

Diagnostyka Systemów

System Identyfikacji Znaków Drogowych

Kacper Stenka 232603 Mateusz Chruściński 232347

Termin zajęć: Wtorek 11:15

Prowadzący: Dr inż. Łukasz Jeleń

1 Cel i zakres projektu

Celem projektu było zaimplementowanie systemu służącego do identyfikacji znaków drogowych przy użyciu analizy obrazu. Wykorzystane do tego zostało nauczanie maszynowe na podstawie głębokich sieci neuronowych.

Jako środowisko programistyczne został użyty program PyCharm, natomiast jako język programowania została wybrany Python w wersji 3.7. Głównymi bibliotekami użytymi do procesu obróbki zdjęć były: OpenCV, Keras 2.4.3 oraz TensorFlow 2.3.0.

W projekcie została wykorzystana sieć konwolucyjna. Jest to algorytm głębokiego nauczania specjalizujący się w wykrywaniu najważniejszych obiektów na obrazie. Głównym zadaniem sieci konwolucyjnych jest redukcja wymiarowości danych, do prostej przetwarzalnej formy, przy zachowaniu klu-czowych cech.

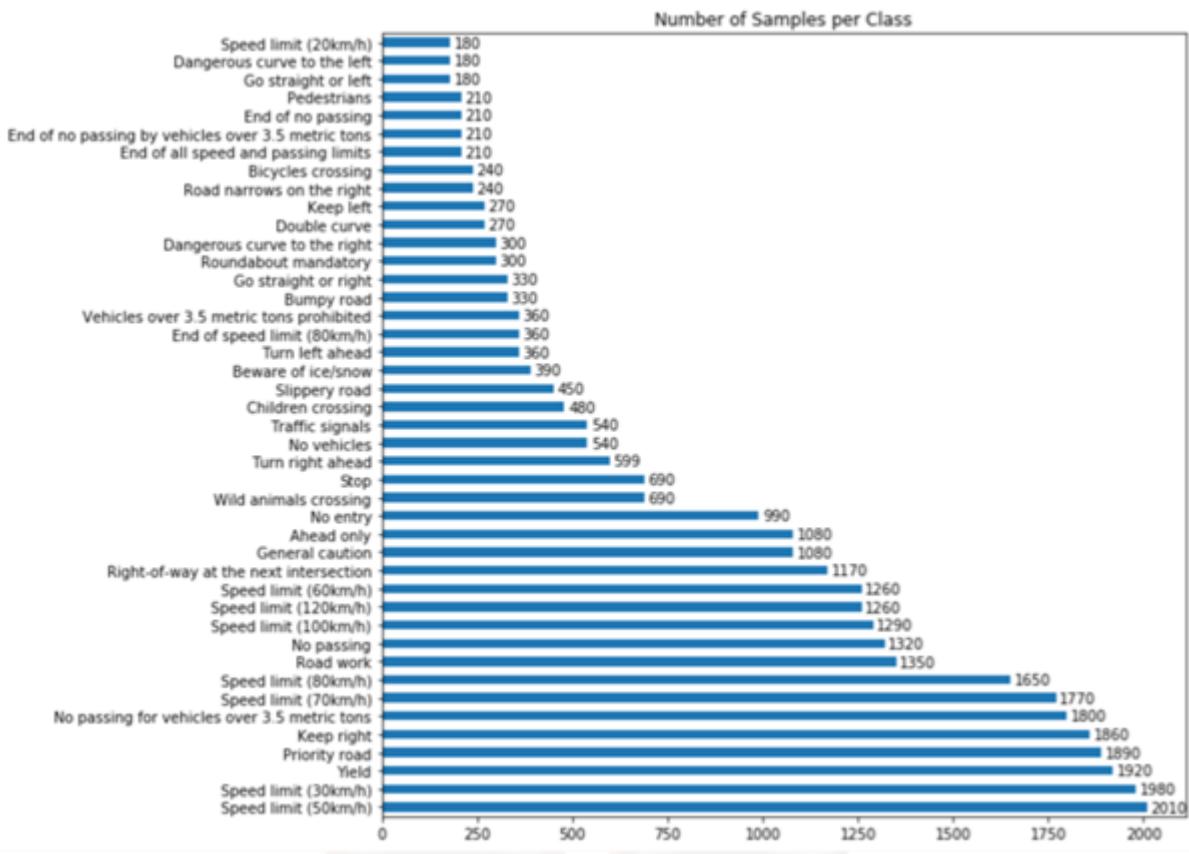
2 Przebieg projektu

2.1 Opis bazy danych

Bazą zastosowaną podczas realizacji projektu była baza „German Traffic Sign Recognition Benchmark (GTSRB)”. Jest to baza zawierająca ponad 30 tys. zdjęć znaków, każde zdjęcie zapisane jest w formacie .jpg w rozdzielczości

32x32 piksele. W bazie znajduje się ponad 40 klas, przy czym każde zdjęcie należy tylko do jednej klasy.

Podczas procesu uczenia modelu za pomocą konwolucyjnej sieci neuronowej warto zwrócić uwagę na ilość danych zawierających się w każdej kategorii. Na Rysunku 1 znajdują się kategorie opisane wartościami występującymi w nich elementów. Na podstawie zbiorów z największymi i najmniejszymi ilościami danych, w późniejszej fazie testów modelu, wykonano badania odnoszące się do precyzji wykrywanych znaków.



Rysunek 1: Baza danych GTSRB

2.2 Opis obróbki zdjęć

Pierwszym etapem obróbki zdjęć jest operacja „cv2.equalizeHist(img)”. Ma ona na celu wzmacnianie kontrastu pomiędzy obszarami obrazu, w których występują jaśniejsze oraz ciemniejsze miejsca.

Drugim etapem jest poddanie zdjęcia operacji „cv2.cvtColor”. Ma ona na celu przetworzenie obrazu do techniki czarno-białej oraz zmniejszenie wagiego zdjęcia z (32x32x3) na (32x32x1).

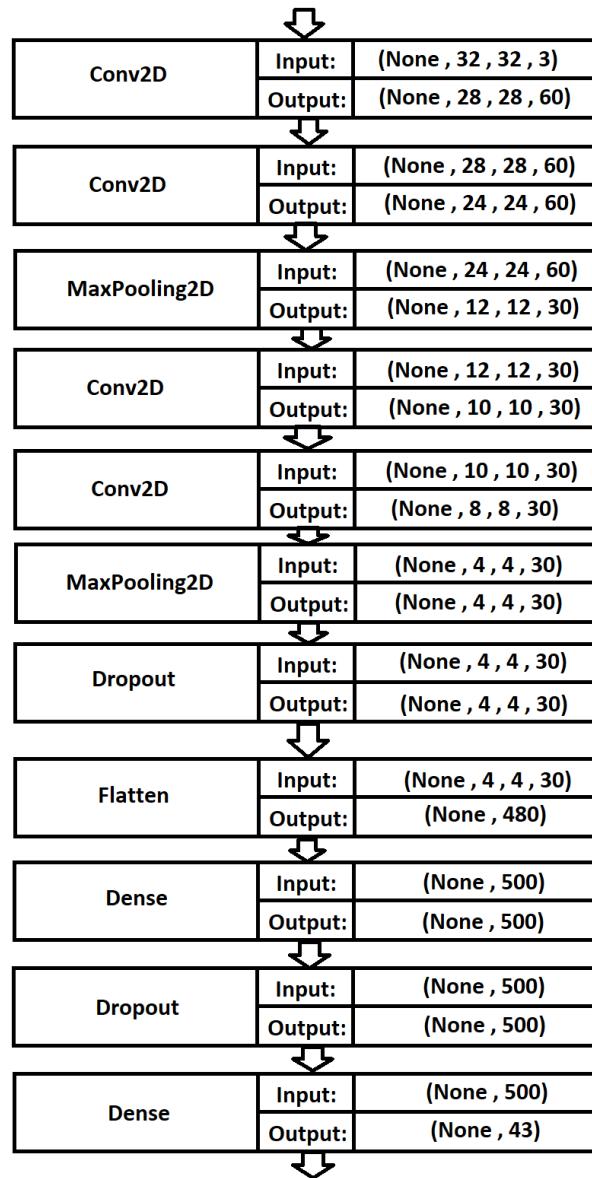
Baza danych została poddana procesowi powiększenia (ang. Augmentation) za pomocą komendy „ImageDataGenerator”. Zabieg ten miał na celu zwiększenia bazy treningowej potrzebnej do nauki modelu. Dodane obrazy bazują na posiadanych już wcześniej zdjęciach, które poddawane są procesom obróbki takim jak np.: przesunięcie, obrót, przybliżenie, oddalenie.

2.3 Zastosowany model

Na podstawie bibliotek Keras i TensorFlow stworzony został model sekwencyjny. Model ten po wstępnych testach zawierał w sobie 12 warstw:

- Warstwa Conv2D – warstwa konwolucyjna mająca na celu, za pomocą zdefiniowanego jądra/filtru reprezentowanego przez macierz, przesuwanie się po macierzy wejściowej i poddawanie kolejnych obszarów obrazu operacji mnożenia macierzy. Służy to do wychwycenia pewnych cech takich jak np.: krawędzie, kolory itp. Pojedyncza warstwa konwolucyjna może składać się z więcej niż jednego jądra, przy czym każde z nich specjalizuje się w detekcji zdefiniowanej cechy.
- Warstwa MaxPooling2D – głównym zadaniem tej warstwy jest redukcja przestrzenności macierzy cech z warstw konwolucyjnych (np. Conv2D). Funkcja ta ma również na celu wyciągnięcie dominującej cechy zbioru, a także ustabilizowanie i zachowanie efektywności treningu.
- Warstwa Dropout – warstwa mająca na celu ustawienie jednostki wejściowej na wartość 0. Proces ten zachodzi z ustaloną częstotliwością. Pozostałe wartości są skalowane, tak aby suma wszystkich wejść się nie zmieniła.
- Warstwa Flatten – warstwa spłaszczająca dane wejściowe.

- Warstwa Dense – warstwa implementująca operację aktywacji, gdzie aktywacja jest funkcją aktywacji warstwy, a jądro to macierz wag.

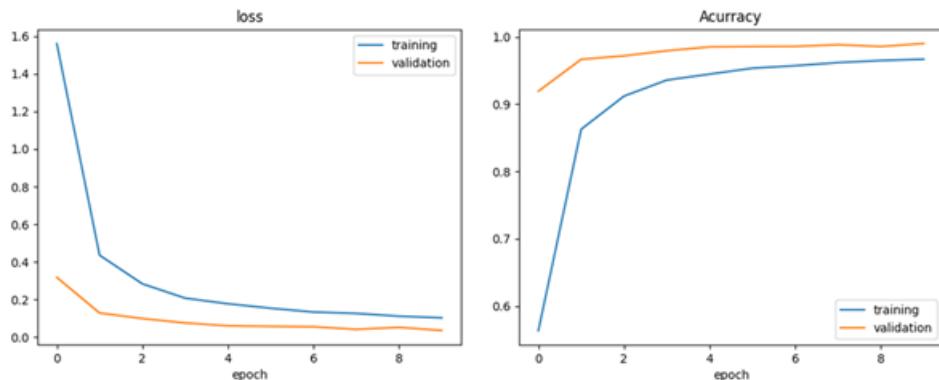


Rysunek 2: Architektura warstw modelu konwolucyjnego

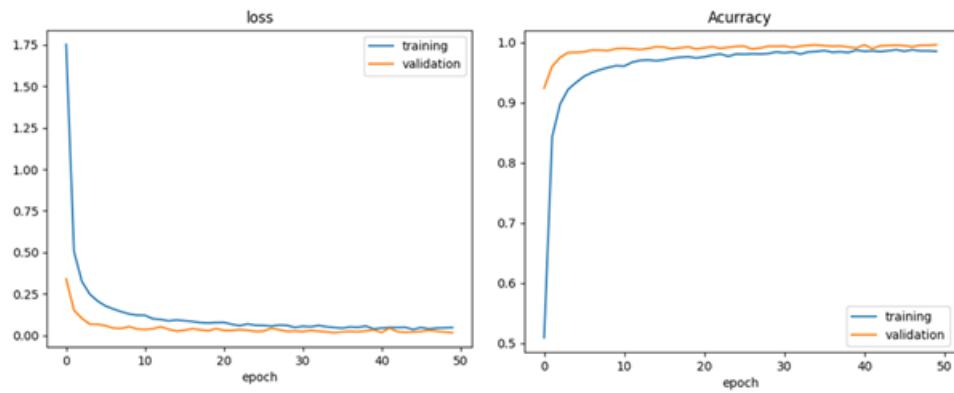
2.4 Trening oraz testowanie modelu

W celu wytrenowania modelu, zbiór danych podzielono na trzy kategorie. Pierwszą kategorią była kategoria danych przeznaczonych do treningu, zbiór ten zawierał 22271 elementów. Pozostałe zbiory to zbiory służące do testu oraz walidacji, posiadające kolejno 6960 oraz 5568 elementów.

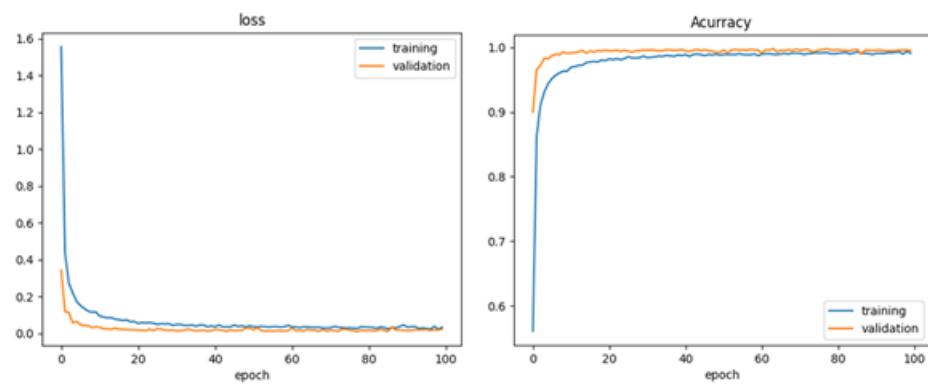
Aby sprawdzić poprawność działania modelu zmieniane zostały parametry służące do jego szkolenia. Pierwszym parametrem, który został poddany badaniom był parametr „epochs”, który definiuje, ile razy algorytm uczenia będzie działał na zestawie danych i jest podawany w komendzie „model.fit()”. Wartości tego parametru zostały kolejno ustawiane na wartości 10/50/100/200/500, przy stałej wartości „batch_size = 50”. Na Rysunkach 3, 4, 5, 6, i 7, pokazano, w jaki sposób zachowuje się szkolony model po zaimplementowaniu wyżej wymienionych parametrów.



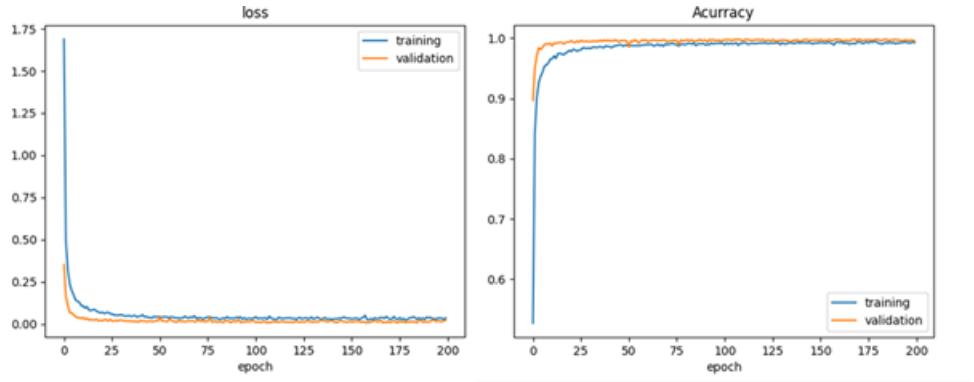
Rysunek 3: Charakterystyki dokładności i straty modelu dla wartości epochs = 10



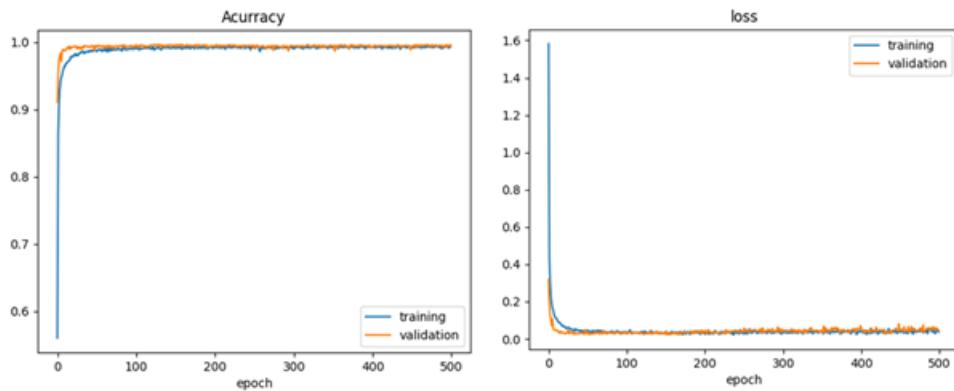
Rysunek 4: Charakterystyki dokładności i straty modelu dla wartości epochs = 50



Rysunek 5: Charakterystyki dokładności i straty modelu dla wartości epochs = 100



Rysunek 6: Charakterystyki dokładności i straty modelu dla wartości epochs = 200



Rysunek 7: Charakterystyki dokładności i straty modelu dla wartości epochs = 500

Następnie porównane zostały wyniki dla dwóch grup znaków. W pierwszej grupie znaków znajdowały się znaki takie jak „Ograniczenie prędkość do 50km/h” oraz „Droga z pierwszeństwem przejazdu”. Kategorie te posiadały jedne z największych ilości próbek, kolejno 1890 oraz 2010 zdjęć.

Jak można zauważyć na Rysunku 8, program działający na podstawie wytrenowanego modelu nie miał problemu z wykryciem znaku oraz poprawnym określeniem rodzaju znaku.



Rysunek 8: Porównanie precyzji wykrywania obiektu dla modeli wyszkolonych na wartościach epoch równych A)10, B)50, C)100, D)200, E)500

W przypadku znaku „Ograniczenie prędkości do 50km/h” (Rysunek 9) można zauważyć, że przy wartości epoch = 500, wartość precyzji wykrycia znaku spada. Jest to spowodowane przetrenowaniem modelu.



Rysunek 9: Porównanie precyzji wykrywania obiektu dla modeli wyszkolonych na wartościach epochs równych A)10, B)50, C)100, D)200, E)500

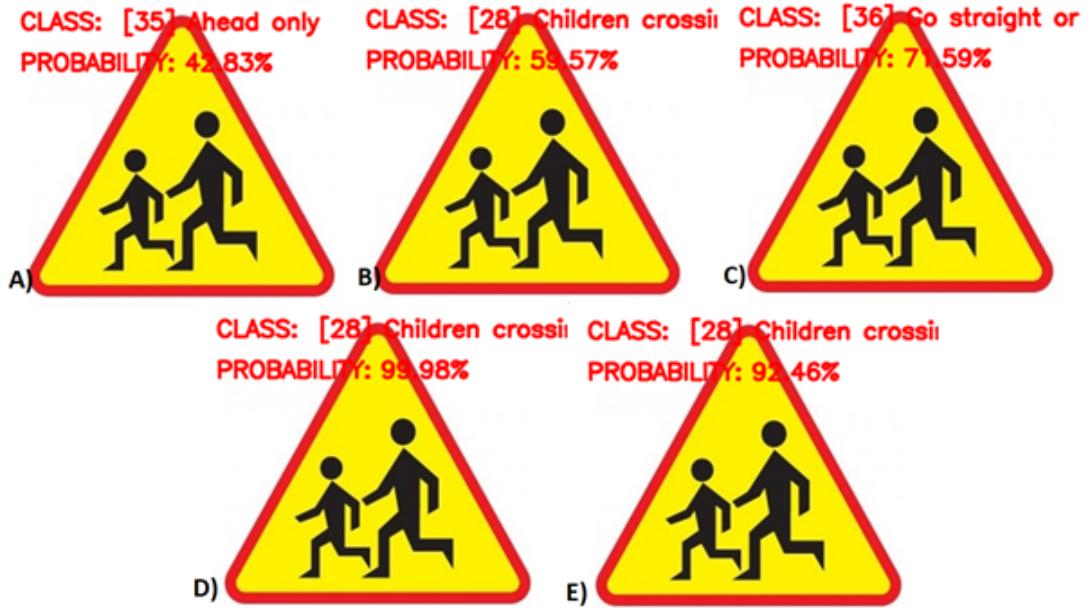
W drugiej grupie znajdowały się znaki, których ilość próbek w porównaniu do reszty znaków była jedną z najniższych. Pierwszym znakiem z tej grupy był znak „Ruch okrężny”, liczba próbek wynosiła 300.



Rysunek 10: Porównanie precyzji wykrywania obiektu dla modeli wyszkolonych na wartościach epoch równych A)10, B)50, C)100, D)200, E)500

Jak można zauważyć, ze względu na niską ilość próbek przy wartościach epoch = 10 oraz 50, algorytm oparty na wytrenowanym modelu wskazywał zły znak. Najlepszą wartością w tym wypadku okazała się wartość epoch = 200, ponieważ przy wyższych wartościach np.: 500, model ulegał przetrenowaniu, a precyzja wykrycia znaku spadała.

Ostatnim przykładem znaku z drugiej grupy był znak „Uwaga dzieci”. Posiadał on w swojej bazie 480 próbek. Jak można zauważyć na Rysunku 11, przy wartościach epoch = 10/50/100 model nie był w stanie poprawnie określić przynależności znaku. Przy wartości epoch = 200, modelowi udało się niemal ze 100% dokładnością określić przynależność znaku, natomiast przy zwiększeniu tego parametru do wartości 500, model uległ przetrenowaniu, a wartość precyzji spadła.



Rysunek 11: Porównanie precyzji wykrywania obiektu dla modeli wyszkolonych na wartościach epoch równych A)10, B)50, C)100, D)200, E)500

Jak można zauważyć w Tabeli 1., najlepszą wartością okazała się wartość epochs=200. Przy niższych wartościach osiągano niższą precyzję (w przypadku wartości podkreślonych na czerwono identyfikacja była błędna), natomiast w przypadku zwiększenia wartości epochs, dochodziło do przeuczenia modelu, a co za tym idzie spadku wartości precyzji identyfikacji znaków.

Tabela 1.

Zestawienie precyzji identyfikacji znaku dla wartości batch_size = 50, przy różnych wartościach epochs.

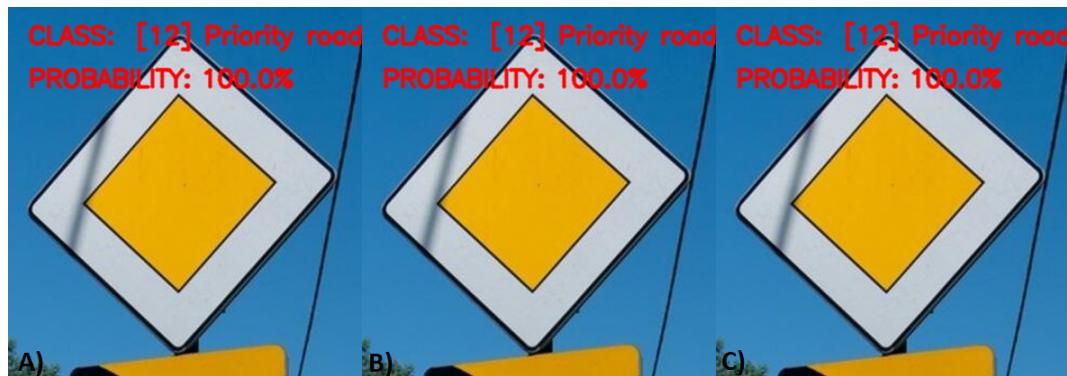
Epochs	Ruch okrężny	Pierwszeństwo	Ograniczenie 50 km/h	Uwaga dzieci
10	61,23	99,99	99,99	42,83
50	99,68	100	100	59,57
100	99,99	100	100	71,59
200	100	100	100	99,98
500	99,99	100	100	92,46

2.5 Testowanie pod względem batch_size

Następnie przy ustalonej wartości epoch=200, która okazała się najlepsza w poprzednim badaniu, zmodyfikowana została wartość batch_size, która definiuje liczbę próbek, które musi przejść sieć przed aktualizacją wewnętrznych parametrów i która w poprzednim eksperymencie przyjmowała wartość 50. Wartość ta w celach badawczych została zmieniana na wartości 20 oraz 100.



Rysunek 12: Porównanie precyzji wykrywania obiektu dla modeli wyszkolonych na wartościach batch_size równych A)20, B)50, C)100



Rysunek 13: Porównanie precyzji wykrywania obiektu dla modeli wyszkolonych na wartościach batch_size równych A)20, B)50, C)100



Rysunek 14: Porównanie precyzji wykrywania obiektu dla modeli wyszkolonych na wartościach batch_size równych A)20, B)50, C)100



Rysunek 15: Porównanie precyzji wykrywania obiektu dla modeli wyszkolonych na wartościach batch_size równych A)20, B)50, C)100

Tabela 2.

Zestawienie precyzji identyfikacji znaku dla wartości epochs = 200, przy różnych wartościach batch_size.

Batch_size	Ruch okrężny	Pierwszeństwo	Ograniczenie 50 km/h	Uwaga dzieci
20	99,09	100	59,91	30,52
50	100	100	100	99,98
200	82,92	100	100	48,78

Jak można zauważyć w Tabeli 2., zmiana wartości batch_size wpłynęła negatywnie na precyzję wykrywania znaków. Otrzymana poprzednio dokładność okazała się wystarczająca.

2.6 Badanie modelu w czasie rzeczywistym

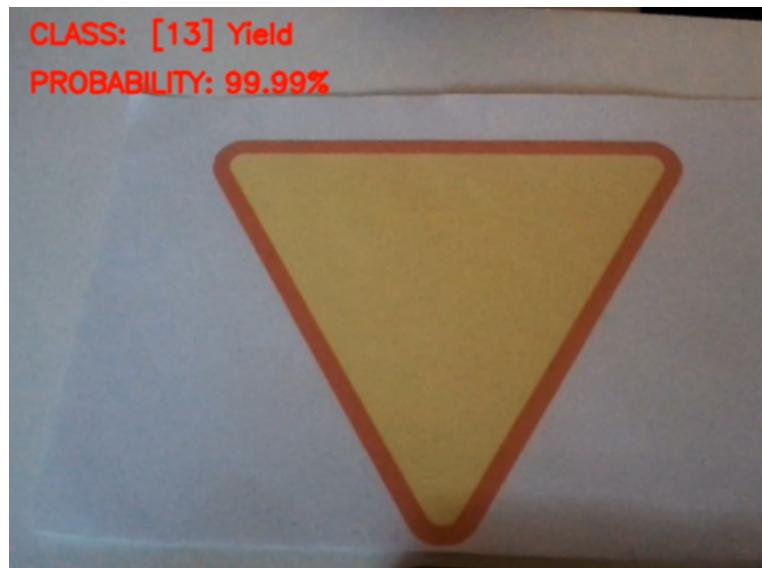
Badanie modelu w czasie rzeczywistym zostało przeprowadzone w dwóch podejściach. Pierwszym podejściem było wykorzystanie kamery z telefonu „ASUS ZenFone 3 ZE520KL”. Kamera została przenierowana poprzez sieć wi-fi za pomocą aplikacji Droid Cam, a następnie wykorzystana w algorytmie służącym do identyfikacji znaków drogowych w czasie rzeczywistym. Jako obiekty zostały zastosowane trzy znaki wydrukowane na kartakach A4. Znaki te to kolejno: Ograniczenie prędkości do 50km/h, Droga z pierwszeństwem przejazdu oraz Ustęp pierwszeństwa. Wyniki badań były pozytywne, udało się uzyskać precyzję w granicy 95% dla każdego ze znaków. Podczas drugiego podejścia zastosowano lepszą jakościowo kamerkę internetową „Trust Tyro Full HD”. Wyniki badań okazały się wyraźnie lepsze w porównaniu do poprzedniego podejścia, średnia precyzja wynosiła około 98,5%.



Rysunek 16: Badanie precyzji wykrywania znaku przy pomocy kamerki internetowej w czasie rzeczywistym



Rysunek 17: Badanie precyzji wykrywania znaku przy pomocy kamerki internetowej w czasie rzeczywistym



Rysunek 18: Badanie precyzji wykrywania znaku przy pomocy kamerki internetowej w czasie rzeczywistym

3 Wnioski

Za pośrednictwem środowiska programistycznego PyCharm oraz narzędzi takich jak biblioteki „OpenCV” (służącej do obróbki obrazu) oraz bibliotek „Keras” i „TensorFlow” (służących do skonstruowania modelu głębokich sieci neuronowych) udało się wyszkolić model pozwalający na identyfikację znaków drogowych z wysoką precyzją.

Wykorzystana baza danych „German Traffic Sign Recognition Benchmark” pozwoliła na uzyskanie satysfakcjonujących rezultatów. Baza ta jest bazą specyficzną, ponieważ posiada dane w małym formacie, 32x32 piksele. Specyfika ta sprawdza się w momencie, kiedy konwertowany obraz z kamery również przetwarzany jest do postaci 32x32 piksele. W wyniku takiego przetwarzania okazało się, że przy ustawieniu niskiej wartości granicznej wykrycia obiektu w czasie rzeczywistym, na wyjściu algorytmu pojawiały się informacje o wykryciu znaków, których na obrazie nie było. Takie informacje pojawiały się w momencie ustawienia wartości granicznej wykrycia w granicach 40-60%. Można temu zapobiec stosując dwie techniki. Pierwszą techniką było podniesienie wartości granicznej wykrycia w celu uniknięcia pojawiania się niewłaściwych informacji na wyjściu algorytmu. Drugą było wykorzystanie odpowiedniego filtra. Działanie filtru polegało na wyświetleniu informacji o identyfikacji znaku dopiero po potwierdzeniu tej samej informacji w 20 kolejnych klatkach. Zastosowany filtr spełnił założenia teoretyczne oraz nie wpływał w widoczny sposób na czas identyfikacji znaku.

Zaimplementowana architektura modelu konwolucyjnego została skonstruowana poprawnie, o czym świadczą wysokie wyniki precyzji. Należy jednak pamiętać, że dana architektura sprawdza się jedynie w przypadku, kiedy na obrazie widoczny jest jeden element, który należy zidentyfikować. W celu próby wykrycia kilku znaków, należałoby zaimplementować architekturę oraz algorytmy mające na celu przeszukiwanie obszarów przetwarzanego obrazu w celu znalezienia obiektu a następnie porównania go do modelu, przykładami takich architektur są np.: modele RetinaNet oraz Yolov3.

Podczas badań w czasie rzeczywistym należy pamiętać, że obraz który przekazywany jest do algorytmu w celu identyfikacji znaków również poddawany jest preprocessingowi, w którego skład wchodzą takie procesy jak konwertowanie obrazu do postaci czarnobiałej oraz operacja „equalize” mająca na celu wzmacnianie kontrastu pomiędzy obszarami obrazu. Dlatego też podczas pracy algorytmu należy zwrócić szczególną uwagę na rozchodzenie się światła, ponieważ ma ono duży wpływ na precyzję identyfikacji.