

DESIGN LABORATORY
PROJECT
SUBJECT:
GESTURE RECOGNITION BY A NEURAL
NETWORK

Authors: Jakub Guza

Adam Zielina

Wojciech Możdzeń

Leader: PhD Jakub Gałka

AGH UST

Cracow (C) 2022

1. Introduction

Feasibility Study:

Dobór odpowiednich narzędzi projektowych. Naszym językiem programowania będzie Python z użyciem biblioteki numpy do obrazowania obiektów wektorowych oraz OpenCV jako narzędzia wspierającego przechwytywanie danych obrazowych z rejestratora. Wybór takiego języka wydaje się naturalny w obliczu zagadnień przetwarzania danych multimedialnych. Alternatywa dla DSP matlab jest językiem stricte pod obliczenia i analizę matematyczną i nie dysponuje tak różnorodnymi bibliotekami jak Python, szczególnie jeśli chodzi o rozwiązania hardwarowe jak przechwytywanie kamery. Wykorzystane zostanie IDE Microsoft Visual Studio wraz z wbudowanym debuggerem stanowi bardzo mocne narzędzie programistyczne, jest bardzo popularny co ułatwi troubleshooting. W celu rozpoznawania ruchu użyjemy trackera, następnie użyjemy biblioteki keras która na podstawie naszych danych nauczy naszą sieć odpowiedniej detekcji. Przewidziane zostaną testy w różnych momentach ewaluacji co pozwoli na uniknięcie późniejszych błędów.

Testy funkcjonalne sprawdzające działanie systemu i jego funkcjonalność np. testowanie działanie kamery, sprawdzenie wyników detekcji.

Testy нефункционалне teoretyczne sprawdzenie działania, analiza dokumentacji, testy wydajnościowe, testowanie działania sieci przy ograniczonej ilości danych treningowych.

SWOT:

S: dobre sposoby szukania gotowych rozwiązań:) w tych czasach jest wiele poradników do bibliotek oraz materiałów naukowych

W: słaba wiedza z sieci neuronowych / brak "normalnych" laboratoriów, potrzeba doszkalania się we własnym zakresie

O: doszkalanie się z wiedzy sieci neuronowych / computer vision, jest mocnym plusem na rynku pracy

T: nie działający kod, rozwiązaniem jest poszukiwanie odpowiedzi w Internecie i innych dostępnych źródłach, komunikacja z prowadzącym lub innymi członkami zespołu

nie odbycie się spotkania z przyczyn technicznych lub prywatnych , w takim wypadku należy odrobić spotkanie w najbliższym możliwym terminie.

znajomość wszystkich rozwiązań technicznych w takim wypadku należy przeszukać datasheety i poznać dane rozwiązania lub wybrać prostszy wariant.

2. Implementation report

Projekt działa następująco:

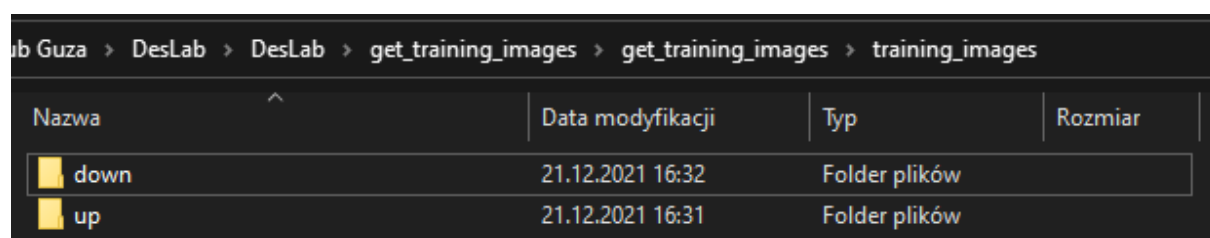
zbieramy zdjęcia z kamery poprzez skrypt w pythonie do nauki sieci neuronowej, następnie tworzymy sieć neuronową w Keras i uczymy ją zebranych przez nas zdjęć.

Przed całym procesem przygotowujemy zdjęcia w taki sposób, że zmniejszamy je do jednego rozmiaru i zmieniamy kolory na czarno-białe.



W taki sposób nauczamy sieć neuronową, następnie w pętli głównej w głównym programie, wrzucamy klatka po klatce z kamery do sieci neuronowej, która dokonuje detekcji.

3. Testing report

- **Testy pierwszego kamienia milowego:** demo kodu Nov 24, 2021–Dec 8, 2021



The screenshot shows a file explorer window with the path: `lab Guza > DesLab > DesLab > get_training_images > get_training_images > training_images`. Below the path, there is a table listing the contents of the `training_images` directory.

Nazwa	Data modyfikacji	Typ	Rozmiar
 down	21.12.2021 16:32	Folder plików	
 up	21.12.2021 16:31	Folder plików	

Program poprawnie tworzy foldery we wskazanej sciezce, oraz umieszcza w nim zdjecia, dzięki którym możemy uczyć naszą sieć neuronową.

- Testy drugiego kamienia milowego: Prototyp Dec 15, 2021–Dec 22, 2021

Oto test sieci neuronowej, jak widzimy mogła by działać troszeczkę lepiej, skuteczność wykrywania danych to 6/8, użyłem zdjęć z internetu, by potwierdzić lepiej poprawność działania,

In [2]:

```
prediction = model.predict([prepare("up_t.png")])
print(CATEGORIES[int(prediction[0][0])])

prediction3 = model.predict([prepare("up_test3.png")])
print(CATEGORIES[int(prediction3[0][0])])
```

down
up

nie
tylko
tych

In [3]:

```
prediction1 = model.predict([prepare("down_t.png")])
print(CATEGORIES[int(prediction1[0][0])])

prediction2 = model.predict([prepare("blog-1-image.jpg")])
print(CATEGORIES[int(prediction2[0][0])])

prediction4 = model.predict([prepare("dd.jpg")])
print(CATEGORIES[int(prediction4[0][0])])

print("\nmy\n")

prediction7 = model.predict([prepare("156.jpg")])
print(CATEGORIES[int(prediction7[0][0])])

prediction8 = model.predict([prepare("840.jpg")])
print(CATEGORIES[int(prediction8[0][0])])

prediction9 = model.predict([prepare("873.jpg")])
print(CATEGORIES[int(prediction9[0][0])])
```

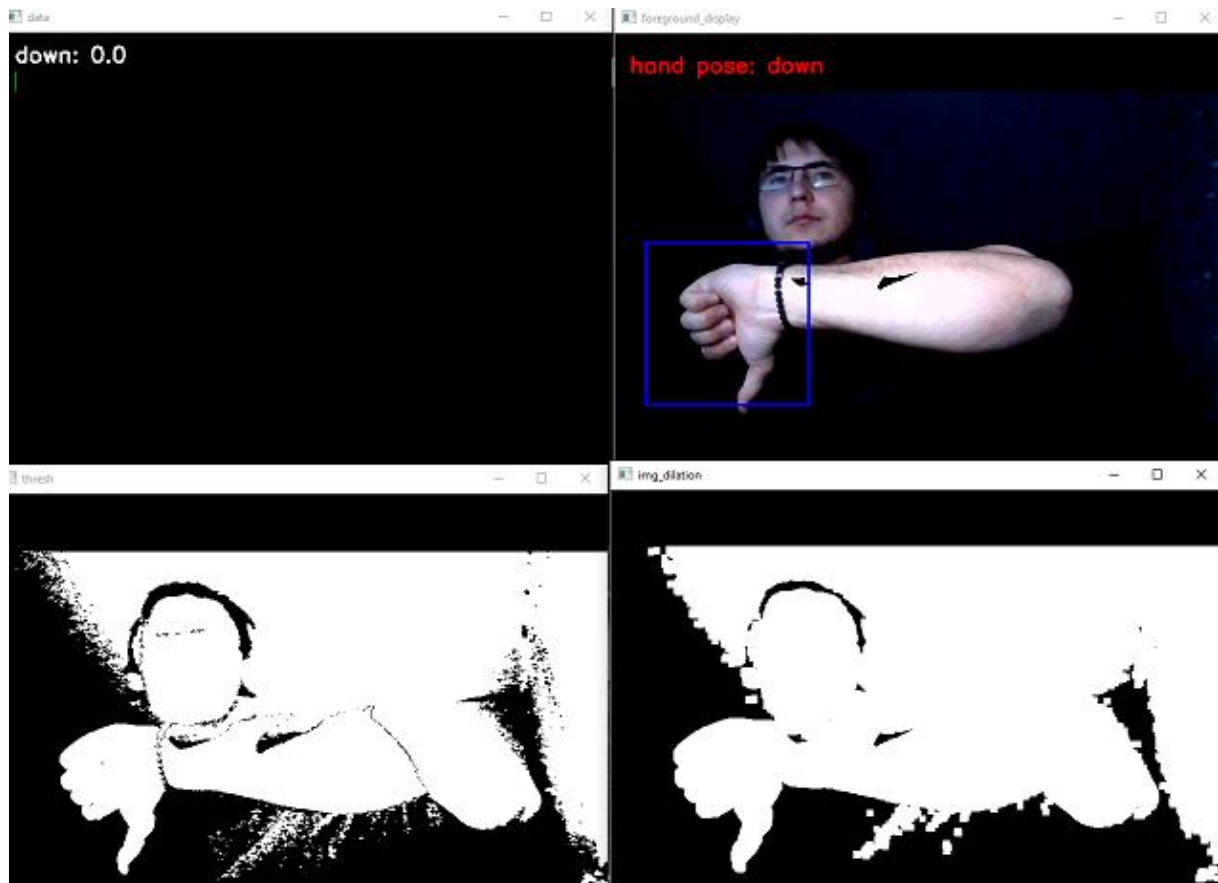
down
up
down

my

down
down
up

wykonanych z kamery.

ToDo w ostatnim kamieniu milowym, zoptymalizować sieć neuronową, by miała większą skuteczność, na danych z "internetu", prawdopodobnie trzeba użyć większej ilości danych treningowych, lub zoptymalizować samą sieć (ilość warstw etc.).



Powyżej test wykrywania gestów realtime, jak widzimy do dopracowania jest background subtraction (dużo szumów) oraz tracker.

- Testy trzeciego kamienia milowego: Poprawa prototypu

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

model.fit(X, y, batch_size = 40, epochs = 9, validation_split = 0.3) # batch size to ilość podawanych próbek w jednym czasie

Epoch 7/9
28/28 [=====] - 5s 196ms/step - loss: 0.5444 - accuracy: 0.6929 - val_loss: 0.5072 - val_accuracy: 0.8333
Epoch 5/9
28/28 [=====] - 5s 194ms/step - loss: 0.3982 - accuracy: 0.8652 - val_loss: 0.3179 - val_accuracy: 0.8896
Epoch 6/9
28/28 [=====] - 6s 197ms/step - loss: 0.2503 - accuracy: 0.9134 - val_loss: 0.2285 - val_accuracy: 0.9312
Epoch 7/9
28/28 [=====] - 5s 196ms/step - loss: 0.1898 - accuracy: 0.9330 - val_loss: 0.2195 - val_accuracy: 0.9250
Epoch 8/9
28/28 [=====] - 5s 195ms/step - loss: 0.1434 - accuracy: 0.9545 - val_loss: 0.1330 - val_accuracy: 0.9542
Epoch 9/9
28/28 [=====] - 5s 196ms/step - loss: 0.1002 - accuracy: 0.9723 - val_loss: 0.1015 - val_accuracy: 0.9729

Out[1]: <tensorflow.python.keras.callbacks.History at 0x2e77e52c5b0>

In [46]: model.save("upvsdown.model")

INFO:tensorflow:Assets written to: upvsdown.model\assets
```

Powyższe zdjęcie, ukazuje jeden z ostatnich procesów nauki sieci neuronowej, niestety wnioski są następujące: Wykorzystywane są tylko moje zdjęcia (1 osoba, podobna sceneria, brak różnorodności barw etc.) co może powodować overfitting, sieć uczy się na podobnych danych wejściowych, przez co coraz ciężiej jest jej później rozróżnić obiekty. Możliwe, że w tym przypadku im więcej danych treningowych, tym gorzej. Nie udało nam się zoptymalizować sieci z Tensorboard, ponieważ nie mogliśmy tam uruchomić odpowiednich oraz ważnych opcji poglądowych takich jak: validation loss, accuracy. Ciężko było dojść do tego, co jest tego przyczyną.

```
from tensorflow.keras.callbacks import TensorBoard #analizowanie sieci neuronowej
import pickle
import time

tf.keras.callbacks.TensorBoard(
    log_dir='logs', update_freq='epoch',
    profile_batch=0, # <-- default value is 2
)

#gpu_options = tf.GPUOptions(per_process_gpu_memory_fraction=0.333)
#sess = tf.Session(config=tf.ConfigProto)(gpu_option=gpu_options)

X = pickle.load(open("X.pickle", "rb"))
y = pickle.load(open("y.pickle", "rb"))

X = X/255.0 #miedzy 0 a 1

import time

dense_layers = [0, 1, 2]
layer_sizes = [32, 64, 128]
conv_layers = [1, 2, 3]

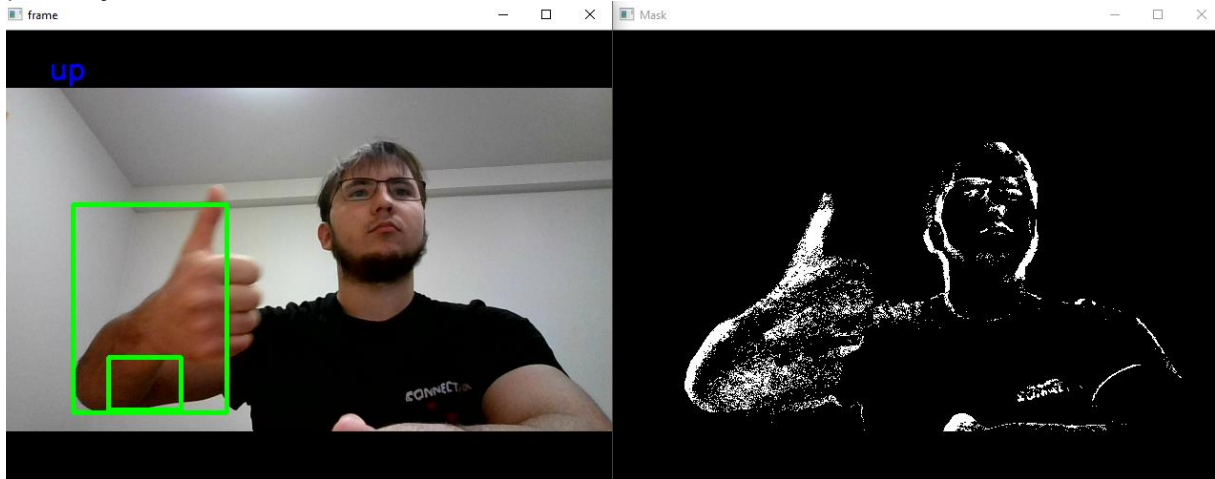
for dense_layer in dense_layers:
    for layer_size in layer_sizes:
        for conv_layer in conv_layers:

            NAME = "{}-conv-{}-nodes-{}-dense-{}".format(conv_layer, layer_sizes, dense_layer, int(time.time()))
            tensorboard = TensorBoard(log_dir='logs/{}'.format(NAME))
            print(NAME)
            model = Sequential()

            model.add( Conv2D(layer_size, (3,3), input_shape = X.shape[1:]) ) #warstwa "spłotowa" 64? potem okno 3x3
            model.add(Activation("relu"))
            model.add(MaxPooling2D(pool_size=(2,2))) #2x2 rozmiar
```

Była próba stworzenia list różnych warstw i testowania ich w pętlach, czyli uczenie sieci neuronowej a potem analiza ile warstw itd. jest najoptymalniejsza dla sieci, lecz z góry zaznaczam nie udało nam się to. Jedyne dane, dla których wykresy pokazywał tensorboard, to był epoch_loss.

Udało się zaaplikować tracker EuclideanDistTracker, który otacza w prostokąt/y obiekty które się poruszają.



Powyżej test trackera, działa lepiej od poprzedniego, poprawiony został background subtraction, dużo mniej szumów.

- **Testy czwartego kamienia milowego:** Ostateczna wersja

W ostatecznej wersji mamy już działający tracker, zdjęcia powyżej. Dodana jest też funkcjonalność przyciszania/pogłaśniania dźwięku w systemie Windows dzięki bibliotece pyautogui.