Rozpoznawanie ruchu 2D myszki niezależne od czasu wykonywania ruchu

Jakub Rakuś, Konrad Reczko

26 stycznia 2024

1 Wstęp

Projekt został utworzony w celu ułatwienia korzystania i zarządzania zasobami komputera. Pozwala użytkownikowi tworzyć dowolne symbole oraz skrypty. Podczas wykonywania ruchu myszki analizowana jest jej trajektoria ruchu w celu wykrycia symbolu. Program odczytuje informację o powiązanym skrypcie z symbolem i go uruchamia. Aplikacja może być pomocnicza dla programistów w zarządzania środowiskiem deweloperskim oraz dla zwykłych użytkowników w celu zautomatyzowania wielu procesów. Zakres projektu obejmuje:

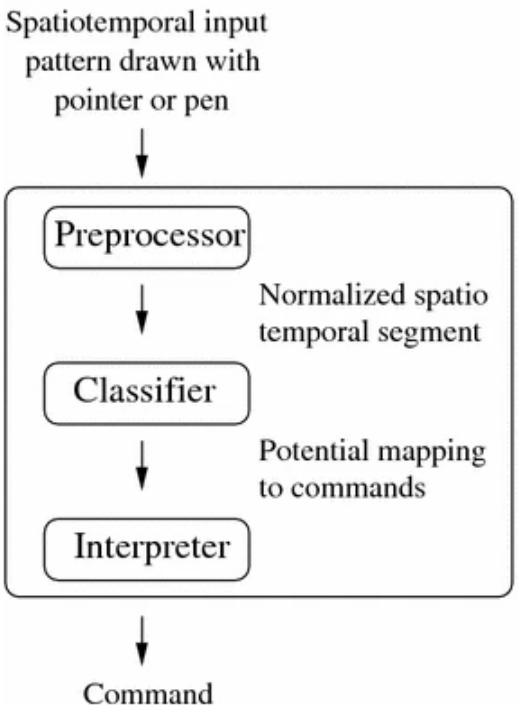
- Opracowanie narzędzia do zbierania i walidacji danych
- Zebranie danych
- Stworzenie odpowiedniego klasyfikatora do rozpoznawnia symboli
- Utworzenie interfejsu graficznego

2 Przegląd literatury

W literaturze mogliśmy znaleźć wiele projektów związanych z przetwrzaniem wejściowych symboli. Ostatecznie postanowiliśmy wzorować się na pracy 'Symbol design: a user-centered method to design pen-based interfaces and extend the functionality of pointer input devices' [MGU06], który zawierał implementację rozwiązania problemu najbliżej spokrewnionwego do naszego. Główny model przedstawiony na rysunku 1 zawierał trzy główne komponenty:

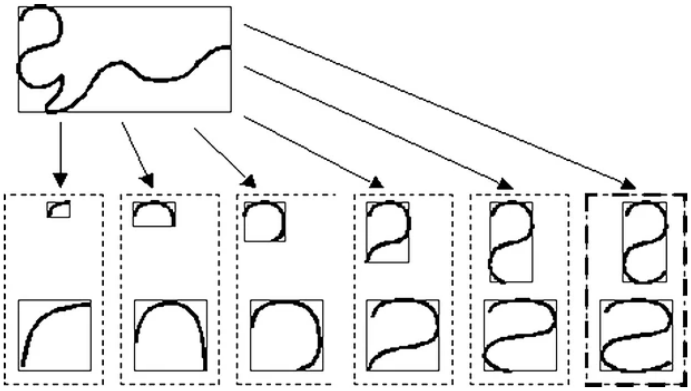
- Przetwarzanie wstępne
- Klasyfikator
- Interpreter

W naszym systemie postanowiliśmy zaimplementować analogiczny sposób działania. W pierwszym etapie wprowadzony wzór miał zostać odpowiednio przetworzony, aby klasyfikator był w stanie przyjąć go na wejściu. W Kolejnym kroku klasyfikator powinien określić jaki to symbol. Ostatecznie interpreter miał wydać odpowiednią komendę znając wykonany symbol.



Rysunek 1: Model systemu [MGU06]

W pracy [MGU06] mogliśmy także znaleźć sposób na odpowiednią implementację przetwarzania wstępnego przedstawionego na rysunku 2. Powinna ona polegać na analizowaniu danego symbolu w małych porcjach. Użytkownik zaczyna wykonywać pewne ruchy i w małych odstępach czasu preprocesor składa otrzymane fragmenty, przetwarza i ostatecznie przekazuje do klasyfikatora. Jeśli klasyfikator wykryje nasz symbol z odpowiednio dużym prawdopodobieństwem to czyścimy bufor w preprocesorze. Odpowiednio kiedy użytkownik przestanie wprowadzać nowe dane (np. przestanie ruszać myszką przez 0.1 sekundy) to także bufor zostanie wyczyszczony i przygotowany do analizy nowych symboli. Jednak w opisywanej pracy zakładano użyteczność systemu dla różnych typów urządzeń wejściowych, w naszym projekcie natomiast narzucaliśmy użytkownikowi myszkę komputerową jako narzędzie do wprowadzania danych. W związku z tym postanowiliśmy uprościć cały proces przetwarzania wstępnego zmuszając użytkownika do przytrzymania odpowiednich klawiszy w momencie wprowadzania danego symbolu. Pozbawiło to system dodatkowych błędów, mogących wynikać z identyfikacji konkretnych wzorców w przypadkowych ruchach myszką.



Rysunek 2: Przetwarzanie wstępne [MGU06]

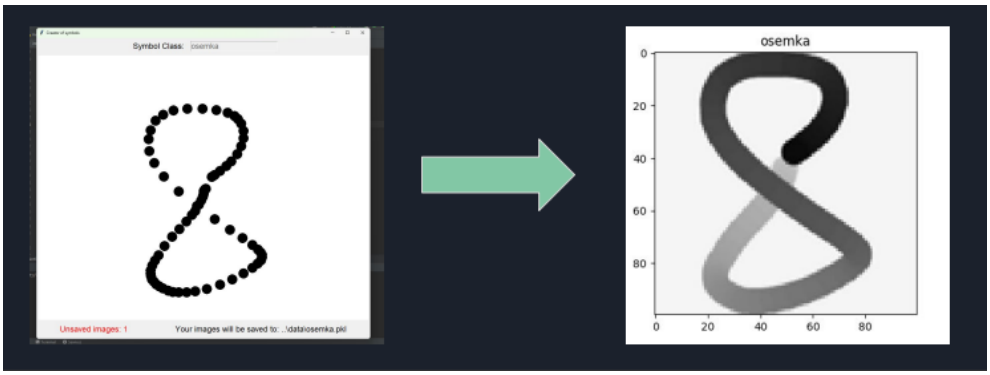
Kolejnym zadaniem było dobranie odpowiedniego klasyfikatora. Korzystając z założeń, że nasz zbiór rozpoznawalnych symbol to będą proste znaki o nieskomplikowanej strukturze mogliśmy porównać go do MNIST database. Przejrzeliśmy zatem zawartość strony z rankingiem najlepszych klasyfikatorów dla MNIST database `https://paperswithcode.com/sota/image-classification-on-mnist` [web] i porównaliśmy najlepsze klasyfikatory dla tego systemu. Po przeanalizowaniu rankingu, postanowiliśmy użyć konwolucyjnej sieci neuronowej jako nasz klasyfikator. Wynikało to z jej wysokich pozycji w rankingu oraz naszego wcześniejszego doświadczenia w implementowaniu tego typu sieci.

3 Użyta metoda / algorytm

3.1 Zbieranie danych

Symbole zostały utworzone za pomocą narzędzia do zbierania danych oraz za pomocą narzędzia do przetwarzania danych, które dołączone są do aplikacji. Początkowo poproszono kilka osób o narysowanie po 1000 symboli dla każdego z sześciu rodzai wymienionych w rozdziale 3. Kolejno listę pikseli przez które przeszedł kursor myszki przekazywano do wspomagających narzędzi, gdzie przechodziły przez następujące etapy:

- Na podstawie przesłanej listy został określony minimalny rozmiar prostokątnej ramki w której mieści się obraz. W ten sposób dobrano odpowiedni kształt dla nowego płótna na którym miał powstawać obrazek
- Każda lista punktów została odczytana i odwzorowana w obrazek przy pomocy biblioteki Pillow. W miejscach oznaczonych punktów rysowane zostało małe koło. Koło miało odpowiedni kolor z zakresu od 0 do 235 w skali szarości w zależności od indeksu. Miało to na celu odróżnienie identycznych symboli rysowanych w innych kierunkach (przykład: litera S ukazana na obrazku 4b i odwrócona litera S ukazana na grafice 4c). Rozmiar promienia koła równy był pięciu procentom długości lub szerokości, w zależności, który z nich był większy.
- W celu nadania obrazowi odpowiednich kształtów użyliśmy interpolacji do połączenia narysowanych kółek. Między wszystkimi dwoma sąsiednimi kołami został utworzony prostokąt, którego dwa równoległe boki pokrywały się odpowiednio ze średnicą jednego koła i ze średnicą drugiego koła. Natomiast kolejne dwa równoległe boki były długości odległości między kołami.



Rysunek 3: Proces interpolacji

- Kolejno obrazek był skalowany do rozmiarów 32x32 piksele. Pierwotne płótno było rozszerzane do rozmiarów najmniejszego możliwego kwadratu w którym mieścił się ten obrazek poprzez dodanie pustego białego tła. Kolejno było skalowane za pomocą funkcji z biblioteki Pillow. W ten sposób otrzymywaliśmy obrazek gotowy do przetworzenia przez sieć neuronową.

- Ostatnim etapem było utworzenie większej ilości danych z jednego obrazka. Chcąc osiągnąć zamierzony efekt obrazek został przetworzony przy pomocy biblioteki SciPy. Wszystkie transformacje jakim został poddany symbol to:

  1. Obrót względem środa obrazu o 4 stopnie
  2. Obrót względem środa obrazu o 7 stopnie
  3. Obrót względem środa obrazu o 10 stopnie
  4. Obrót względem środa obrazu o -4 stopnie
  5. Obrót względem środa obrazu o -7 stopnie
  6. Obrót względem środa obrazu o -10 stopnie
  7. Rozciągnięcie w poziomie o 5 procent
  8. Rozciągnięcie w poziomie o 10 procent
  9. Rozciągnięcie w pionie o 5 procent
  10. Rozciągnięcie w pionie o 10 procent

3.2 Dane

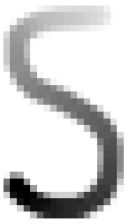
Końcowo w aplikacji mamy dostęp do jednego pakietu zawierającego 6 wyuczonych symboli:

- Koło - 4a
- Litera 'S' - 4b
- Odwrócona litera 'S' - 4c
- Litera 'W' - 4d
- Trójkąt - 4e
- Symbol Błyskawicy - 4f

Każdy z nich zawiera po 11 000 rysunków danego symbolu i ma przydzielony jeden z trzech domyślnych skryptów: uruchomienie visual studio code, wyświetlenie repozytorium projektu, wypisanie 'hello world' w terminalu.

(a) Koło

(b) Litera "S"



(c) Odwrócona litera "S"

(d) W

(e) Trójkąt

(f) Symbol Błyskawicy

Rysunek 4: Symbole

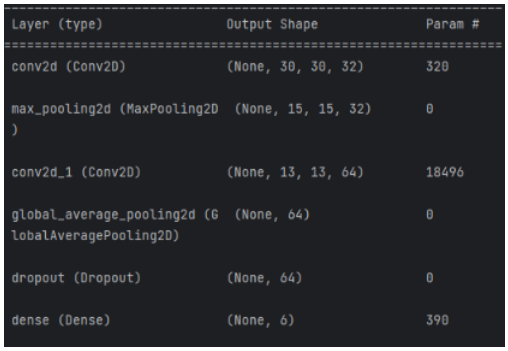
4 Zbiór danych testowych i treningowych

W celu prawidłowego wytrenowania sieci neuronowej zdecydowaliśmy się na odpowiedni podział danych. W jednym zbiorze umieściliśmy cały pakiet symboli (symbol wprowadzony przez użytkownika oraz jego 10 transformacji). Inaczej mówiąc dany symbol i jego transformacje nie były rozdzielane. Wszystkie trafiały do tego samego zbioru danych (testowego lub trenującego). Taki proces miał na celu wykluczenie fałszywego testowania, w którym dane byłyby testowane na swoich transformacjach. Chcemy mieć pewność, że transformowane dane nie będą przypadkiem zawyżać wyników testów. Ilość danych dla zbioru trenującego to 80% wszystkich danych (52 800 symboli). Odpowiednio ilość danych zbioru testowego to 20% wszystkich danych (13 200 symboli).

5 Rezultaty treningu oraz testów

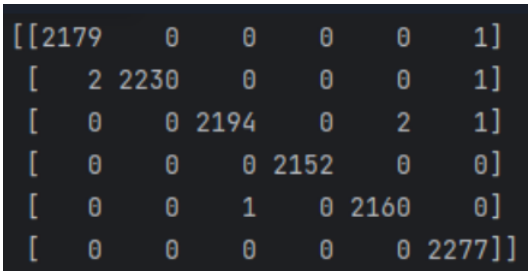
Jako klasyfikator użyta została konwolucyjna sieć neuronowa w konfiguracji pierwszej ukazanej na rysunku 5a, wykonana przy pomocy biblioteki Tensorflow i Keras. Poniższe zdjęcia przedstawiają różne testowane modele oraz ich confiusion matrix dla zebranych danych 3.2. Dla wszystkich modeli został zastosowany stosunkowo wysoki dropout (0.4) w celu uniknięcia przetrenowania.

W pierwszej konfiguracji został użyty optymalizator RMS, ponieważ jest to mały model, a z testów wynikało, że mniejsze sieci osiągają lepsze wyniki dla RMS niż optymalizator Adam. Architektura sieci wygląda tak jak przedstawiono na rysunku 5a. Po procesie trenowania udało się uzyskać dokładność bliską 100%. Jedynie 8 wyników z puli 13 200 zostało niepoprawnie sklasyfikowanych. Zostało to ukazane na obrazku 5b.



Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	320
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18496
global_average_pooling2d (GlobalAveragePooling2D)	(None, 64)	0
dropout (Dropout)	(None, 64)	0
dense (Dense)	(None, 6)	390

(a) Model

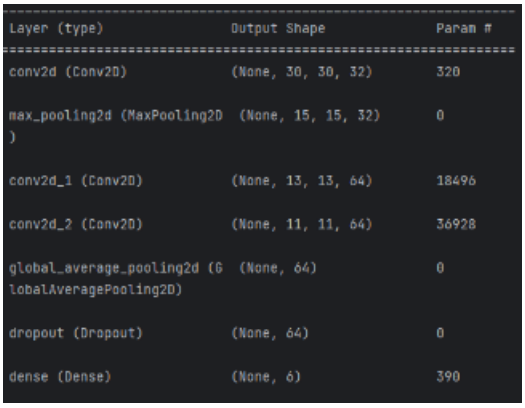


```
[[2179    0    0    0    0    1]
 [   2 2230    0    0    0    1]
 [   0    0 2194    0    2    1]
 [   0    0    0 2152    0    0]
 [   0    0    1    0 2160    0]
 [   0    0    0    0    0 2277]]
```

(b) Confusion Matrix

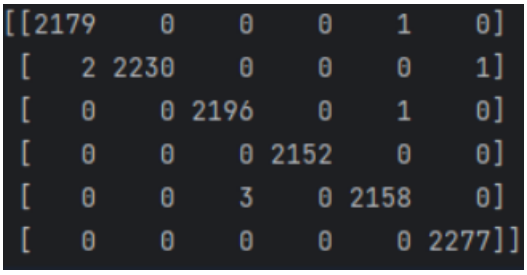
Rysunek 5: Konfiguracja 1

W Konfiguracji drugiej przedstawionej na rysunku 6a, dodano 2D convolution warstwę, jednak nie wpłynęło to w żaden sposób na skuteczność modelu. Confusion Matrix 6b wskazuje na to, że taka sama ilość symboli została poprawnie sklasyfikowana.



Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	320
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18496
conv2d_2 (Conv2D)	(None, 11, 11, 64)	36928
global_average_pooling2d (GlobalAveragePooling2D)	(None, 64)	0
dropout (Dropout)	(None, 64)	0
dense (Dense)	(None, 6)	390

(a) Model

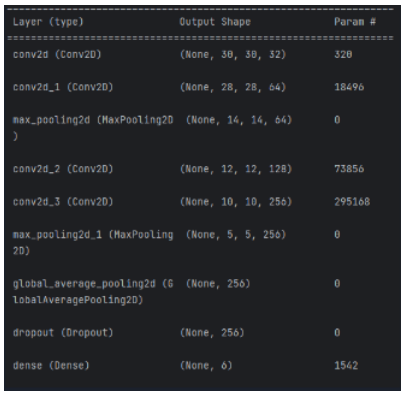


```
[[2179    0    0    0    1    0]
 [   2 2230    0    0    0    1]
 [   0    0 2196    0    1    0]
 [   0    0    0 2152    0    0]
 [   0    0    3    0 2158    0]
 [   0    0    0    0    0 2277]]
```

(b) Confusion Matrix

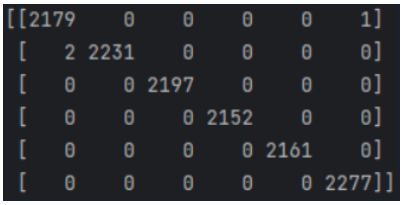
Rysunek 6: Konfiguracja 2

W konfiguracji trzeciej zastosowano bardziej rozbudowany model ukazany na obrazku 7a. Wprowadzono kolejną 2D convolution warstwę oraz zaimplementowano dodatkową warstwę max pooling 2D w porównaniu do modelu drugiego 6a. To poprawiło uzyskany wynik i zaledwie 3 symbole zostały niepoprawnie sklasyfikowane. Zostało to ukazane na grafice 7b.



Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	320
conv2d_1 (Conv2D)	(None, 28, 28, 64)	18496
max_pooling2d (MaxPooling2D)	(None, 14, 14, 64)	0
conv2d_2 (Conv2D)	(None, 12, 12, 128)	73856
conv2d_3 (Conv2D)	(None, 10, 10, 256)	295168
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 256)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 256)	0
dropout (Dropout)	(None, 256)	0
dense (Dense)	(None, 6)	1542

(a) Model



```
[[2179    0    0    0    0    1]
 [   2 2231    0    0    0    0]
 [   0    0 2197    0    0    0]
 [   0    0    0 2152    0    0]
 [   0    0    0    0 2161    0]
 [   0    0    0    0    0 2277]]
```

(b) Confusion Matrix

Rysunek 7: Konfiguracja 3

Jak wynika z przedstawionych confusion matrix, dokładność jest bliska 100% dla wszystkich konfiguracji, zatem w celu optymalizacji szybkości działania programu, został zaimplementowany model pierwszy (najmniej rozbudowany). Nie było potrzeby implementowania bardziej złożonych sieci.

6 Architektura

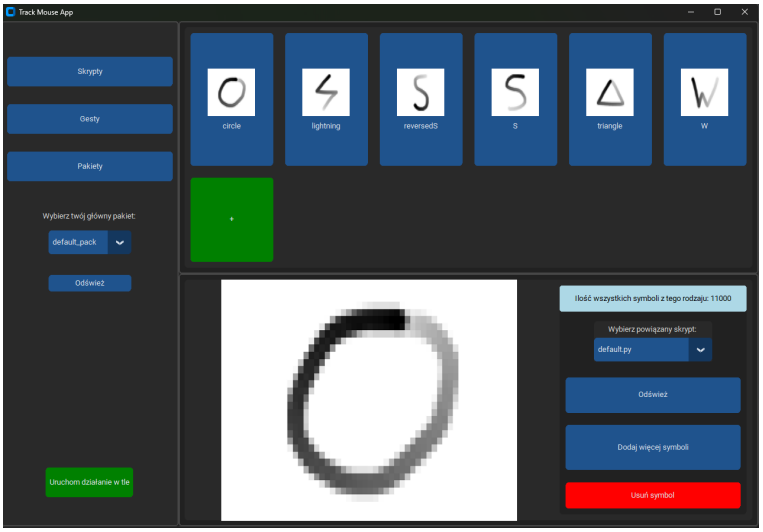
Całość aplikacji została wykonana w języku Python. W Programie możemy wyróżnić pięć głównych komponentów:

- Interfejs graficzny
- Narzędzie do tworzenia nowych symboli
- Narzędzie do przetwarzania zdjęć
- Klasyfikator
- Skrypt śledzący ruch myszki

6.1 Interfejs graficzny

Interfejs graficzny ukazany na rysunku 8 został dodany, aby wygodnie zarządzać procesem tworzenia scryptów, symboli oraz pakietów. Każdy pakiet zawiera pewną ilość przypisanych symboli dla których istnieją przydzielone skrypty. Do wykonania GUI została użyta biblioteka CustomeTkinter. Każdy pakiet ma także odpowiadający mu model sieci neuronowej, która pozwala wykrywać wprowadzone symbole. Zatem w trakcie użytkowania może być aktywny tylko jeden zestaw symboli, a w raz z nim model sieci.

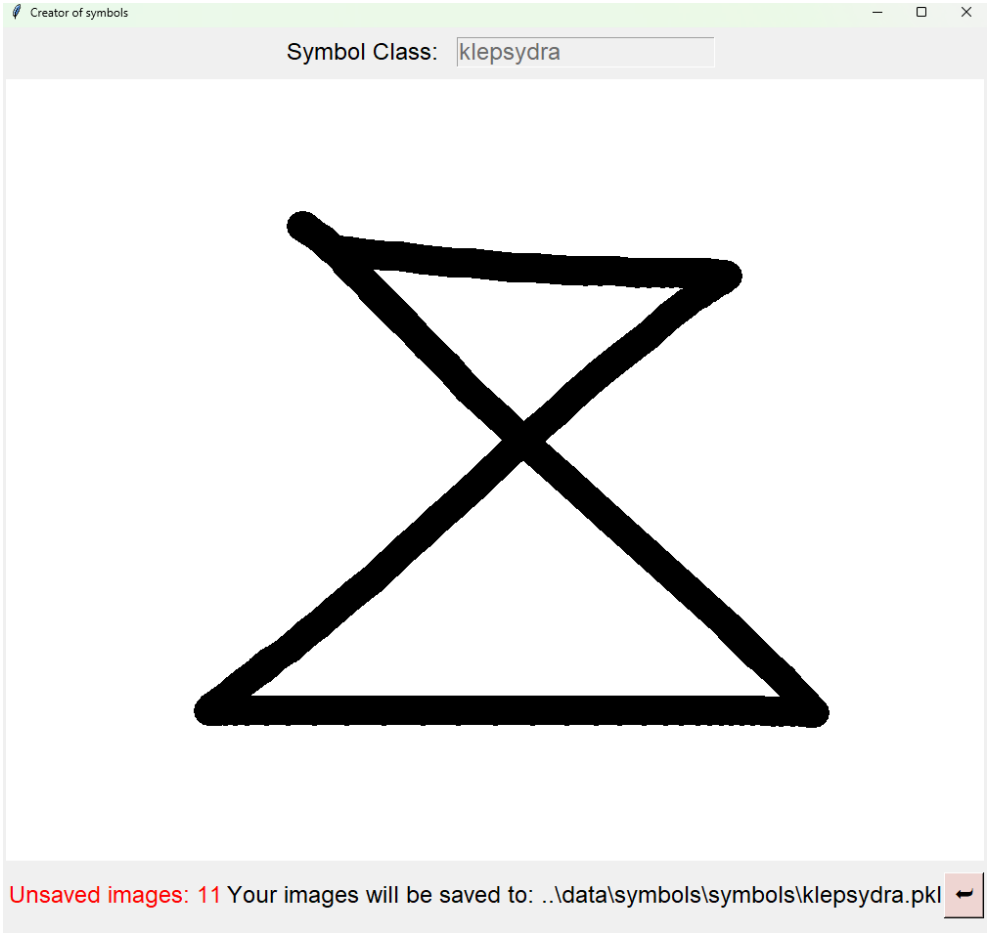
Interfejs pozwala także na łatwe przełączanie się między funkcjonalnościami. Mamy możliwość szybkiego przejścia pomiędzy systemem zarządzania skryptami, symbolami i pakietami. Kolejnym zadaniem GUI jest powiązanie klasyfikatora oraz narzędzie do zbierania danych z odpowiednimi przyciskami, zapewniając łatwy dostęp do odrębnych części aplikacji.



Rysunek 8: Interfesj graficzny

6.2 Narzędzie do tworzenia nowych symboli

Narzędzie do zbierania danych przedstawione na grafice 9 pozwala na łatwe tworzenie nowych symboli poprzez ich bezpośrednie rysowanie za pomocą myszki na tablicy. Program śledzi ruch myszki zapamiętując każdy oznaczony piksel. Kolejno przekazuje te dane do narzędzia przetwarzającego zdjęcia.



Rysunek 9: Narzędzie do zbierania danych

6.3 Narzędzie do przetwarzania zdjęć

Narzędzie to pozwala na odpowiednie obrabianie zdjęć. Jest wykorzystywane w skalowaniu zdjęć do odpowiedniego formatu, aby nadawało się do uczenia maszynowego. Moduł ten pozwala także na dokonywanie dziesięciu różnych transformacji na obrazach w celu szybszego zbierania danych. Zatem za każdym razem, gdy użytkownik wprowadza nowy symbol, dodawanych jest 11 nowych do listy. Dokładny opis przetwarzania został opisany w rozdziale 3.

6.4 Klasyfikator

Jako klasyfikator użyta została konwolucyjna sieć neuronowa wykonana przy pomocy biblioteki Tensorflow i Keras. Implementacja została szczegółowo omówiona w rozdziale 5.

6.5 Skrypt śledzący ruch myszki

Skrypt śledzący ruch myszki został utworzony przy pomocy bibliotek: Pyautogui, Keyboard. Pierwsza z nich została wykorzystana do pobierania odpowiednich pikseli z monitora. Natomiast druga została użyta do sprawdzania wciśniętych klawiszy przez użytkownika. Skrypt uruchamia aktualnie wybrany model przez użytkownika w celu predykcji aktualnie narysowanego symbolu.

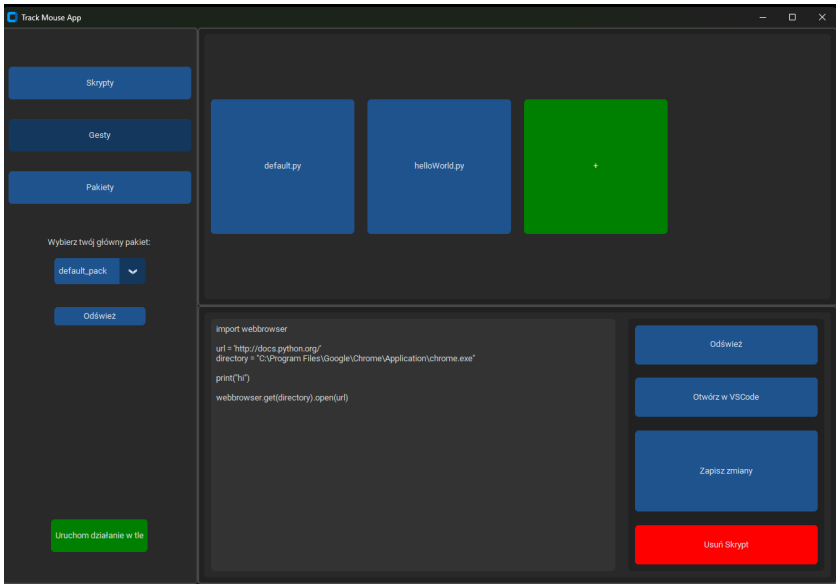
7 Przykład korzystania z Aplikacji

7.1 Wymagania wstępne

Przed uruchomieniem aplikacji użytkownik powinien pobrać odpowiednie zewnętrzne biblioteki pythonowe t.j: Pyautogui, Keyboard, Tensorflow, Pillow, Numpy, CustomeTkinter, SciPY, Scikit-learn, Matplotlib, Pandas. Więcej informacji o wersjach danych bibliotek znajduje się w pliku requirements.txt, który umieszczony jest bezpośrednio w folderze projektu.

7.2 Zakładka skryptów

Po pobraniu będzie możliwość uruchomienia aplikacji z poziomu wierzchołka projektu, za pomocą wpisania komendy w terminalu: 'python ./src/main.py'. Po wprowadzeniu komendy powinniśmy zobaczyć interfejs graficzny jak przedstawiony na grafice 10.

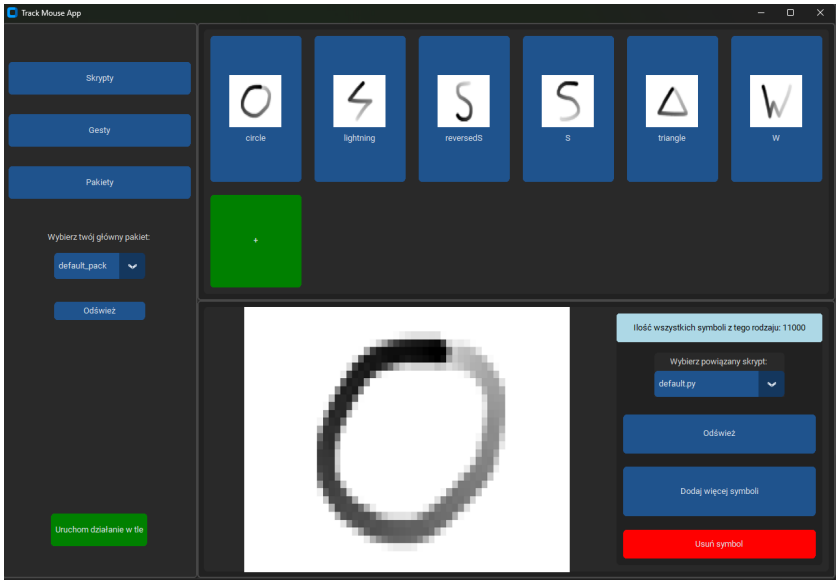


Rysunek 10: Zakładka skryptów

Jest to zakładka skryptów. Możemy dodawać tutaj dowolne pythonowe skrypty, które będą wykonywane podczas rozpoznawania symboli. Mamy także możliwość otwarcia skryptu w Visual Studio Code (jeśli go posiadamy) lub modyfikowania skryptu bezpośrednio z interfejsu. Po zakończeniu modyfikacji należy pamiętać o kliknięciu 'zapisz zmiany'.

7.3 Zakładka symboli

Po lewej stronie widnieje panel boczny z którego możemy się przełączać między zakładami. Po kliknięciu na przycisk 'Gesty' powinniśmy zaobserwować na naszym ekranie zmianę zakładki na taką jak ukazana na obrazku 11.

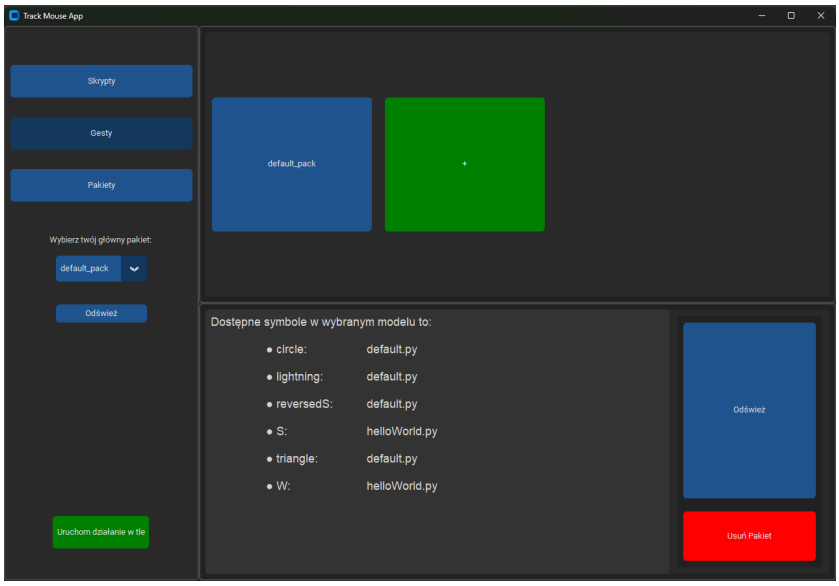


Rysunek 11: Zakładka symboli

W tej części możemy szybko i łatwo dodawać nowe symbole oraz przypisywać do nich wybrane skrypty. Klikając na przycisk 'Dodaj więcej symboli' zostaniemy przekierowani do narzędzia do zbierania danych w którym będziemy mogli rozpocząć proces tworzenia nowych zdjęć.

7.4 Zakładka pakietów

W celu przełączenia się do ostatniej zakładki powinniśmy kliknąć przycisk z lewej strony ekranu 'Pakiety'. Jej wygląd prezentuje się tak jak przedstawiono na rysunku 12.



Rysunek 12: Zakładka pakietów

Tutaj możemy przeglądać nasze pakiety oraz tworzyć nowe. Klikając na przycisk plus zostaniemy przekierowani do menadżera tworzenia pakietów. Będziemy mogli tam wybrać dowolne symbole z wszystkich istniejących i przekazać je do nauki sieci neuronowej. Po zakończeniu procesu tworzenia zostaniemy przekierowani z powrotem do naszej zakładki pakietów.

7.5 Uruchomienie aplikacji

Ostatecznie możemy wybrać jeden z naszych pakietów, ustawiając go jako główny w panelu bocznym. Następnie klikając przycisk 'Uruchom działanie w tle' rozpoczynamy proces śledzenia ruchu myszki na ekranie. Jednak aby piksele zostały przekazane do narzędzia do przetwarzania danych i kolejno do sieci w celu predykcji, musimy najpierw przytrzymać klawisze 'Ctrl' oraz 'q'. Po puszczeniu klawiszy powinniśmy zaobserwować skutki odpowiedniego skryptu.

8 Wnioski

Projekt, mający na celu rozpoznawanie ruchu 2D myszki niezależne od czasu wykonywania ruchu, osiągnął sukces w realizacji założonych celów. Przy użyciu narzędzia do zbierania i walidacji danych, zdołaliśmy gromadzić informacje o trajektorii ruchu myszki i przetwarzać je w odpowiednie symbole. Opracowany klasyfikator efektywnie rozpoznaje stworzone symbole, co umożliwia skojarzenie ich z odpowiednimi skryptami. Natomiast intuicyjny interfejs graficzny pozwala na łatwe tworzenie nowych skryptów, symboli i pakietów (zbiorów symboli) oraz zarządzanie nimi.

9 Link do repozytorium

https://github.com/this0is0kuba/symbolRecognizerGUI

Literatura

[MGU06] Betke Margrit, Oleg Gusyatin, and Mikhail Urinson. Symbol design: a user-centered method to design pen-based interfaces and extend the functionality of pointer input devices. *Universal Access in the Information Society 4*, 223–236, (2006).

[web] Image classification on mnist: https://paperswithcode.com/sota/image-classification-on-mnist (data dostępu 26.01.2024).