

Mateusz Erezman 171675
Michał Hajdasz 172156
Dariusz Kobiela 175656

TASK 1

Klasyfikacja typów pojazdów naziemnych

Zaawansowane przygotowanie danych w uczeniu maszynowym

Opis problemu

System będzie rozpoznawał zdjęcia pojazdów naziemnych i odpowiednio je klasyfikował. Wyróżnione klasy to:

- **Samochody osobowe** (wraz z minivanami)
- **Samochody ciężarowe** wraz z autobusami i furgonetkami (zakładamy, że furgonetki to zarówno dostawczaki, samochody typu Kangoo oraz osobowe 9-osobowe czyli busiki)
- **Pojazdy dwukołowe** - motocykle, skutery, rowery, hulajnogi elektryczne

Na zdjęciu powinien znajdować się tylko jeden rodzaj pojazdu. W przypadku gdy na zdjęciu znajdować się będą pojazdy różnych klas, zachowanie systemu nie będzie zdefiniowane.

Dane Wejściowe

Zdjęcia przetworzone do rozdzielczości około 150x150 pixeli w formacie jpg, w kolorystyce RGB gdzie każdy kolor opisany jednym bajtem. Na zdjęciu powinien być widoczny tylko jeden typ pojazdu.

Dane wejściowe będą pozyskiwane ręcznie jako zdjęcia w formacie 1:1, przez członków zespołu wykorzystując aparaty zawarte w telefonach komórkowych. Jeżeli okaże się to niewystarczającą liczbą danych lub liczbość grup okaże się zbyt niezbalansowana, większa liczba danych może zostać pobrana z dostępnych w internecie źródeł.

Przykładowe obrazy wejściowe:

- **Samochody osobowe**



- **Samochody ciężarowe**



- **Pojazdy dwukołowe**



Dane Wyjściowe

System zwróci klasę pojazdu znajdującego się na obrazie oraz odpowiednią wartość prawdopodobieństw.

Możliwe będą 3 klasy:

1. Samochód osobowy - zdjęcie na którym znajduje się samochód osobowy lub minivan
2. Pojazd dwukołowy - zdjęcie na którym znajduje się motocykl, skuter, rower lub hulajnoga elektryczna

3. Samochód ciężarowy - zdjęcie na którym znajduje się samochód ciężarowy, autobus, busik, furgonetka lub dostawczak

Zostaną zwrócone wartości prawdopodobieństwa dla każdej rozpoznanej klasy. Dlatego system będzie miał trzy wyjścia numeryczne, po jednym dla każdej klasy. Każda wartość będzie liczbą rzeczywistą z zakresu [0,1], gdzie 0 = 0% ufności, a 1 = 100% ufności.

Przykładowe dane wyjściowe:

	Samochód osobowy - 0.95 Pojazd dwukołowy - 0.01 Samochód ciężarowy - 0.04
	Samochód osobowy - 0.04 Pojazd dwukołowy - 0.93 Samochód ciężarowy - 0.03
	Samochód osobowy - 0.10 Pojazd dwukołowy - 0.05 Samochód ciężarowy - 0.85

Mateusz Erezman 171675
Michał Hajdasz 172156
Dariusz Kobiela 175656

TASK 2

Klasyfikacja typów pojazdów naziemnych

Zaawansowane przygotowanie danych w uczeniu maszynowym

1. Opisz dane, które są potrzebne do celów treningu i testowania.

Po konsultacji części pierwszej projektu zdecydowaliśmy się na rozszerzenie liczby klas do 5. Nasze dane muszą zawierać zdjęcia pojazdów należących do grup pojazdów samochodowych, ciężarowych, autobusów, motorów oraz rowerów. Chcielibyśmy, aby liczебность tych grup była do siebie jak najbardziej zbliżona, co jednak nie będzie łatwe do zapewnienia. Między innymi dlatego, oraz z powodu dużej liczby potrzebnych danych, nie wystarczą nam dane, które zbierzymy sami, więc do zrobionych przez nas zdjęć będziemy musieli dołożyć obrazy wyszukane w internecie i je ze sobą odpowiednio połączyć.

Na pojedynczym zdjęciu nie powinien znajdować się więcej niż jeden z rozpoznawanych przez nas typów pojazdów. Fotografowany obiekt powinien znajdować się w centrum zdjęcia i wypełniać jego znaczną część (być głównym elementem zdjęcia). Zdjęcia będą robione z osobami obok/na/w pojeździe lub bez tych osób (ten czynnik nie będzie brany pod uwagę).

Zdjęcie zostaną przez nas przetworzone do oczekiwanej przez model rozdzielczości. Używanie zdjęć niższej rozdzielczości spowoduje nie tylko przyspieszenie procesu uczenia się, ale również poprawę jego jakości, ponieważ zwiększenie rozmiaru tzw. batcha sieci pozwoli jej widzieć więcej zależności między danymi. Przeglądając internetowe bazy danych o treści przypominającej naszą możemy zauważyc, że używane w nich są zdjęcia o rozdzielczości około 300x300px (https://ai.stanford.edu/~jkrause/cars/car_dataset.html).

2. Zaplanuj procedurę zbierania danych

- a) Spróbuj przewidzieć, jakiego rodzaju dane będą potrzebne i jakie są potencjalne źródła danych

Do naszego treningu potrzebujemy danych z wszystkich czterech typów, które chcemy rozróżnić, czyli:

- Samochody osobowe (wraz z minivanami) [**class: car**]
- Samochody ciężarowe wraz z furgonetkami (zakładamy, że furgonetki to samochody dostawcze i samochody typu Kangoo) [**class: truck**]
- Autobusy - autobusy miejskie oraz osobowe 9-osobowe czyli busiki [**class: bus**]
- Motocykle - motocykle, skutery [**class: motorcycle**]
- Pojazdy dwukołowe - rowery, hulajnogi elektryczne [**class: bicycle**]

Potencjalnymi źródłami tych danych mogą być:

- własnoręcznie zrobione zdjęcia pojazdów na ulicach telefonem komórkowym
- internetowe zbiory danych np. http://mmlab.ie.cuhk.edu.hk/datasets/comp_cars/index.html
- ogólnodostępne zdjęcia w internecie np. Google images (szczególnie przydatne dla motocykli, ponieważ ta klasa będzie prawdopodobnie najmniej liczna w danych zebranych z dwóch powyższych źródeł)

- b) Spróbuj oszacować ilość danych, których będziesz potrzebować dla każdej klasy.

Zakładamy, że każda osoba będzie w stanie wykonać około 200 zdjęć pojazdów, co daje nam teoretycznie 200 zdjęć na klasę (zakładając idealne zbalansowanie klas). Jest to stanowczo zbyt mała liczba do dobrego wytrenowania sieci, dodatkowo część z tych danych zostanie użyta do utworzenia zbioru testowego i walidacyjnego.

Będziemy musieli skorzystać z pozostałych wymienionych źródeł w celu zwiększenia liczebności klas oraz ich zbalansowania. Końcowo zakładamy po 600 zdjęć na każdą klasę. Nie chcemy zakładać zbyt dużej ilości danych, ponieważ wszystkie te dane (zarówno zrobione przez nas oraz ściągnięte z internetu) będziemy musieli etykietować, co zakładając 10 sekund na etykietowanie jednego zdjęcia, zajmie nam łącznie $1800 * 10 / 3600 \text{ h} = 5 \text{ h}$.

- c) Spróbuj przewidzieć potencjalne problemy związane z gromadzeniem danych, opisz sposoby minimalizowania tego ryzyka

- Zbalansowanie klas

Aby późniejszy proces uczenia przebiegał sprawnie chcielibyśmy, aby nasze klasy były jak najbardziej zbalansowane. Jednakże zebranie podobnej ilości zdjęć samochodów osobowych, motocykli, rowerów, autobusów oraz samochodów ciężarowych będzie trudne, ponieważ w mieście o tej porze roku zdecydowana większość pojazdów to samochody osobowe, zdecydowanie mniej jest samochodów ciężarowych, a motocykli aktualnie praktycznie nie widujemy na ulicach. Autobusy i rowery również mogą nie być łatwe do odnalezienia.

Dysproporcje postaramy się skorygować poprzez dodanie do odpowiednich grup danych z internetu oraz poprzez augmentację danych w odpowiednich klasach.

- **Zdjęcia wykonywane różnymi urządzeniami**

Każdy z nas będzie fotografował pojazdy swoim telefonem, który różni się parametrami od pozostałych urządzeń. Ponadto zebrane przez nas dane będą łączone z danymi pobranymi z internetu, które mogą znacząco się różnić od naszych danych m. in. jakością, rozdzielczością, stosunkiem wysokości do szerokości, barwą itp.

Dodatkowo przeprowadzimy preprocessing danych, aby wprowadzić jednakowy rozmiar dla danych z internetu oraz zebranych przez nas.

- **Zdjęcia wykonywane w różnych warunkach**

Zdjęcia będziemy wykonywali prawdopodobnie przez kilka dni o różnych porach dnia i w różnej pogodzie, więc mogą się pojawić między nimi nieuchciane zależności.

Będziemy starać się wykonywać zdjęcia o podobnej porze dnia, gdy będzie jasno (**w ciągu dnia, nie w nocy**).

- **Wiele pojazdów na zdjęciach**

W mieście najczęściej jest tłoczno, a pojazdy poruszają się oraz są zaparkowane obok siebie co sprawi, że na wielu naszych zdjęciach znajdzie się kilka pojazdów jednocześnie.

Podczas robienia zdjęć będziemy starali się, aby w kadrze nie znalazło się wiele pojazdów na raz, a przede wszystkim, aby nie były to pojazdy należące do różnych, zdefiniowanych przez nas typów.

- **Prostokątne zdjęcia**

Zdjęcia najczęściej są wykonywane w stosunku szerokości do wysokości 16:9 lub 4:3, a na wejściu do sieci chcielibyśmy posiadać obrazy kwadratowe (1:1), dlatego zdjęcia będziemy wykonywać (lub obrabiać) do formatu 1:1.

Jeśli będzie to konieczne, przed wprowadzeniem do sieci będziemy musieli przerobić nasze zdjęcia oraz dane z internetu na kwadratowe np. poprzez ich rozciągnięcie lub wypełnienie przestrzeni jednolitymi pasami.

- Wysoka rozdzielcość zdjęć

W ogólności dążymy do tego, aby rozdzielcość fotografii była jak najwyższa, ponieważ do pewnego stopnia poprawia to ich ogólną jakość. Następnie zdjęcia zostaną przez nas przetworzone do oczekiwanej przez model rozdzielcości.

3. Zbierz dane stosując powyższą procedurę.

Zaproponuj kilka przykładów i opisz problemy, jakie napotkałeś.

Największym problemem podczas zbierania danych była duża reprezentatywność klasy "car" oraz mniejsza reprezentatywność pozostałych klas.

W celu pozyskania większej liczby zdjęć, zostały one dodatkowo pobrane z dostępnych w internecie źródeł udostępniających zbiory danych zdjęć różnego typu. Pobrane zostały zdjęcia każdej klasy, łącznie prawie 8000 zdjęć. Po pobraniu zdjęcia były sprawdzone ręcznie w celu zweryfikowania co się na nich znajduje, usunięcia niepoprawnych zdjęć, jak na przykład rysunki ciężarówek czy grafiki przedstawiające latające motocykle z przeszłości.

Kolejnym celem przeglądu danych było poprawienie części źle otagowanych zdjęć co było bardzo czasochłonnym zajęciem.

Niestety liczba dostępnych zdjęć dla klas również nie jest równa, klasa bicycle jest mniejsza od pozostałych, ponadto część zdjęć klasy bicycle jest formatu bmp podczas gdy wszystkie pozostałe zdjęcia są formatu jpg.

Zdjęcia te powinny zostać przekonwertowane do formatu jpg.

4. Zaplanuj procedurę adnotacji danych

a) Wybierz narzędzie do adnotacji, którego będziesz używać:

Na początku mieliśmy zamiar używać programu Label Studio napisanego w języku Python, jednak po konfiguracji środowiska i poetykietowaniu części zdjęć zdecydowaliśmy, że jako doświadczeni użytkownicy komputera obejdziemy się bez specjalnych programów i będziemy etykietować zdjęcia poprzez przenoszenie ich do odpowiednich folderów, co powinno przyśpieszyć cały proces.

Label Studio

Projects / carsZPD

Default									
Tasks		Columns		Filters		Order		not set	LF
ID	?	H	S	+	Annotated by	?	image	img	
1		1	0	0	(BR)				
2		1	0	0	(BR)				
3		1	0	0	(BR)				
4		1	0	0	(BR)				
5		1	0	0	(BR)				
6		1	0	0	(BR)				
7		1	0	0	(BR)				
8		1	0	0	(BR)				
9		1	0	0	(BR)				
10		1	0	0	(BR)				

Google drive: nazwa folderu to nazwa klasy. Poszczególne zdjęcia należy przenosić do właściwych folderów. Gdyby było to potrzebne, dodatkowo można wykorzystać istniejący skrypt do tagowania (w języku python).

- b) Stwórz precyzyjne instrukcje dotyczące wymagań związanych z adnotacją danych, sytuacji typowych i przypadków brzegowych

[class: bicycle] - tutaj wszystkie rowery i hulajnogi elektryczne

[class: bus] - tutaj wszystkie autobusy miejskie oraz busiki (osobowe 9-osobowe lub większe). Zwróćcie uwagę, że w przypadku dwóch wyglądających podobnie busików zwykle różnica jest taka, że busik dostawczy nie ma okien dla pasażerów (tylko na poziomie kierowcy), natomiast busik przewożący ludzi będzie miał okna na poziomie pasażerów.

[class: truck] - tutaj ciężarówki (małe i duże), samochody dostawcze. Zwróćcie uwagę, że w przypadku dwóch wyglądających podobnie busików zwykle różnica jest taka, że busik dostawczy nie ma okien dla pasażerów (tylko na poziomie kierowcy), natomiast busik przewożący ludzi będzie miał okna na poziomie pasażerów. Także dostawcze auta typu Kangoor.

[class: motorcycle] - tutaj wszystkie motory i skutery

[class: car] - tutaj wszystkie auta osobowe oraz minivany. Także osobowe auta typu Kangoor.

W przypadku jakichś niejasności, należy włożyć zdjęcie do osobnego folderu “**TRUDNOSC_Z_PRZYPISANIEM**”. Te zdjęcia zostaną otagowane na późniejszym etapie przez wszystkich adnotatorów poprzez głosowanie w arkuszu excel. Adnotujący dla każdego zdjęcia ma do wyboru wszystkie z pięciu dostępnych klas oraz dodatkową opcję “zdjęcie nieprzydatne”. Zdjęcie takie zostanie przydzielone do klasy o największej liczbie głosów. Jeśli zdjęcie nie zostało w większości przypisane do jednej konkretnej klasy lub zostało przypisane jako “zdjęcie nieprzydatne” to w takim wypadku zostanie ono odrzucone i nie będzie użyte w procesie wytwarzania modelu, ponieważ jest ono zbyt niejednoznaczne.

- c) Rozdziel surowe dane pomiędzy członków zespołu w celu późniejszej adnotacji.

Wszystkie zebrane przez nas dane umieściliśmy we wspólnym folderze. Następnie zdjęciom zostały nadane unikalne losowe nazwy, aby usunąć korelację czasową (zdjęcia zrobione kolejno po sobie posiadają podobne nazwy, są posortowane po kolejności ich zrobienia) oraz osobową (różne aparaty posiadają różne formaty nazewnictwa plików, przez co zdjęcia są pogrupowane po urządzeniach, którymi wykonano zdjęcia). W kolejnym kroku podzieliliśmy zbiór na trzy części i każdy z trzech annotatorów dostał jedną z nich do otagowania.

5. Dokonaj adnotacji na danych stosując zdefiniowaną procedurę. Przedstaw kilka przykładów i opisz problemy, jakie napotkałeś.

Adnotacja danych przebiegła stosunkowo płynnie - instrukcja była wyraźnie napisana, a w razie problemów annotatorzy konsultowali się ze sobą. Dane zostały rozdzielone na 5 klas. Poniżej przykłady z każdej klasy.

Głównymi napotkanymi problemami były busiki oraz auta osobowe typu Kangoor. **Potrzeba było większego zastanowienia, aby poprawnie rozdzielić je między klasy car, bus i truck. Jednym z głównych kryteriów były okna dla pasażerów w busikach (i na tej podstawie: okna/brak okien rozdzielenie między bus i truck).** W razie dużych problemów, zdjęcie było przenoszone do klasy "TRUDNOSC_Z_PRZYPISANIEM", aby potem wspólnie annotatorzy podjęli decyzję o danej klasie dla zdjęcia.

[class: car]



[class: truck]



[class: bus]



[class: motorcycle]



[class: bicycle]



6. Pomiar inter-observer variability/reliability

- Znajdź 50-100 trudnych lub niejednoznacznych obrazów w zbiorze danych

Zdjęcia wykonywaliśmy zgodnie ze stworzoną wcześniej instrukcją, która dobrze opisywała należyty sposób zbierania danych, dlatego też wśród zebranych przez nas danych nie znaleźliśmy dużej ilości niejednoznacznych obrazów. Po przejrzeniu zbioru danych udało nam się znaleźć 17 trudnych przypadków, które przekazaliśmy do adnotacji przez wszystkich członków zespołu.

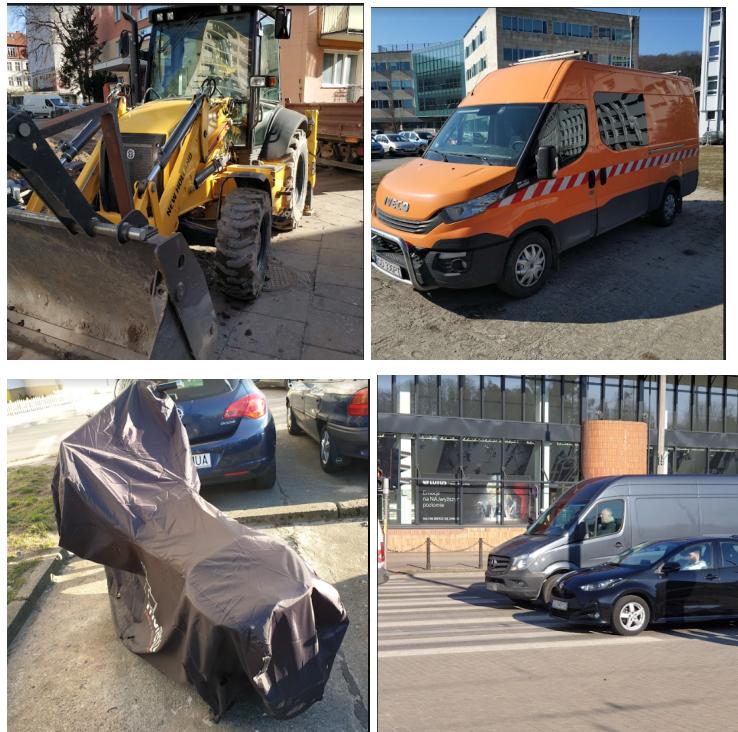
- Przekaż wszystkie te obrazy wszystkim członkom zespołu do adnotacji

Każdy z członków otrzymał dostęp do współdzielonego na dysku Google arkusza kalkulacyjnego ze stworzoną tabelą:

IMAGE_NAME	CLASS_BY_MH	CLASS_BY_ME	CLASS_BY_DK
TR1	2	2	5
TR2	0	0	0
TR3	2	1,2	1
TR4	2	1,2	1
TR5	2	1,2	1
TR6	0	0	0
TR7	0	0	0
TR8	2	2	5
TR9	2	2	5
TR10	1	1	1
TR11	1	1	1
TR12	1	1	1
TR13	2	1	1
TR14	2	1	1
TR15	2	1	1
TR16	0	3	3
TR17	0	0	3

- Oblicz zgodność adnotacji (inter-observer variability/reliability)

Do adnotacji przekazaliśmy 17 najtrudniejszych do otagowania obrazów. Wśród nich znalazły się m.in. :



Najtrudniejsze dane zostały otagowane przy użyciu arkusza excel i 6 dostępnych adnotacji:

CLASS_DICTIONARY	
CLASS_NUMBER	CLASS_NAME
1	car
2	truck
3	motorcycle
4	bicycle
5	bus
0	- (dana nieużyteczna)

Każdy annotator osobno ocenił przynależność zdjęć trudnych do odpowiednich klas. Wyniki zostały zagregowane do jednego pliku csv. Następnie obliczony został inter-observer variability przy użyciu biblioteki nltk w pythonie. Wyniośł on 57%.

```
1 print("Average observed agreement across all coders and items: " +str(ratingtask.avg_Ao()))
```

```
Average observed agreement across all coders and items: 0.5686274509803921
```

Kappa coefficient jest nieco niższy (około 45%) i wskazuje na moderate agreement (umiarkowaną zgodę) pomiędzy annotatorami.

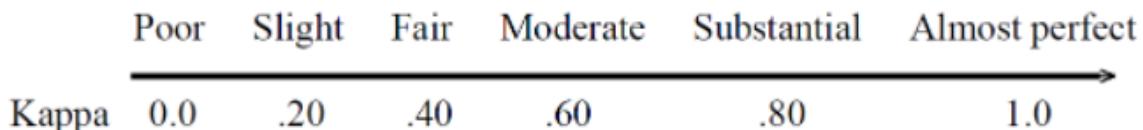
```

1 print("Kappa coefficient: " + str(ratingtask.kappa()))
2 print("Weighted kappa coefficient: " + str(ratingtask.weighted_kappa()))

```

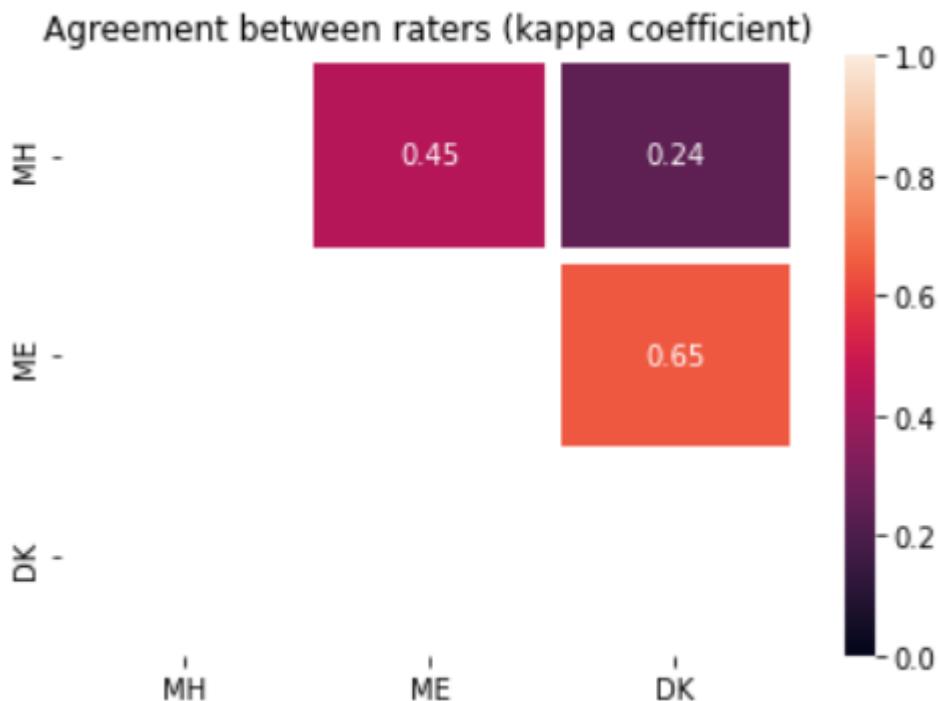
Kappa coefficient: 0.44630371458217866
 Weighted kappa coefficient: 0.44630371458217866

Interpretation of Kappa



<u>Kappa</u>	<u>Agreement</u>
< 0	Less than chance agreement
0.01–0.20	Slight agreement
0.21– 0.40	Fair agreement
0.41–0.60	Moderate agreement
0.61–0.80	Substantial agreement
0.81–0.99	Almost perfect agreement

Poniższa heatmapa przedstawia inter-observed agreement pomiędzy każdą parą z annotującymi. Najwyższa wystąpiła między ME i DK - 65%, a najniższa między MH i DK - 24%. MH i ME osiągnęli wyniki 45%.



7. Przeanalizuj utworzony zbiór danych

- Oblicz i przedstaw statystyki dotyczące danych (liczba obrazów, liczba przykładów, nazwy klas i ich liczba, itp.)

Łącznie udało nam się zebrać 8716 obrazów. Ręcznie zebranych zostało 736 obrazów, a z baz internetowych pobranych zostało 7980 obrazów.

	class_name	collected_images	downloaded_images	sum_of_images_in_the_class
0	bicycle	66	365	431
1	bus	46	1059	1105
2	car	488	3045	3533
3	motorcycle	50	2487	2537
4	truck	69	1024	1093
5	TRUDNOSC_Z_PRZYPISANIEM	17	0	17

Total sum of all images (collected + downloaded): 8716

Total sum of collected images: 736

Total sum of downloaded images : 7980

Dane są w różnych formatach. Zdjęcia wykonane przez nas są w całości w formacie JPEG. Natomiast zdjęcia pobrane z baz internetowych są w różnych formatach.

Collected data

jpg images: 736

all images: 736

Downloaded data

jpg images: 7428

all images: 7980

Formaty są 3: JPEG, BMP i PNG.

JPEG 8147

BMP 365

PNG 187

Name: format, dtype: int64

Zdjęcia mają różne rozdzielczości. Wszystkie są w formacie 1:1, natomiast każdy annotator posługiwał się innym telefonem, a więc różnią się liczbą pikseli.

Image from MH annotator

Image Dimension : (2976, 2976, 3)
Image Height : 2976
Image Width : 2976
Number of Channels : 3



Image from DK annotator

Image Dimension : (3024, 3024, 3)
Image Height : 3024
Image Width : 3024
Number of Channels : 3

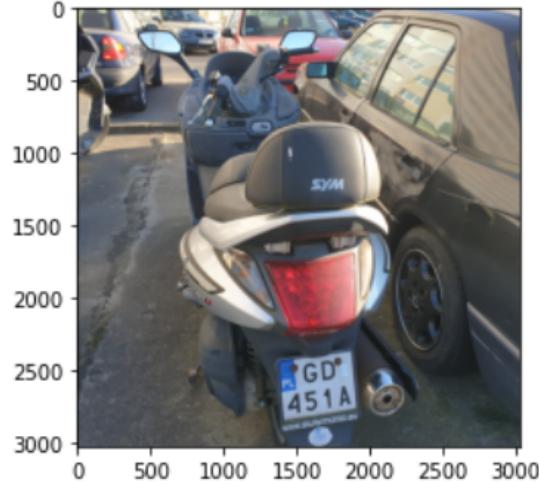


Image from ME annotator

Image Dimension : (4240, 4240, 3)
Image Height : 4240
Image Width : 4240
Number of Channels : 3

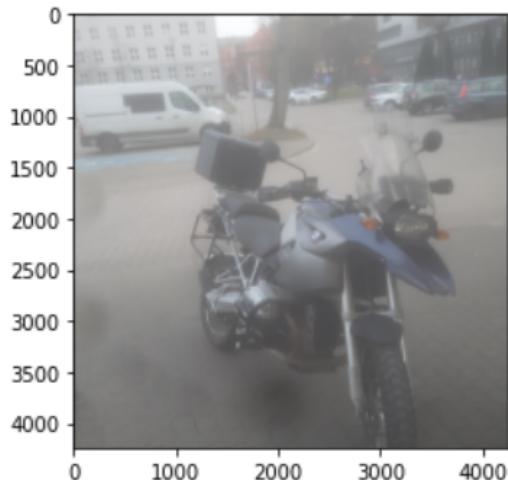
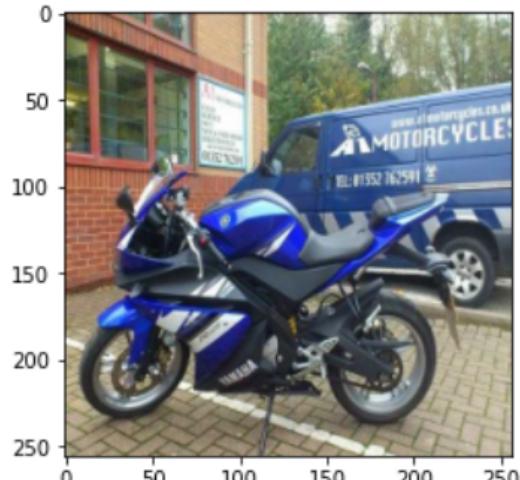


Image from internet database

Image Dimension : (256, 256, 3)
Image Height : 256
Image Width : 256
Number of Channels : 3



Dodatkowo, zdjęcia wykonane przez ME są mało wyraźne, gdyż Jego szybka od aparatu był uszkodzona podczas robienia zdjęć. Jednak te zdjęcia także będą brane pod uwagę - liczymy na to, że pomogą w generalizacji sieci (jako zdjęcia zawierające szum - noise).

Z wszystkich dostępnych zdjęć został wygenerowany plik csv zawierający dane i metadane o zdjęciach. Wygenerowany plik zawiera dane tekstowe, które są dużo łatwiejsze w analizie niż obrazy.

name	mode	format	size	R	G	B	source	datetime		
motorcycle0	RGB	JPEG	(2976, 2976)	[13318, 5107, 37 [6249, 5455, 311 [9755, 3928, 241	collected			2022:03:14 17:1		
motorcycle1	RGB	JPEG	(3024, 3024)	[340, 62, 68, 67, [3, 5, 4, 3, 7, 4, 1 [27, 7, 2, 5, 7, 7,	collected			2022:03:14 14:4		
motorcycle2	RGB	JPEG	(3024, 3024)	[4, 0, 3, 1, 0, 1, 2 [0, 0, 0, 0, 0, 0, 0 [0, 0, 0, 0, 0, 0, 0,	collected			2022:03:14 14:4		
motorcycle3	RGB	JPEG	(3024, 3024)	[12, 4, 6, 16, 12, [0, 0, 0, 0, 0, 0, 0 [0, 0, 0, 0, 0, 1, 0,	collected			2022:03:14 14:4		
motorcycle4	RGB	JPEG	(3024, 3024)	[0, 0, 0, 0, 0, 0, 0 [0, 0, 0, 0, 0, 0, 0 [0, 0, 0, 0, 0, 0,	collected			2022:03:14 14:4		
motorcycle5	RGB	JPEG	(3024, 3024)	[8, 3, 9, 3, 6, 12, [0, 0, 0, 0, 0, 0, 0 [0, 0, 0, 0, 0, 0, 1,	collected			2022:03:14 14:4		
motorcycle6	RGB	JPEG	(3024, 3024)	[1035, 223, 230, [670, 303, 253, 2 [3191, 345, 465,	collected			2022:03:14 17:1		
motorcycle7	RGB	JPEG	(3024, 3024)	[856, 240, 209, 2 [726, 281, 271, 2 [2012, 257, 319,	collected			2022:03:14 17:1		
motorcycle8	RGB	JPEG	(3024, 3024)	[9318, 2777, 226 [6486, 3056, 277 [10842, 2145, 31	collected			2022:03:14 17:1		
motorcycle9	RGB	JPEG	(3024, 3024)	[1291, 360, 361, [539, 387, 346, 3 [1748, 607, 883,	collected			2022:03:14 17:1		
resolutionUnit	Xresolution	Yresolution	brightness	aperture	phonemake	phonemodel	ISOspeedrating	flash	focallength	sensingmethod
2 (72, 1)	(72, 1)	(233, 100)	(169, 100)	Xiaomi	Mi Note 3		138		24 (3820, 1000)	2
2 (72, 1)	(72, 1)	(544, 100)	(252, 100)	samsung	SM-G965F		50		0 (430, 100)	
2 (72, 1)	(72, 1)	(574, 100)	(252, 100)	samsung	SM-G965F		50		0 (430, 100)	
2 (72, 1)	(72, 1)	(546, 100)	(252, 100)	samsung	SM-G965F		50		0 (430, 100)	
2 (72, 1)	(72, 1)	(454, 100)	(252, 100)	samsung	SM-G965F		64		0 (430, 100)	
2 (72, 1)	(72, 1)	(600, 100)	(252, 100)	samsung	SM-G965F		50		0 (430, 100)	
2 (72, 1)	(72, 1)	(382, 100)	(252, 100)	samsung	SM-G965F		100		0 (430, 100)	
2 (72, 1)	(72, 1)	(382, 100)	(252, 100)	samsung	SM-G965F		100		0 (430, 100)	
2 (72, 1)	(72, 1)	(425, 100)	(252, 100)	samsung	SM-G965F		80		0 (430, 100)	
2 (72, 1)	(72, 1)	(403, 100)	(252, 100)	samsung	SM-G965F		100		0 (430, 100)	

Tabela zawiera następujące kolumny:

- **name** - nazwa pliku
- **mode** - model barw np. RGB
- **format** - format obrazu np. JPEG
- **size** - rozmiar obrazu
- **R, G, B** - histogram, liczba pikseli o danej jasności
- **source** - sposób pozyskania obrazu, zdjęcie pozyskane ręcznie lub pobrane z dostępnej bazy (collected, downloaded)
- **datetime** - data i godzina wykonania zdjęcia
- **resolutionUnit** - Jednostka miary dla X i Y Resolution. Możliwe wartości to (1, 2 ,3) gdzie 1 -Brak bezwzględnej jednostki miary. Używany do obrazów, które mogą mieć niekwadratowy współczynnik proporcji, ale nie mają znaczących wymiarów bezwzględnych 2 - cal 3 - centymetr. Domyślna wartość to 2
- **XResolution, YResolution** - Liczba pikseli na ResolutionUnit w odpowiednim kierunku. Nie jest obowiązkowe, aby obraz był faktycznie wyświetlany lub drukowany w rozmiarze sugerowanym przez ten parametr. Od aplikacji zależy, czy wykorzysta te informacje w jakiś sposób.
- **brightness** - ustawienie jasności użyte podczas robienia zdjęcia
- **aperture** - ustawienie apertury użyte podczas robienia zdjęcia
- **phonemake** - producent urządzenia robiącego zdjęcie
- **phonemodel** - model urządzenia robiącego zdjęcie
- **ISOSpeedRatings** - wskazuje czułość ISO i szerokość geograficzną ISO aparatu
- **flash** - Wskazuje stan lampy błyskowej w momencie wykonania zdjęcia.
- **focallength** - Rzeczywista ogniskowa obiektywu w mm
- **sensingmethod** - Wskazuje typ przetwornika obrazu w aparacie lub urządzeniu wejściowym.

Są to w większości metadane które nie przydadzą się do samego porównywania i klasyfikacji zdjęć ale mogą pomóc przy dzieleniu danych na zbiory treningowy, testowy i walidacyjny.

Dalsza analiza pokazała, jakie rozmiary zdjęć występują najczęściej. Mediana (najczęściej występujące rozmiary) to 256x256. Jest ich około 3000/9000 (8800), a więc aż 34% wszystkich danych. Dlatego decydujemy się na skalowanie wszystkich zdjęć do rozmiaru 256x256.

	resolutionUnit	sensingmethod	height	width
count	1240.000000	425.000000	8699.000000	8699.000000
mean	3.051613	1.990588	671.150822	583.275089
std	14.600040	0.137037	888.970043	874.136093
min	2.000000	0.000000	64.000000	58.000000
25%	2.000000	2.000000	256.000000	256.000000
50%	2.000000	2.000000	260.000000	256.000000
75%	2.000000	2.000000	640.000000	480.000000
max	300.000000	2.000000	4608.000000	4240.000000

Image_size; Number_of_images_sizes

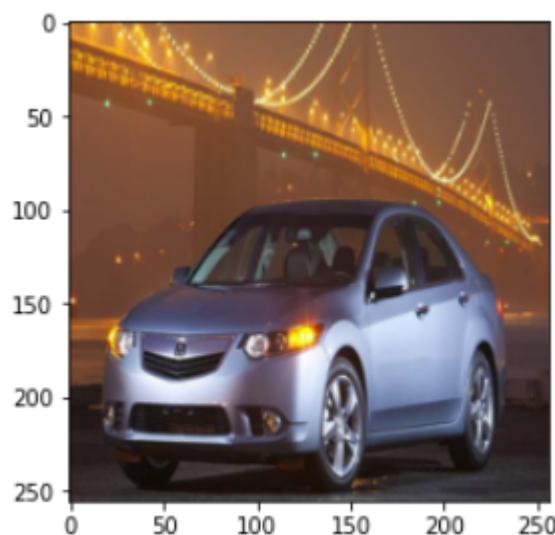
256	3006
64	1060
480	772
3024	287
2976	239
300	228
4240	193
768	187
194	164
375	157
360	99
225	87
600	69
1200	68
400	59
333	55
183	48
500	45
469	30
240	30
640	29
533	26

Zdjęć o rozmiarach mniejszych niż 256x256 jest około 1200. Zdjęcia te zostaną rozciągnięte.

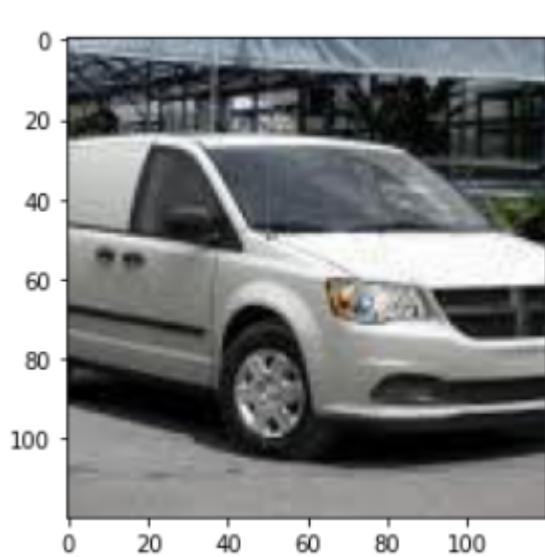
```
1 df.height[df.height < 256].count()
```

```
1171
```

Przykład zmniejszenia obrazu:



Przykład rozciągnięcia zdjęcia:



- b) Przeanalizuj relacje i korelacje w danych (grupy danych, podobne dane, korelacje między etykietami / klasami, źródła, itd.) Zaprezentuj kilka przykładów.

Korelacje zostały obliczone przy użyciu kolumny datetime zawierającą datę i czas zrobienia zdjęcia. Zdjęcia tego samego pojazdu (z różnych stron, w różnych wersjach) zamkają się w przedziale 30 sekund. Ten próg zostanie użyty w późniejszej analizie do automatyzacji szukania skorelowanych zdjęć.

	name	mode	format	size	R	G	B	source	datetime
0	motorcycl	RGB	JPEG	(2976, 297 [13318, 51 [6249, 545 [9755, 392 collected					2022:03:14 17:13:44
1	motorcycl	RGB	JPEG	(3024, 302 [340, 62, 6 [3, 5, 4, 3, [27, 7, 2, 5 collected					2022:03:14 14:49:39
2	motorcycl	RGB	JPEG	(3024, 302 [4, 0, 3, 1, [0, 0, 0, 0, [0, 0, 0, 0, collected					2022:03:14 14:49:32
3	motorcycl	RGB	JPEG	(3024, 302 [12, 4, 6, 1 [0, 0, 0, 0, [0, 0, 0, 0, collected					2022:03:14 14:49:38
4	motorcycl	RGB	JPEG	(3024, 302 [0, 0, 0, 0, [0, 0, 0, 0, [0, 0, 0, 0, collected					2022:03:14 14:49:35
5	motorcycl	RGB	JPEG	(3024, 302 [8, 3, 9, 3, [0, 0, 0, 0, [0, 0, 0, 0, collected					2022:03:14 14:49:34
6	motorcycl	RGB	JPEG	(3024, 302 [1035, 223 [670, 303, [3191, 345 collected					2022:03:14 17:13:40
7	motorcycl	RGB	JPEG	(3024, 302 [856, 240, [726, 281, [2012, 257 collected					2022:03:14 17:13:41
8	motorcycl	RGB	JPEG	(3024, 302 [9318, 277 [6486, 305 [10842, 21 collected					2022:03:14 17:13:37
9	motorcycl	RGB	JPEG	(3024, 302 [1291, 360 [539, 387, [1748, 607 collected					2022:03:14 17:13:39
10	motorcycl	RGB	JPEG	(3024, 302 [810, 290, [413, 310, [741, 147, collected					2022:03:14 17:13:42
11	motorcycl	RGB	JPEG	(2976, 297 [308, 374, [18, 26, 47 [26, 10, 16 collected					2022:03:15 17:27:31
12	motorcycl	RGB	JPEG	(2976, 297 [25686, 76 [10287, 67 [5453, 254 collected					2022:03:15 17:27:35
13	motorcycl	RGB	JPEG	(2976, 297 [1183, 933 [443, 285, [84, 44, 76 collected					2022:03:15 17:27:39
14	motorcycl	RGB	JPEG	(3024, 302 [57, 28, 36 [13, 9, 17, [30, 17, 19 collected					2022:03:13 15:46:28

Kolumna datetime wskazuje na korelację ze sobą zdjęć o indeksach od 1 do 5. Kolejna grupa to obrazy 6-10. Ostatnia widoczna grupa to obrazy 11-13.

- c) Przeanalizuj zakłócenia i błędy w danych (rodzaje zakłóceń, rodzaje i liczba błędów, błędne etykiety, itd.) Zaprezentuj kilka przykładów.

Podczas tworzenia histogramów zauważliśmy, że 3 ze zdjęć pobranych z internetu są w odcieniach szarości. Zdecydowaliśmy jednak, że zostawimy je w naszym zbiorze, jako że nie powinno to mieć dużego wpływu na przebieg treningu.





Zauważaliśmy również, że w klasie "bus" przeważają pobrane z sieci zdjęcia żółtych amerykańskich autobusów, które różnią się znaczco od widywanych na naszych ulicach autobusów oraz busów. Może to spowodować słabe działanie sieci dla naszych danych wejściowych, ponieważ model może skojarzyć klasę z żółtym pojazdem.



W zdjęciach pobranych z internetu niestety znajduje się dużo danych, które nie są zgodne z naszymi wytycznymi z instrukcji, mianowicie na zdjęciach często pojazd nie znajduje się w centrum obrazu i nie jest widoczny w całości, tylko jego niewielka część, po której trudno, nawet człowiekowi, rozpoznać jaki to typ pojazdu, i czy w ogóle na zdjęciu znajduje się pojazd.



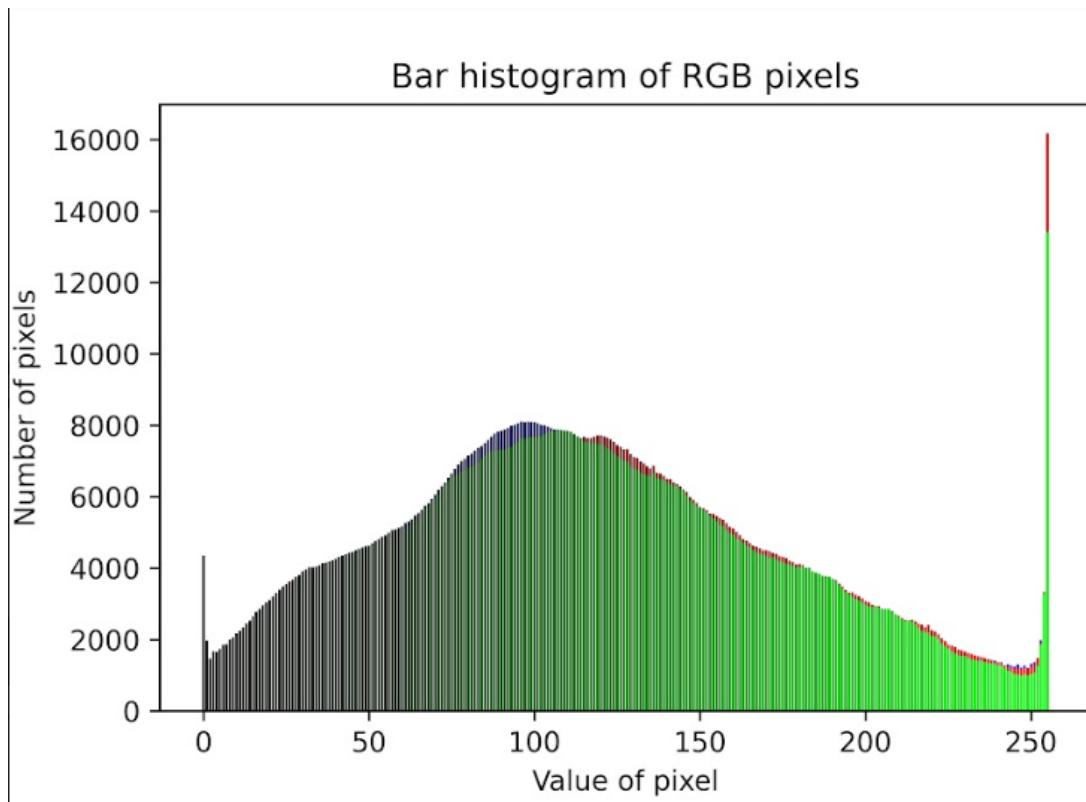


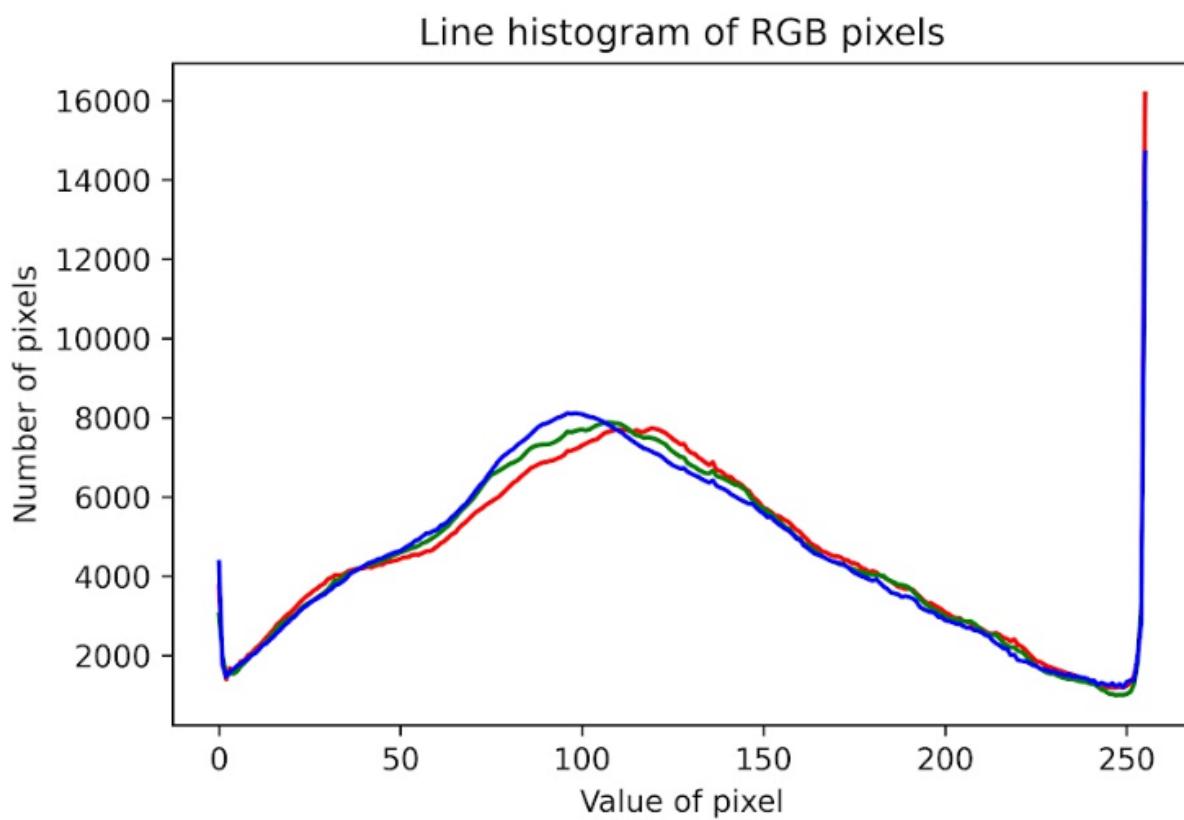
Wszystkie wyżej zamieszczone zdjęcia należą u nas do klasu "bus", jednak nie są to zdjęcia zgodne z naszymi wytycznymi.

Zdecydowaliśmy się jednak nie usuwać tych zdjęć, ponieważ przejrzenie tak dużej ilości danych pobranych z internetu zajęłoby bardzo dużo czasu.

- d) Zbadaj, czy niektóre dane są "trudniejsze" lub "ważniejsze" od innych, zdecyduj, czy trzeba je traktować oddziennie (np. nadawanie wag podczas treningu lub oddzielne zbiory testowe).

Zbiór danych jest bardzo ujednolicony. Analiza wykazała, że nie trzeba traktować specjalnie żadnej części danych. Dodatkowo, wykonane zostały histogramy intensywności barw.





8. Wstępne przetwarzanie danych

- a) Wybierz metodę reprezentacji danych. Skonwertuj dane.
- b) Zaproponuj i wykonaj standaryzację/normalizację danych oraz inne niezbędne przetwarzanie wstępne.

Nasze zdjęcia są w większości w formacie RGB. 187 jest w formacie RGBA, czyli Red, Green, Blue, Alpha - Alpha to transparentność. W celu ujednolicenia wszystkich zdjęć odrzucimy kanał alfa i zapiszemy wszystkie zdjęcia w formacie JPEG (który nie zawiera w sobie informacji o transparentności). Zdjęcia w formacie 'L' oznaczają skalę szarości, czyli tylko jeden kanał. Takie zdjęcia posiadamy tylko 3, jest to około 0,03% naszego zbioru, a więc zostawimy je w takiej formie.

```
1 df[ 'mode' ].value_counts()
```

RGB	8509
RGBA	187
L	3
Name:	mode, dtype: int64

c) Zaproponuj i wykonaj rozszerzenie danych (augmentację danych)

Celem naszej augmentacji danych było nie tylko zwiększenie ilości danych uczących, ale również zbalansowanie klas.

W tym celu postanowiliśmy rozszerzyć dane w klasach mniej licznych tak, aby dorównywały swoją liczebnością klasie o największej ilości danych (klaśa "car").

Augmentację przeprowadzaliśmy na danych z wyłączeniem danych przeznaczonych do zbioru testowego, aby uniknąć ewentualnej korelacji danych pomiędzy zborem treningowym i testowym.

Jako, że na zbiór testowy przeznaczyliśmy 10% wszystkich naszych danych, a chcieliśmy również, aby nasz zbiór testowy był zrównoważony pod względem ilości danych w poszczególnych klasach, to liczby danych do augmentacji prezentuje się następująco:

```
8716 (total images collected) * 0.1 (testing set size) / 5 (number of classes) = 174 (images per class for testing set)

Images left in training set:


- bicycle = 431 - 174 = 257
- bus = 1105 - 174 = 931
- car = 3533 - 174 = 3359
- motorcycle = 2537 - 174 = 2363
- truck = 1093 - 174 = 919



Images to augmented:


- bicycle = 3359 - 257 = 3102
- bus = 3359 - 931 = 2428
- car = 3359 - 3359 = 0
- motorcycle = 3359 - 2363 = 996
- truck = 3359 - 919 = 2440

```

Do załadowania danych użyliśmy funkcji

image_dataset_from_directory z biblioteki Keras dla każdej z klas.

Następnie do augmentacji skorzystaliśmy z klasy *ImageDataGenerator* z następującymi parametrami:

```
datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rotation_range=30,
    height_shift_range=0.1,
    width_shift_range=0.1,
    shear_range=0.2,
    brightness_range=[0.5, 1.5],
    zoom_range=0.1,
    channel_shift_range=50,
    horizontal_flip=True,
    preprocessing_function = generate_noise,
    fill_mode='nearest')
```

gdzie:

- rotation_range - zakres obracania zdjęcia
- height_shift_range - zakres przesunięcia pionowego
- width_shift_range - zakres przesunięcia poziomego
- shear_range - zakres zniekształcenie wzdłuż osi w celu zmiany kąta percepcji
- brightness_range - zakres modyfikacji jasności zdjęcia
- zoom_range - zakres przybliżenia zdjęcia
- channel_shift_range - zakres zmiany kanałów
- horizontal_flip - możliwy obrót poziomy
- fill_mode - wypełnienia powstały pustych przestrzeni (*nearest* - za pomocą koloru z najbliższego piksela)
- preprocessing_function - własna funkcja z dodatkowymi przekształceniemi, gdzie *generate_noise* wygląda następująco:

```
def generate_noise(img):
    noise_img = img/255
    probability = 3 # it means 1/probability chance for every noise addition

    # Add salt-and-pepper noise to the image.
    if random.randrange(probability) == 0:
        noise_img = random_noise(noise_img, mode='s&p', amount=random.random()/20) # 0 - 0.05

    # Add gaussian noise
    if random.randrange(probability) == 0:
        noise_img = random_noise(noise_img, mode='gaussian')

    # Add Contrast Limited Adaptive Histogram Equalization
    if random.randrange(probability) == 0:
        noise_img = skimage.exposure.equalize_adapthist(
            noise_img, clip_limit=random.random()/20) # 0 - 0.05

    # The above function returns a floating-point image
    # on the range [0, 1], thus we changed it to 'uint8' if we want to display it
    # and from [0,255]
    # noise_img = np.array(255*noise_img, dtype = 'uint8')
    return noise_img
```

Z pewnym stopniem prawdopodobieństwa dla każdego obrazu wykonujemy każdą z trzech operacji:

- dodanie szumu pod postacią “soli i pieprzu” w losowej ilości z przedziału (0, 0.05)
- dodanie szumu gaussowskiego
- lokalne wzmacnianie kontrastu z użyciem histogramu, wartość wzmacniania jest losowa z przedziału (0, 0.05)

Główna pętla odpowiedzialna za augmentację danych wygląda następująco:

```

# how many augmented images of each class we need
aug_nums = [('bicycle', 3102), ('bus', 2428), ('motorcycle', 996), ('truck', 2440)]

for aug in aug_nums:
    full_input_path = input_path + aug[0]
    imgs = loadImagesFromDirKeras(full_input_path)
    full_out_path = output_path + aug[0]

    datagen.fit(imgs)

    batch_size = 100
    num_of_iter = aug[1] // batch_size

    i = 0
    for batch in datagen.flow(imgs, batch_size=batch_size, save_to_dir=full_out_path, save_format='jpeg'):
        i += 1
        if i >= num_of_iter:
            break

```

Na początku zadajemy ilość zdjęć, które chcemy otrzymać po augmentacji, następnie dla każdej klasy ładujemy zdjęcia przy pomocy biblioteki Keras i wykonujemy metody fit oraz flow na naszym zbiorze danych. Ilość otrzymanych danych będzie przybliżona do oczekiwanej z dokładnością zależną od zadanej wielkości batch'a. Funkcja flow posiada parametr, który pozwala od razu zapisywać zdjęcia do odpowiedniej biblioteki.

Ostatecznie przy pomocy augmentacji wygenerowaliśmy:

- 3051 zdjęć dla klasy bicycle
- 2344 zdjęć dla klasy bus
- 950 zdjęć dla klasy motorcycle
- 2324 zdjęć dla klasy truck

Po wykonaniu augmentacji, suma danych treningowych (oryginalnych i wygenerowanych) wygląda następująco:

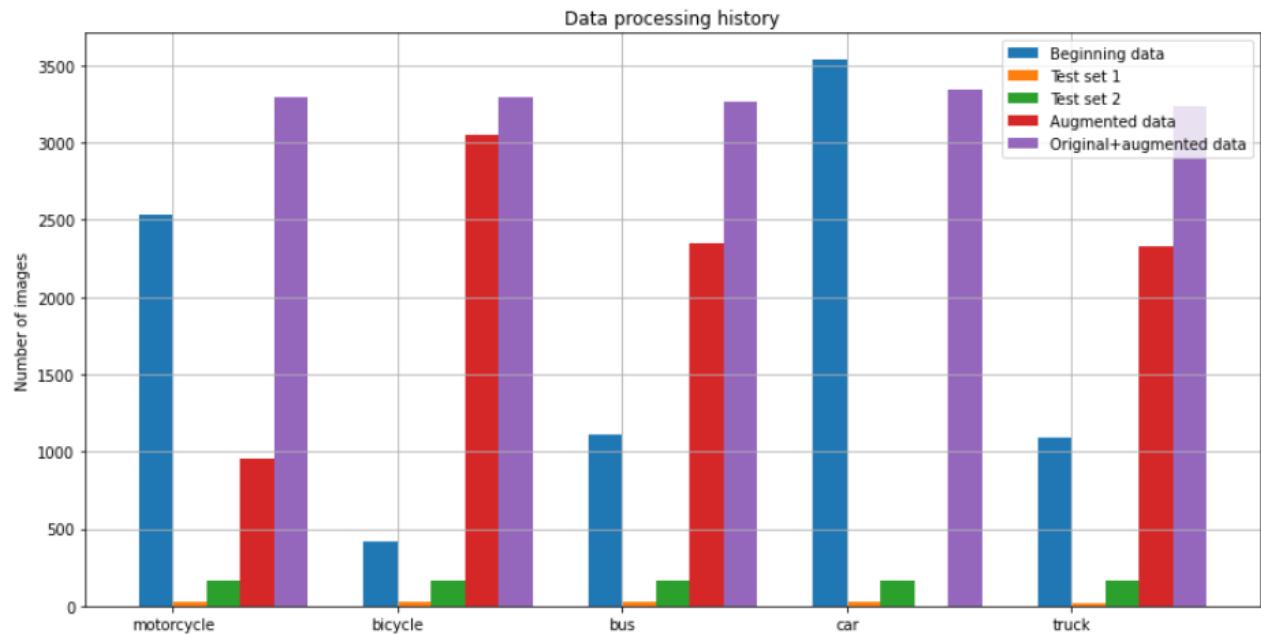
```
{'bicycle': 3297,
'bus': 3266,
'car': 3342,
'motorcycle': 3296,
'truck': 3236})
```

9. Podziel dane na zbiory: treningowy (TRAIN), walidacyjny (VAL) oraz testowy (TEST). Przedstaw kilka statystyk dla wygenerowanych zbiorów.

Stworzyliśmy 2 zbiory testowe. Jeden zawiera tylko dane zebrane przez nas, drugi zawiera tylko dane pobrane z internetu. Test set 1 zawiera około 15% danych oryginalnych (zebranych przez nas), test set 2 zawiera około 10% danych pobranych z internetu. Powodem wyznaczenia takich procentowych wartości zbiorów

testowych jest, aby nie zabrać z żadnej klasy więcej niż 50% danych (chodzi tu o klasę najmniej liczne).

- TEST SET 1: [in this set correlation needs to be checked]
 - $0.15 \times 736 = 110$ images;
 - 110 images / 5 classes = 22 images per class
- TEST SET 2:
 - $0.1 \times 7980 = 798$ images;
 - 798 images / 5 classes = 160 images per class



Przy generowaniu zbioru testowego 1 oraz późniejszych zbiorów walidacyjnych i treningowych sprawdzaliśmy korelację między zdjęciami. Przy generowaniu zbioru testowego do sprawdzenie korelacji została użyta kolumna datetime - czas wykonania zdjęcia i ograniczenie najbliższych 30 sekund.

Z kolei przy generowaniu zbiorów treningowych i walidacyjnych została użyta miara Structural similarity index (SSIM). Ta miara korelacji waha się w przedziale $[0, 1]$.

Wartość maksymalna 1 wskazuje, że dwa sygnały zdjęciowe są doskonale podobne strukturalnie. Natomiast wartość 0 wskazuje na brak podobieństwa.

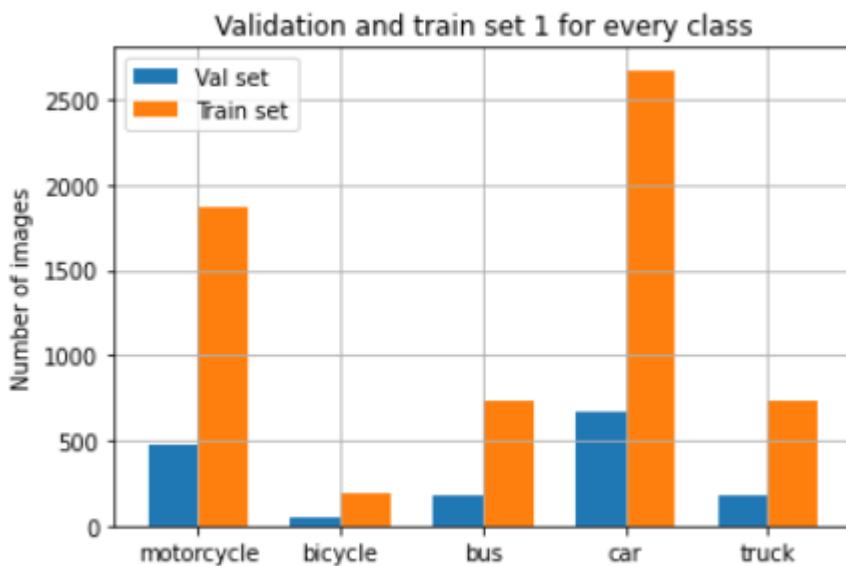
Zdecydowaliśmy, że para zdjęć jest traktowana jako skorelowana gdy miara SSIM jest większa niż 0.49 (tak więc nawet słaba korelacja jest brana pod uwagę).

TABLE I
MAPPING SSIM TO MEAN OPINION SCORE SCALE

SSIM	MOS	Quality	Impairment
≥ 0.99	5	Excellent	Imperceptible
[0.95, 0.99)	4	Good	Perceptible but not annoying
[0.88, 0.95)	3	Fair	Slightly annoying
[0.5, 0.88)	2	Poor	Annoying
< 0.5	1	Bad	Very annoying

- a. SPLIT 1. Original class distribution / proportion, original data (no normalization, no augmentation)

<i>Original train set 1</i>	<i>Val set 1</i>	<i>Train set 1</i>
{'bicycle': 246, {'bicycle': 49, {'bicycle': 197, 'bus': 922, 'bus': 184, 'bus': 738, 'car': 3342, 'car': 668, 'car': 2674, 'motorcycle': 2346, 'motorcycle': 470, 'motorcycle': 1876, 'truck': 912}) 'truck': 182}) 'truck': 730})		



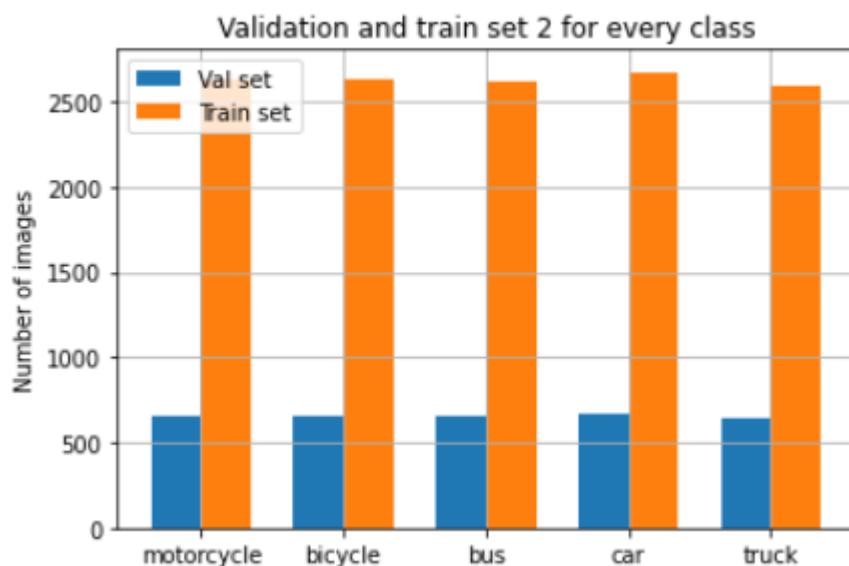
- b. SPLIT 2. Uniform class distribution / proportion, normalized and augmented data

<i>Original train set 2</i>	<i>Val set 2</i>	<i>Train set 2</i>
-----------------------------	------------------	--------------------

```

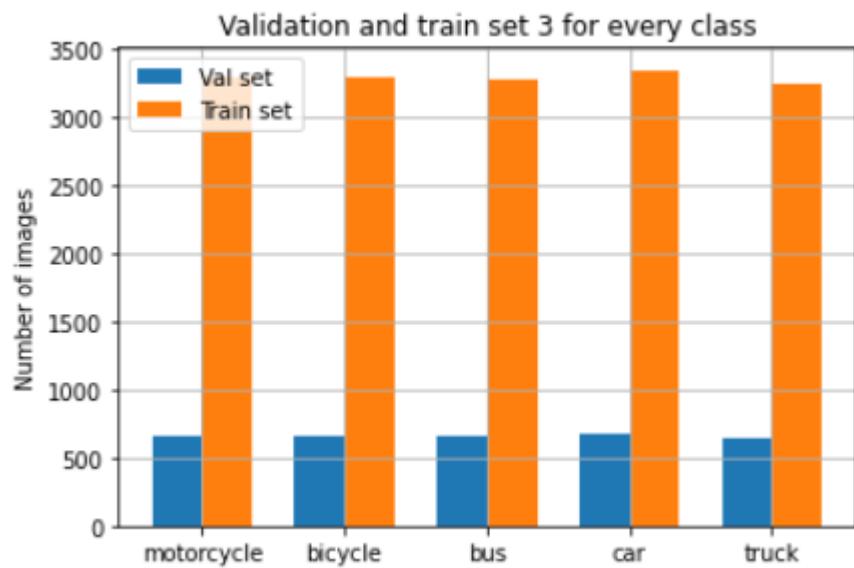
{'bicycle': 3297,      {'bicycle': 659,      {'bicycle': 2638,
'bus': 3266,           'bus': 653,       'bus': 2613,
'car': 3342,           'car': 668,       'car': 2674,
'motorcycle': 3296,    'motorcycle': 659,   'motorcycle': 2637,
'truck': 3236})        'truck': 647})     'truck': 2589})

```



c. SPLIT 3. Like "SPLIT2", but VAL set is added to TRAIN set

<i>Original train set 3</i>	<i>Val set 3</i>	<i>Train set 3</i>
<pre> {'bicycle': 3297, 'bus': 3266, 'car': 3342, 'motorcycle': 3296, 'truck': 3236}) </pre>	<pre> {'bicycle': 659, 'bus': 653, 'car': 668, 'motorcycle': 659, 'truck': 647}) </pre>	<pre> {'bicycle': 3297, 'bus': 3266, 'car': 3342, 'motorcycle': 3296, 'truck': 3236}) </pre>



10. Wszystkie kody źródłowełączamy do raport w formie notatnika jupyter notebook oraz wygenerowane w html.

Mateusz Erezman 171675
Michał Hajdasz 172156
Dariusz Kobiela 175656

TASK 3

Classification of types of ground vehicles

Advanced data preparation in machine learning

1. Basing on the problem you solve and the data you have, discuss and decide:
What is the type of the problem (classification, regression, detection, generation, etc.)
What kind of training methods you can use (supervised training, unsupervised, etc.)
What are possible machine learning methods and models to solve your problem? Choose a machine learning method, model and architecture suitable to your problem.
What are possible loss / fitness functions for your problem? Choose one of them and justify why.
Knowing which machine learning methods you use, describe possible testing metrics and procedures (required metrics: Loss, Accuracy, F-score, ROC, Confusion matrix) which are suitable to your problem.

Because our problem is image data classification, we will use the transfer learning method and adapt MobileNet V2 developed by Google to our problem. Transfer learning enables us to use a pre-trained model (with some data used by creators), and to fine-tune the model to our problem by training it again with data collected by us. It is a supervised learning problem.

[<https://towardsdatascience.com/review-mobilenetv2-light-weight-model-image-classification-8febb490e61c>]

[https://www.tensorflow.org/tutorials/images/transfer_learning]

Because our problem is multi-class classification (labels from 0 to 4), possible loss functions are **Multi-Class Cross-Entropy Loss** (which is the default loss function to use for multi-class classification problems), **Sparse Multiclass Cross-Entropy Loss** (used to problems with a large number of labels, which require one hot encoding process; it is performing the same cross-entropy calculation of error, without requiring that the target variable be one hot encoded prior to training) and **Kullback Leibler Divergence Loss** (most commonly used when using models that learn to

approximate a more complex function than simply multi-class classification, such as in the case of an autoencoder used for learning a dense feature representation under a model that must reconstruct the original input. In this case, KL divergence loss would be preferred. Nevertheless, it can be used for multi-class classification, in which case it is functionally equivalent to multi-class cross-entropy).

[<https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/>]

Implementation details. In keras library, the implementation of Sparse Multiclass Cross-Entropy Loss is called `SparseCategoricalCrossentropy`.

```
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
```

Chosen metrics to evaluate the model:

- **Loss: Sparse Multi-Class Cross-Entropy Loss.** Cross-entropy is the default loss function to use for multi-class classification problems. In this case, it is intended for use with multi-class classification where the target values are in the set {0, 1, 3, ..., n}, where each class is assigned a unique integer value.
- **Accuracy.**
- **F-score.**
- **ROC**
- **Confusion matrix.**

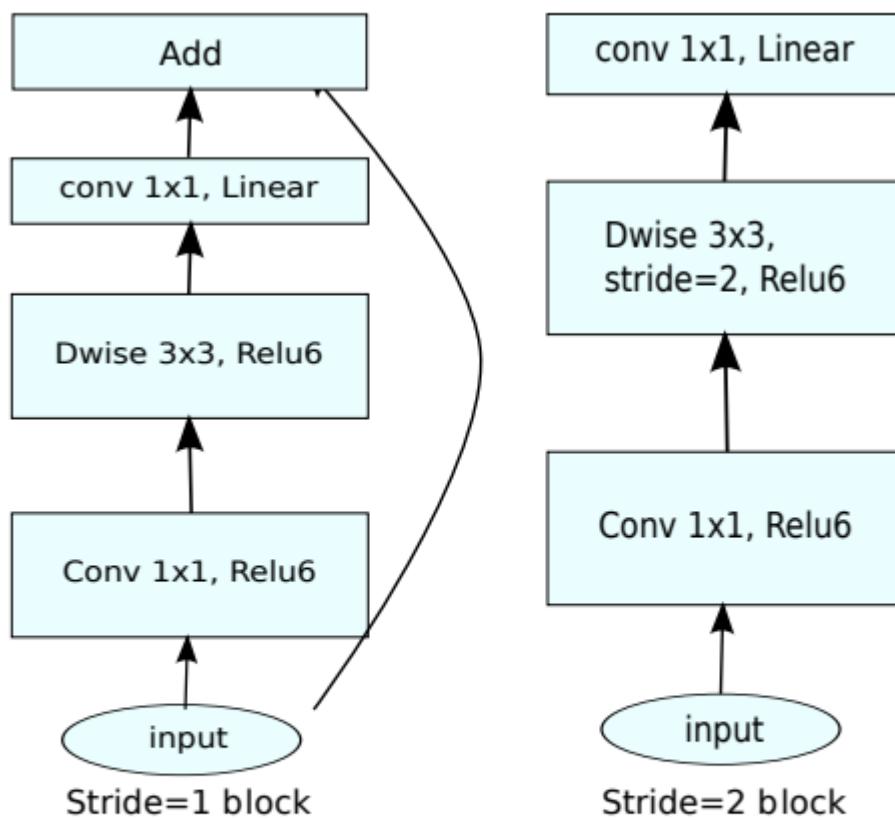
2. Training. Perform training and validation of the chosen model using their corresponding datasets. Calculate and plot all metrics for the whole training (i.e. after every epoch of training). Training should run for at least 10 epochs.

Relying on validation results, choose and save few 'best' models as MODELS_BASELINE

The MobileNetv2 architecture consists of about 53 deep layers, with additional normalization layers (154 layers at all). Its architecture can be described as:

Input	Operator	<i>t</i>	<i>c</i>	<i>n</i>	<i>s</i>
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	-

where t: expansion factor, c: number of output channels, n: repeating number, s: stride. 3×3 kernels are used for spatial convolution.



(d) Mobilenet V2

At the top of the layers, our own Dense layer with 5 neurons (because of the fact of having 5 classes multi-class classification problem) is added.

The architecture consists of 2,257,984 total params, which are initially frozen.

Model: "mobilenetv2_1.00_224"

Layer (type)	Output Shape	Param #	Connected to
<hr/>			
input_1 (InputLayer)	[None, 224, 224, 3]	0	[]
Conv1 (Conv2D)	(None, 112, 112, 32)	864	['input_1[0][0]']
bn_Conv1 (BatchNormalization)	(None, 112, 112, 32)	128	['Conv1[0][0]']
Conv1_relu (ReLU)	(None, 112, 112, 32)	0	['bn_Conv1[0][0]']

.... (more and more layers)

```
Conv_1 (Conv2D)      (None, 7, 7, 1280) 409600  ['block_16_project_BN[0][0]']
Conv_1_bn (BatchNormalization) (None, 7, 7, 1280) 5120    ['Conv_1[0][0]']
out_relu (ReLU)      (None, 7, 7, 1280) 0       ['Conv_1_bn[0][0]']
=====
Total params: 2,257,984
Trainable params: 0
Non-trainable params: 2,257,984
```

The input size (resolution) of the image can be chosen from a selected range values from 96 to 224. Because of the fact that our source images have a resolution of 256x256, we have decided to scale them into 224x224 size (the highest possible). Also, in the loss function parameter from_logits = True, so what comes out of the network is called logits - values that are an argument of a logistic function and this is where the "probability" comes from.

The final architecture with the added layer consists of 6405 trainable params (only last layer is unfrozen to be trained in order not to update weights in the basic MOBILENetV2 architecture):

Model: "MODEL1_BASELINE"

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 224, 224, 3)]	0
tf.math.truediv (TFOpLambda)	(None, 224, 224, 3)	0
tf.math.subtract (TFOpLambda)	(None, 224, 224, 3)	0
mobilenetv2_1.00_224 (Functional)	(None, 7, 7, 1280)	2257984
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0
dropout (Dropout)	(None, 1280)	0
dense (Dense)	(None, 5)	6405
Total params:	2,264,389	
Trainable params:	6,405	
Non-trainable params:	2,257,984	

3. Training on SPLIT1 datasets

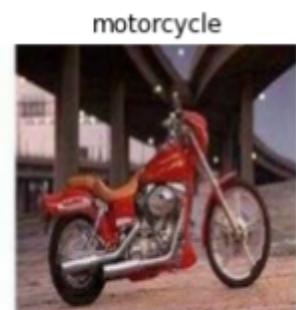
Perform the training on SPLIT1/TRAIN dataset and simple testing on SPLIT1/VAL dataset (the datasets should be created in "Task 2" of the lab).

Observe and present changes in loss/fitness during training. Compare results on TRAIN and VAL datasets.

Try to overfit your model. Describe this process and results.

Choose the best model (according to validation results), save it as MODEL1.

Sample images for classification:



MODEL: MODEL1_BASELINE

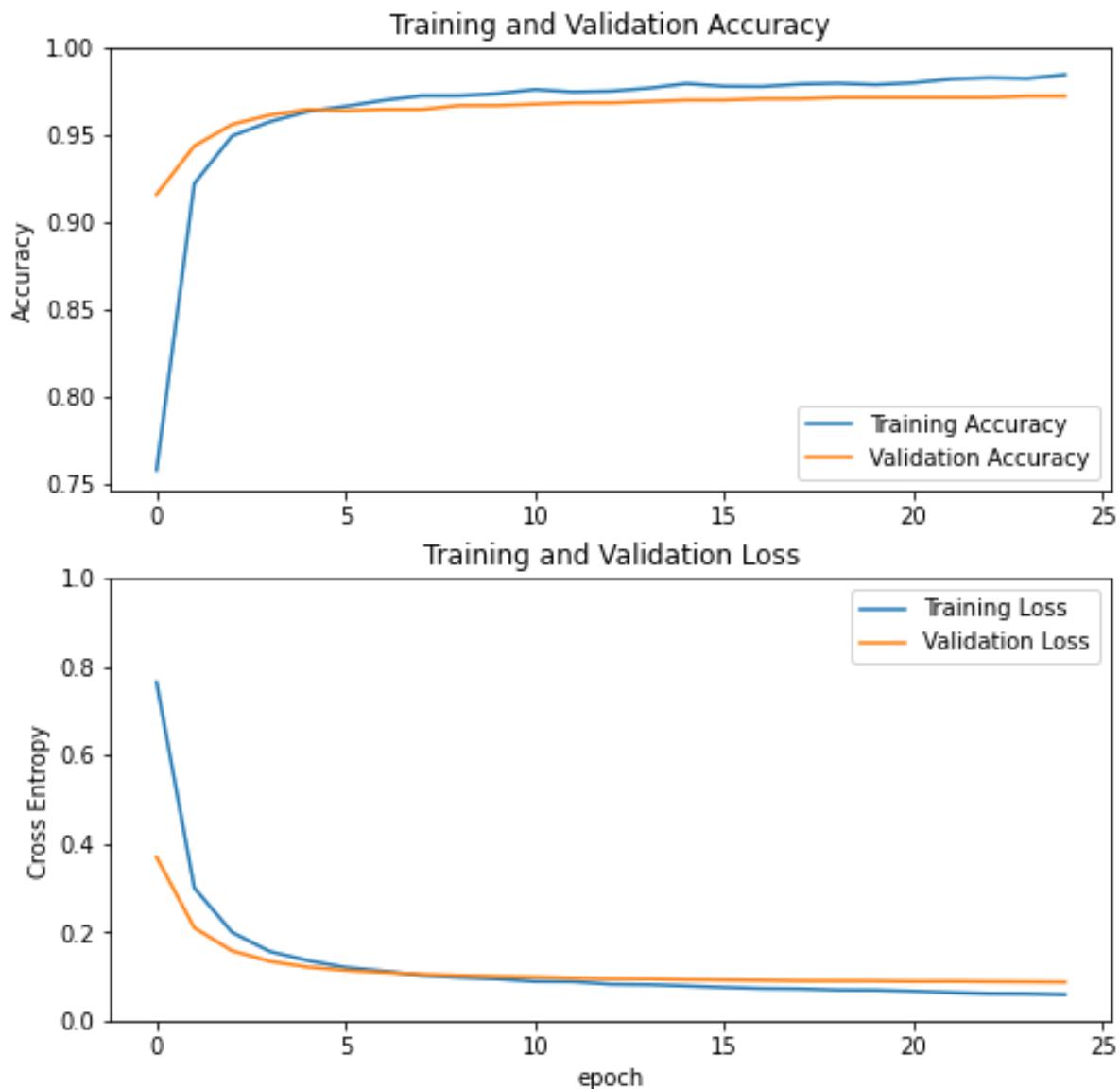
TRAIN: Found 6474 files belonging to 5 classes.

VAL: Found 1294 files belonging to 5 classes.

Number of train batches: 203

Number of validation batches: 41

loss after training: 0.0869
accuracy after training: 0.9722



```
images shape: (224, 224, 3)
IMG_SHAPE: (224, 224, 3)
example batch size (32, 7, 7, 1280)
41/41 [=====] - 228s 2s/step - loss: 1.8018 -
accuracy: 0.2304
initial loss: 1.80
initial accuracy: 0.23
```

```
Epoch 1/25
203/203 [=====] - 927s 2s/step - loss: 0.6184
- accuracy: 0.7974 - val_loss: 0.2497 - val_accuracy: 0.9334
Epoch 2/25
203/203 [=====] - 47s 90ms/step - loss: 0.2204
- accuracy: 0.9387 - val_loss: 0.1672 - val_accuracy: 0.9528
Epoch 3/25
203/203 [=====] - 47s 91ms/step - loss: 0.1647
- accuracy: 0.9518 - val_loss: 0.1368 - val_accuracy: 0.9595
Epoch 4/25
203/203 [=====] - 47s 90ms/step - loss: 0.1389
- accuracy: 0.9583 - val_loss: 0.1193 - val_accuracy: 0.9641
Epoch 5/25
203/203 [=====] - 47s 90ms/step - loss: 0.1214
- accuracy: 0.9633 - val_loss: 0.1082 - val_accuracy: 0.9677
Epoch 6/25
203/203 [=====] - 47s 90ms/step - loss: 0.1116
- accuracy: 0.9645 - val_loss: 0.0997 - val_accuracy: 0.9702
Epoch 7/25
203/203 [=====] - 47s 90ms/step - loss: 0.1033
- accuracy: 0.9681 - val_loss: 0.0930 - val_accuracy: 0.9720
```

Training was set to 25 epochs with an EarlyStopping parameter set to 3 epochs. Initial values of loss and accuracy were, respectively, 1.8 and 0.23. Because of the fact that we have used the Transfer Learning method, the Neural Network architecture was already pre-trained on images consisting of 1000 classes, which also contained the 5 classes defined by us: ['bicycle' 'bus' 'car' 'motorcycle' 'truck']. The ImageNetv2 architecture is so broad, that it was even more detailed in the class definition (eg. distinguishing types of buses and trucks).

Learning rate was set to 0.001 (small learning rate) in order not to overfit the model. While using the Transfer Learning method, usually there is a need to set a very small learning rate. Batch size parameter was set to 32. Monitored metric was accuracy.

Validation and training loss converges quickly. That is a good sign - the chosen learning rate was neither too small nor too big. Only the first epoch learning took about 15 minutes (because of the fact that the model started updating weights). The next epochs' iterations took about 50 seconds each. It was possible to train the model longer than 25 epochs without overfitting, but we have decided to stop here as the BASELINE_MODEL.

In order to overfit the model, the number of epochs can be simply set to a big number (like 800 hundred) without the EarlyStopping parameter. Then, the achieved results will be high only on the training dataset, and low on the validation of test sets.

4. Training on SPLIT2 datasets *[10 points]*

Perform the training on SPLIT2/TRAIN dataset and simple testing on SPLIT2/VAL dataset.

Compare results (time, accuracy, etc.) to the training on SPLIT1

Choose the best model (according to validation results), save it as MODEL2.

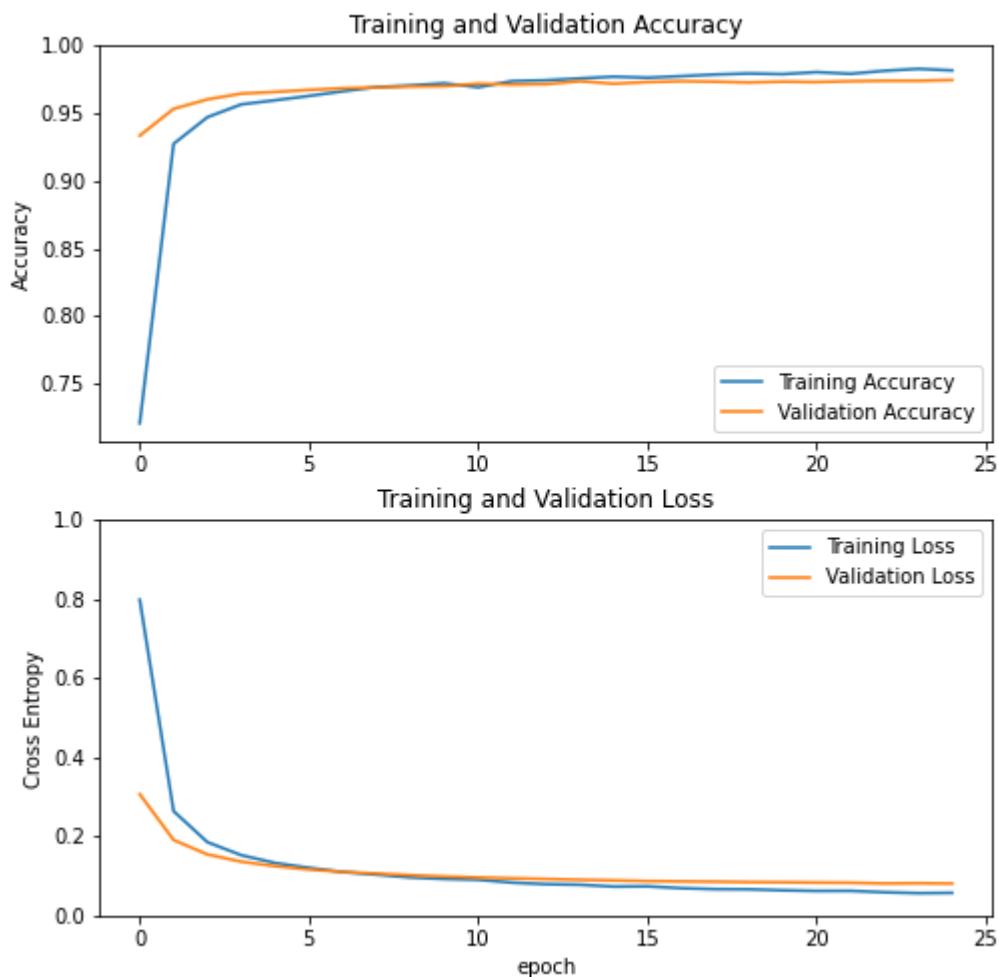
Sample examples for classification:



MODEL: MODEL2_BASELINE

Train: Found 13151 files belonging to 5 classes.

Val: Found 3286 files belonging to 5 classes.



Number of train batches: 411

Number of validation batches: 103

loss after training: 0.08169762790203094

accuracy after training: 0.9744369983673096

Loss function converges fastly and smoothly. Training loss is lower than validation loss, so the model is not overfitted. The achieved validation accuracy is higher than in MODEL1_BASELINE, because here we have more data (augmentation used) and balanced classes (the same number of learning examples).

5. Training on SPLIT3 datasets *[5 points]*

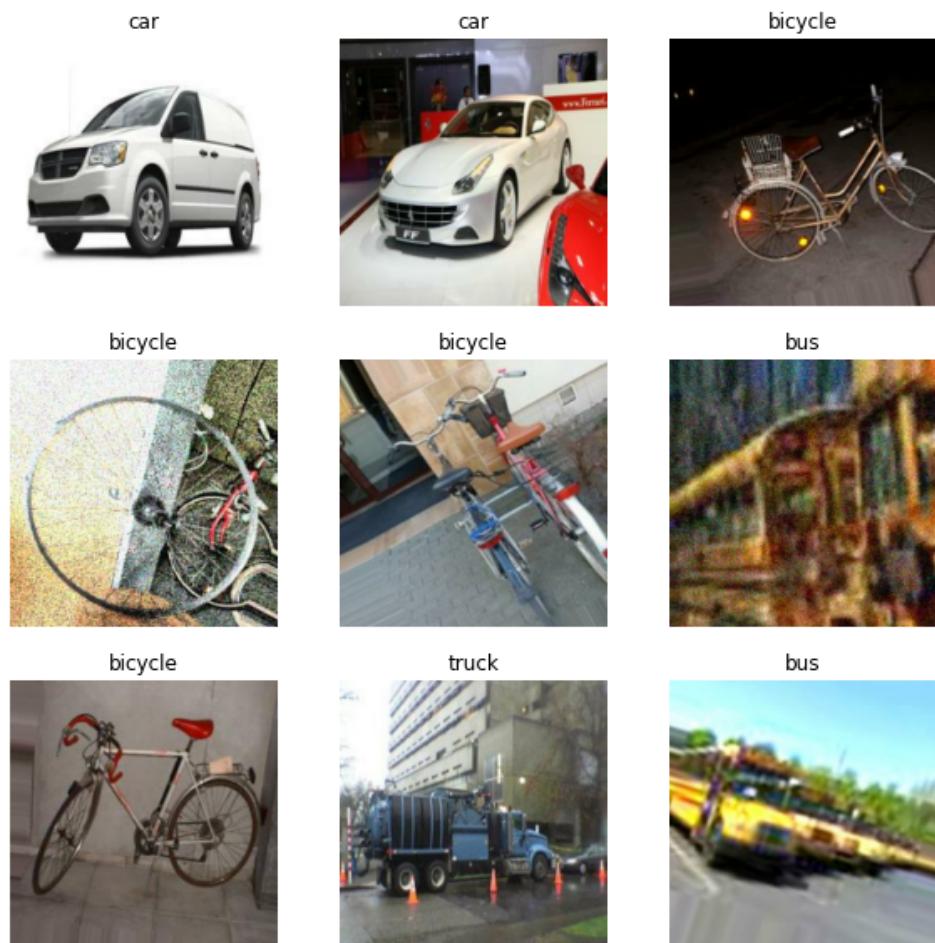
Perform the training on SPLIT3/TRAIN dataset and simple testing on SPLIT3/VAL dataset

Compare results (time, accuracy/loss on TRAIN dataset, accuracy/loss on VAL dataset) to the training on SPLIT2

Choose the best model (according to validation results), save it as MODEL3.

NOTE. SPLIT3 shall be exactly the same as SPLIT2, with one exception: VAL set must be added to TRAIN set.

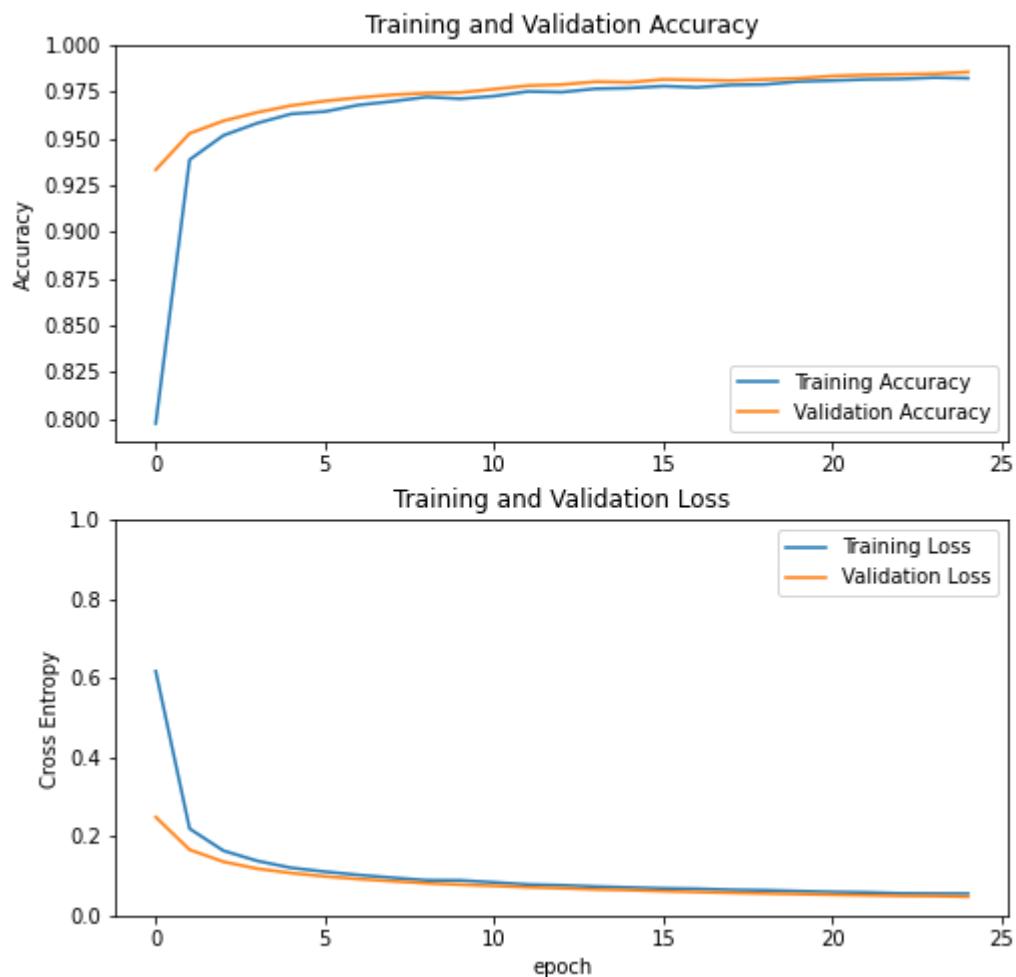
Sample examples for classification:



MODEL: MODEL3_BASELINE

Train: Found 16437 files belonging to 5 classes.

Val: Found 3286 files belonging to 5 classes.



Number of train batches: 514

Number of validation batches: 103

loss after training: 0.048375129699970703

accuracy after training: 0.9856969118118286

This model is exactly the same as MODEL2_BASELINE with one exception: here we have added a validation set to the train set.

Loss function converges fastly and smoothly. The achieved validation accuracy is higher than in MODEL2_BASELINE, because we have added a validation set to the train set. However, adding the validation set to the training set brings risk, that the model can overfit (learn the training examples by heart). Thus, we cannot reliably evaluate the results of the model.

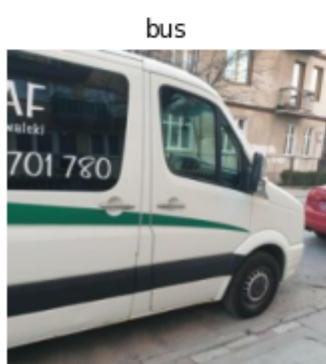
6. Test your models created in Task 3 (MODEL1_BASELINE, MODEL2_BASELINE, MODEL3_BASELINE) using all defined measures. For each test, make some useful plots and result visualizations.

Test on TRAIN subsets. Test on VAL subsets. Test on TEST subsets

Discuss the results. Discussion may include: differences and similarities in results for different models and datasets, analysis of errors and their causes (according to data analysis from Task 2), differences in results for various test metrics (you shall describe what these differences mean), and others.

Sample testset1 examples:





Sample testset2 examples:





bus



bus



bus



bus



bus



bus



bus



bus



bus

At the beginning, data about labels and predictions is taken:

- test_dataset_labels:

```
<PrefetchDataset element_spec=(TensorSpec(shape=(None, 224, 224, 3), dtype=t-
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1]
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
[2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3]
[3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
[4 4 4]
```

- test_dataset_binarized_labels:

- test_dataset_predicted_probs:

```
array([[ 3.3810773 , -5.4127207 , -3.6946068 ,  1.7537885 , -5.3273964 ],  
       [ 5.0070186 , -5.17487 , -3.2289896 , -0.10815454, -3.8657198 ],  
       [ 5.0694475 , -5.0153027 , -3.6647482 , -1.490962 , -4.3484716 ],  
       [ 3.3227968 , -4.34748 , -3.1767912 ,  1.3998142 , -5.7245064 ],  
       [ 6.035137 , -5.0809507 , -3.850081 ,  0.3358262 , -6.2734246 ],  
       [ 3.5464375 , -6.4722114 , -4.8000503 , -0.1835701 , -5.3611765 ],  
       [ 5.4124613 , -4.024412 , -2.469121 , -1.2755262 , -4.5455475 ],  
       [ 5.3162746 , -3.8215678 , -2.55028 , -1.391141 , -4.66854 ],  
       [ 5.9005013 , -4.4253206 , -3.0238295 , -1.3518044 , -4.6868696 ],  
       [ 3.1337602 , -4.9183016 , -3.540956 ,  2.8302474 , -6.221245 ],  
       [ 6.324378 , -5.7644105 , -4.389067 ,  0.06812227, -5.569676 ],  
       [ 4.254759 , -4.9978604 , -2.3757915 , -3.8448176 , -5.0382586 ],  
       [ 4.44098 , -5.727446 , -2.4284787 , -3.6458142 , -6.240299 ],  
       [ 1.768094 , -5.9870033 , -3.113018 , -2.6688874 , -3.89831 ]])
```

- test_dataset_predicted_labels:

Then, the model is assessed using defined metrics.

MODEL1_BASELINE

MODEL1_BASELINE	testset1	testset2	trainset	valset
Accuracy	0,770992	0,965	0,985944	0,972179
F1 score	0,71651	0,964818	0,985893	0,971833
ROC AUC	0,96605	0,99593	0,997542	0,994443
Precision	0,836037	0,965819	0,985945	0,971881
Recall	0,770992	0,965	0,985944	0,972179
Cohens kappa	0,709277	0,95625	0,97972	0,95979

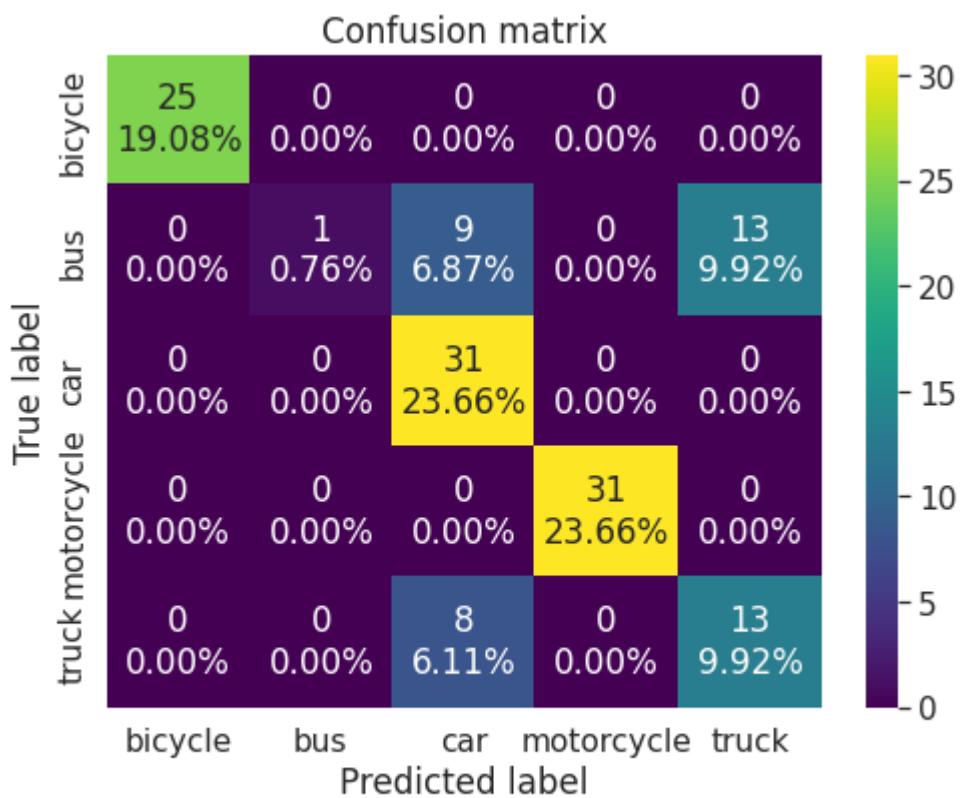
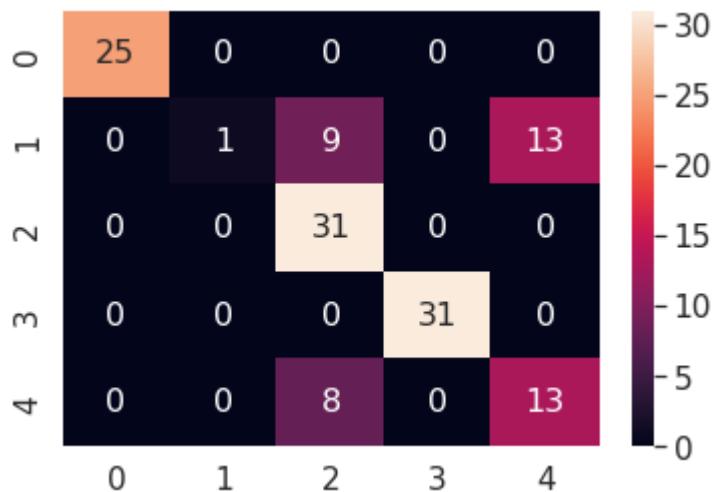
Results obtained by the model are relatively high. Accuracy on the “harder” test set (testset1) has about 77%, and is supported by f1-score equal to 72%. Area under the curve (AUC) covers 96% of the data. However, from the confusion matrix of the testset1 we can see that the model has problems with distinguishing between truck and bus, car and bus and car and truck. It may be connected with the fact that in the collected data set some examples of car, bus and trucks were distinguishable only by having windows next to the passenger seat and by the size of the vehicle.

Accuracy on the “easier” test set (testset2) was 96,5%. F1-score was also about 96,5 %. Area under the curve covered almost 100% of the data. The results are very satisfactory.

Of course, the highest results were obtained by evaluating the model using train and validation sets. Accuracy took, respectively, 98,6% and 97% (f1-score very similar). It was exactly the same data as used for training, so it was predictable. However, the model did not receive 100% accuracy, which is a good sign - it is not overfitted. The model had rather created classification rules to follow than to learn the training examples “by heart”. AUC took almost 100% of the data. Confusion matrix looks very good - small number of mistakes.

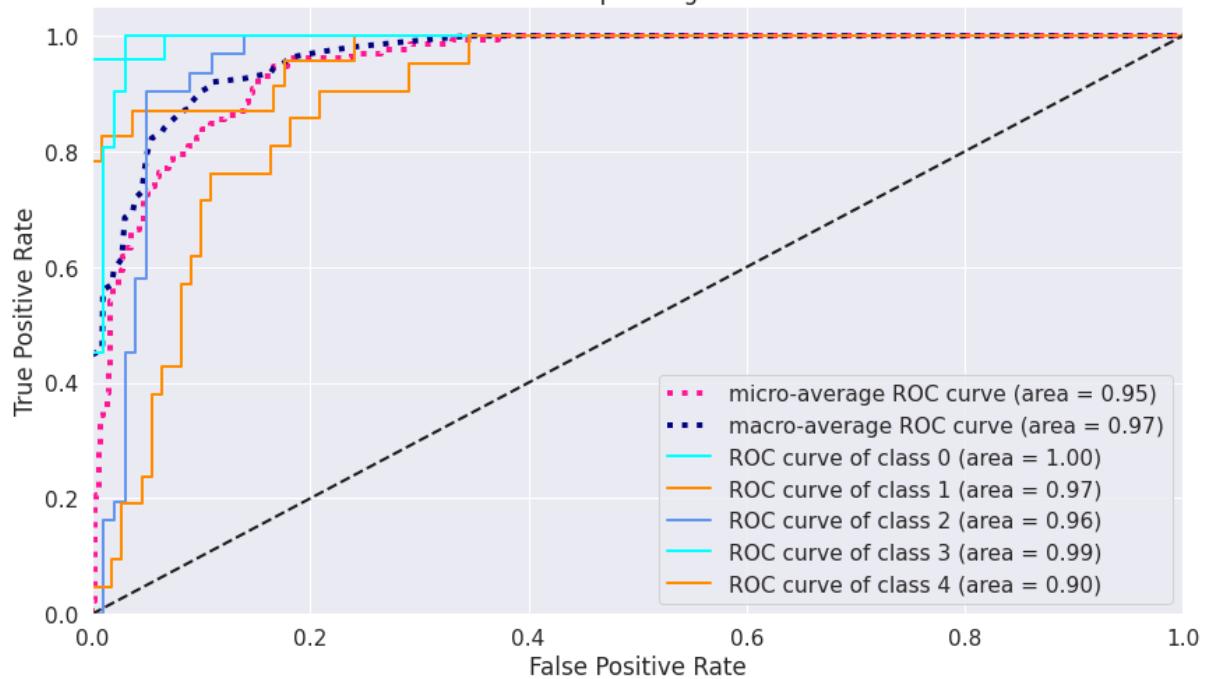
Evaluation results on the other models using training and validation dataset will be similar. We will not conduct them, because it takes a lot of time to prepare such a big amount of data for the evaluation purposes. Only testset1 and testset2 will be taken into consideration.

- **testset1**

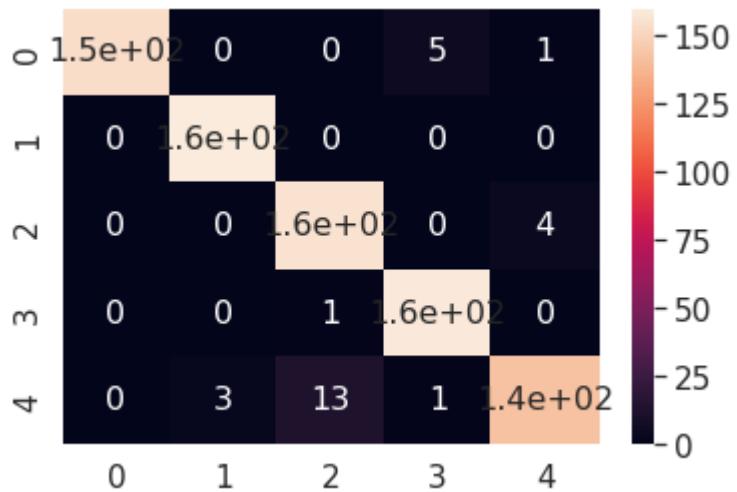


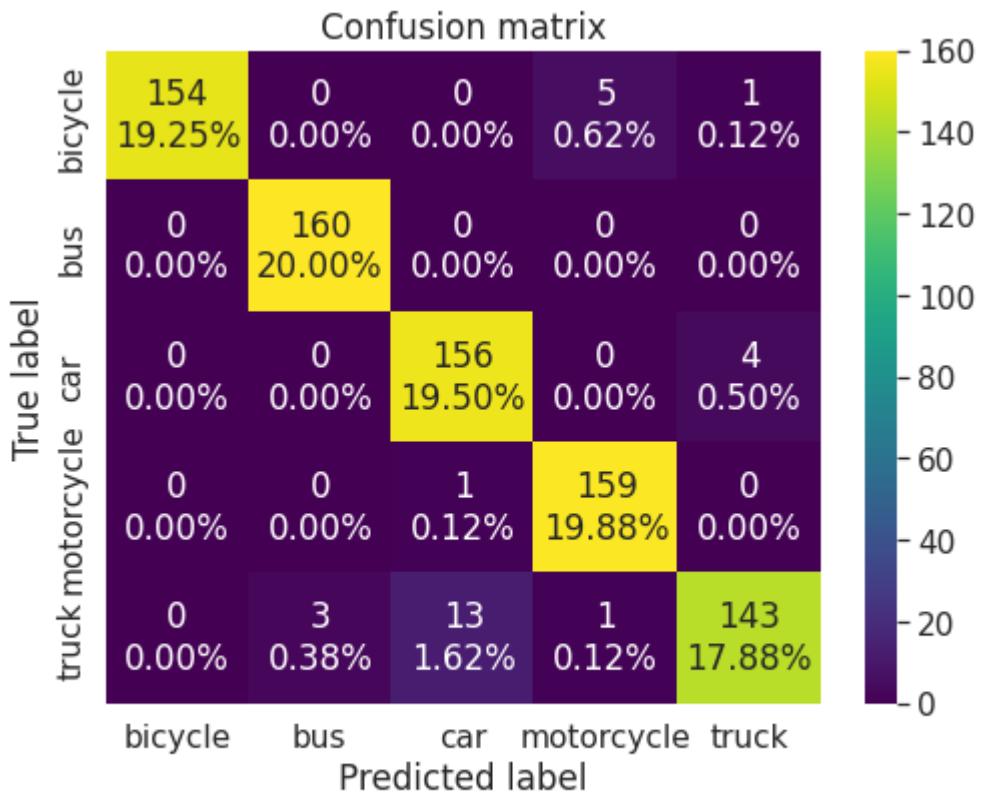
Accuracy=0.771

Some extension of Receiver operating characteristic to multiclass

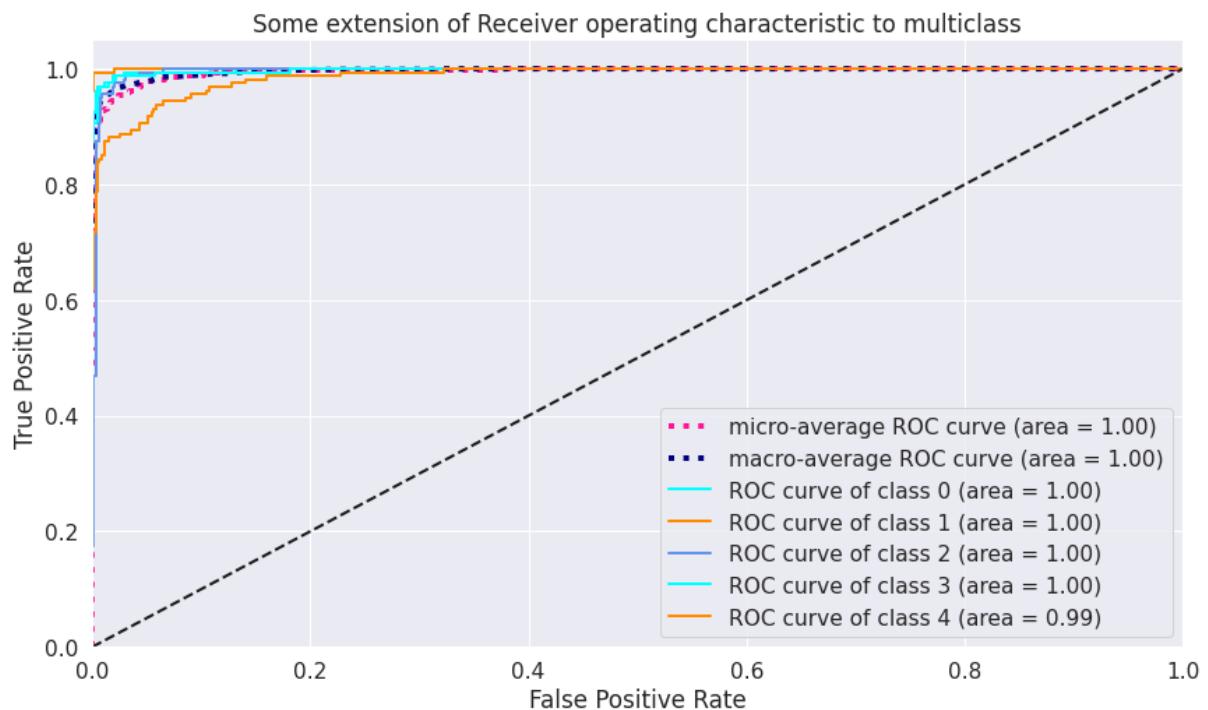


• **testset2**

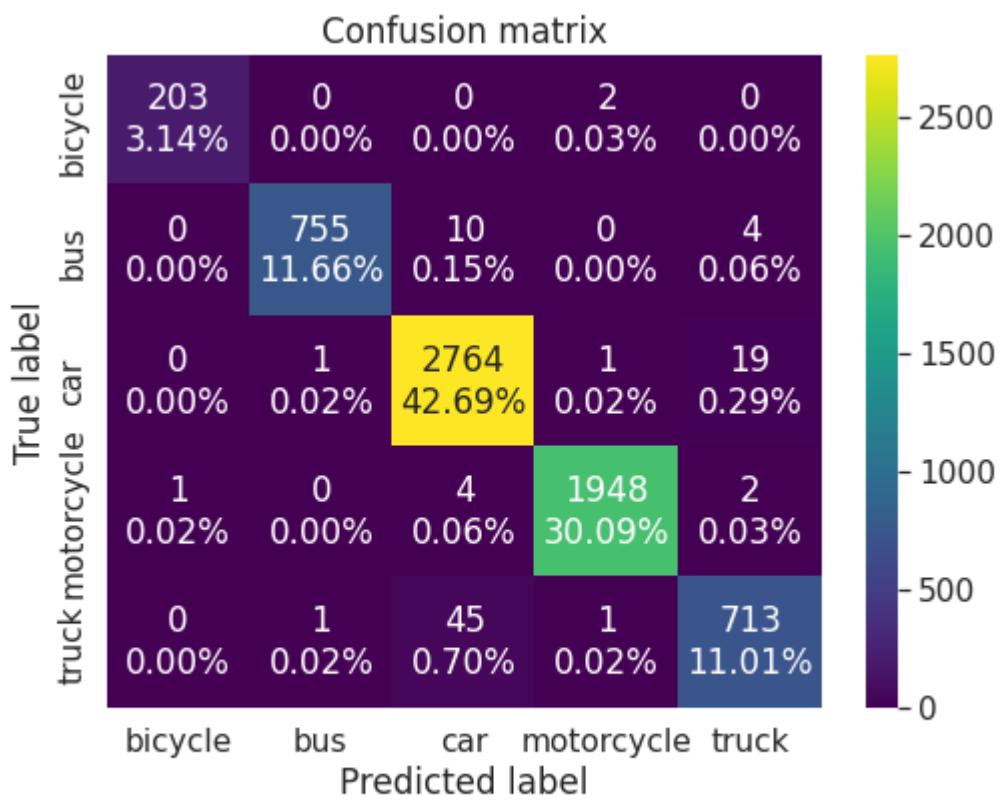
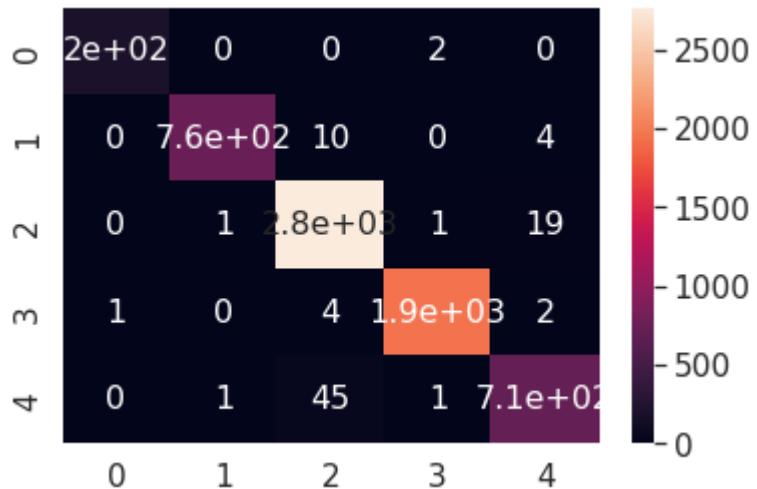




Accuracy=0.965

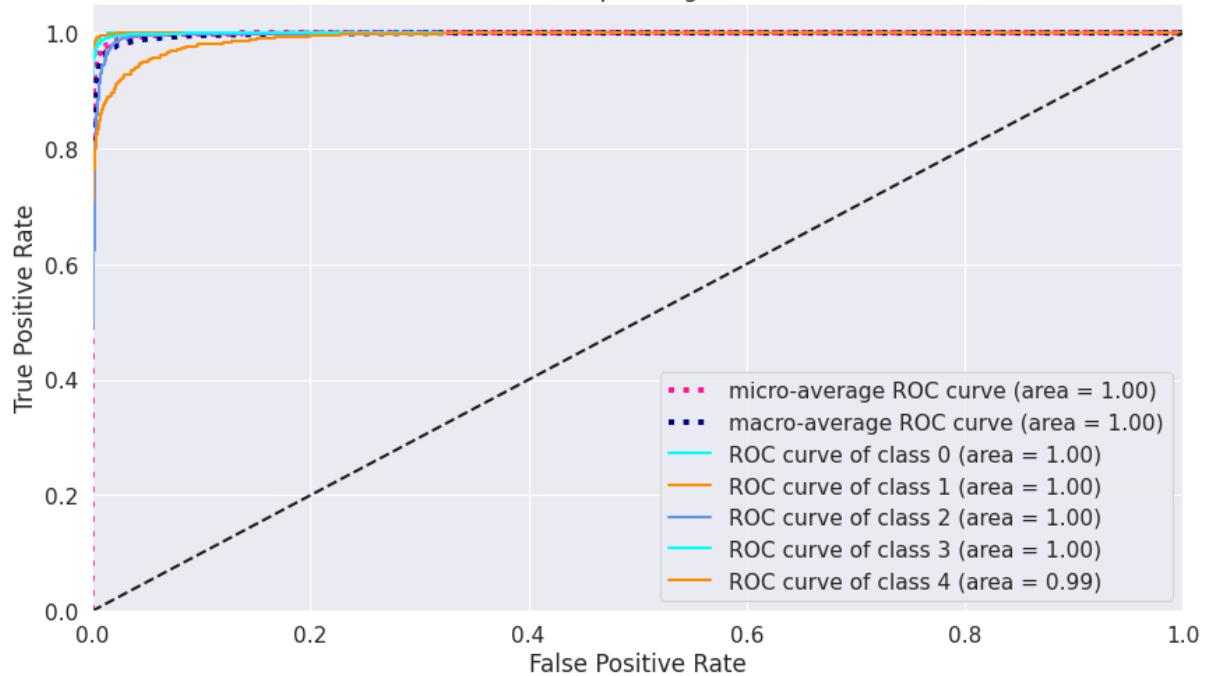


- **trainset**



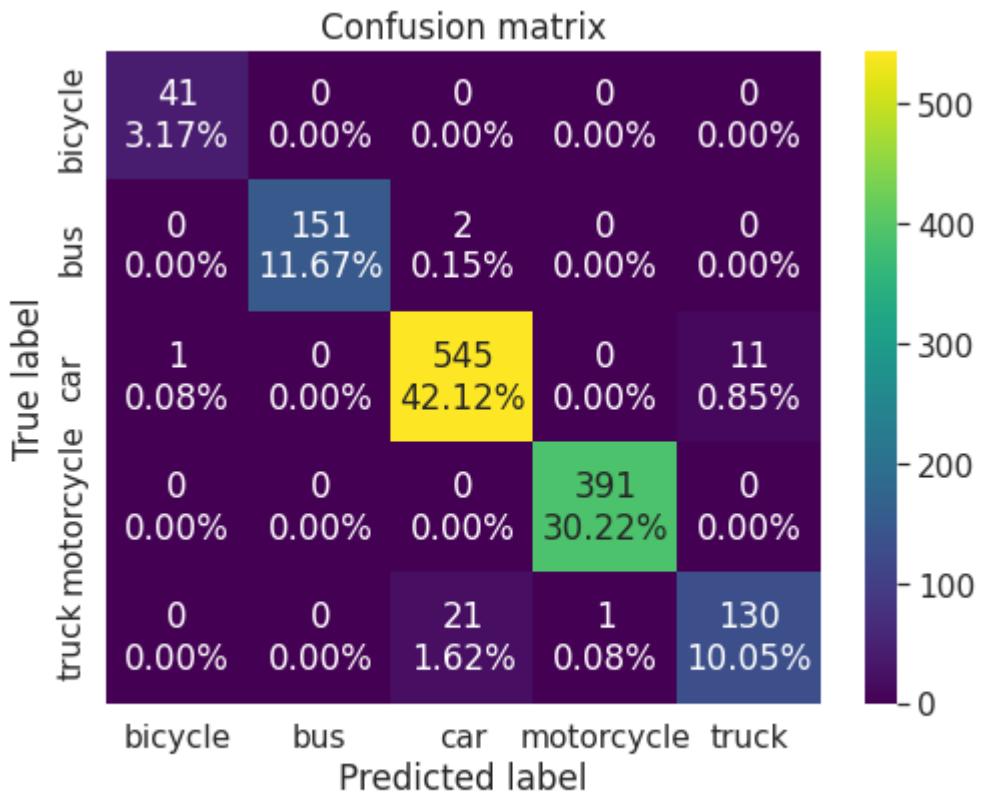
Accuracy=0.986

Some extension of Receiver operating characteristic to multiclass

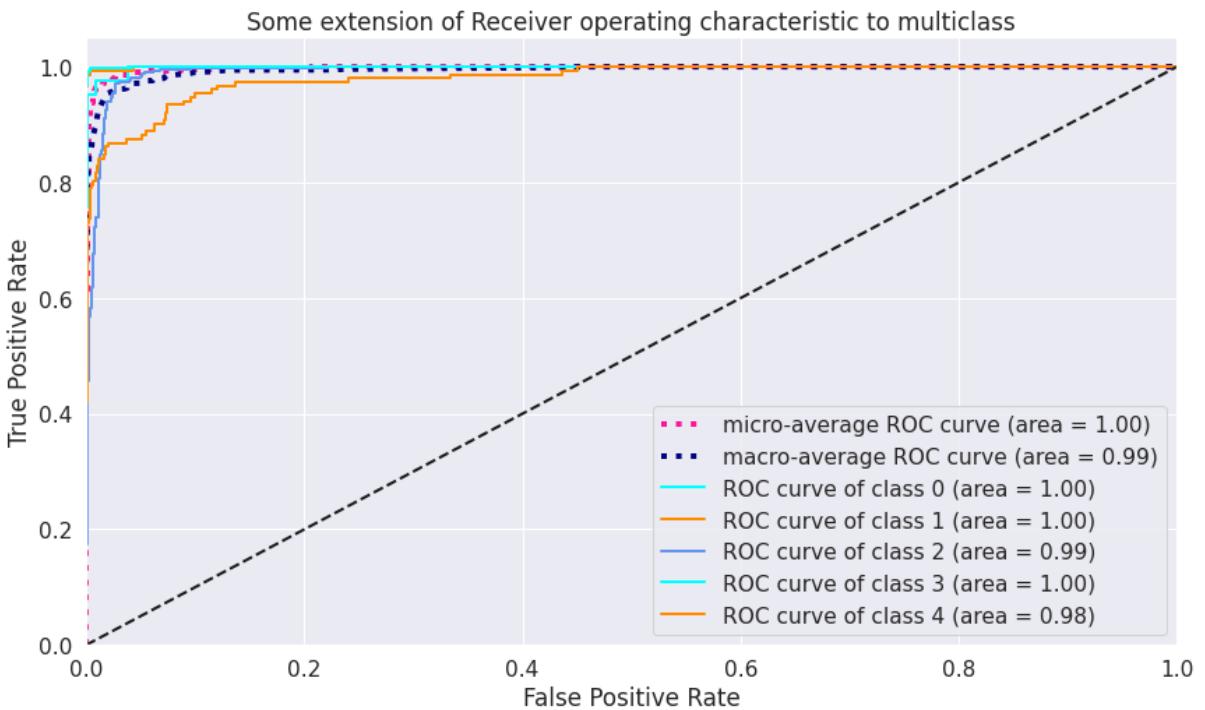


- validation set





Accuracy=0.972



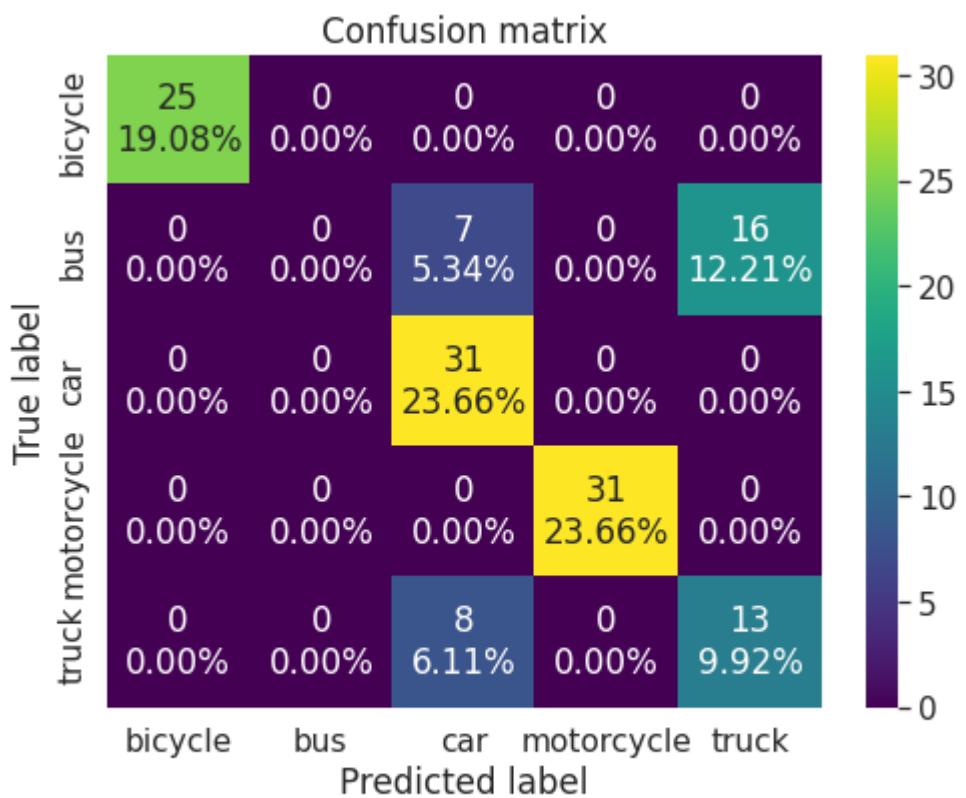
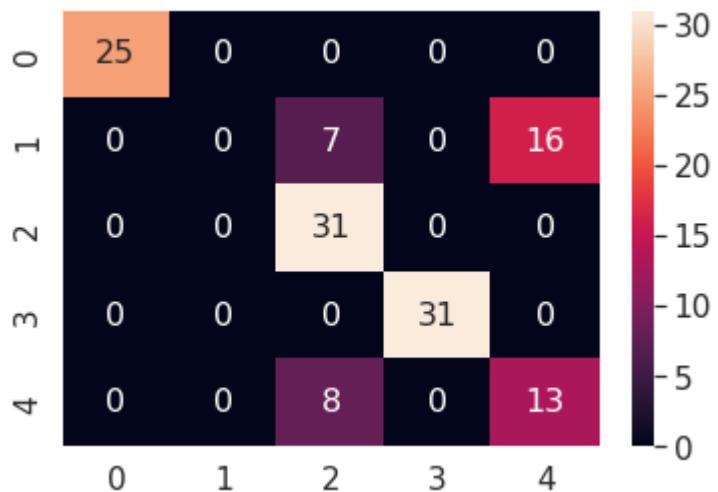
MODEL2_BASELINE

MODEL2_BASELINE	testset1	testset2
Accuracy	0,763359	0,9775
F1 score	0,701382	0,977309
ROC AUC	0,95734	0,996211
Precision	0,658818	0,977768
Recall	0,763359	0,9775
Cohens kappa	0,700074	0,971875

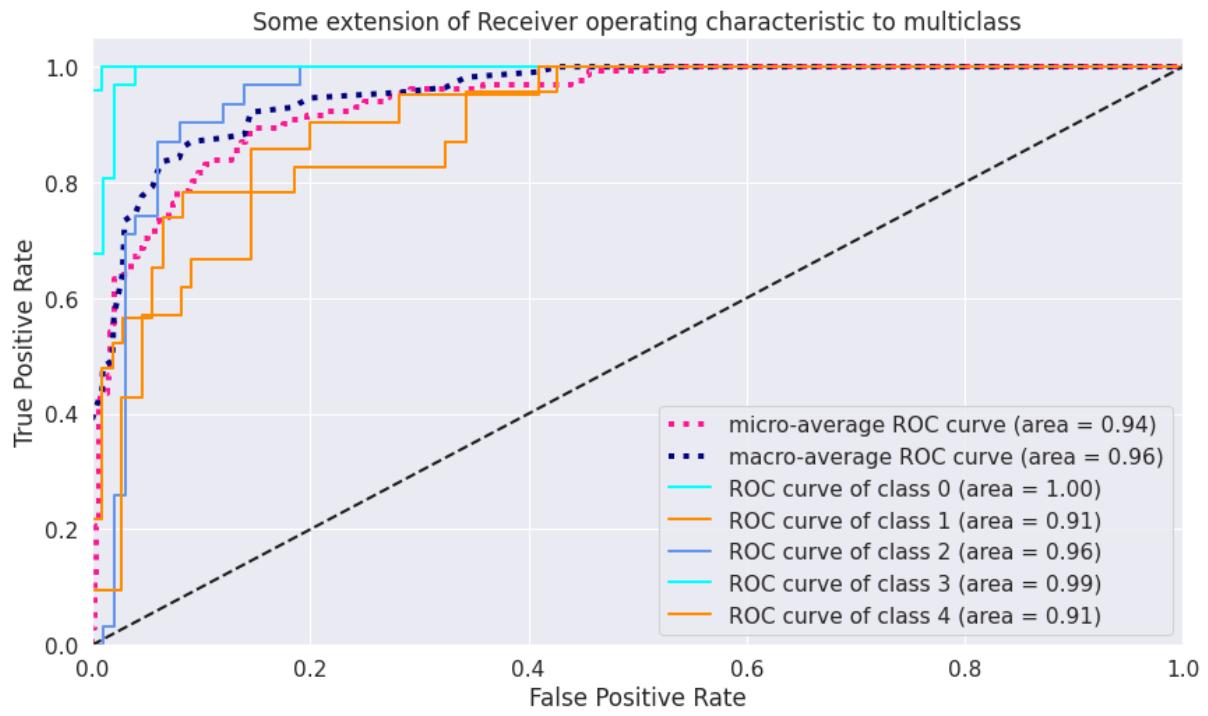
Once again, results obtained on the testset1 are worse than on the testset2. On the first set, accuracy is about 76%, but the f1-score is worse (only 70%). Confusion matrix shows a lot of mistakes, especially in the car-bus-truck recognition. However, this results may not be so bad, if the difference between car, bus and truck is hard to obtain even for the human eye.

Second test set performs much better. Accuracy on the level of 97,8%, f1-score about 97,7%.

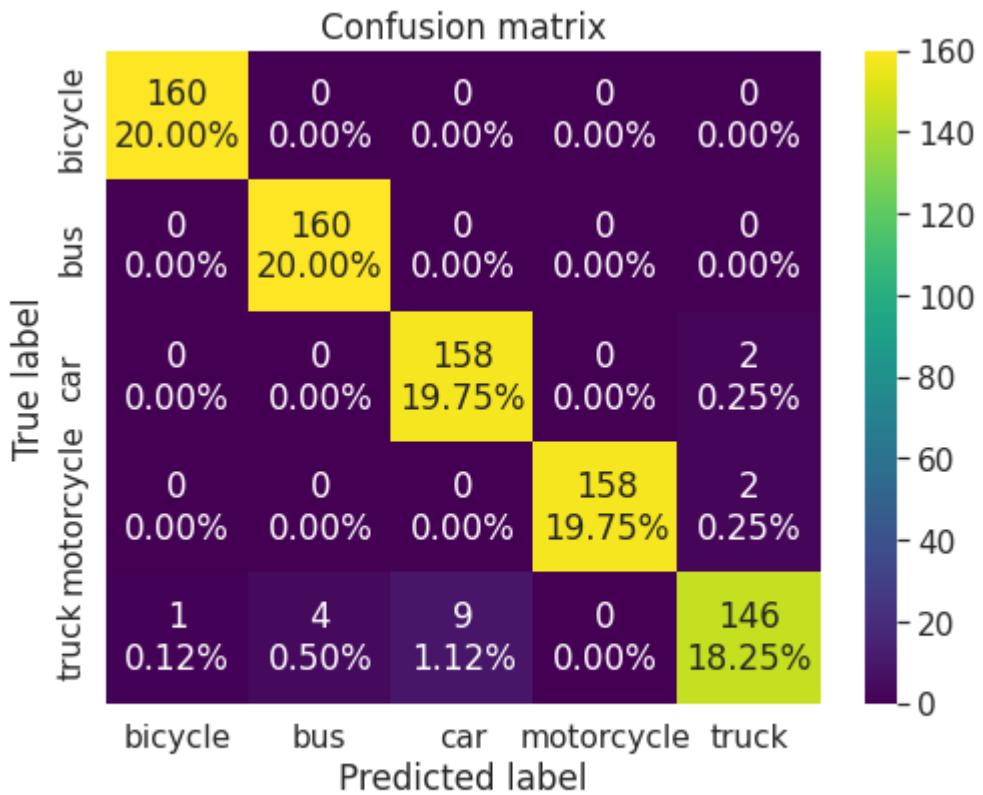
- testset1



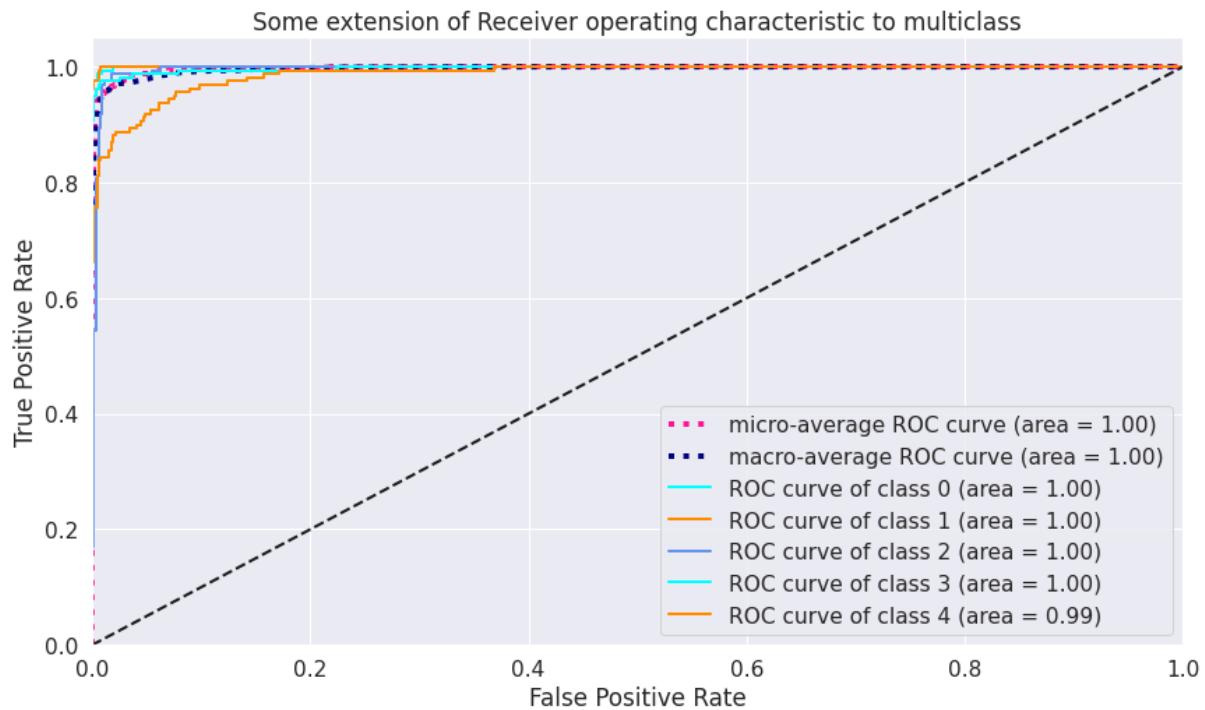
Accuracy=0.763



- **testset2**



Accuracy=0.978

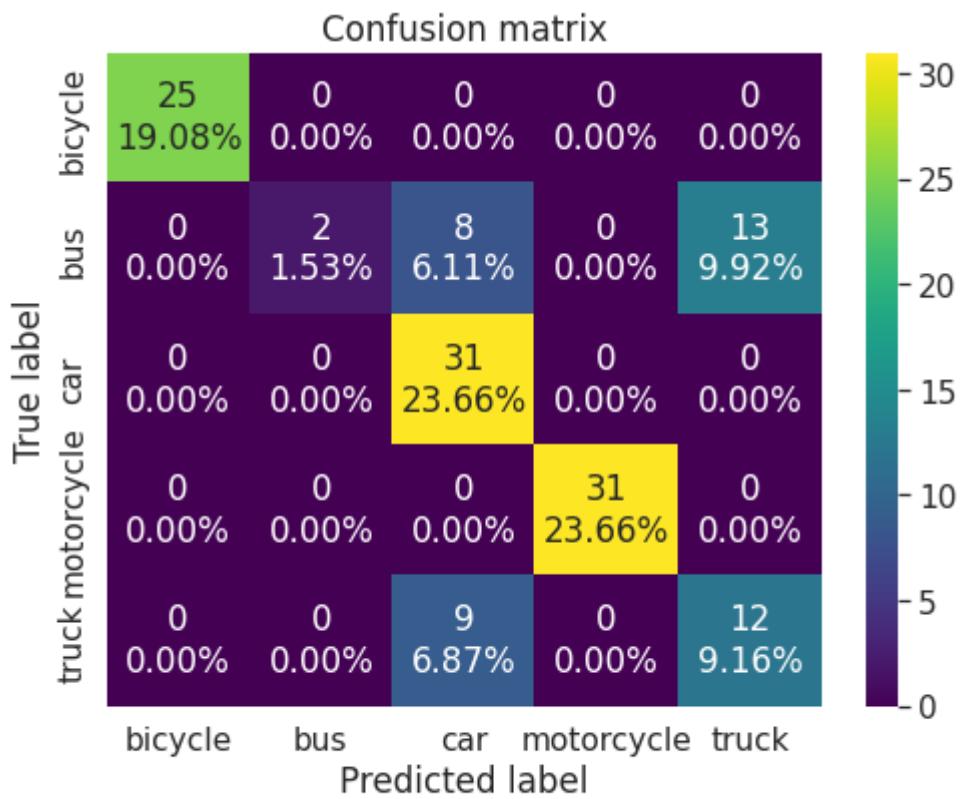
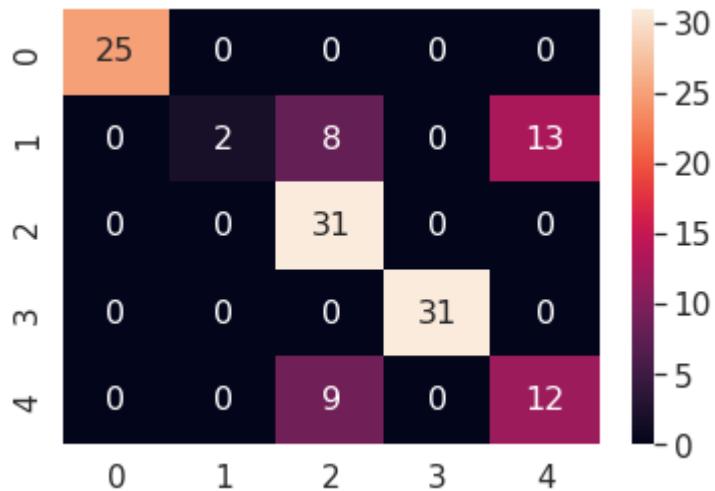


MODEL3_BASELINE

MODEL3_BASELINE	testset1	testset2
Accuracy	0,770992	0,97875
F1 score	0,724929	0,978626
ROC AUC	0,967337	0,996771
Precision	0,832831	0,979164
Recall	0,770992	0,97875
Cohens kappa	0,709234	0,973437

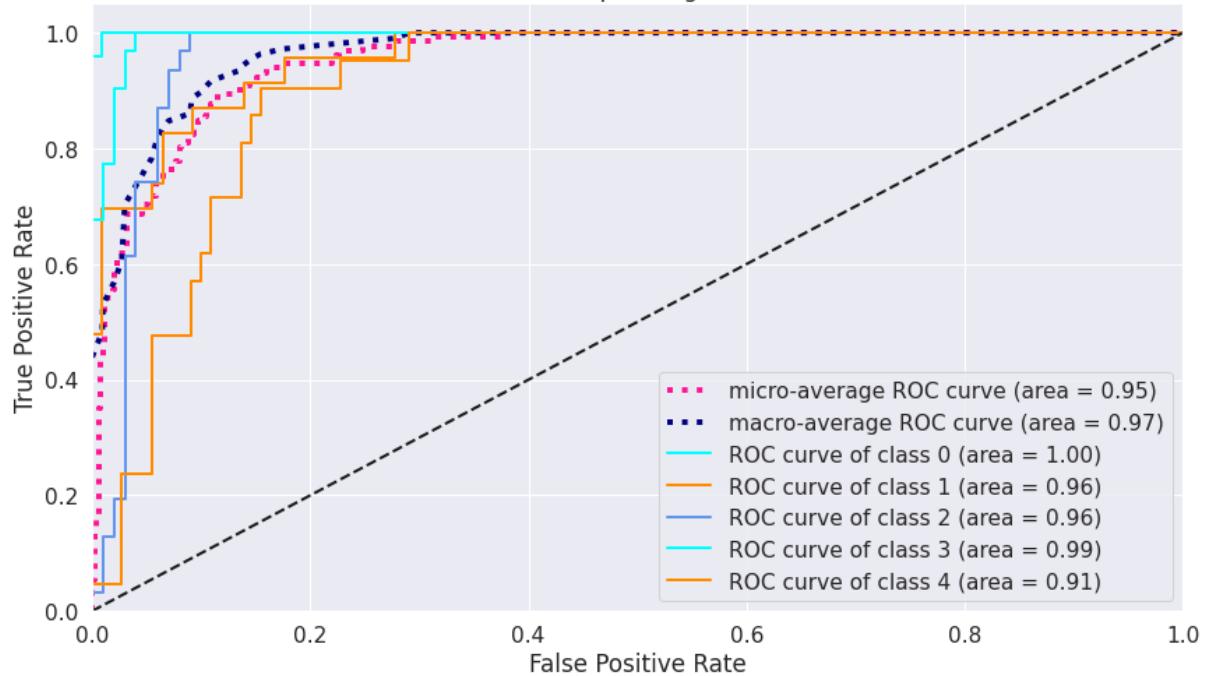
Metrics values obtained by this model are slightly better than in MODEL2_BASELINE. It comes from the fact that the validation set is added to the train set. On the testset1 - accuracy about 77%, f1-score 72,5%, ROC 97%. On the testset2 - accuracy 97,9%, f1-score 97,9%, AUC 99,7%.

- **testset1**

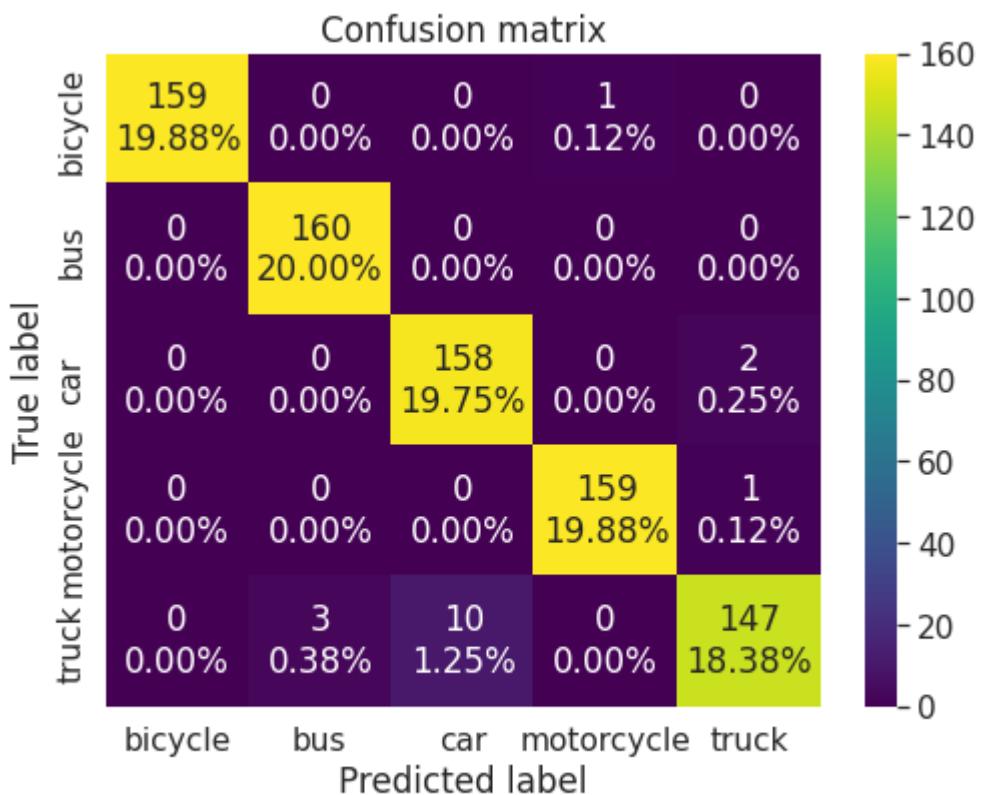


Accuracy=0.771

Some extension of Receiver operating characteristic to multiclass

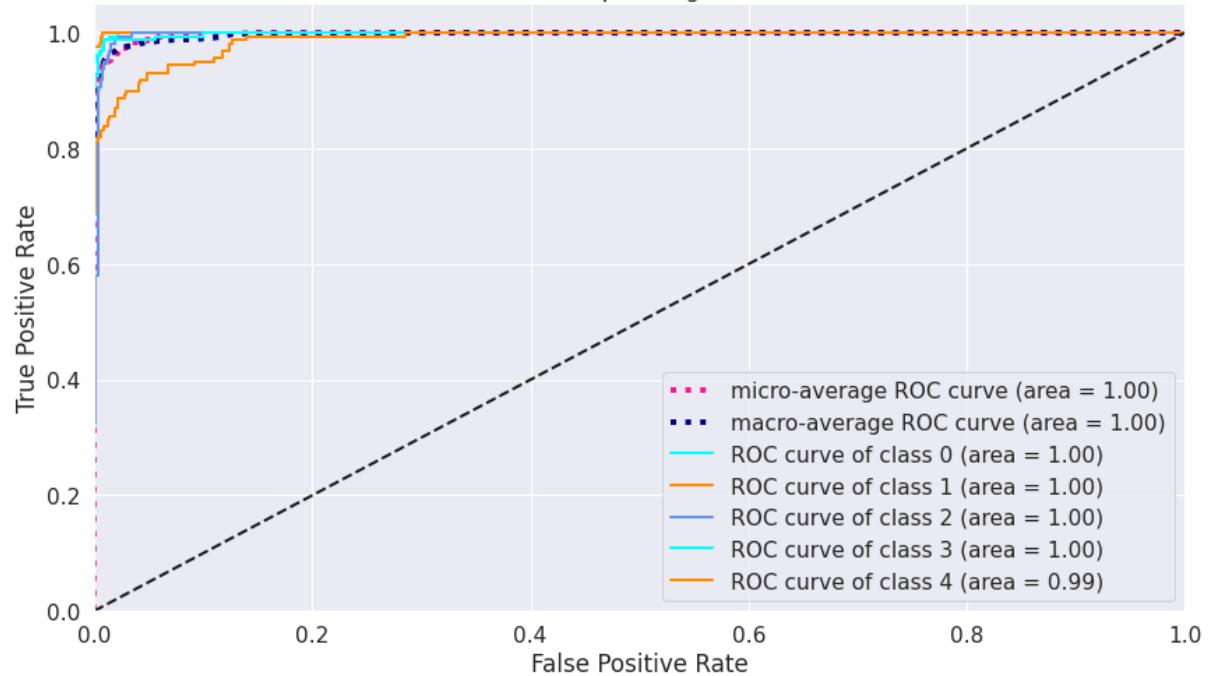


- **testset2**



Accuracy=0.979

Some extension of Receiver operating characteristic to multiclass



7. Try to improve your results. You are advised (but not limited) to do at least some of following tasks:

Repeat the training and testing on SPLIT2 datasets.

During the training, calculate all defined testing measures on TRAIN and VAL datasets

Plot measurements' results for the whole training.

Choose "best" models basing on different test measures on VAL datasets

Test selected "best" models on TEST dataset. Do some plots and discuss the results.

Do hyperparameter optimization and/or cross-validation

Perform hard-example mining and retraining

Propose improvements in the whole workflow (data collection, preprocessing, models, training, optimization, hyperparameters, etc.)

Implement these improvements, do the training and testing, and analyze and discuss results.

Do whatever other you want, to improve your models.

Experiment with your models, data, parameters etc. Be creative :) If your experiment gives worse results - no worries, you can still include it in the report.

Relying on validation results, choose and save a few 'best' models as MODELS_IMPROVED.

MODEL2_IMPROVED exemplary training data:



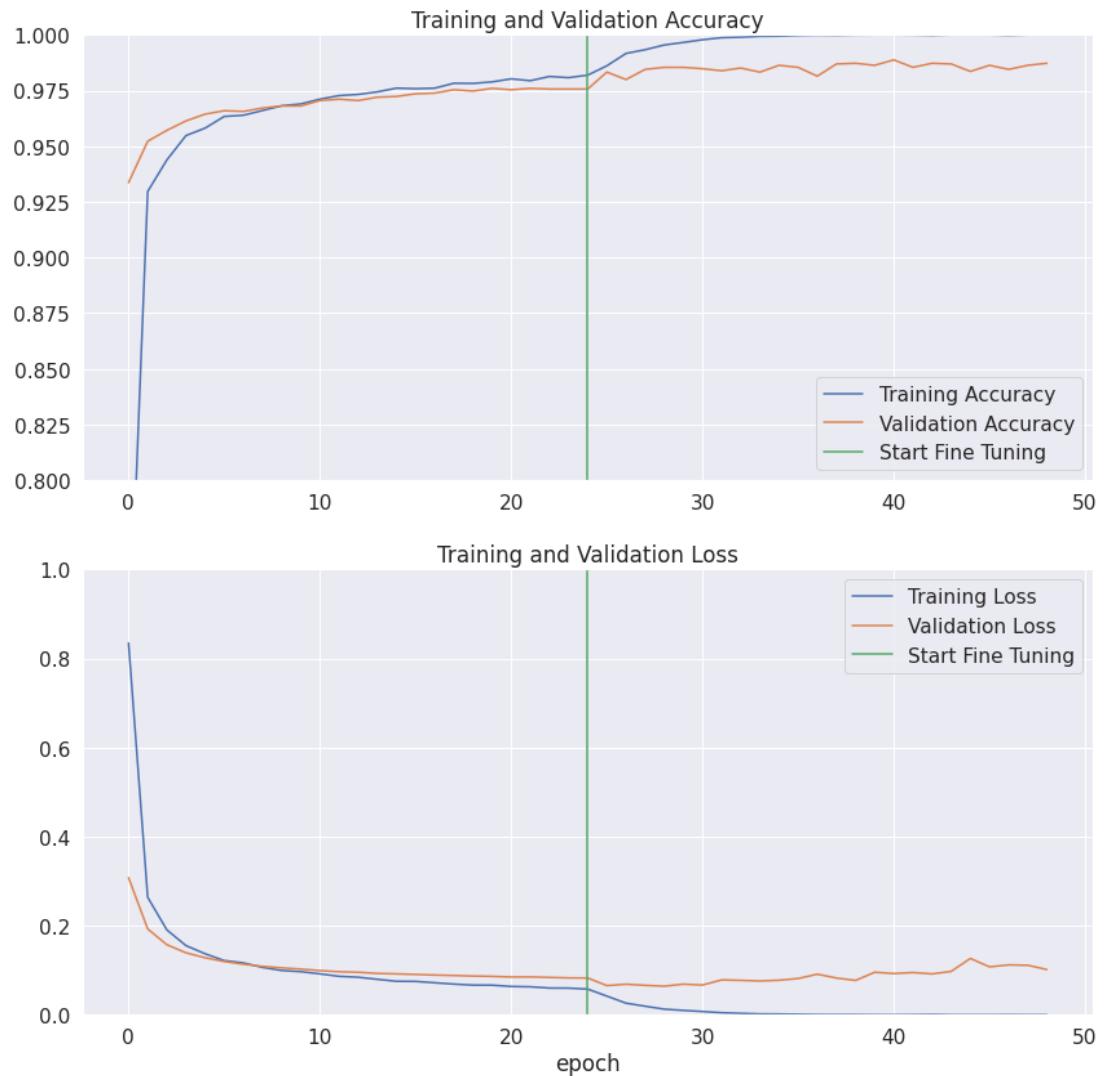
Because of the fact that MODEL2_BASELINE has the biggest potential to be furtherly explored (the biggest number of training and validation data, without breaking

learning rules as in the case of MODEL3_BASELINE), we have decided to perform fine-tuning on this model. In order to do so, we unfreezed the top 54 layers from MobileNetv2 architecture (layers from 100 to 154) and continued learning the model for the next 25 epochs (50 in total with the previous 25). As the tensorflow documentation tells us, in most convolutional networks, the higher up a layer is, the more specialized it is. The first few layers learn very simple and generic features that generalize to almost all types of images. As you go higher up, the features are increasingly more specific to the dataset on which the model was trained. The goal of fine-tuning is to adapt these specialized features to work with the new dataset, rather than overwrite the generic learning.

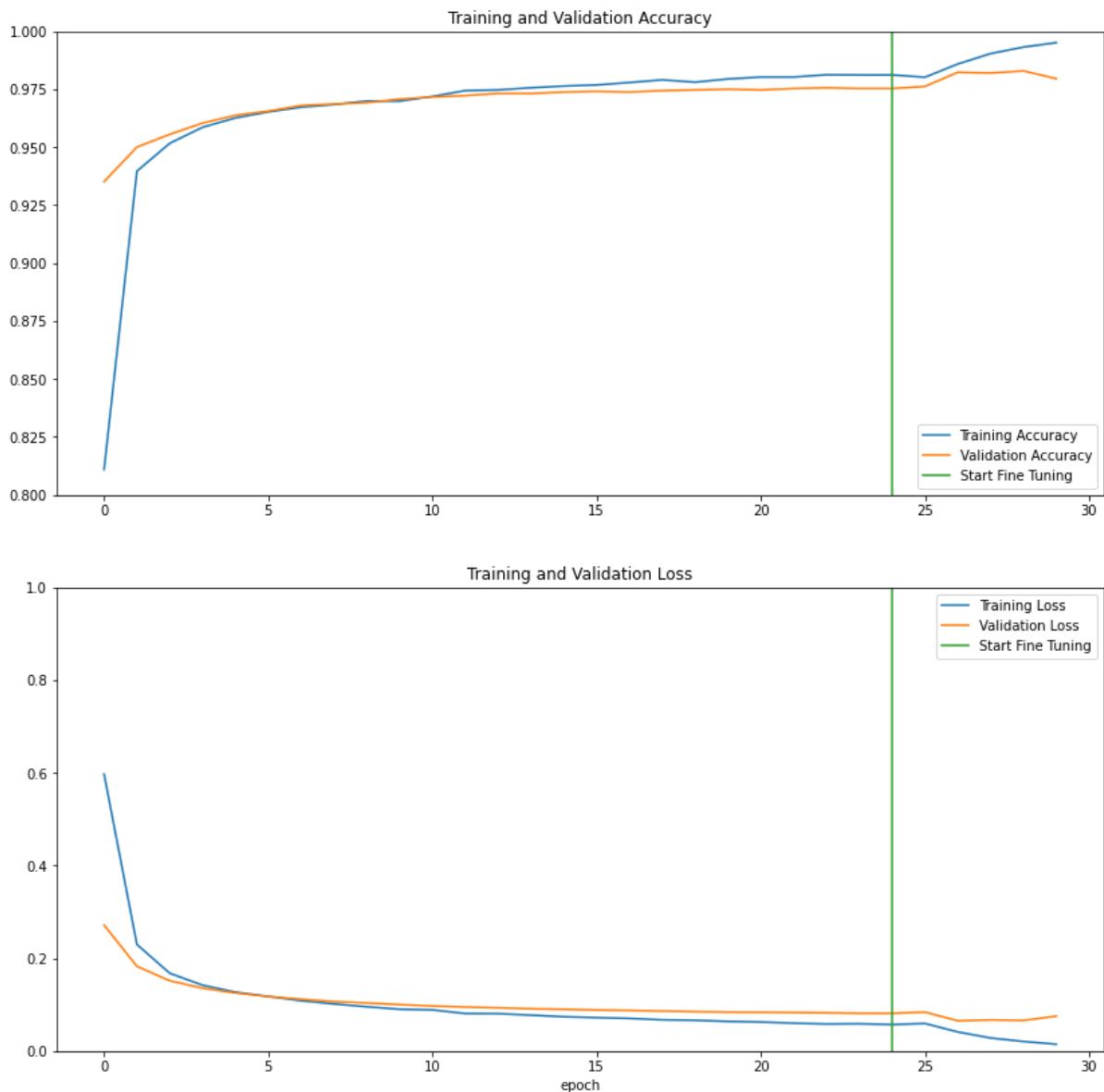
The learning rate for fine-tuning was 10 times smaller than before (learning_rate=0.00001), because otherwise the model could overfit very quickly. Early stopping was still set to 3 epochs.

Model: "MODEL2_IMPROVED"

Layer (type)	Output Shape	Param #
<hr/>		
input_8 (InputLayer)	[(None, 224, 224, 3)]	0
tf.math.truediv_3 (TFOpLambda)	(None, 224, 224, 3)	0
tf.math.subtract_3 (TFOpLambda)	(None, 224, 224, 3)	0
mobilenetv2_1.00_224 (Functional)	(None, 7, 7, 1280)	2257984
global_average_pooling2d_3 (GlobalAveragePooling2D)	(None, 1280)	0
dropout_3 (Dropout)	(None, 1280)	0
dense_3 (Dense)	(None, 5)	6405
<hr/>		
Total params: 2,264,389		
Trainable params: 1,867,845		
Non-trainable params: 396,544		



During training, 23 epochs were performed (48 in total). However, the usage of EarlyStopping parameter worked and cut the learning in the 30th epoch.



8. Final model evaluation and discussion

Evaluate MODELS_IMPROVED on the testing dataset using all defined metrics

Plot the results and discuss them. Discussion may include for example:

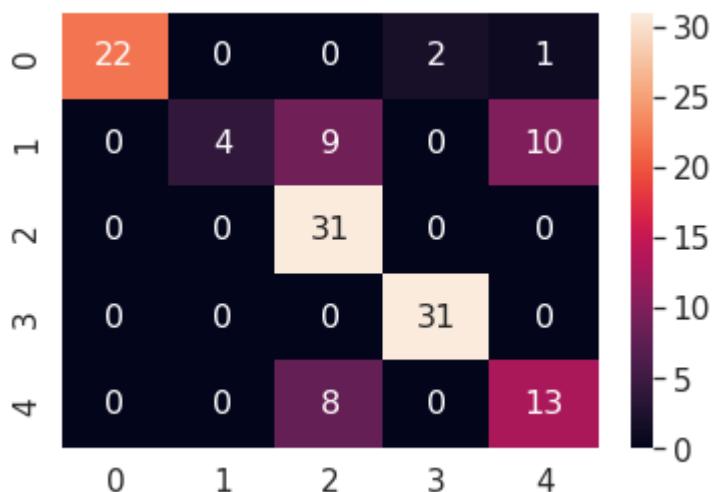
- Differences and similarities in results for different models and optimizations
- Analysis of errors and their causes
- Differences in results for various test metrics (you shall describe what these differences mean)
- Other observations, considerations and conclusions.

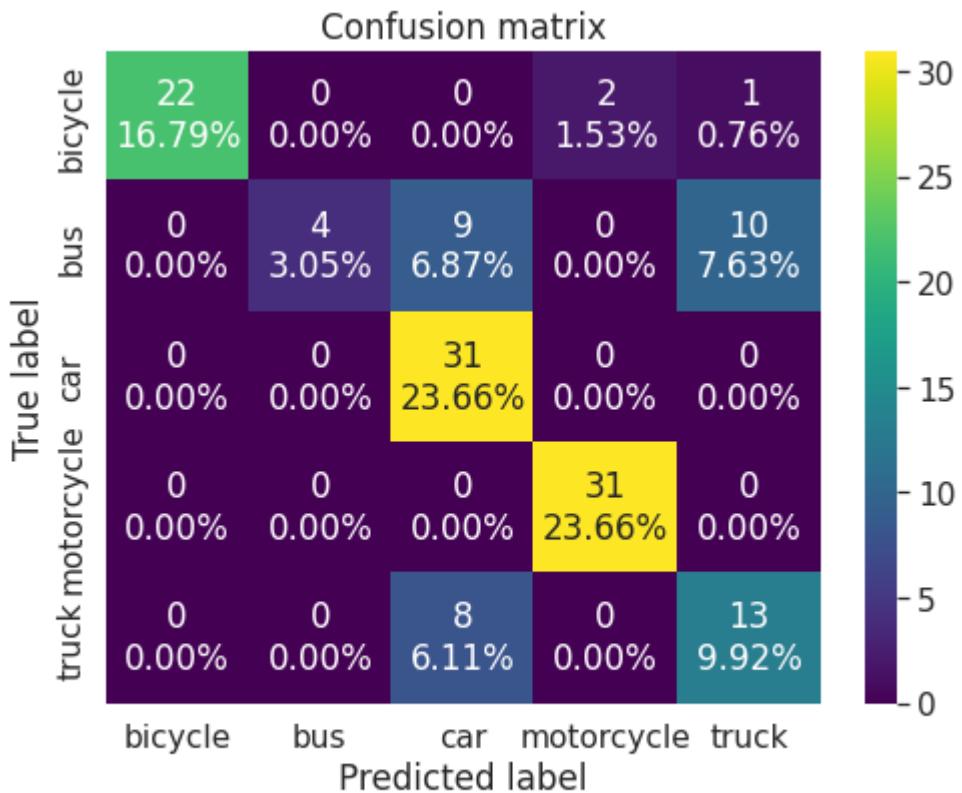
MODEL2_IMPROVED

MODEL2_IMPROVED	testset1	testset2	valset	trainset
Accuracy	0,770992339	0,97625	0,9872	1
F1 score	0,738265	0,976119	0,8932	0,8887
ROC AUC	0,970014	0,998572	-	-
Precision	0,828374	0,977136	0,7997	0,7968
Recall	0,770992	0,97625	1,016	1,0102
Cohens kappa	0,708975	0,970313	-	-

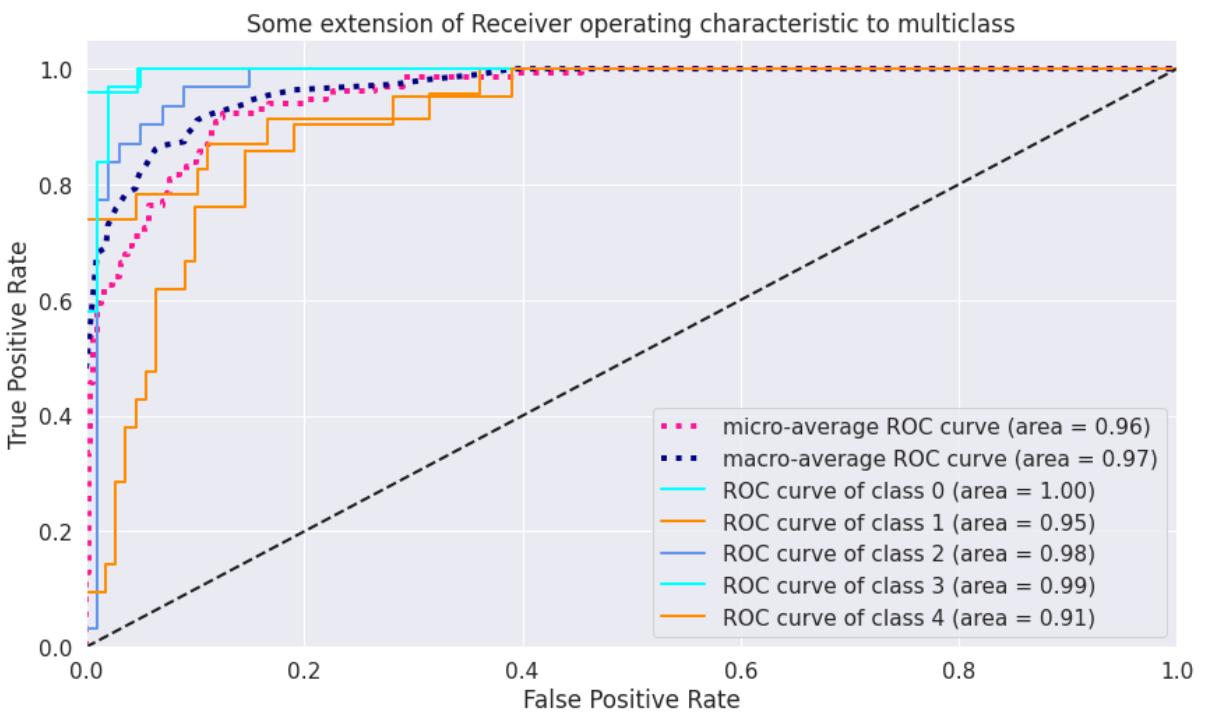
Tuned model achieves results similar to the MODEL2_BASELINE model. On the first test set, accuracy reached 77,1%, f1-score 73.9%. On the second test set, accuracy reached 97,6%, f1-score also 97,6%.

- **testset1**

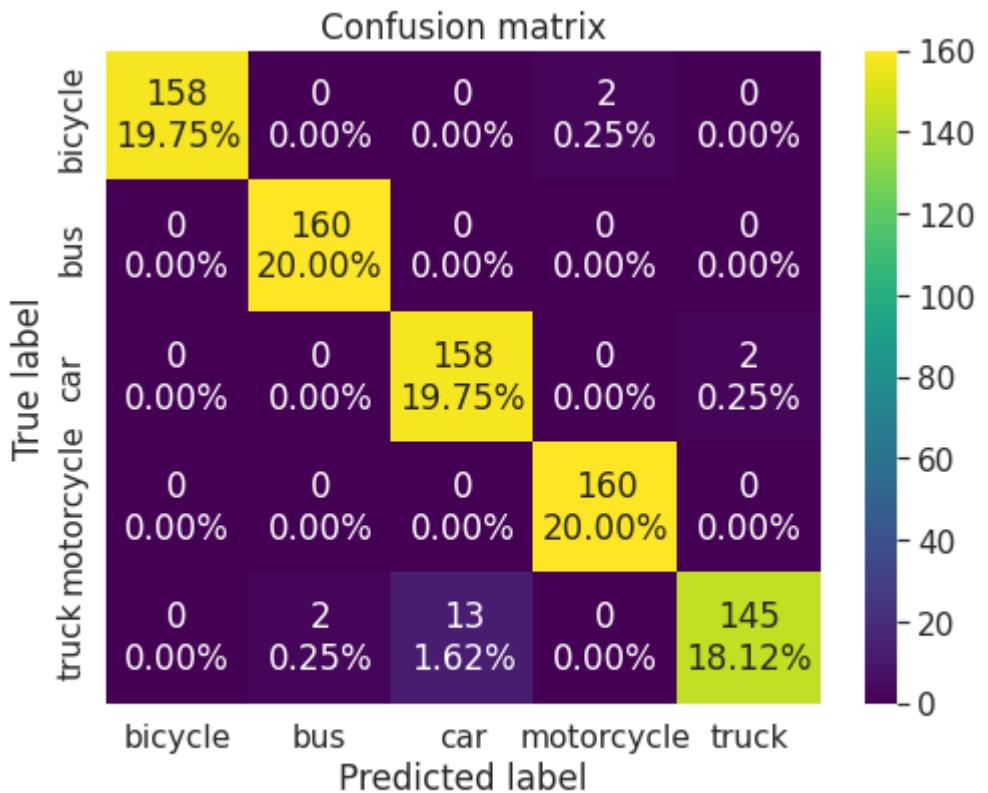




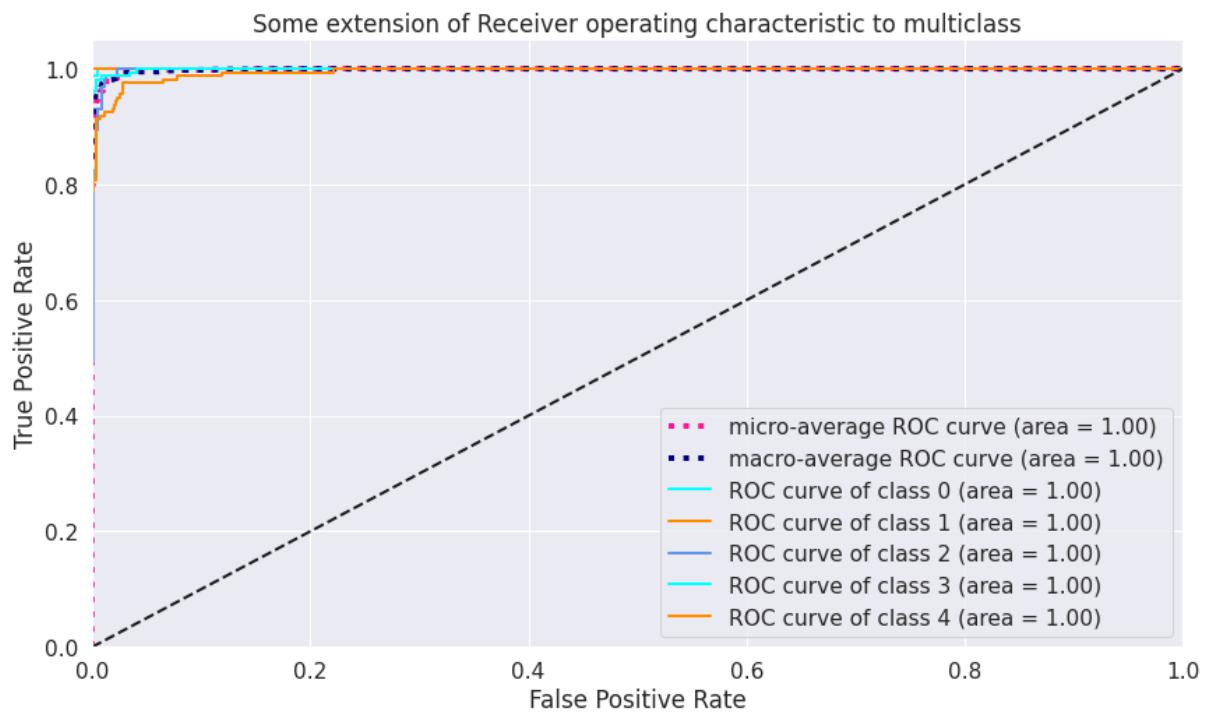
Accuracy=0.771



- testset2



Accuracy=0.976



9. Project summary. Summarize the entire project. Write about problems you encountered and what you learned.

The goal of the entire project was to create the system (machine learning model) which will recognize ground vehicles in the photos and classify them accordingly. The goal was fully achieved and the results are satisfactory. The featured classes were car, bus, truck, motorcycle and bicycle. In the case of motorcycles and bicycles the classification went perfectly - the model has no problem classifying it and usually makes no mistakes. In the case of car, bus and truck mistakes occurs. The reason for that is the problem with distinguishing minibus-like passenger cars and vans. The possible solution for the problem would be to create a separate class for the minivan-like vehicles and store all ambiguous examples there.

The table below presents the comparison of all the models used and evaluated in the project. The source code is attached to the report.

model_name	MODEL1_BASELINE	MODEL2_BASELINE	MODEL3_BASELINE	MODEL2_IMPROVED
train_examples	6474	13151	16437	13151
val_examples	1294	3286	3286	3286
epochs_performed	25	25	25	30
learning_rate	0.0001	0.0001	0.0001	0.00001
optimizer	Adam	Adam	Adam	RMSprop
batch_size	32	32	32	32
val_loss	0.086922	0.081698	0.048375	0.074404
val_accuracy	0.972179	0.974437	0.985697	0.987523
test1_loss	0.832436	0.912332	0.803597	1.662321
test1_accuracy	0.770992	0.763359	0.770992	0.755725
test2_loss	0.099143	0.068278	0.065667	0.086333
test2_accuracy	0.965	0.9775	0.97875	0.97625
test1_f1_score	0.71651	0.701382	0.724929	0.705492
test1_ROC	0.96605	0.95734	0.967337	0.968055
test2_f1_score	0.964818	0.977309	0.978626	0.976119
test2_ROC	0.99593	0.996211	0.996771	0.998572