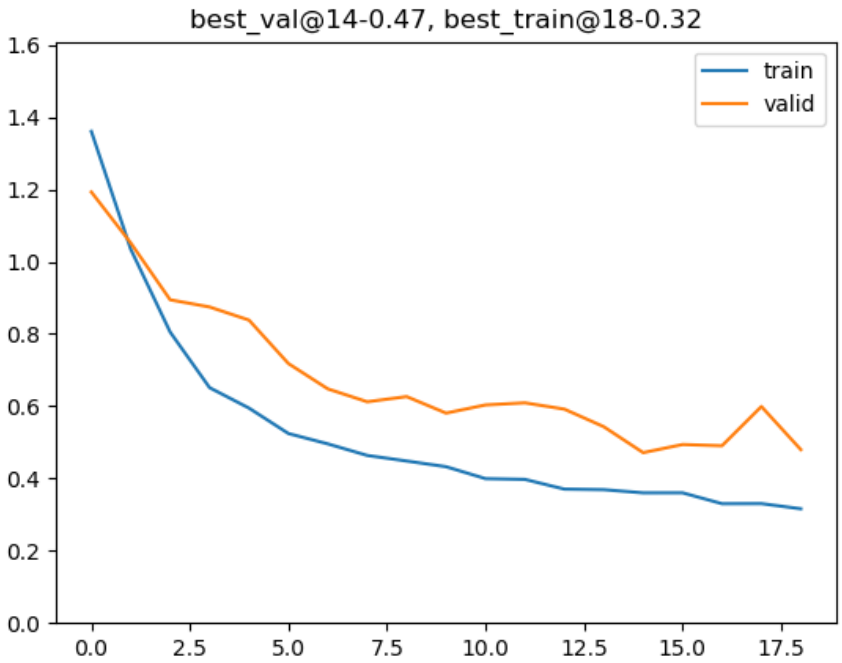
Zastosowany model 6-warstwowy charakteryzował się opadającą funkcją kosztu (rys. 10) oraz wznoszącą się funkcją precyzji (rys. 11).



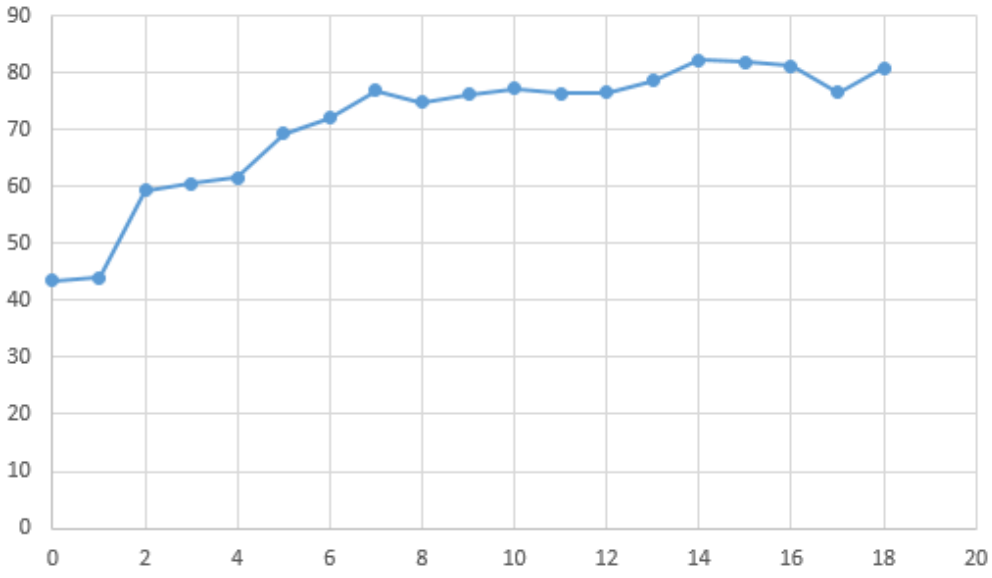
Rys. 10. Wykres zależności kosztu od epoki dla modelu 6-warstwowego 3D CNN + LSTM

Dorobić drugi wykres

Drugi model oparty o tą samą architekturę, posiada podobną charakterystykę funkcji kosztu (rys. 12) oraz precyzji (rys. 13).



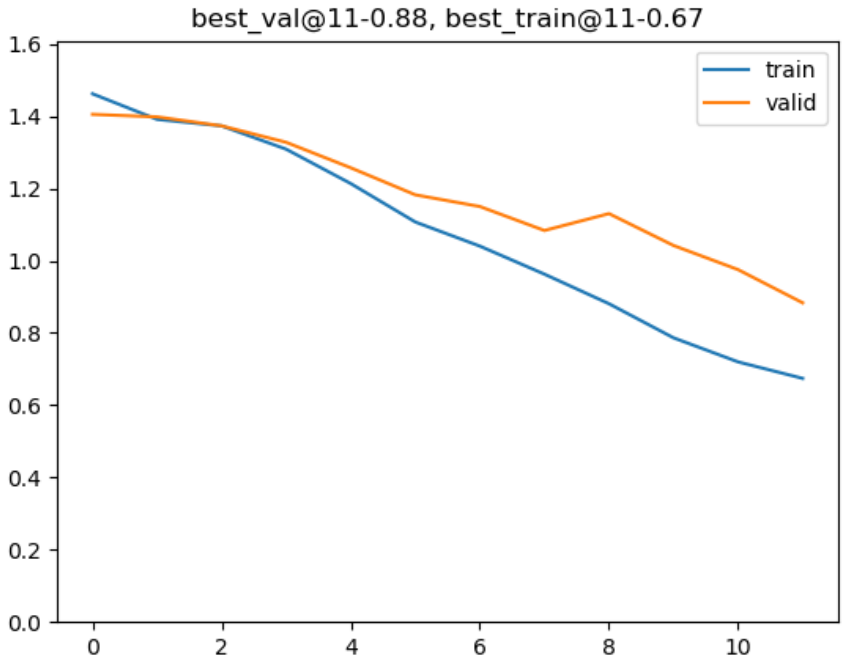
Rys. 12. Wykres zależności kosztu od epoki dla modelu 9-warstwowego 3D CNN+LSTM



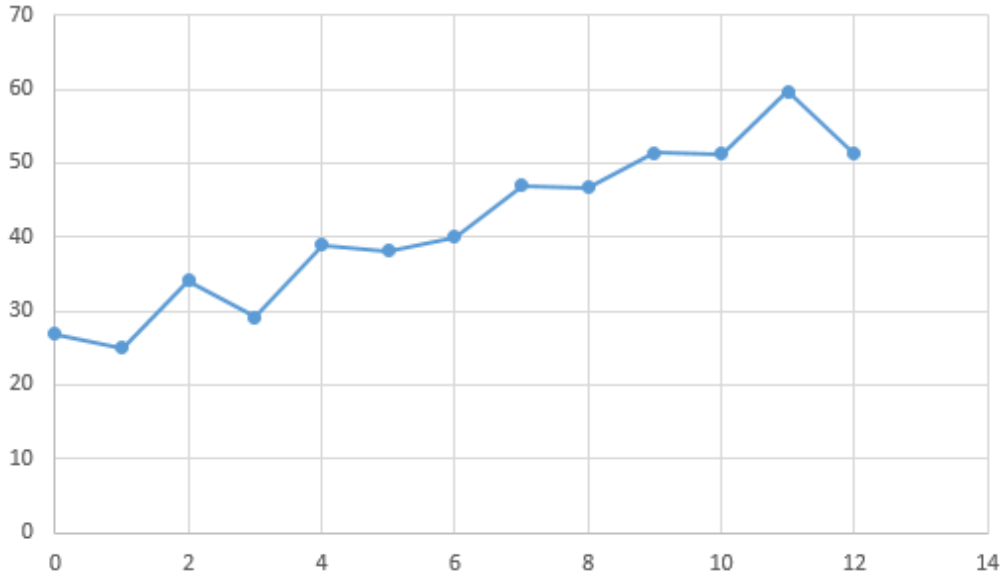
Rys. 13. Wykres zależności precyzji od epoki dla modelu 9-warstwowego 3D CNN+LSTM

6.2. Wyniki architektury równoległej 3D CNN + LSTM

Model charakteryzuje się powoli opadającym zboczem funkcji kosztu (rys. 14) i precyzji (rys.15).



Rys. 14. Wykres zależności kosztu od epoki dla modelu sieci równoległej 3D CNN + LSTM



Rys. 15. Wykres zależności precyzji od epoki dla modelu sieci równoległej 3D CNN+LSTM

7. Podsumowanie i wnioski

Rozdział ten obejmuje następujące elementy:

- uzyskane wyniki w świetle postawionych celów i zdefiniowanych wymagań,
- kierunki ewentualnych dalszych prac (rozbudowa funkcjonalna, …),
- problemy napotkane w trakcie pracy.

Bibliografia

[1]	Twenty Billion Network. *The 20BN-Jester Dataset – Twenty Billion Network*. `https://20bn.com/datasets/jester` [data dostępu: 2018-09-30].
[2]	Imię Nazwisko, Imię Nazwisko. *http://openaccess.thecvf.com/content_ICCV_2017_workshops/papers/w44/Zhang_Learning_Spatiotemporal_Features_ICCV_2017_paper.pdf*. Wydawnictwo, Warszawa, 2017.
[3]	Krzysztof Odrzywołek. Wykorzystanie głębokich sieci neuronowych w weryfikacji mówcy: Praca Magisterska, str. 5 Kraków 2016.
[4]	Jacek Bartman. Sieci rekurencyjne. *Wykłady*, http://www.neurosoft.edu.pl/media/pdf/jbartman/sztuczna_inteligencja/NTI9.pdf. Na dzień 07.12.2018
[5]	https://trends.google.pl/trends/explore?date=all&q=deep%20learning. Na dzień 06.12.2018
[6]	Benteng Ma, Yong Xia. Autonomous Deep Learning: A Genetic DCNN Designer for Image Classification. arXiv:1807.00284v1. 1 July 2018, str. 5.
[7]	Yanting Pei, Yaping Huang, Qi Zou, Hao Zang, Xingyuan Zhang, Song Wang. Effects of Image Degradations to CNN-based Image Classification. arXiv:1810.05552v1. 12 October 2018, str. 2
[8]	Jie Hu, Li Shen, Samuel Albanie, Gang Sun, Enhua Wu. Squeeze-and-Excitation Networks. arXiv:1709.01507v3. 25 October 2018
[9]	Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition. arXiv:1512.03385v1. 10 december 2015, str. 6

[10]	Stefan Braun. LSTM Benchmarks for Deep Learning Frameworks. arXiv:1806.01818v1. 5 June 2018
[11]	Markku Hinkka, Teemu Lehto, Keijo Heljanko, Alexander Jung. arXiv:1809.05896v1. 16 september 2018
[12]	Ben Krause, Iain Murray, Steve Renals, Liang Lu. Multiplicative Lstm For Sequence Modelling. arXiv:1609.07959v3. 12 October 2017
[13]	Agne Grinciunaite, Amogh Gudi1, Emrah Tasli1,Marten den Uyl1. Human Pose Estimation in Space and Time using 3D CNN. arXiv:1609.00036v3. 19 October 2016. Str. 6.
[14]	Jaebok Kim, Khiet P. Truong, Gwenn Englebienne, and Vanessa Evers. Learning spectro-temporal features with 3D CNNs for speech emotion recognition. arXiv:1708.05071v1. 14 August 2017. Str. 4.
[15]	Yue Luo, Jimmy Ren, Zhouxia Wang, Wenxiu Sun, Jinshan Pan, Jianbo Liu, Jiahao Pang, Liang Lin. LSTM Pose Machines. arXiv:1712.06316v1. 18 December 2017.
[16]	Andres Sanin, Conrad Sanderson, Mehrtash T. Harandi, Brian C. Lovell. Spatio-Temporal Covariance Descriptors for Action and Gesture Recognition. arXiv:1303.6021v1. 25 March 2013
[17]	https://20bn.com/datasets/jester
[18]	Stefan Braun. LSTM Benchmarks for Deep Learning Frameworks. arXiv:1806.01818v1. 5 June 2018. Str. 7.
[19]	2017 Twenty Billion Neurons GmbH, Berlin, Germany. https://github.com/TwentyBN/GulpIO-benchmarks. Na dzień 08.12.2018
[20]	Udacity. https://github.com/udacity/CVND---Gesture-Recognition. Na dzien 08.12.2018.
[21]	Clevert, Djork-Arné; Unterthiner, Thomas; Hochreiter, Sepp. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). arXiv:1511.07289. 22 February 2016. Str. 5.

Spis skrótów i symboli

DNA	kwas deoksyrybonukleinowy (ang. *deoxyribonucleic acid*)
MVC	model – widok – kontroler (ang. *model–view–controller*)
N	Liczebność zbioru danych

Zawartość dołączonej płyty

Na płycie DVD dołączonej do dokumentacji znajdują się następujące materiały:

- praca w formacie pdf,
- źródła programu,
- zbiory danych użyte w eksperymentach.

Spis rysunków

Spis tabel