Побудова моделі для класифікації сміття

Постановка задачі:

1) Розробити модель CV для класифікації різних видів сміття, таких як пластик, метал, папір тощо.
2) Під час тренування моделі врахувати реальні кейси її застосування, тобто модель має навчитися розрізняти різні відходи, незважаючи на зміни умов освітлення та інші зміни навколишнього середовища;

Критерії валідації:

1) Тренувальна точність моделі нейронної мережі – 90% і більше;
2) Валідаційна точність моделі нейронної мережі – 85% і більше;
3) Тестування моделі на реальних (зроблених самостійно) зображеннях сміття.

Представлені граничні моменти:

Враховані граничні моменти:

1) Різні кути, масштаб, освітлення, контрастність, шуми, фони на зображеннях/фотографіях сміття.

Невраховані граничні моменти:

1) Наявність інших класифікацій сміття, відмінних від досліджуваної;
2) Не проводилося тестування поведінки моделі для зображень, які не містять сміття.

План виконання завдання

1) Знайти та дослідити датасет для вирішення поставленої задачі;
2) Визначити та реалізувати необхідні техніки аугментації з метою збільшення обсягу датасету та імітування зображень, з якими потенційно у майбутньому може мати справу модель;
3) Визначити оптимальну архітектуру моделі для тренування та оцінити показники ефективності натренованої моделі;
4) Протестувати модель на самостійно зроблених зображеннях сміття;
5) Створити демонстраційний варіант використання моделі у вигляді Streamlit-застосунку.

Етап 1: Пошук і дослідження датасету

Для виконання завдання було обрано доволі популярний (рис. 1) датасет Garbage Classification Dataset (Kaggle): https://www.kaggle.com/datasets/asdasdasasdas/garbage-classification.

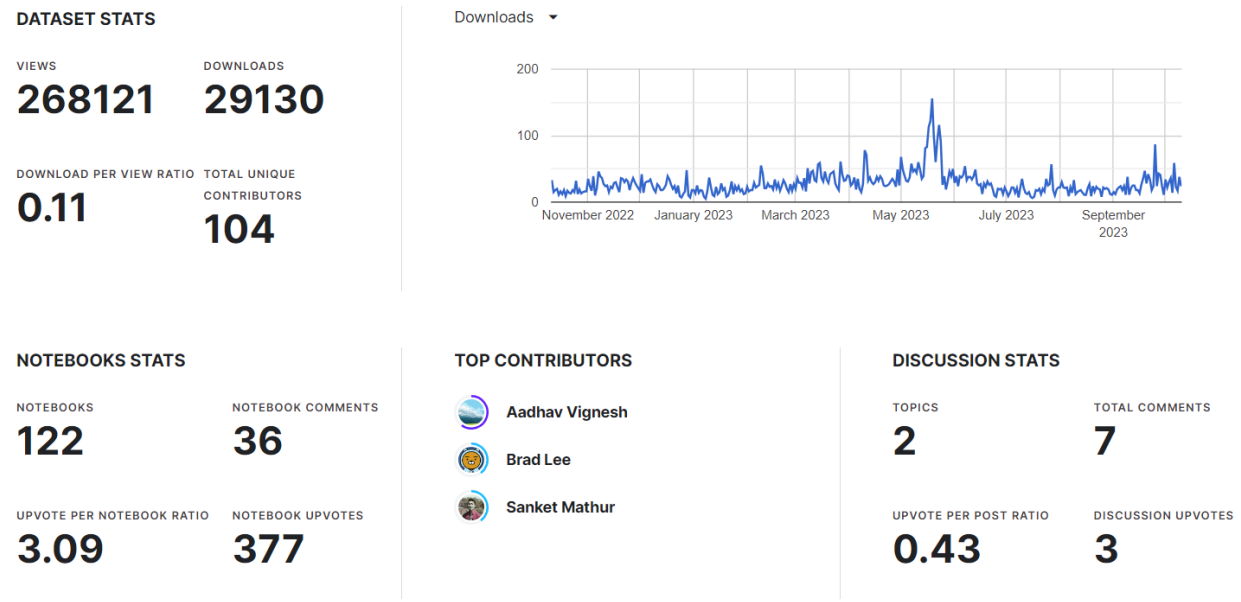

Рис. 1. Статистика обраного датасету

Датасет містить 2527 зображень сміття розміром 512×384, поділених на 6 класів (рис. 2):

1) Картон (cardboard – рис. 3);
2) Скло (glass – рис. 4);
3) Метал (metal – рис. 5);
4) Папір (paper – рис. 6);
5) Пластик (plastic – рис. 7);
6) Інше сміття (trash – рис. 8).

About Dataset

Garbage Classification Data

The ***Garbage Classification Dataset*** contains 6 classifications: cardboard (393), glass (491), metal (400), paper(584), plastic (472) and trash(127).

Рис. 2. Інформація про розподіл класів сміття датасету

Рис. 3. Приклади зображень класу cardboard

Рис. 4. Приклади зображень класу glass

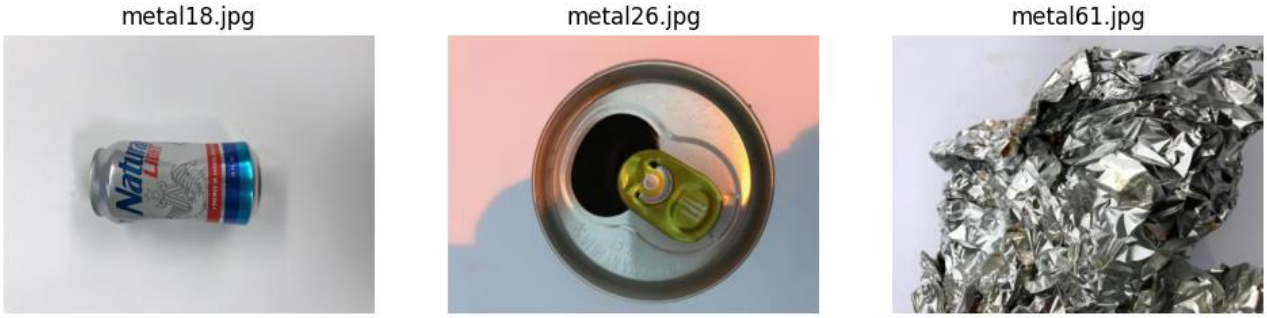

Рис. 5. Приклади зображень класу metal

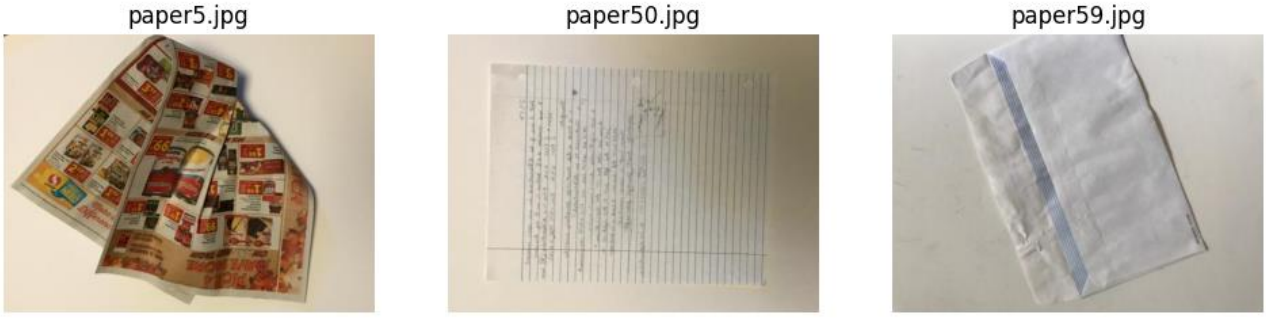

Рис. 6. Приклади зображень класу paper

Рис. 7. Приклади зображень класу plastic

Рис. 8. Приклади зображень класу trash

Досліджуваний датасет є незбалансованим (рис. 9-10): найбільше екземплярів класу paper, найменше – класу trash (менше 10%).

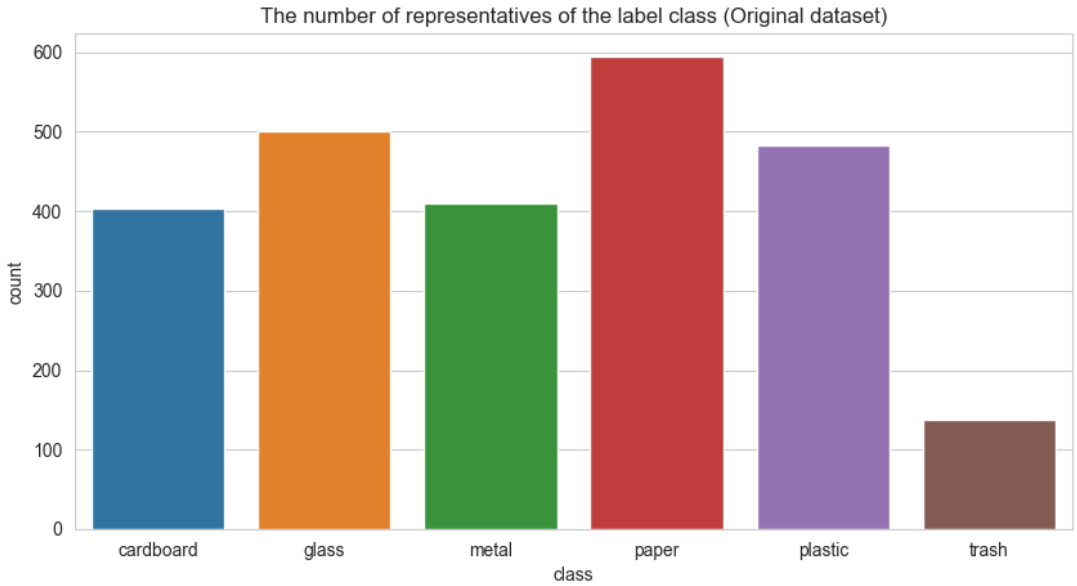

Рис. 9. Аналіз розподілу класів досліджуваного датасету

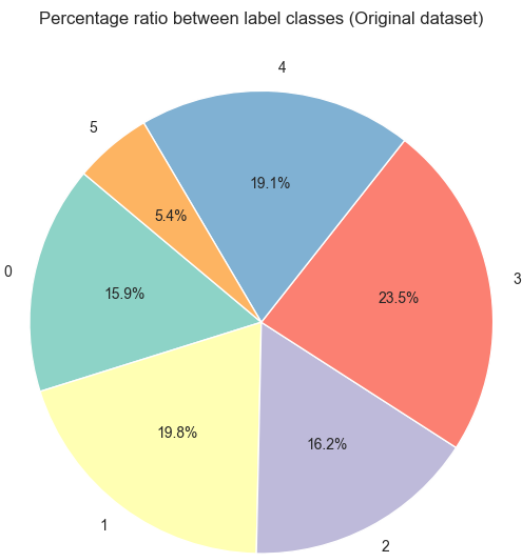

Рис. 10. Аналіз розподілу класів досліджуваного датасету

Досліджуваний датасет було поділено спочатку на тренувальну/тестову вибірки у співвідношенні 90/10, а тренувальну в свою чергу – на тренувальну і валідаційну у співвідношенні 85/15 так, щоб в кожній вибірці зберігалося співвідношення між класами сміття (рис. 11). В результаті, для тренування моделі було виділено 1927 зображень, для валідації – 344, для тестування – 256 зображень.

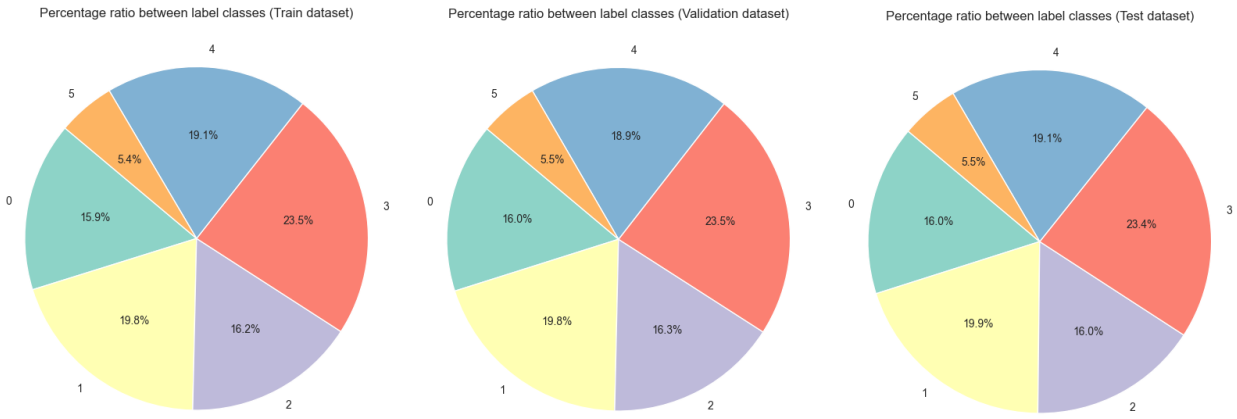

Рис. 11. Аналіз розподілу класів сміття у тренувальному/валідаційному/тесту вальному датасеті відповідно

Етап 2: Визначення та реалізація технік аугментації

Значна частина технік аугментації, які виконувалися над тренувальними і валідаційними зображеннями, була реалізована з допомогою функцій бібліотеки OpenCV, що забезпечило такі переваги (порівняно із використанням класу ImageDataGenerator фреймворку TensorFlow) під час підготовки даних:

1) Клас ImageDataGenerator забезпечує не всі необхідні для вирішення задачі техніки аугментації;
2) Повний контроль над аугментацією зображень і розуміння застосованих технік;

3) Можливість створювати різні комбінації реалізованих аугментацій;
4) Необхідність зберігати аугментовані зображення в папках із тренувальними і валідаційними даними.

Також варто зазначити, що деякі техніки аугментації все ж передбачали використання класу ImageDataGenerator.

Для реалізації завдання було визначено такі техніки аугментації:

1) Повороти (rotation augmentation): Обертання обробляє зміни в орієнтації об’єкта через різні розташування або кути, під якими може бути сфотографовано сміття (рис. 12). Цю техніку аугментації було реалізовано з допомогою функцій getRotationMatrix2D() та warpAffine() бібліотеки OpenCV з метою чітко контролювати кількість аугментованих з її допомогою зображень та відповідні кути повороту.

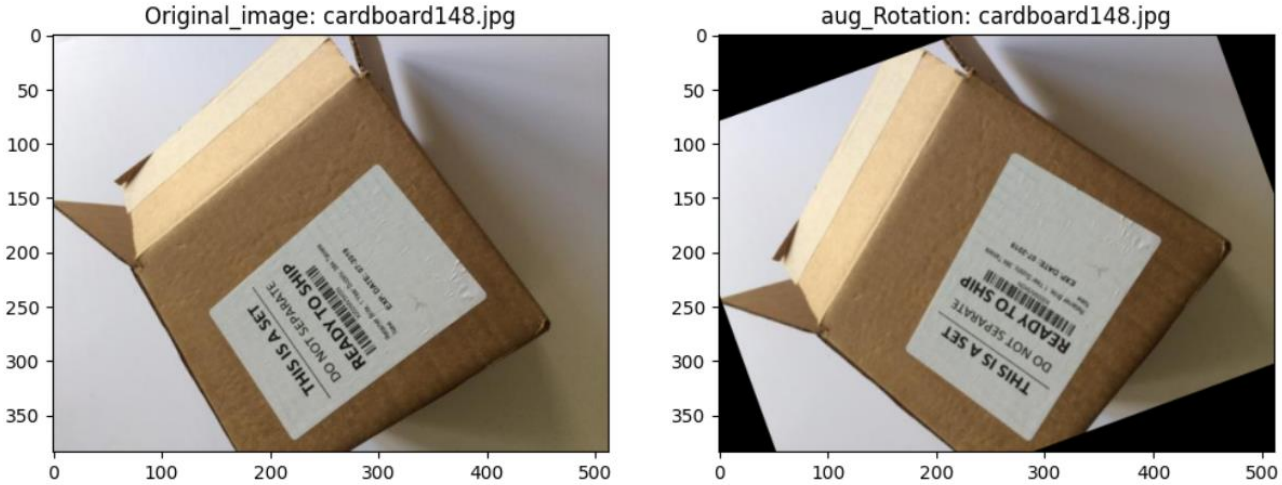

Рис. 12. Приклад rotation-аугментації

2) Зміщення по ширині та висоті (width and height shifts augmentation): таке зміщення забезпечує надійність моделі проти змін у горизонтальному та вертикальному розташуванні об’єктів сміття, забезпечуючи точні передбачення, навіть якщо об’єкти не ідеально відцентровані в кадрі (рис. 13-14). Ці техніки аугментації було реалізовано з допомогою проміжних обчислень та функції warpAffine() бібліотеки OpenCV з метою створення для кожного зображення чотирьох аугментованих: з нульовими пікселями зверху, знизу, зліва та справа (ImageDataGenerator проводить аугментацію зображень рандомно, що призводило до випадків, коли для одного зображення утворювалися два аугментовані з нульовими векторами зверху замість того, щоб були зверху і знизу – рис. 15).

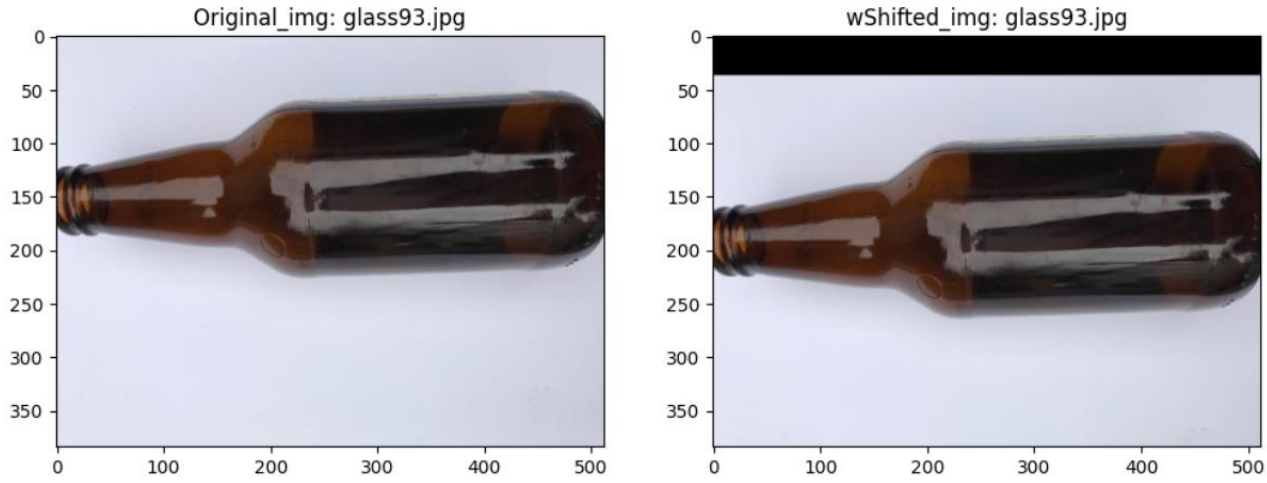

Рис. 13. Приклад width_shift-аугментації

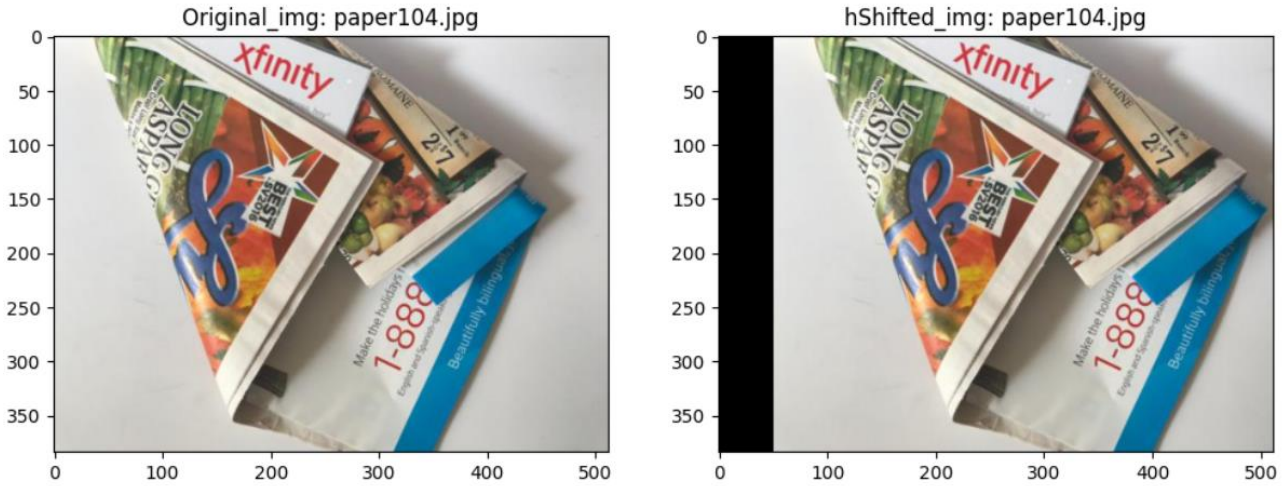

Рис. 14. Приклад height_shift-аугментації

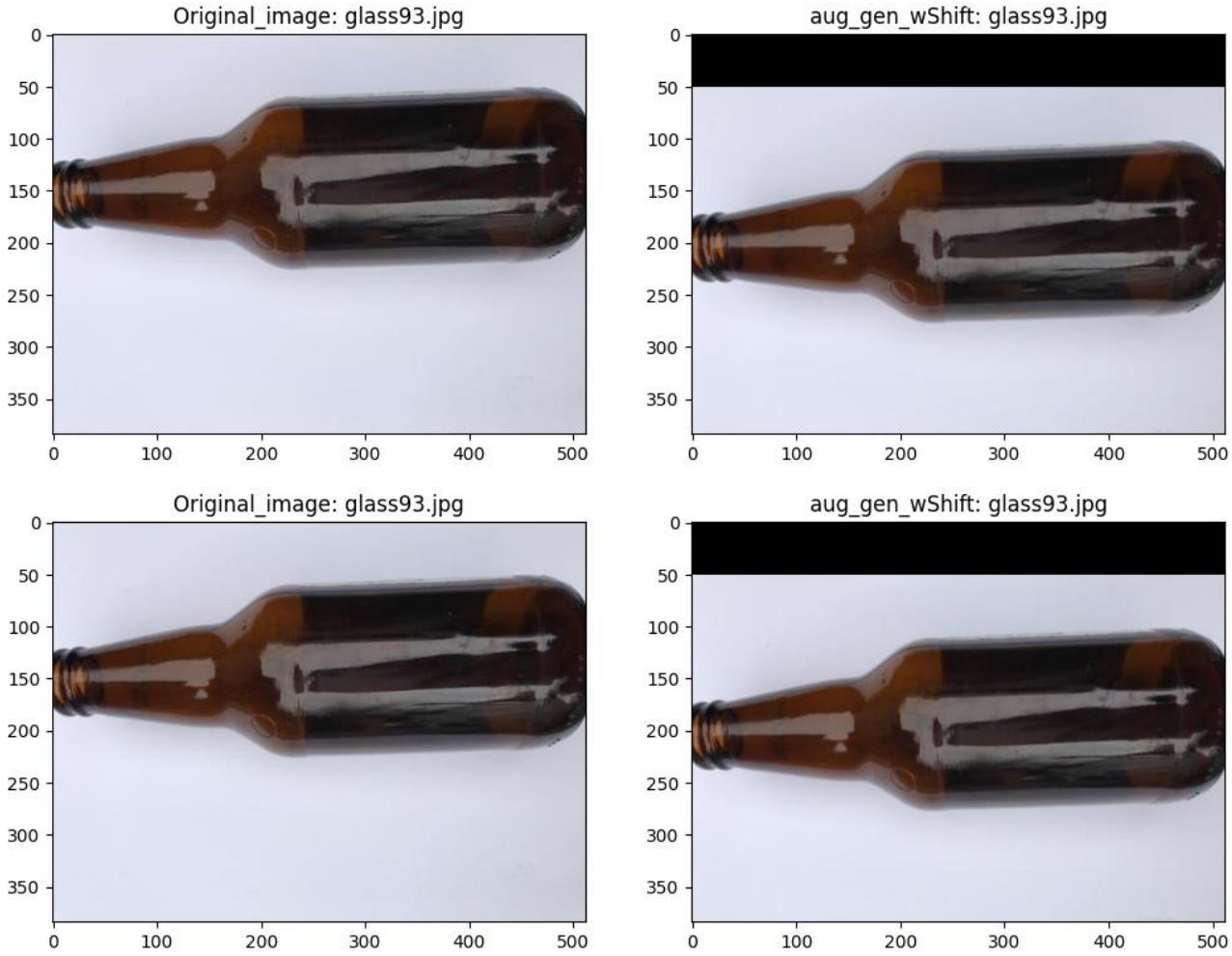

Рис. 15. Приклад width_shift-аугментації з допомогою класу ImageDataGenerator

3) Горизонтальне та вертикальне відзеркалення (horizontal and vertical flips augmentation): створення дзеркальних зображень для обробки різних кутів, під якими може бути сфотографоване сміття (рис. 16-17). Реалізація передбачала використання функції flip() бібліотеки OpenCV та була здійснена у зв’язку із необхідністю зберігати аугментовані зображення в окремій папці, а ImageDataGenerator проводить ці техніки аугментації із шансом 50%, що призводило до зберігання в папці оригінальних зображень, позначених як аугментовані (рис. 18).

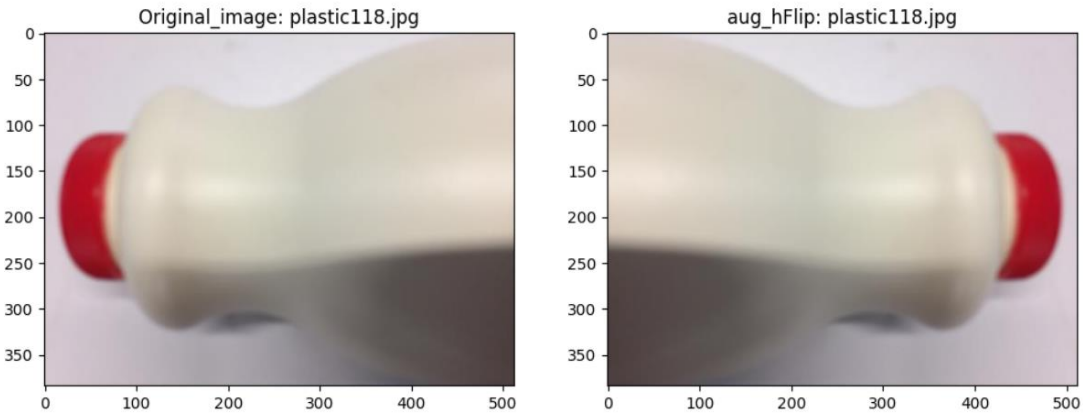

Рис. 16. Приклад horizontal_flip-аугментації

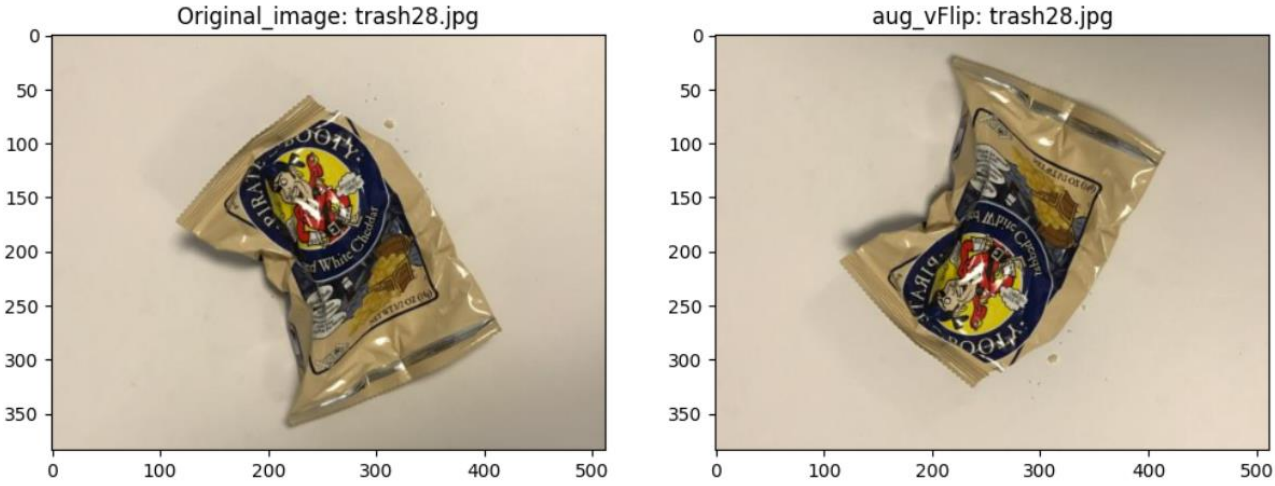

Рис. 17. Приклад vertical_flip-аугментації

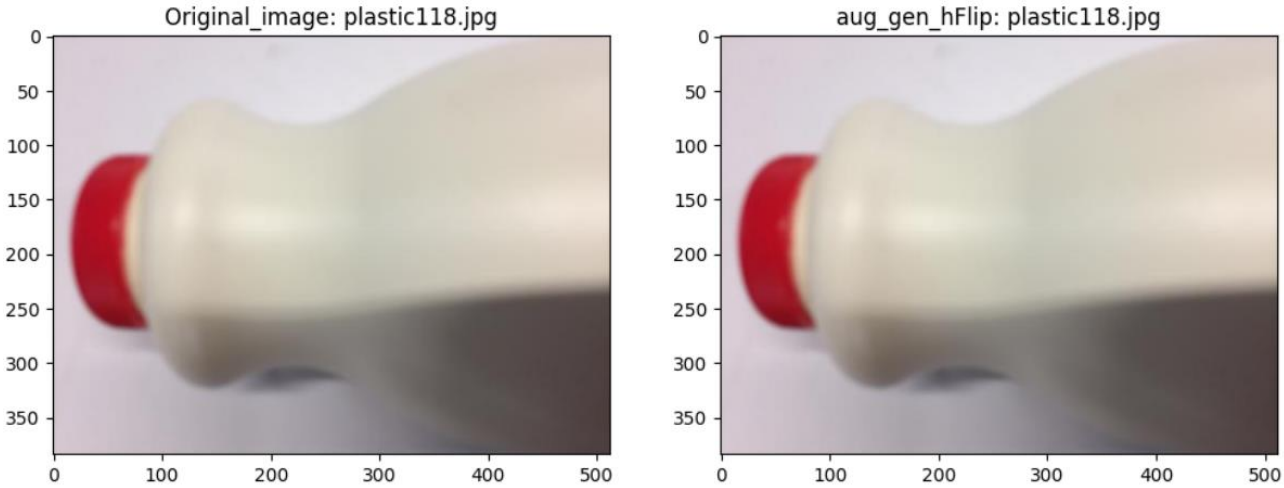

Рис. 18. Приклад horizontal_flip-аугментації з допомогою класу ImageDataGenerator

4) Масштабування зображення (zoom=scaling augmentation): збільшення або зменшення масштабу зображень максимум на 20% для імітації варіацій відстані камери (рис. 19-20). Ця техніка аугментації передбачала використання класу ImageDataGenerator із параметром zoom_range=(0.8, 1) (increase_zoom-аугментація) та zoom_range=(1, 1.2) (decrease_zoom-аугментація).

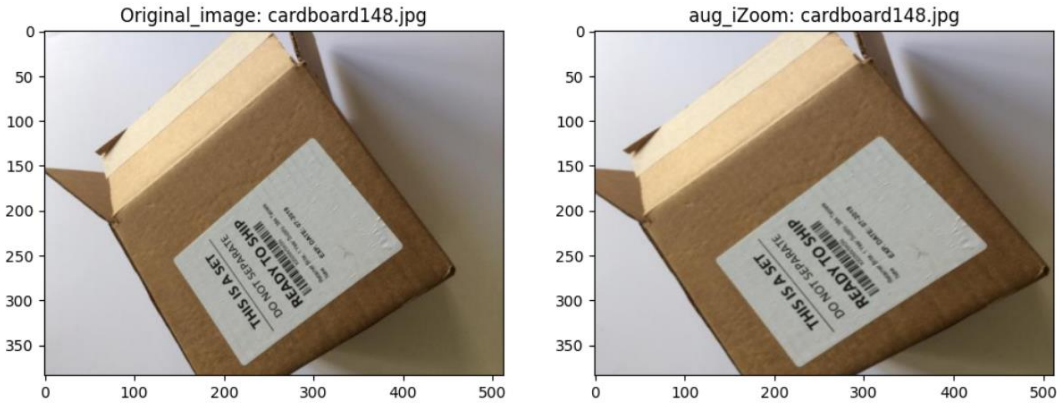

Рис. 19. Приклад increase_zoom-аугментації

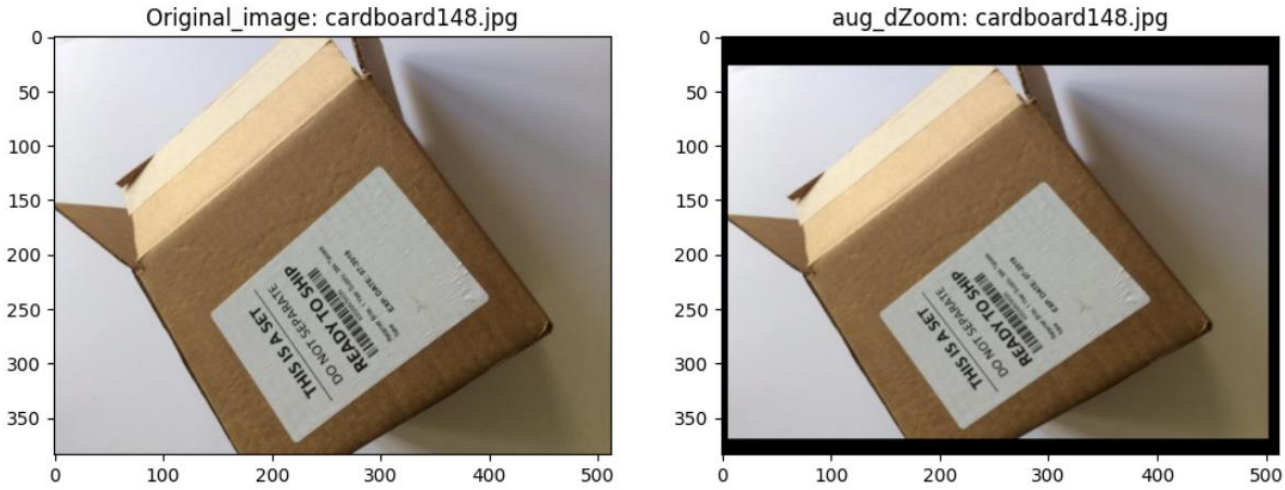

Рис. 20. Приклад decrease_zoom-аугментації

5) Зміна яскравості зображення (brightness augmentation): імітація різної інтенсивності освітлення шляхом регулювання яскравості зображень (рис. 21-22). Для реалізації використовувався клас ImageDataGenerator із параметром brightness_range=(0.5, 0.65) (blackBrightness-аугментація) та brightness_range=(1.10, 1.25) (lightBrightness - аугментація).

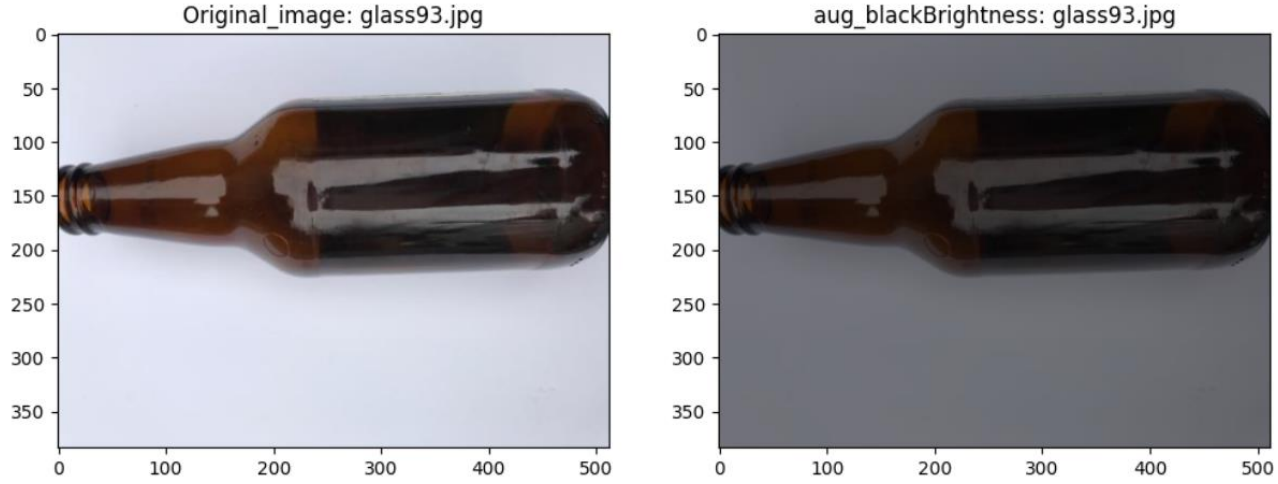

Рис. 21. Приклад blackBrightness-аугментації

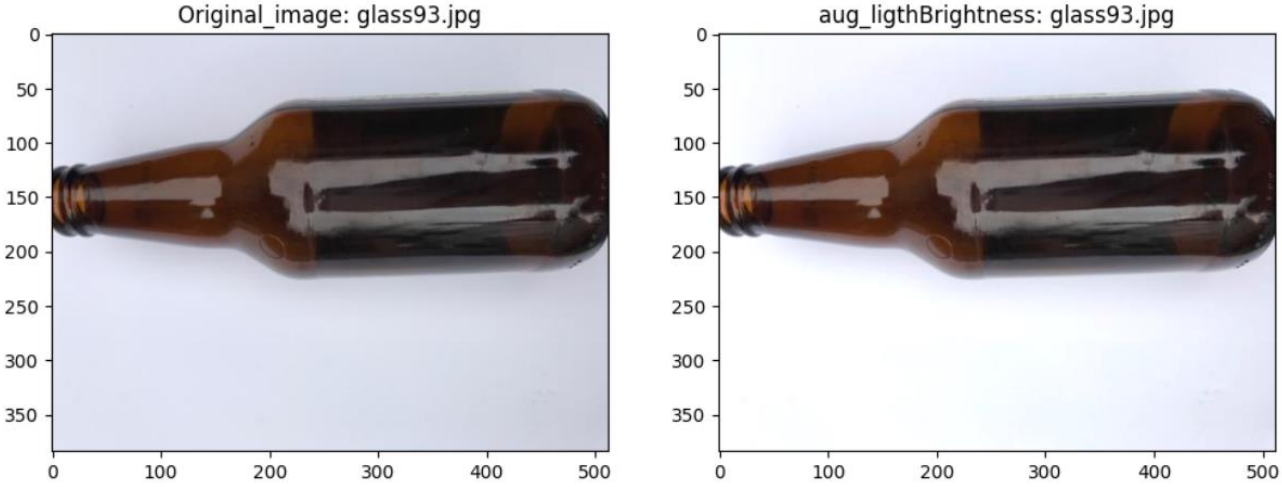

Рис. 22. Приклад lightBrightness-аугментації

6) Зміна контрастності зображень (contrast augmentation): зміна рівня контрастності для врахування різних умов освітлення (рис. 23). Для реалізації цієї техніки аугментації було застосовано CLAHE (Contrast Limited Adaptive Histogram Equalization) до L-каналу (функція createCLAHE() та clahe.apply()).

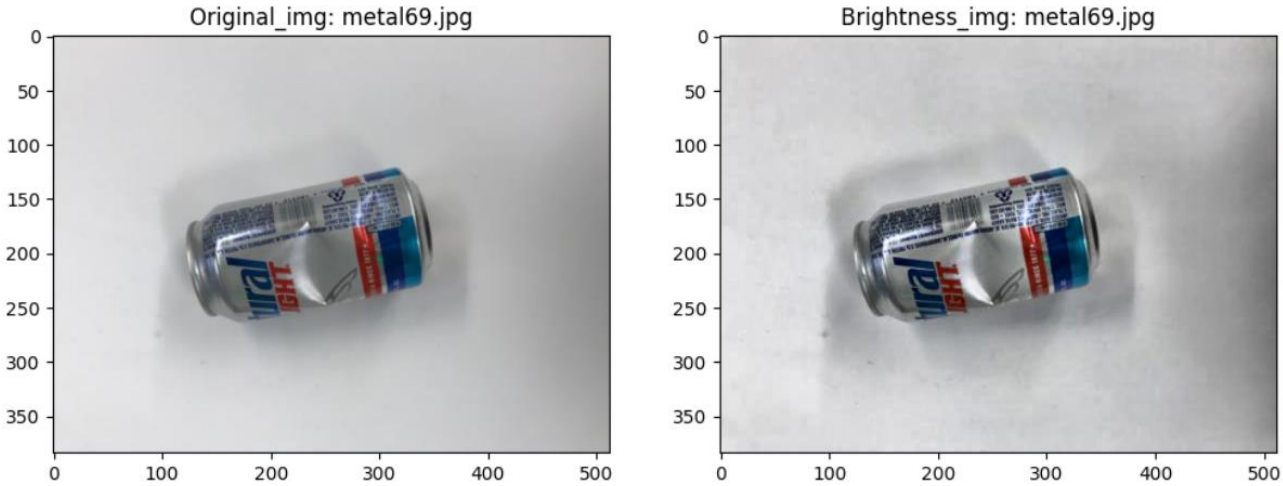

Рис. 23. Приклад contrast-аугментації

7) Зміна фону зображення (backgrounds augmentation): поєднання зображень сміття з різним фоном для імітації різноманітних контекстів навколишнього середовища (рис. 24). Результати аугментації для одного із зображень наведено на рис. 25. Реалізація backgrounds augmentation була найважчою серед усіх реалізованих самостійно аугментацій і передбачала використання функції remove() бібліотеки rembg для видалення фону оригінального зображення сміття та використання бінарних масок (функції threshold(), bitwise_not(), bitwise_and() бібліотеки OpenCV) для накладання зображення сміття без фону на нове зображення фону.

Рис. 24. Зображення, які використовуватимуться в якості фонових для background-augmentation

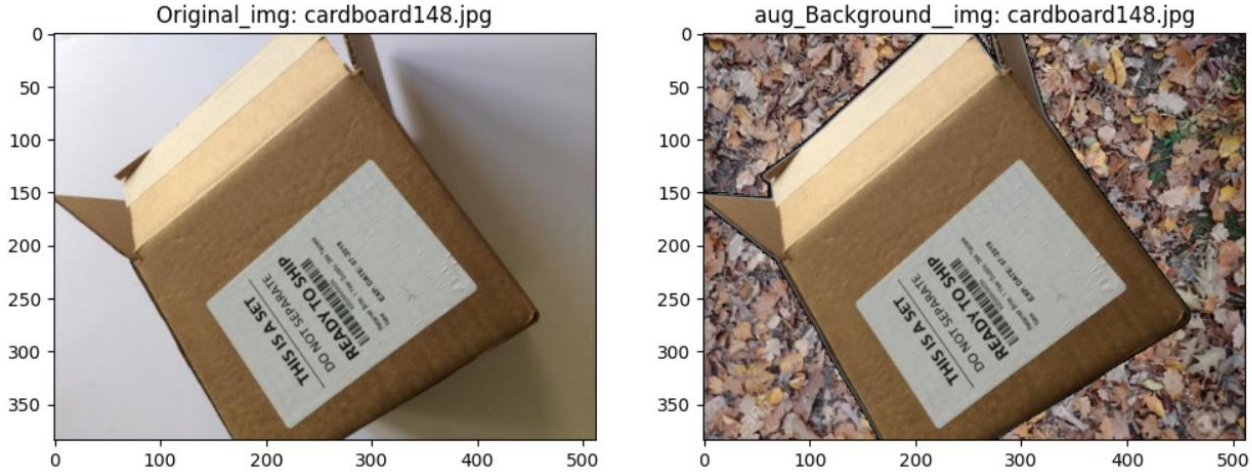

Рис. 25. Приклад background-аугментації

8) Трансформації колірного простору (HSV augmentation): зміна колірного простору може допомогти моделі працювати з різними колірними температурами в різних сценаріях освітлення (рис. 26). Для реалізації здійснювався перехід зображення у модель HSV та зміна відповідних характеристик зображення (під час аугментації змінювався лиш тон (Hue), однак реалізована функція get_hsv_image() дозволяє змінювати і насиченість (Saturation), і значення кольору (Value)).

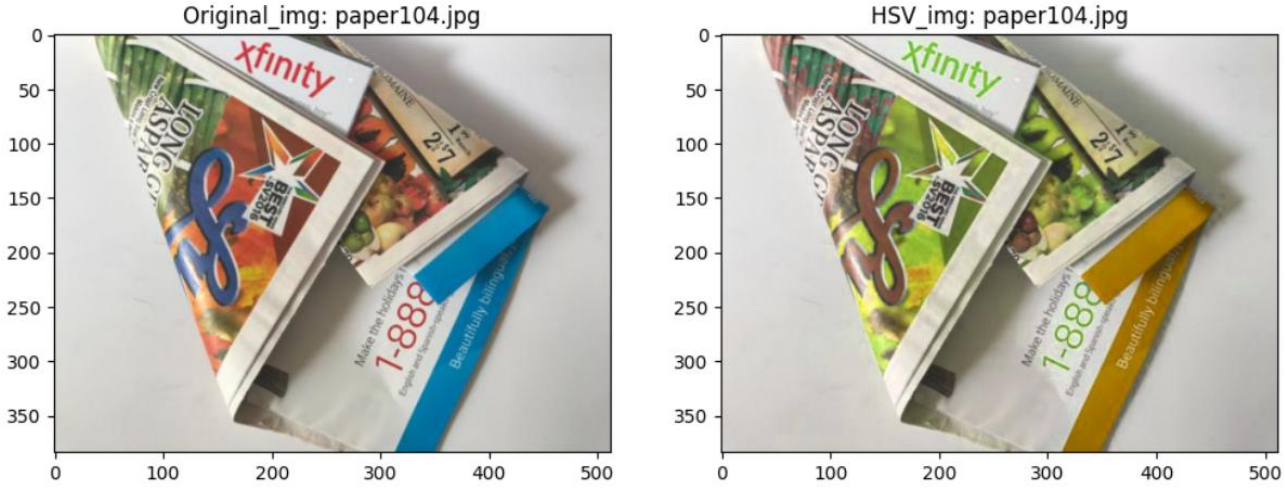

Рис. 26. Приклад hsv-аугментації

9) Додавання шуму (noise addition augmentation): додавання випадкового шуму до зображень, щоб зробити модель стійкою до варіацій на рівні пікселів (рис. 27). Реалізація передбачала використання статистичних параметрів – середнє статистичне (mean) та стандартне відхилення (standard deviation) і застосування функцій random.normal() бібліотеки numpy та add() бібліотеки OpenCV.

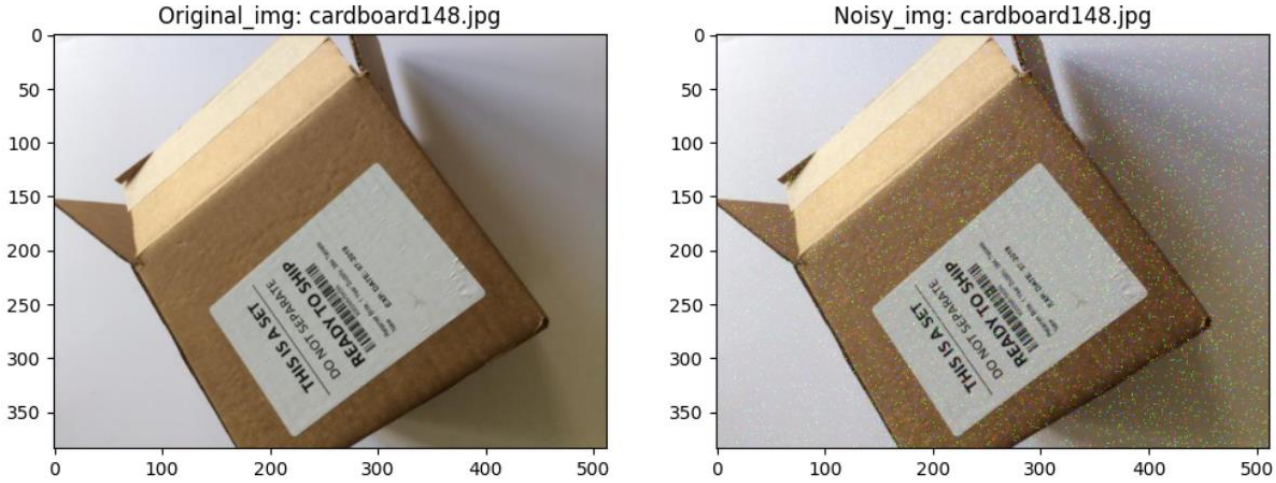

Рис. 27. Приклад noise_addition-аугментації

Як відомо, поєднання різних методів аушментації до навчальних даних підвищує різноманітність і якість набору даних, що використовується для навчання моделей машинного навчання. Тому деякі техніки аугментації було об’єднано з метою збільшити обсяг тренувальної і валідаційної вибірок та імітувати умови навколишнього середовища:

1) Зменшення зображення (zoom augmentation) + зміна фону (background augmentation): оскільки на значній частині зображень датасету зображення сміття займає майже весь простір, його зменшення дозволить отримати правдоподібніші результати аугментації (рис. 28).

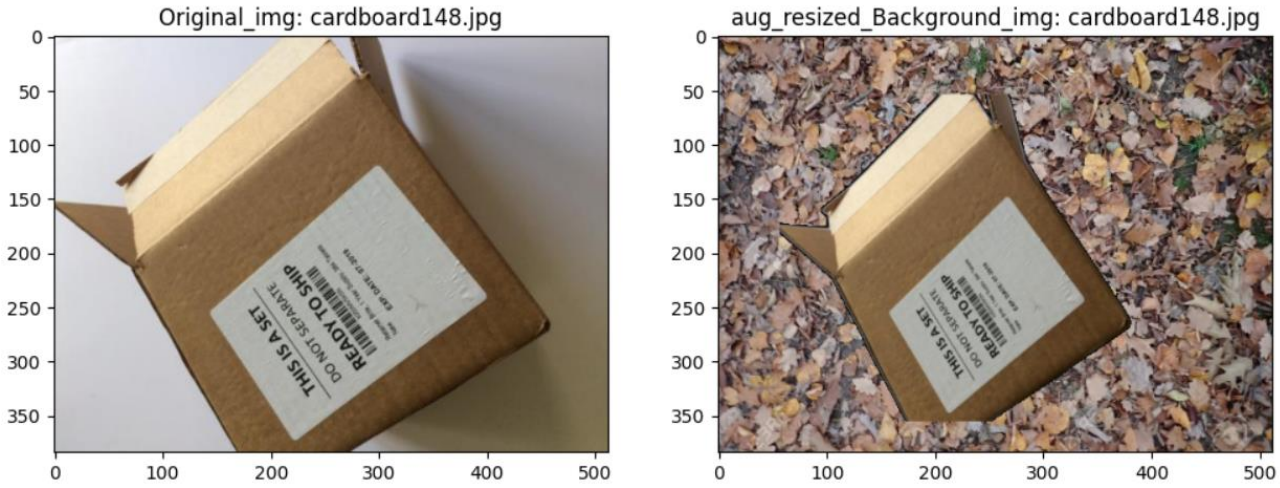

Рис. 28. Приклад resized_Background-аугментації

2) Зміна яскравості (dark brightness augmentation) + зміна контрастності (contrast augmentation) зображень: додаткова імітація різної інтенсивності освітлення (рис. 29).

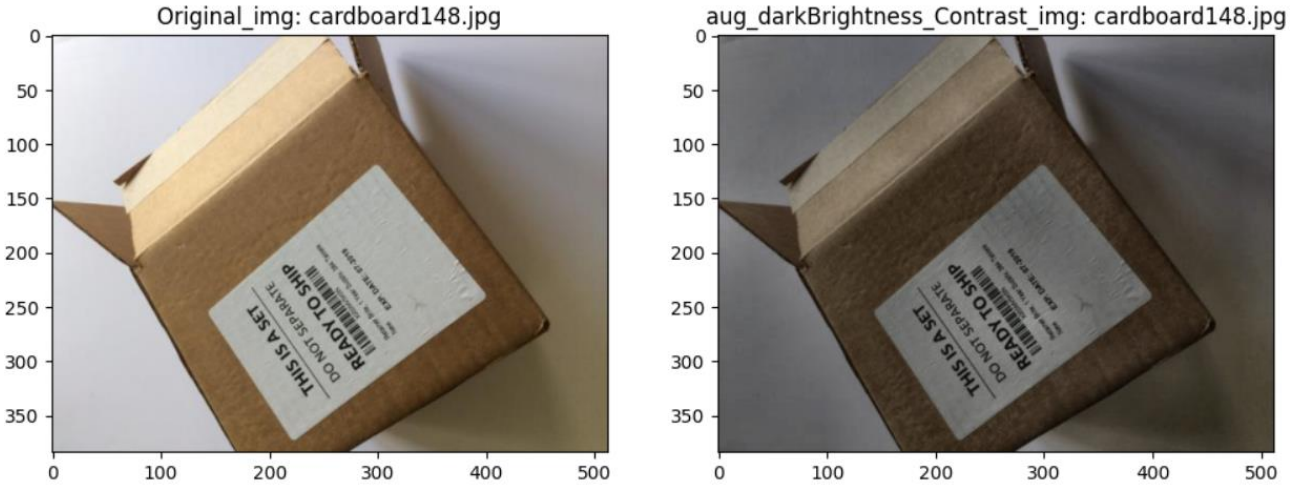

Рис. 29. Приклад darkBrightness_Contrast-аугментації

3) Зміна яскравості (light brightness augmentation) + зміна контрастності (contrast augmentation) + трансформації колірного простору (HSV augmentation) зображень: додаткова імітація різної інтенсивності освітлення та колірних температур (рис. 30).

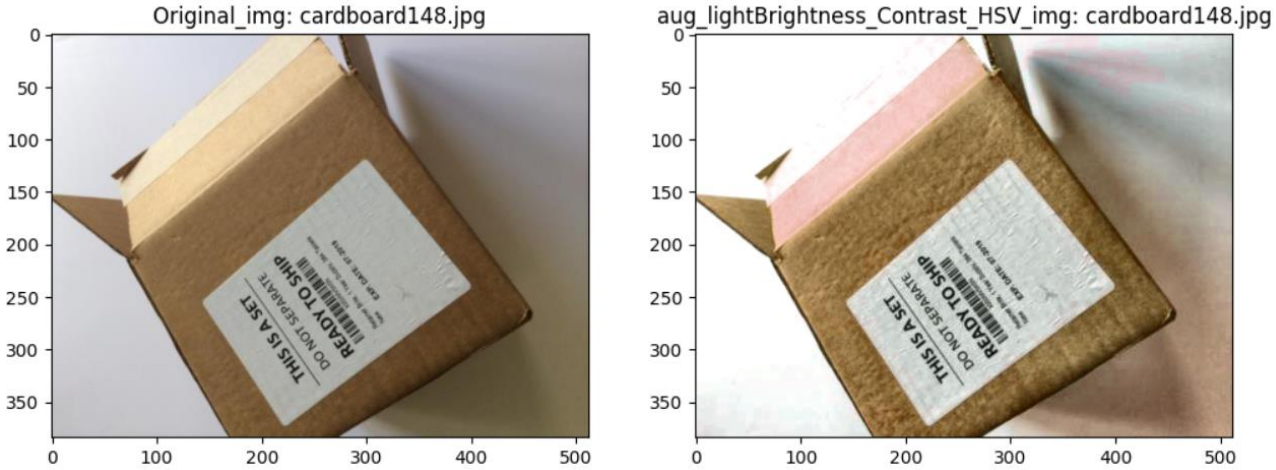

Рис. 30. Приклад lightBrightness_Contrast_HSV-аугментації

Етап 3: Визначення оптимальної архітектури моделі, її тренування та оцінка ефективності

Перед тренуванням моделі нейронної мережі було збільшено обсяг датасету за рахунок використання реалізованих технік аугментації (на момент першого тренування моделі не всі з перелічених вище технік аугментації були реалізовані):

1) Rotation augmentation;
2) Width shift and height shift augmentation;
3) Horizontal flip and vertical flip augmentation;
4) Zoom augmentation;
5) Brightness shift augmentation;
6) Contrast augmentation;
7) Color Space Transformations (HSV) augmentation.

В результаті аугментації розмір тренувального/валідаційного датасетів збільшився від 1927/344 до 36613/6536 зображень (оскільки до кожного оригінального зображення застосовувалися всі згадані техніки аугментації), тобто обсяги зросли в 19 разів.

Для відстеження ефективності тренування моделі нейронної мережі та передчасної її зупинки було використано власний клас myCallback (рис. 31) та EarlyStopping (рис. 32).

```
class myCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if(logs.get('accuracy')>0.99):
            print("\nReached 99% accuracy so cancelling training!")
            self.model.stop_training = True
```

Рис. 31. Клас myCallback

```
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=5)
my_callback = myCallback()
```

Рис. 32. Використані під час тренування моделі нейронної мережі колбеки

Архітектура моделі (рис. 33) передбачає використання таких шарів:

- Conv2D - виконує операції згортки над вхідними 2D-даними, вилучаючи ієрархічні характеристики та шаблони, які є важливими для таких завдань, як розпізнавання зображень і комп'ютерне бачення;
- MaxPooling2D - зменшує дискретизацію вхідних просторових розмірів, зберігаючи найважливішу інформацію, вибираючи максимальні значення з групи сусідніх пікселів;
- Flatten - перетворює багатовимірний вихід із попередніх шарів на одновимірний вектор, що дозволяє мережі з'єднувати повністю пов'язані рівні для подальшої обробки та прийняття рішень.
- Dense - виконує обчислення *output = activation(weight * input + bias)*.

```
model = tf.keras.models.Sequential(layers=[
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(300, 300, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(6, activation='softmax')
])
```

Рис. 33. Архітектура моделі нейронної мережі

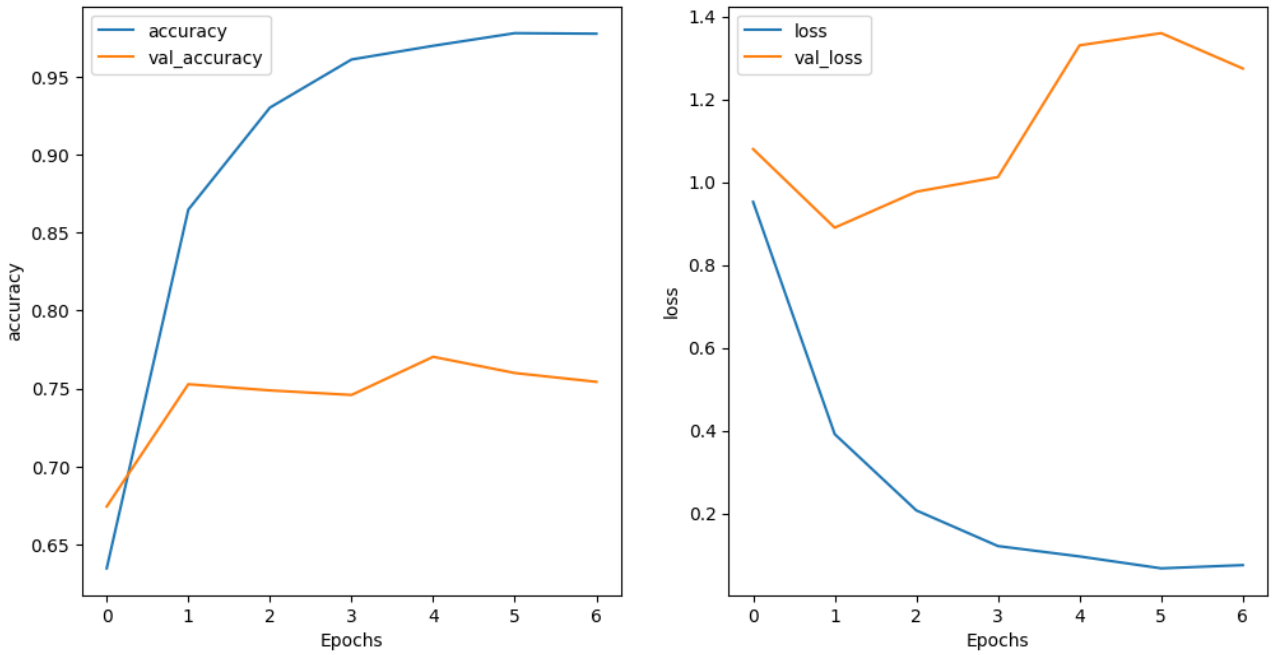

Рис. 34. Історія тренування моделі

Із рис. 34 видно, що незважаючи на високу тренувальну точність (понад 95%), валідаційна точність протягом другої-шостої епох перебувала в діапазоні 74-78%. Після другої епохи val_loss почала різко зростати, що призвело до передчасної зупинки тренування моделі (спрацював EarlyStopping callback).

Для уникнення процесу перенавчання архітектуру моделі було трохи спрощено, а також було додано Dropout шар і L2-регуляризацію (рис. 35). Окрім того, параметр patience EarlyStopping колбеку було зменшено від 5 до 3.

```
model = tf.keras.models.Sequential(layers=[
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(300, 300, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.01)),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(6, activation='softmax')
])

model.summary()
```

Рис. 35. Спрощена архітектура моделі нейронної мережі

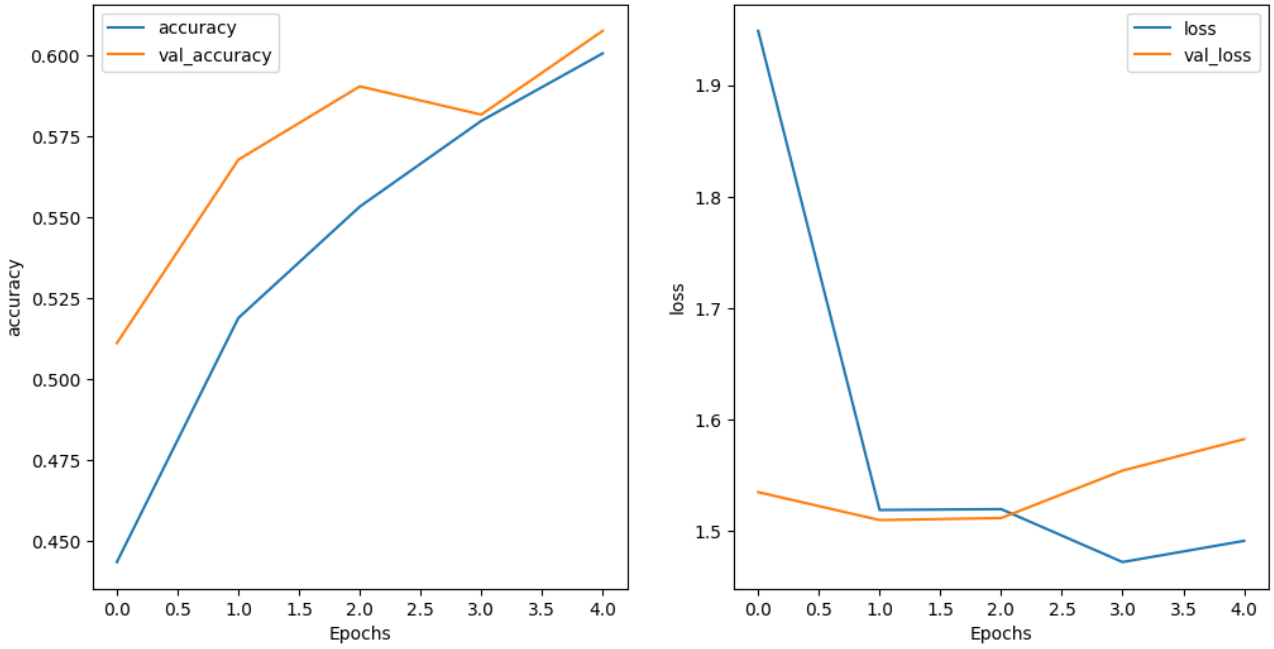

Рис. 36. Історія тренування моделі

Із рис. 36 видно, що тренувальна і валідаційна точності зростали протягом тренування, але val_loss після другої епохи почала зростати, що призвело до передчасної зупинки тренування моделі. На тестових даних модель показала дуже посередні результати для більшості класів і дуже погані для класу trash (рис. 37, 38).

```
4/4 [==============================] - 1s 230ms/step
              precision    recall  f1-score   support

   cardboard       0.81      0.85      0.83        41
       glass       0.51      0.78      0.62        51
       metal       0.67      0.39      0.49        41
       paper       0.67      0.85      0.75        60
     plastic       0.81      0.51      0.62        49
       trash       0.33      0.07      0.12        14

    accuracy                           0.66       256
   macro avg       0.63      0.58      0.57       256
weighted avg       0.67      0.66      0.64       256
```

Рис. 37. Класифікаційний звіт натренованої моделі

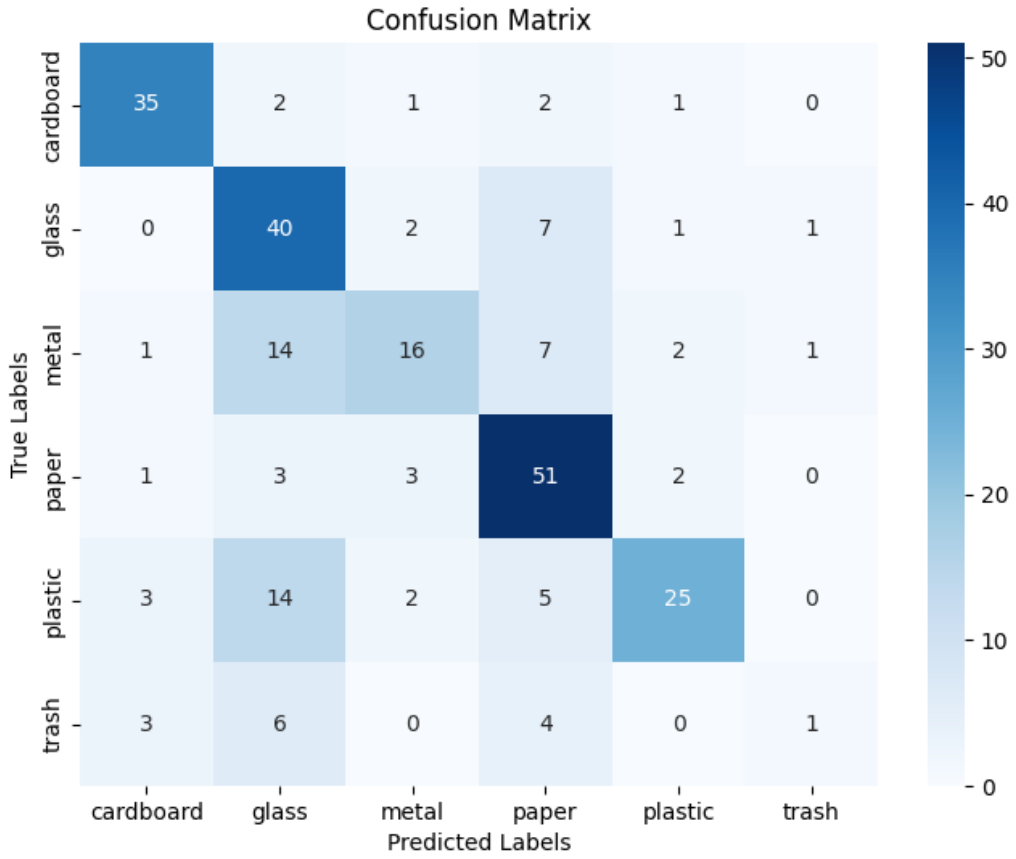

Рис. 38. Матриця плутанини натренованої моделі

На реальних зображеннях сміття, зроблених власноруч модель теж показала доволі погані результати. Однак, сам факт тестування на таких зображеннях наштовхнув на думку про інші техніки аугментації, як от аугментація із зміною фону зображення та додавання шумів, а також про комбінування різних технік аугментації.

Перед наступним тренуванням моделі нейронної мережі було здійснено декілька змін:

1) Реалізовано нові техніки аугментації для імітації умов навколишнього середовища;
2) Прийнято рішення вилучити деякі техніки аугментації (width та height shifts) та зменшити обсяги аугментації (зростання розміру датасету не у 19 разів, а в 14: тренувальний/валідаційний датасети розміром 26978/4816 зображень);
3) Для EarlyStopping callback параметр restore_best_weights встановити в True (рис. 39);
4) Під час тренування моделі врахувати незбалансованість датасету з допомогою ваг кожного класу сміття (рис. 40);
5) Зменшити відсоток відкинутих нейронів у Dropout шарі від 50 до 30.

```
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
my_callback = myCallback()
```

Рис. 39. Використані під час тренування моделі нейронної мережі колбеки

```
true_labels = train_generator.classes
class_weights = compute_class_weight('balanced',
                                     classes=np.unique(true_labels),
                                     y=true_labels)
class_weight_dict = dict(zip(np.unique(true_labels), class_weights))
```

Рис. 40. Обчислення ваг кожного класу сміття

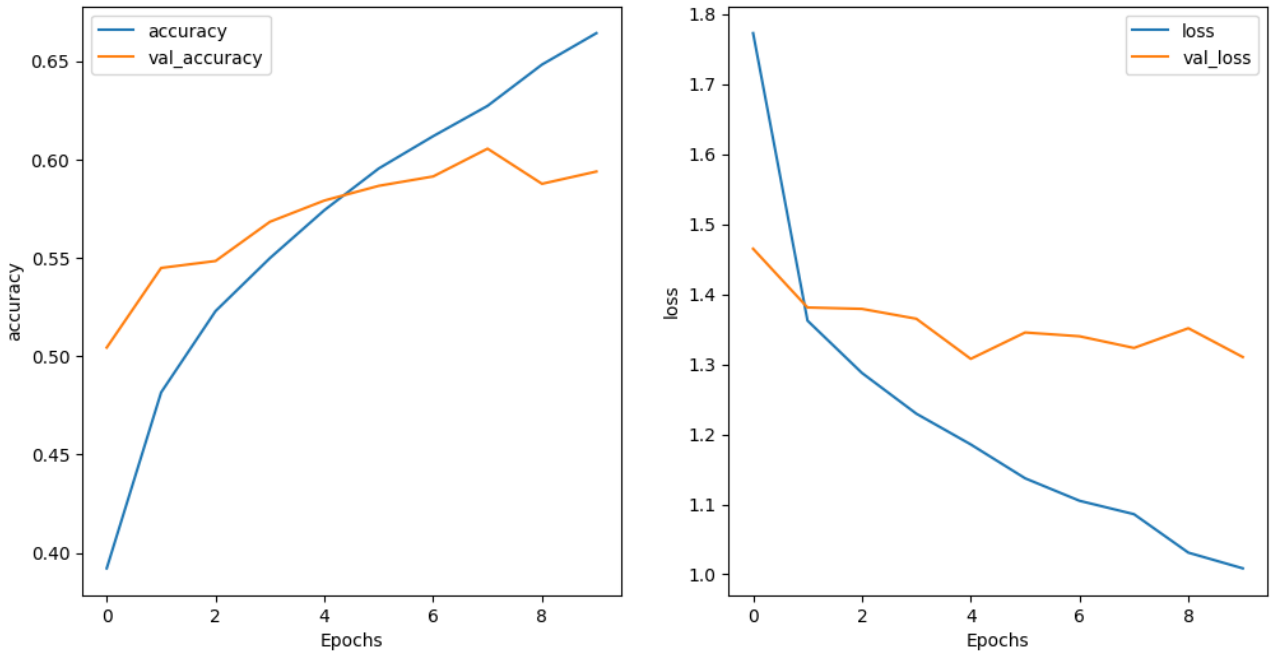

Рис. 41. Історія тренування вдосконаленої моделі

Із рис. 41 видно, що внесені зміни не допомогли уникнути процесу перенавчання моделі, тому наступним кроком стало використання Transfer Learning для вирішення завдання. Для цієї цілі було обрано модель ResNet152 - глибоку згорточну нейронну мережеву архітектуру, яка містить 152 рівні. ResNet152 спочатку навчався на великомасштабному наборі даних ImageNet,

що містить мільйони позначених зображень у тисячах класів. До базової моделі ResNet152 було додано декілька додаткових шарів (рис. 42), серед яких слід виділити шар BatchNormalization, який нормалізує вхідні дані рівня нейронної мережі шляхом масштабування та центрування активацій, стабілізації та прискорення процесу навчання.

Для чистоти експерименту модель resnet_152_1 (введено умовну назву моделі для зручності посилання на неї) тренувалася на тих самих зображеннях, що і вищезгадана згорткова нейронна мережа. Окрім того, для моделі, яка використовує ResNet152, також враховано незбалансованість датасету.

```
base_model = tf.keras.applications.resnet.ResNet152(input_shape=IMG_SHAPE,
                                        include_top=False,
                                        weights='imagenet')
model = tf.keras.models.Sequential()

base_model.trainable=False

model.add(base_model)

model.add(tf.keras.layers.GlobalAveragePooling2D())

model.add(tf.keras.layers.Dense(units=512, activation='relu'))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dropout(0.2))

model.add(tf.keras.layers.Dense(units=128, activation='relu'))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dropout(0.2))

model.add(tf.keras.layers.Dense(units=6, activation='softmax'))
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])

history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples/train_generator.batch_size ,
    epochs=30,
    validation_data=valid_generator,
    validation_steps=valid_generator.samples/valid_generator.batch_size,
    verbose=1,
    callbacks = [early_stopping_callback, my_callback],
    class_weight=class_weight_dict)
```

Рис. 42. Архітектура моделі resnet_152_1 нейронної мережі

```
Epoch 1/100
843/843 [==============================] - 289s 327ms/step - loss: 0.5051 - accuracy: 0.8197 - val_loss: 0.3527 - val_accur
acy: 0.8833
Epoch 2/100
843/843 [==============================] - 270s 320ms/step - loss: 0.1635 - accuracy: 0.9444 - val_loss: 0.3103 - val_accur
acy: 0.8951
Epoch 3/100
843/843 [==============================] - 270s 320ms/step - loss: 0.0902 - accuracy: 0.9713 - val_loss: 0.2824 - val_accur
acy: 0.9074
Epoch 4/100
843/843 [==============================] - 267s 317ms/step - loss: 0.0615 - accuracy: 0.9816 - val_loss: 0.2991 - val_accur
acy: 0.9111
Epoch 5/100
843/843 [==============================] - 264s 313ms/step - loss: 0.0435 - accuracy: 0.9869 - val_loss: 0.2976 - val_accur
acy: 0.9097
Epoch 6/100
844/843 [==============================] - ETA: 0s - loss: 0.0337 - accuracy: 0.9906
Reached 99% accuracy so cancelling training!
843/843 [==============================] - 265s 315ms/step - loss: 0.0337 - accuracy: 0.9906 - val_loss: 0.3286 - val_accur
acy: 0.9068
```

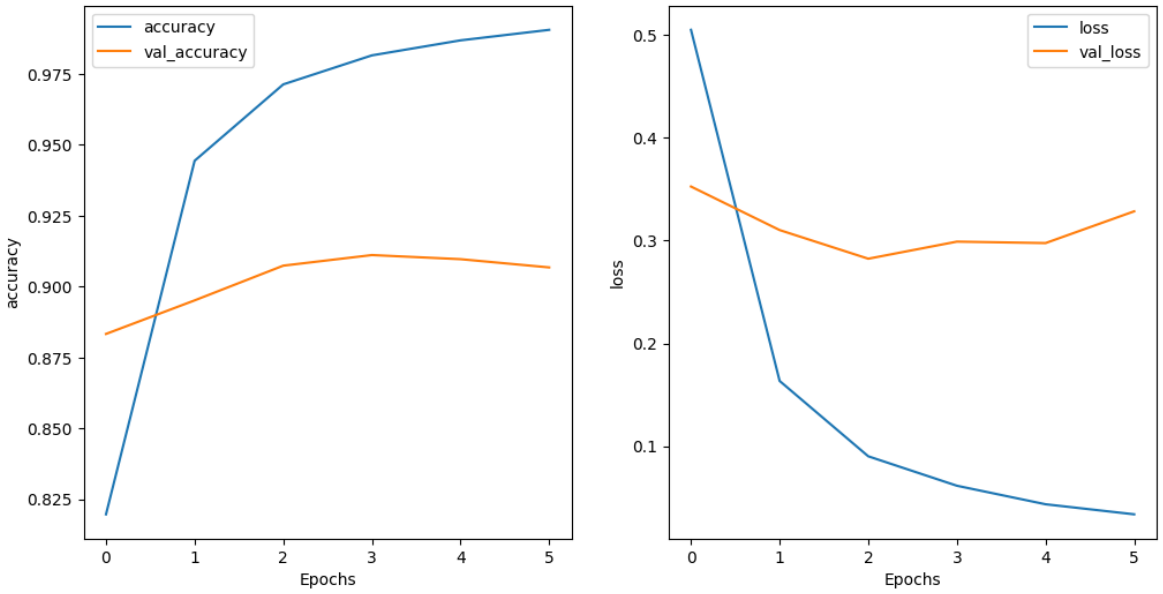

Рис. 43. Історія тренування моделі resnet_152_1

Із рис. 43 видно, що процес тренування моделі resnet_152_1 було перервано колбеком my_callback внаслідок досягнення моделлю на 6-ій епосі тренувальної точності більше 99%. Водночас бачимо, що після 2-ої епохи валідаційні втрати (validation loss) почали зростати, а валідаційна точність становила приблизно 90-91%.

На тестових даних модель також показала доволі хороші результати (рис. 44-45).

	precision	recall	f1-score	support
cardboard	1.00	0.90	0.95	41
glass	0.92	0.94	0.93	51
metal	0.88	0.88	0.88	41
paper	0.94	0.97	0.95	60
plastic	0.98	0.92	0.95	49
trash	0.78	1.00	0.88	14
accuracy			0.93	256
macro avg	0.92	0.93	0.92	256
weighted avg	0.93	0.93	0.93	256

Рис. 44. Класифікаційний звіт натренованої моделі resnet_152_1

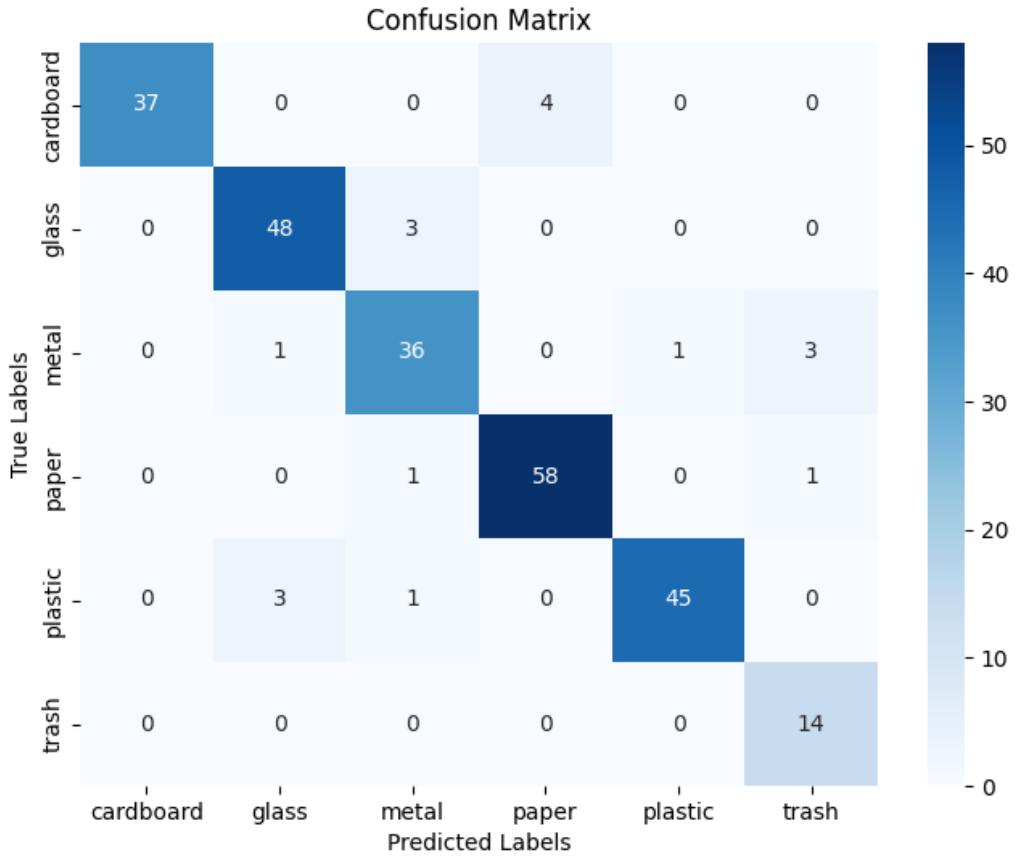

Рис. 45. Матриця плутанини моделі resnet_152_1

Розглянувши матрицю плутанини, можна побачити, що модель іноді плутала класи glass і plastic, metal і glass, paper і cardboard, trash і metal. Однак така поведінка є доволі очікуваною, враховуючи, що деякі зображення, які містять обидва класи кожної із перерахованих пар, доволі схожі (рис. 46-49).

Рис. 46. Схожі зображення класів glass і plastic

Рис. 47. Схожі зображення класів glass і metal

Рис. 48. Схожі зображення класів cardboard і paper

Рис. 49. Схожі зображення класів metal і trash

Розглянувши історію тренування моделі resnet_152_1 (рис. 43) було зроблено висновок, що краще встановити меншу точність для припинення тренування у класі myCallback, а потім за необхідності дотренувати модель.

Нарешті, було зроблено ще одну спробу натренувати модель resnet_152_2, яка використовує ResNet152, зменшивши обсяг тренувального і валідаційного датасетів до 19460/3471 (збільшення розміру оригінальних дата

сетів приблизно в 10 разів) (рис. 50) та точність, при якій припиняється тренування моделі до 98%.

```
target_size=(224, 224)

datagen = ImageDataGenerator(preprocessing_function=preprocess_input)
train_generator = datagen.flow_from_directory(directory=train_dir_path,
                                              batch_size=32,
                                              class_mode='categorical',
                                              target_size=target_size)

valid_generator = datagen.flow_from_directory(directory=valid_dir_path,
                                              batch_size=32,
                                              class_mode='categorical',
                                              target_size=target_size)
```

```
Found 19460 images belonging to 6 classes.
Found 3471 images belonging to 6 classes.
```

Рис. 50. Розміри тренувального та валідаційного датасетів

```
Epoch 1/30
608/608 [==============================] - 217s 316ms/step - loss: 0.5834 - accuracy: 0.7955 - val_loss: 0.3741 - val_accur
acy: 0.8724
Epoch 2/30
608/608 [==============================] - 189s 311ms/step - loss: 0.2032 - accuracy: 0.9305 - val_loss: 0.3145 - val_accur
acy: 0.8908
Epoch 3/30
608/608 [==============================] - 203s 334ms/step - loss: 0.1213 - accuracy: 0.9605 - val_loss: 0.2909 - val_accur
acy: 0.9078
Epoch 4/30
608/608 [==============================] - 190s 312ms/step - loss: 0.0863 - accuracy: 0.9738 - val_loss: 0.2878 - val_accur
acy: 0.9104
Epoch 5/30
609/608 [==============================] - ETA: 0s - loss: 0.0601 - accuracy: 0.9814
Reached 98% accuracy so cancelling training!
608/608 [==============================] - 189s 311ms/step - loss: 0.0601 - accuracy: 0.9814 - val_loss: 0.2947 - val_accur
acy: 0.9110
```

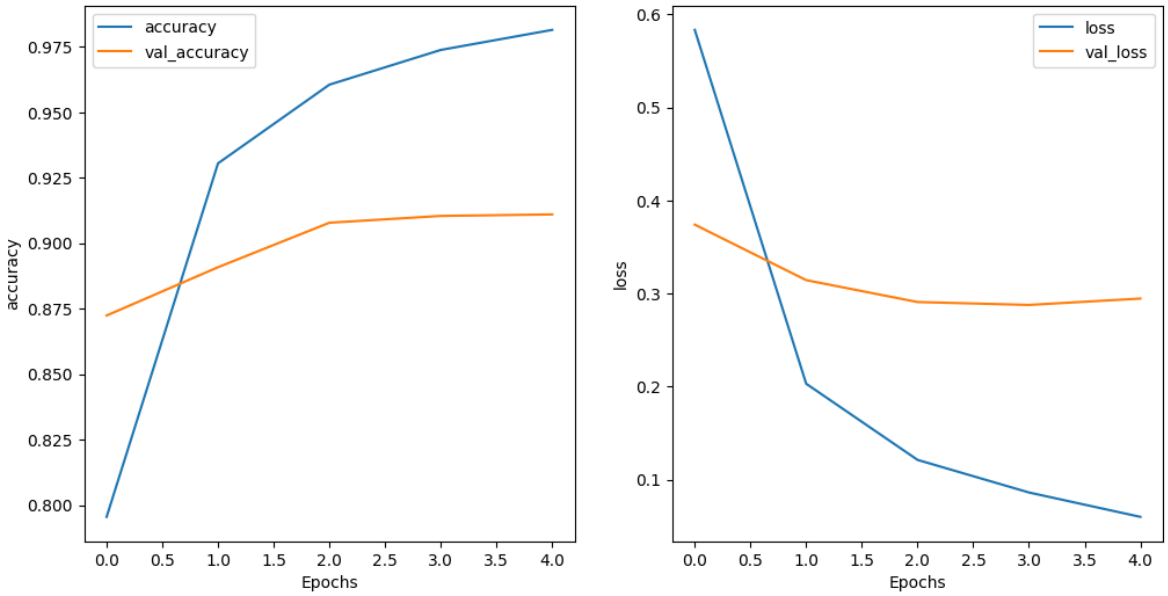

Рис. 51. Історія тренування моделі resnet_152_2

	precision	recall	f1-score	support
cardboard	1.00	0.93	0.96	41
glass	0.84	0.96	0.90	51
metal	0.88	0.85	0.86	41
paper	0.93	0.93	0.93	60
plastic	0.96	0.92	0.94	49
trash	0.77	0.71	0.74	14
accuracy			0.91	256
macro avg	0.90	0.88	0.89	256
weighted avg	0.91	0.91	0.91	256

Рис. 52. Класифікаційний звіт натренованої моделі resnet_152_2

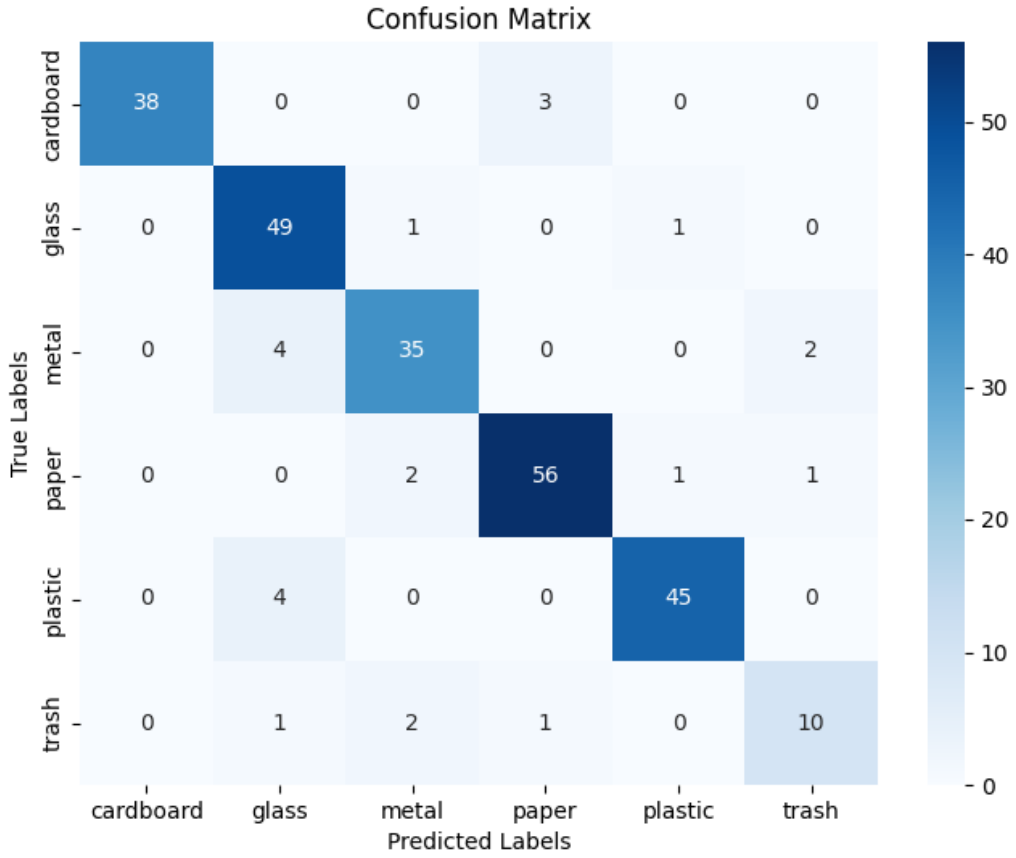

Рис. 53. Матриця плутанини моделі resnet_152_2

Порівнявши класифікаційні звіти моделей resnet_152_1 (рис. 44) і resnet_152_2 (рис. 52) зроблено висновок, що перша модель показала кращі результати, ніж друга. Тому було прийнято рішення спробувати дотренувати модель resnet_152_2 до точності 99% і зберегти отриманий результат як модель resnet_152_3 (рис. 54).

```
Epoch 1/30
608/608 [==============================] - 187s 308ms/step - loss: 0.0421 - accuracy: 0.9886 - val_loss: 0.2946 - val_accur
acy: 0.9153
Epoch 2/30
608/608 [==============================] - 190s 312ms/step - loss: 0.0405 - accuracy: 0.9876 - val_loss: 0.2942 - val_accur
acy: 0.9179
Epoch 3/30
609/608 [==============================] - ETA: 0s - loss: 0.0334 - accuracy: 0.9904
Reached 99% accuracy so cancelling training!
608/608 [==============================] - 189s 311ms/step - loss: 0.0334 - accuracy: 0.9904 - val_loss: 0.3137 - val_accur
acy: 0.9139
```

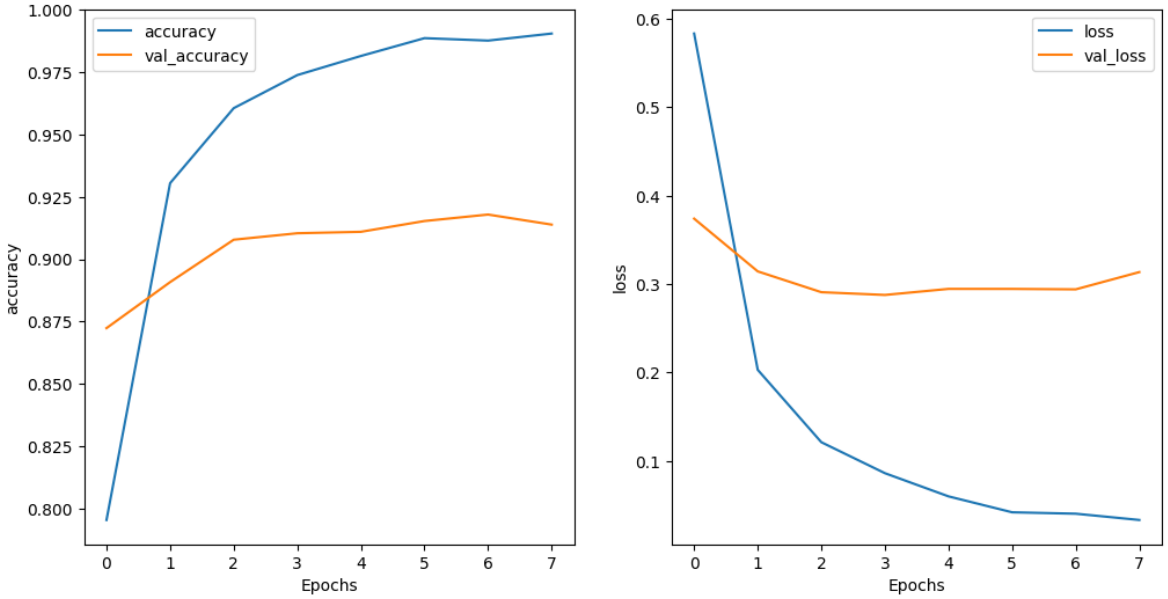

Рис. 54. Історія тренування моделі resnet_152_3 (дотреновування моделі resnet_152_2 до точності 99%)

	precision	recall	f1-score	support
cardboard	1.00	0.93	0.96	41
glass	0.85	0.98	0.91	51
metal	0.92	0.83	0.87	41
paper	0.95	0.95	0.95	60
plastic	0.98	0.88	0.92	49
trash	0.72	0.93	0.81	14
accuracy			0.92	256
macro avg	0.90	0.92	0.91	256
weighted avg	0.93	0.92	0.92	256

Рис. 55. Класифікаційний звіт натренованої моделі resnet_152_3

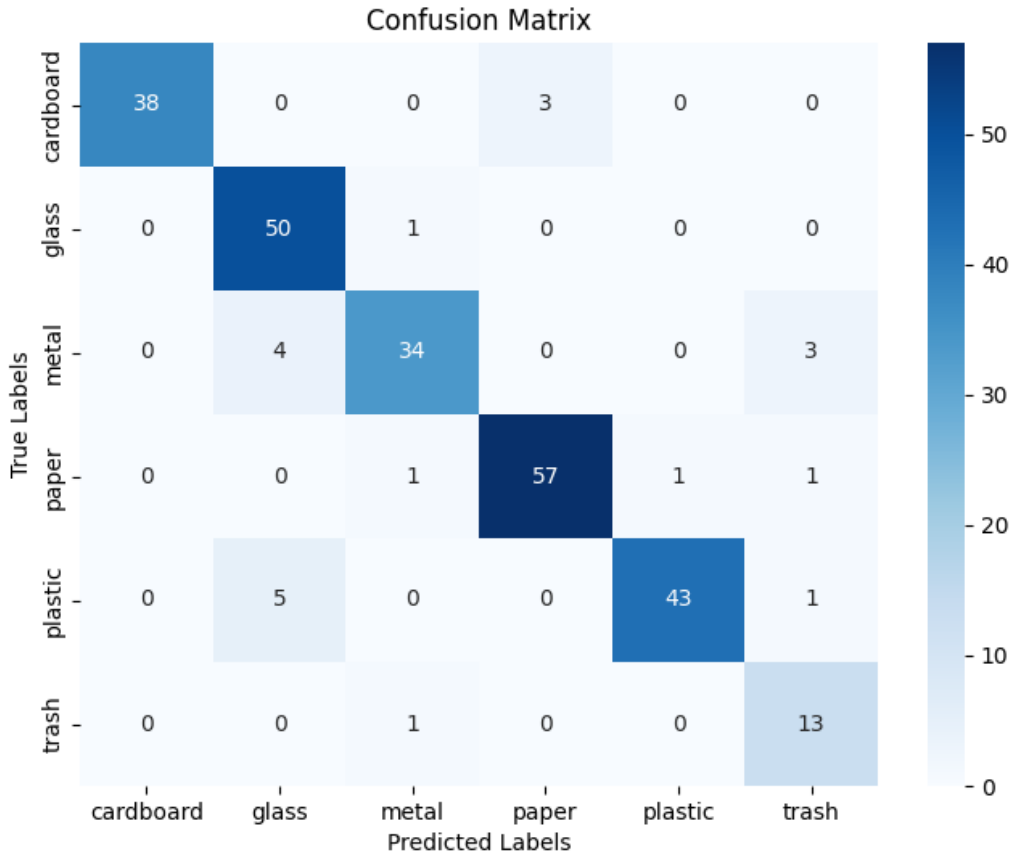

Рис. 56. Матриця плутанини моделі resnet_152_3

Порівнявши класифікаційні звіти моделей resnet_152_1 (рис. 44) і resnet_152_3 (рис. 55) зроблено висновок, що перша модель показала кращі результати, ніж друга.

Однак вирішальним у прийнятті рішення стосовно того, яку модель використовувати у демонстраційному Streamlit-застосунку, стало тестування трьох моделей (resnet_152_1, resnet_152_2 і resnet_152_3) на реальних зображеннях сміття, зроблених на власний смартфон.

Етап 4: Тестування моделей на самостійно зроблених зображеннях сміття

Для перевірки здатності моделі правильно класифікувати різні види сміття було зібрано власний тестовий міні-датасет, який містить 55 зображень різних видів сміття (рис. 57).

Кожну із трьох моделей (resnet_152_1, resnet_152_2 і resnet_152_3) було протестовано на отриманому датасеті; отримані результати зображено на рис. 58-59.

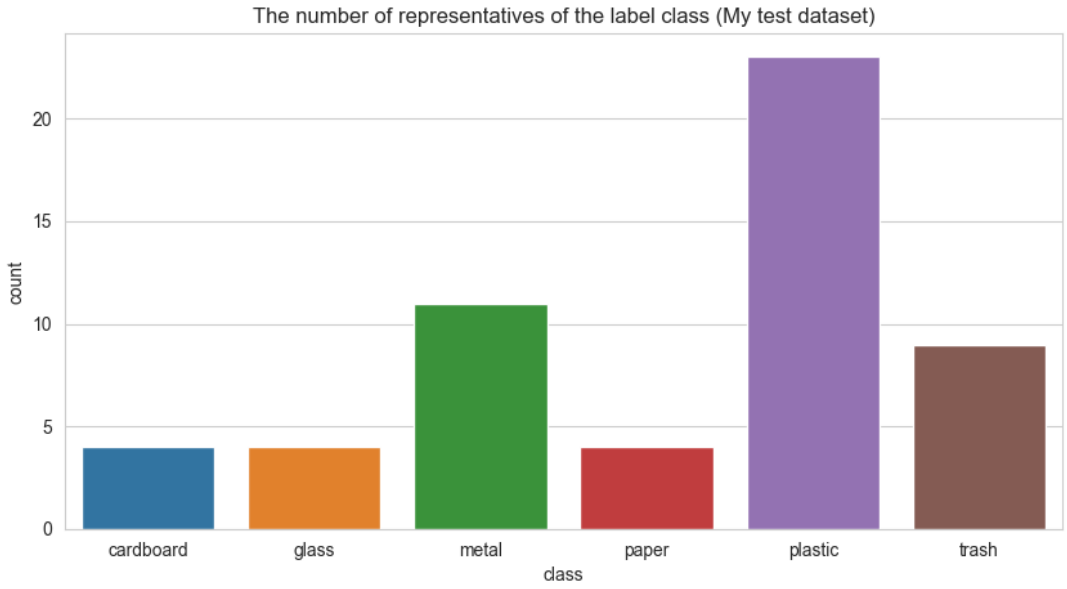

Рис. 57. Аналіз розподілу класів зібраного власноруч тестового датасету

	precision	recall	f1-score	support
cardboard	0.44	1.00	0.62	4
glass	0.50	0.25	0.33	4
metal	0.57	0.73	0.64	11
paper	0.40	1.00	0.57	4
plastic	0.93	0.61	0.74	23
trash	0.60	0.33	0.43	9
accuracy			0.62	55
macro avg	0.57	0.65	0.55	55
weighted avg	0.70	0.62	0.62	55

	precision	recall	f1-score	support
cardboard	0.57	1.00	0.73	4
glass	0.22	0.50	0.31	4
metal	0.67	0.36	0.47	11
paper	0.25	0.50	0.33	4
plastic	0.94	0.65	0.77	23
trash	0.44	0.44	0.44	9
accuracy			0.56	55
macro avg	0.52	0.58	0.51	55
weighted avg	0.67	0.56	0.59	55

	precision	recall	f1-score	support
cardboard	0.36	1.00	0.53	4
glass	0.20	0.25	0.22	4
metal	0.71	0.45	0.56	11
paper	0.25	0.75	0.38	4
plastic	1.00	0.65	0.79	23
trash	0.60	0.33	0.43	9
accuracy			0.56	55
macro avg	0.52	0.57	0.48	55
weighted avg	0.72	0.56	0.59	55

Рис. 58. Класифікаційні звіти моделей (resnet_152_1 - зліва, resnet_152_2 – посередині і resnet_152_3 - справа) під час їх тестування на власному міні-датасеті

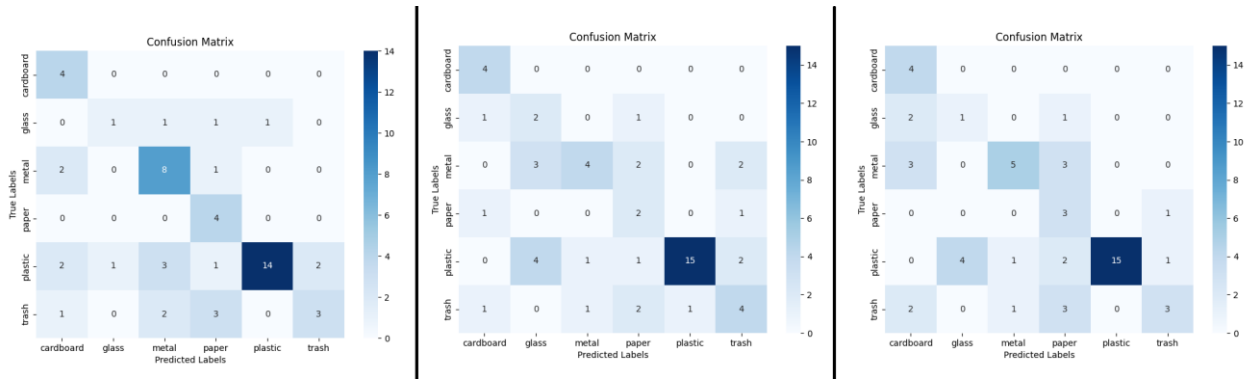

Рис. 59. Матриці плутанини моделей (resnet_152_1 - зліва, resnet_152_2 – посередині і resnet_152_3 - справа) під час їх тестування на власному міні-датасеті

Отримані результати дозволяють стверджувати, що найкращою є модель resnet_152_1, тому для демонстрації її роботи наведено декілька зображень із тестового датасету, які вона класифікувала правильно (рис. 60-61) і ті, на яких модель помилилася (рис. 62). Варто зазначити, що заголовок кожного зображення містить усю необхідну інформацію:

- Зліва (до символа ‘:’) – назва файла, яка відображає клас сміття;

- Справа (після символа ‘:’) – прогнозований моделлю клас сміття + у дужках наведена максимальна впевненість моделі, тобто найвищий показник впевненості моделі серед прогнозованих класів.

Рис. 60. Правильно класифіковані моделлю resnet_152_1 зображення сміття

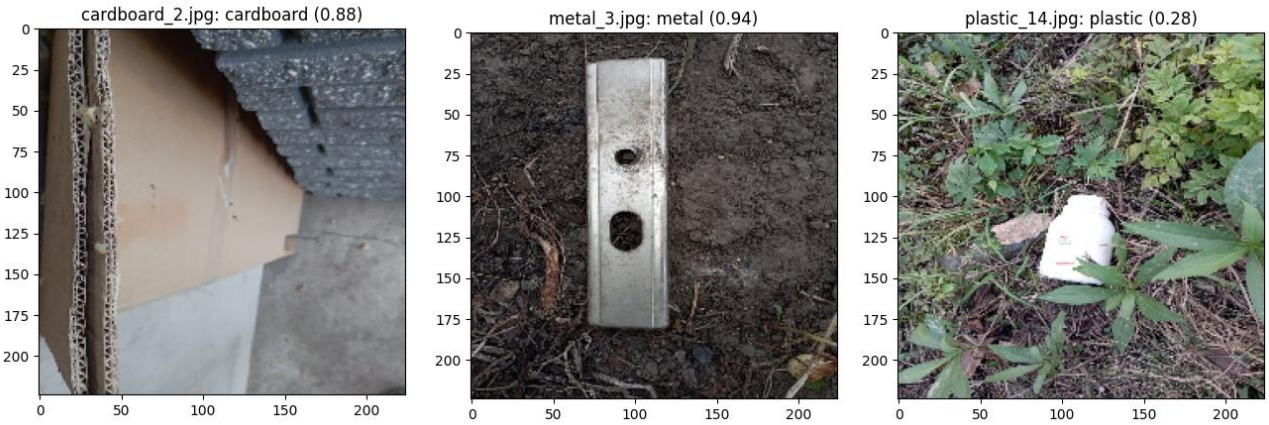

Рис. 61. “Складніші” зображення сміття, які модель resnet_152_1 змогла правильно класифікувати

Рис. 62. Неправильно класифіковані моделлю resnet_152_1 зображення сміття

Етап 5: Створення демонстраційного варіанту використання моделі у вигляді Streamlit-застосунку

Для демонстрації одного із можливих варіантів застосування натренованої моделі було створено Streamlit-застосунок (рис. 63), який надає користувачу такі можливості:

1) Завантажити будь-яке зображення сміття із власного пристрою та переглянути результати прогнозування класу сміття з допомогою натренованої моделі resnet_152_1 (рис. 64);
2) Переглянути приклади зображень кожного із шести класів сміття – наведено по 3 зображення для кожного класу, взяті із досліджуваного датасету Garbage Classification Dataset (Kaggle) (рис. 65).

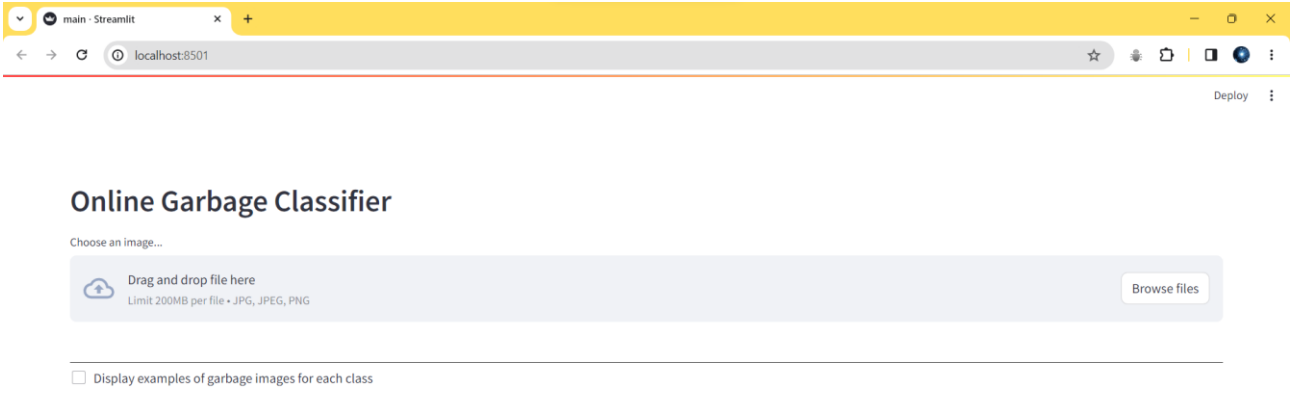

Рис. 63. Вигляд головної сторінки Streamlit-застосунку

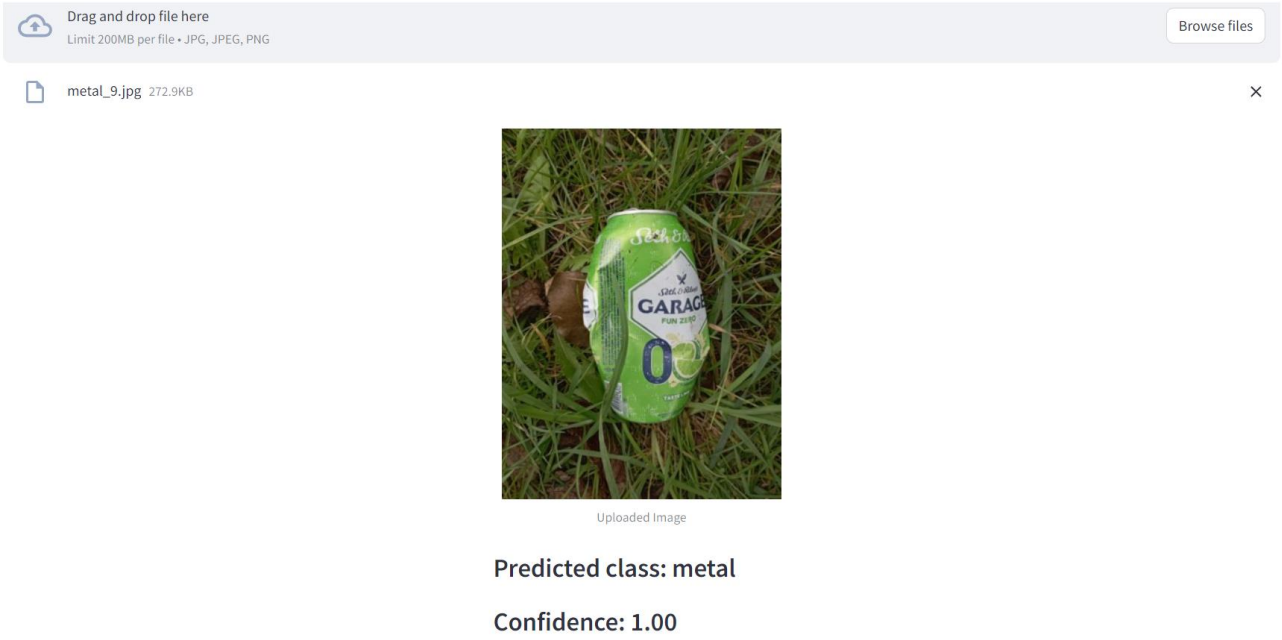

Рис. 64. Відображення прогнозованого моделлю класу сміття для завантаженого користувачем зображення

Рис. 65. Можливість переглянути класичних представників кожного класу сміття

Висновки

В ході реалізації завдання було натреновано три моделі нейронної мережі (resnet_152_1, resnet_152_2, resnet_152_3), побудовані на основі архітектури ResNet152 з метою покращення виділення ознак і підвищення точності класифікації. Для тренування використовувався датасет Garbage Classification Dataset (Kaggle), який містить 2527 зображень сміття. Датасет було поділено на тренувальний та тестувальний у співвідношенні 90/10, а тренувальний у свою чергу – на тренувальний і валідаційний у співвідношенні 85/15. Під час тренування моделей було враховано незбалансованість використаного датасету з допомогою ваг класів; з допомогою використання різних технік аугментації

імітовано різні положення сміття на зображеннях, умови освітлення та навколишнього середовища.

Найкращі результати продемонструвала модель resnet_152_1:

- Тренувальна точність становить 99%;
- Валідаційна точність – 91%;
- Точність на тестових даних із досліджуваного датасету – 93%;
- Точність на тестових даних, зібраних власноруч – 62%.

Розглянувши отримані результати, можна стверджувати, що виконано всі критерії валідації.

Варто зазначити, що точність на тестових даних не відображає цілком можливість натренованої моделі класифікувати зображення сміття (оскільки згаданий тестовий датасет є незбалансованим) і його було використано для того, щоб оцінити здатність навченої моделі до узагальнення та її продуктивність у сценаріях реального світу, забезпечуючи таким чином практичну оцінку за межами навчального набору даних.

Для демонстрації одного із можливих варіантів застосування натренованої моделі було створено Streamlit-застосунок, використовуючи який користувач може завантажити будь-яке зображення сміття із власного пристрою та переглянути результати прогнозування класу сміття з допомогою натренованої моделі resnet_152_1.

Майбутні плани та напрямки розвитку проекту:

1) Використання для тренування моделі не тільки архітектури ResNet152, але й інші архітектури для Transfer Learning (наприклад, InceptionV3, MobileNetV2) з метою отримання кращих результатів та розширення потенційної області використання натренованих моделей;
2) Тренування нових моделей нейронної мережі на інших датасетах з метою розширення кількості класів сміття;
3) Об'єднання результатів класифікації кількох моделей за допомогою ансамблевих методів з метою отримати кращі результати.
4) Балансування досліджуваного датасету (наприклад, доповнення класу trash зображеннями з мережі Інтернет) та здійснення ще декількох спроб отримати кращі результати;
5) Створення повноцінного вебзастосунку з допомогою HTML, CSS та Vue.js для покращення досвіду користувача (User Experience) та розширення можливостей програми.