Розпізнавання невдалого друку за допомогою нейроних мереж

Кирило Байбула

22 листопада 2024 р.

Анотація

Дана робота розглядає методи рішення проблеми детекції невдалих 3д друків FDM принтерів. Розглядається глибоке навчання моделі, що за допомогою методів ком'ютерного зору знаходить потенційно небезпечні для процесу дефекти. Також розглядається проблема мінімізації ком'ютерних ресурсів необхідних для використання моделі на практиці для інтегрування моделі в існуючі моделі принтерів.

Зміст

1 Вступ

FDM 3д принтери — це принтери, що використовують технологію пошарового вичавлювання пластику шар за шаром по визначеному заздалегіть контору. Хоча й технологія вже існує деякий час і в сфері виходять нові інновації як у сенсі заліза так і програмного забезпечення, проте стабільність якості, особливо у домінуючому сегменті

любительских принтерів, колосально варіюєця від виробника до воробника, від пластику до пластику. Саме тому дана робота розглядає спосіб як економії матеріалів, так і особистої безпеки у критичних ситуаціях досліджуючи питання простої універсальної швидкої детекції "невдалих" друків на єтапі їх утворення.

1.1 Що таке "невдалий" друк

У данній роботі "невдалими", або "спагетті" друками будуть вважатися такі, що утворюють нитко подібні шари пластика, що не злиплися із основною моделлю, через що далі продовжувати роботу не є можливим (див Рисунок 1). На практиці, при достатній кількості навчальних данних можливо знаходити і інші типи проблем, проте для спрощення вирішено ідентифікувати не тип проблеми, а саме неможоливість подальшого друку.

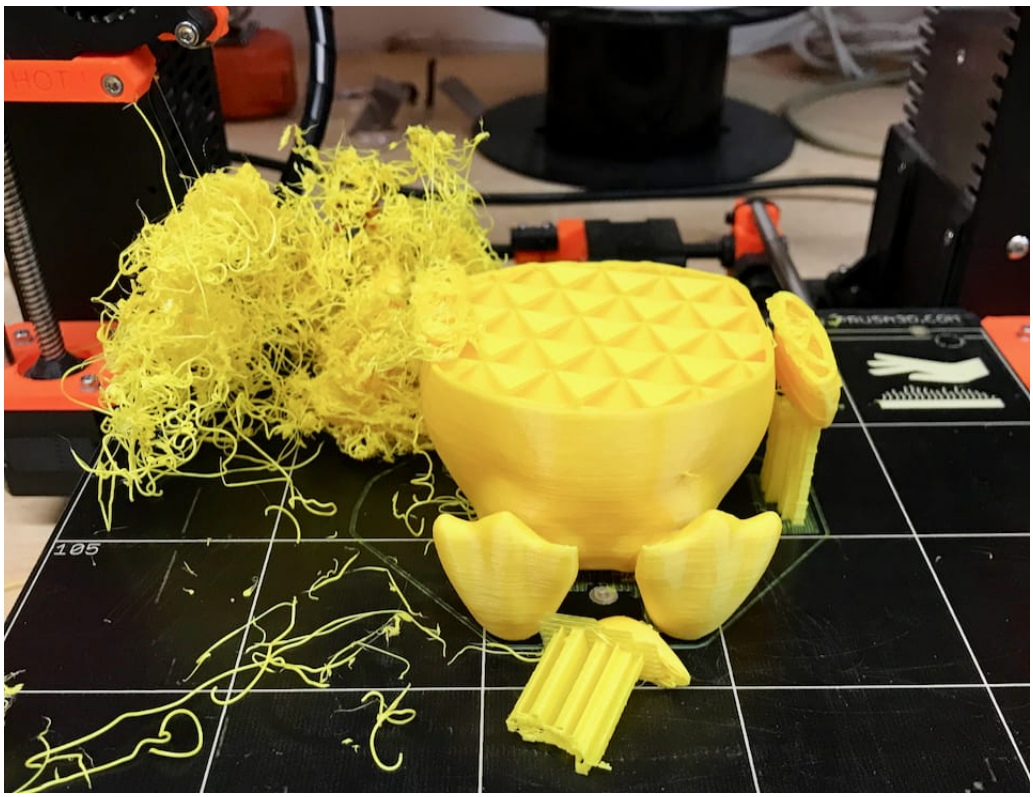

Рис. 1: Приклад невдалого друку

1.2 Існуючі рішення

На данний момент існує невелика кількість саме /відкритих/ рішень, так як деякі підприємці використовують подібну модель як зовнішній сервіс, що з одного боку дозво-

ляє розвантажити і так обмежену кількість ресурсів, проте з іншого змушує віддавати доступ до камери комусь ззовні. Саме тому пропонується спочатку розглянути саме відкриті рішення.

Один з користувачів сайту “Hugging Face” виклав дотренову модель [7] іншої нейроної мережі YOLOv5 [1]. До самої моделі він також виклав невеличкий тренувальний датасет [6] на 73 зображення із виділеними соплом принтера, частиною що друкується, та “спагетті”, тобто місце що можна вважати потенційно “невдалим” (див. Рисунок 2).

Також вартими уваги є: [5], [4], [2] та [3].

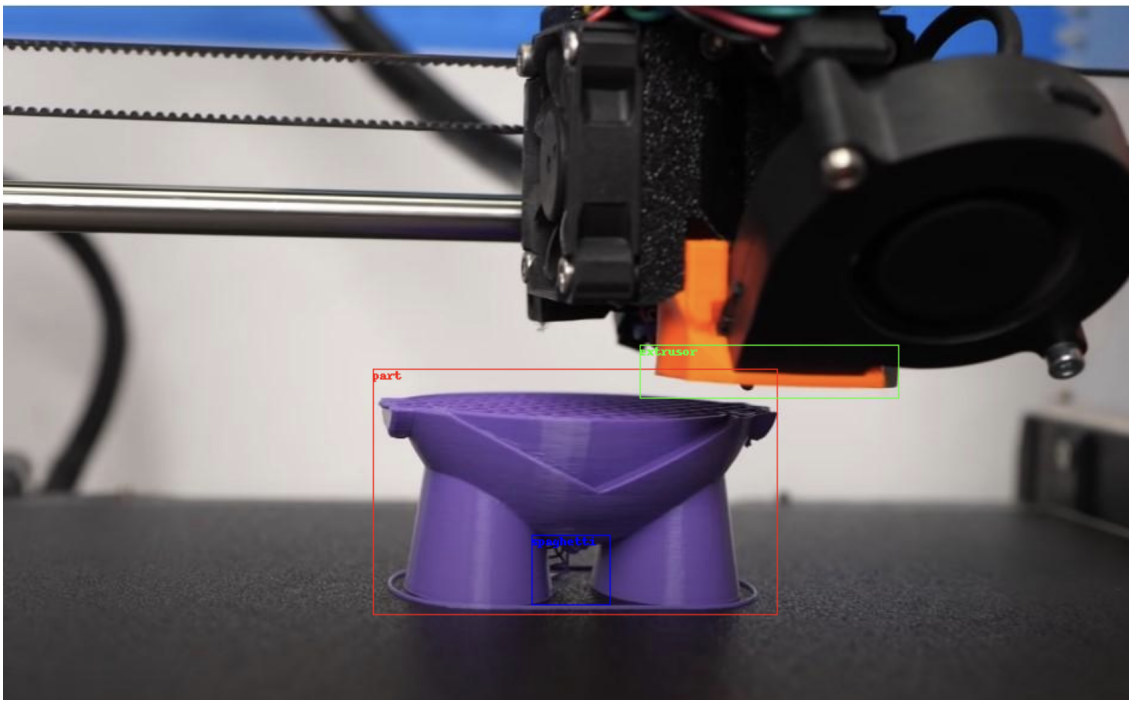

Рис. 2: Приклад результату роботи детекції від Javier Villarroel

2 Огляд датасету

Для задачі глибокого навчання було обрано суміш датасетів з Kaggle різного наповнення вигляду як на рисунку 3 розподіленого у два класи “спагетті” та “ок”.

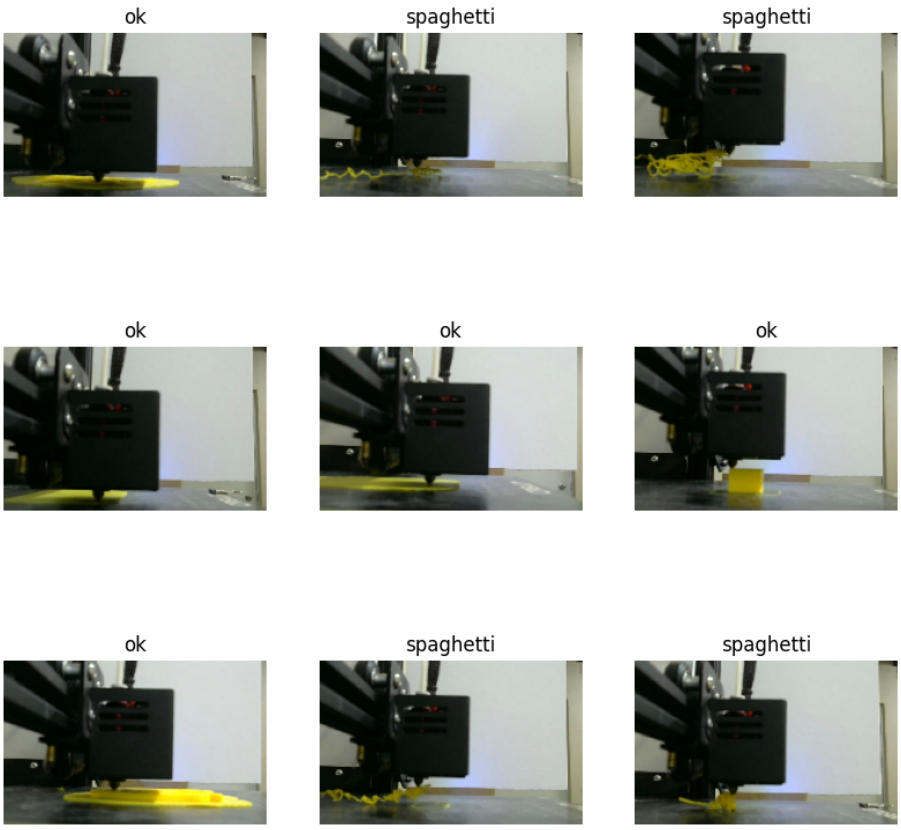

Рис. 3: Приклад зображень датасету з маркуваннями

Так як датасет не збалансований, проте різниця по кількості у кожному класі незначна, було застосовано “undersampling” із прибиранням із класу “ok” частини зображень. Таким чином на кожен клас було виділено до 700 зображень.

2.1 Аугментація зображень

Для запобігання перенавчання моделі, було застовано преобробку зображень за допомогою вбудованих методів бібліотеки `tensorflow`:

```
tf.keras.Sequential([
    tf.keras.layers.RandomFlip('horizontal'),
    tf.keras.layers.RandomRotation(0.2),
```

```
    tf.keras.layers.GaussianNoise(0.1),
])
```

що послідовно застосовує випадкове відзеркалення по горизонталі, випадковий поворот зображення та Гаусівський шум поверх зображення відповіно. Розмір вхідних данних 120 на 192 RGB пікселя.

3 Мала модель: нейрона мережа з двома щільними слоями

Для малої модель було вирішено застосувати просту нейрону мережу з двома щільними слоями, рисунок **??**. Кількість нейронів в слої обмежена розміром вагів у 8 мегабайтів, так як кінцева НН буде запускатися на вбудованих системах та мікроком'ютерах.

```
inputs = tf.keras.Input(shape=(img_height, img_width, 3))
x = data_augmenter(inputs)
x = tf.keras.layers.Flatten(input_shape=(img_height, img_width, 3))(inputs)
x = tf.keras.layers.Dense(32, activation='relu', kernel_initializer='HeNormal')(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = tf.keras.layers.Dense(1, activation='sigmoid', kernel_initializer='HeNormal')(x)
```

Поглянувши на історію тренування, рисунок 4, моделі можна побачити, що точність і loss не збігаться із єпохами, то очевидно, що даної складності моделі категорично недостатньо задля використання для данної задачі.

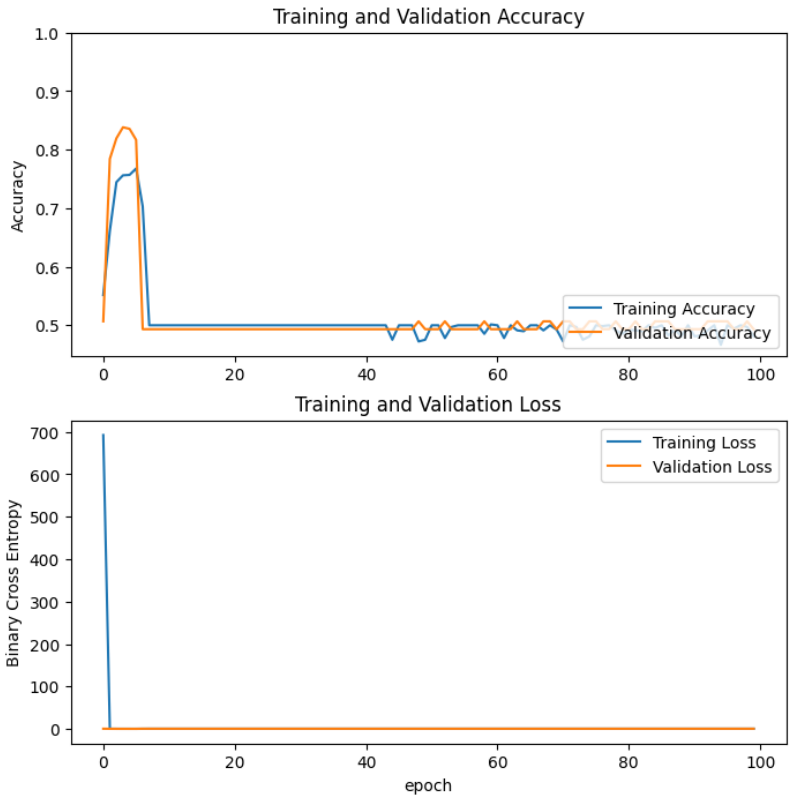

Рис. 4: Історія тренування для малої моделі

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 120, 192, 3)]     0

 flatten_1 (Flatten)         (None, 69120)             0

 dense_2 (Dense)             (None, 32)                2211872

 dropout_1 (Dropout)         (None, 32)                0

 dense_3 (Dense)             (None, 1)                 33

=================================================================
Total params: 2211905 (8.44 MB)
Trainable params: 2211905 (8.44 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

4 Середня модель: Класична згорткова нейрона мережа

Для середньої моделі було застосовано класичну архітектуру згорткової мережі — послідовність згорткових слоїв із "maxpool" слоєм, де кількість фільтрів послідовно збільшується та закінчується щільними слоями.

```
inputs = tf.keras.Input(shape=(img_height, img_width, 3))
x = data_augmenter(inputs)
x = tf.keras.layers.Conv2D(32, (3, 3), activation='relu')(x)
x = tf.keras.layers.MaxPooling2D((2, 2))(x)
x = tf.keras.layers.Conv2D(64, (3, 3), activation='relu')(x)
x = tf.keras.layers.MaxPooling2D((2, 2))(x)
x = tf.keras.layers.Conv2D(128, (3, 3), activation='relu')(x)
x = tf.keras.layers.MaxPooling2D((2, 2))(x)
x = tf.keras.layers.Conv2D(256, (3, 3), activation='relu')(x)
x = tf.keras.layers.MaxPooling2D((2, 2))(x)
x = tf.keras.layers.Flatten()(x)
x = tf.keras.layers.Dense(128, activation='relu', kernel_initializer='HeNormal')(x)
x = tf.keras.layers.Dropout(0.5)(x)
x = tf.keras.layers.Dense(64, activation='relu', kernel_initializer='HeNormal')(x)
x = tf.keras.layers.Dropout(0.3)(x)
outputs = tf.keras.layers.Dense(1, activation='sigmoid', kernel_initializer='HeNormal')(x)
```

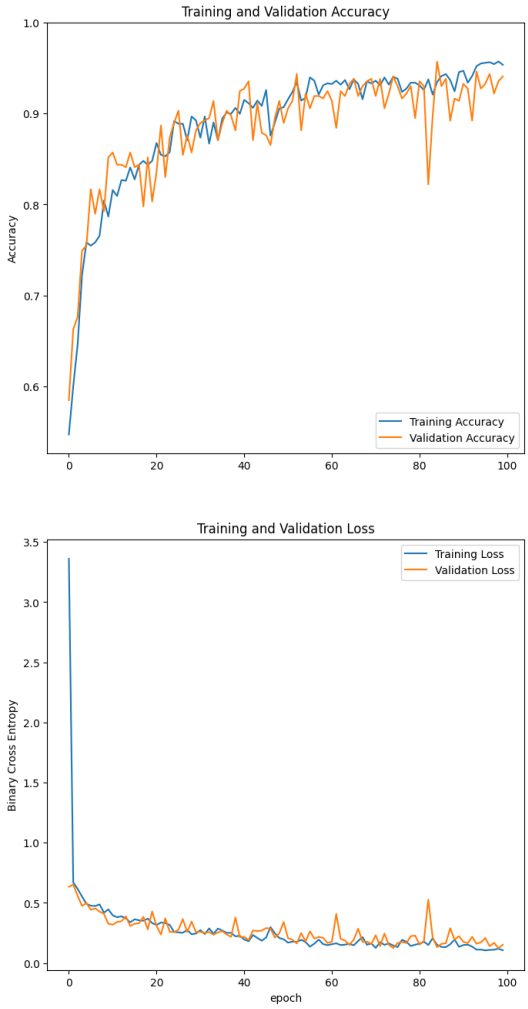

Рис. 5: Історія тренування для середньої моделі

По історії тренування, рисунок 5, можна побачити, що показники збігаються, отже модель навчається, проте показники F1 все ще не ідеальні, див. рисунок 6.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 120, 192, 3)]     0

 sequential (Sequential)     (None, 120, 192, 3)       0

 conv2d (Conv2D)             (None, 118, 190, 32)      896
```

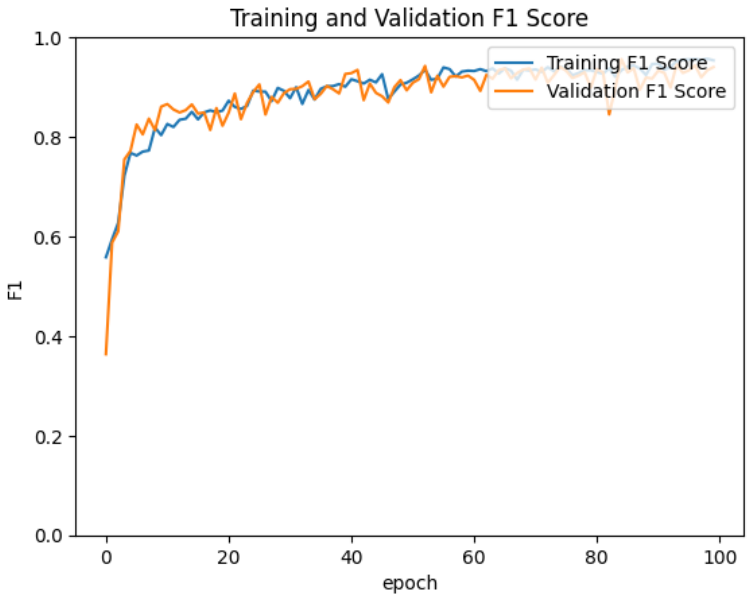

Рис. 6: F1 показники середньої моделі

max_pooling2d (MaxPooling2D)	(None, 59, 95, 32)	0
conv2d_1 (Conv2D)	(None, 57, 93, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 28, 46, 64)	0
conv2d_2 (Conv2D)	(None, 26, 44, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(None, 13, 22, 128)	0
conv2d_3 (Conv2D)	(None, 11, 20, 256)	295168
max_pooling2d_3 (MaxPooling2D)	(None, 5, 10, 256)	0
flatten_2 (Flatten)	(None, 12800)	0
dense_4 (Dense)	(None, 128)	1638528

```
 dropout_2 (Dropout)         (None, 128)               0

 dense_5 (Dense)             (None, 64)                8256

 dropout_3 (Dropout)         (None, 64)                0

 dense_6 (Dense)             (None, 1)                 65

=================================================================
Total params: 2035265 (7.76 MB)
Trainable params: 2035265 (7.76 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

5 Велика модель: MobileNetV3 fine-tuning з додатковими слоями

Для великої моделі було вирішено застосувати і модифікувати архітектуру існуючої моделі — `MobileNetV3`, із існуючими до неї вагами "ImageNet" та додаванням одного щільного слоя, як в малій моделі, та `GlobalMaxPooing2D`.

Варто зауважити, що `MobileNetV3` працює з існуючими вагами "ImageNet" в `tensorflow` лише з розміром зображення у 224 на 224 пікселя. І хоча роздільна здатність стала більшою, проте кількість параметрів сильно меньше ніж у інших архітектур і зберігає обмеження.

```
base_model = tf.keras.applications.MobileNetV3Small(
    input_shape=(img_height, img_width, 3),
    include_top=False,
)

inputs = tf.keras.Input(shape=(img_height, img_width, 3))
x = data_augmenter(inputs)
x = tf.keras.applications.mobilenet_v3.preprocess_input(x)
x = base_model(x, training=True)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Dense(64, activation='relu', kernel_initializer='HeNormal')(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = tf.keras.layers.Dense(1, activation='sigmoid', kernel_initializer='HeNormal')(x)
```

На рисунку 7 зображена історія навчання. Всі показники сильно збільшилися у кращу сторону, а loss впав майже на порядок.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_7 (InputLayer)        [(None, 120, 192, 3)]     0

 sequential (Sequential)     (None, 120, 192, 3)       0

 MobilenetV3small (Function  (None, 4, 6, 576)         939120
 al)

 global_average_pooling2d_2  (None, 576)               0
  (GlobalAveragePooling2D)

 dense_5 (Dense)             (None, 64)                36928
```

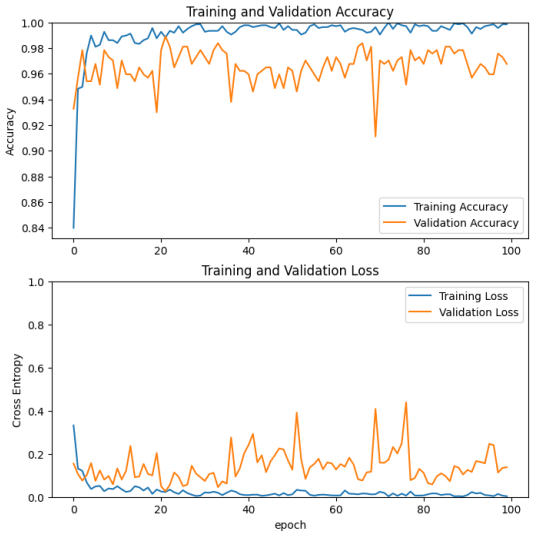

Рис. 7: Історія тренування для великої моделі

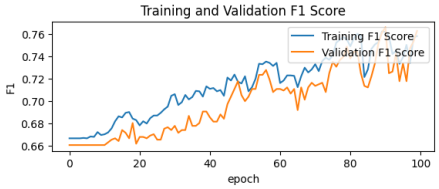

Рис. 8: Історія тренування для великої моделі F1

```
 dropout_3 (Dropout)         (None, 64)                0

 dense_6 (Dense)             (None, 1)                 65

=================================================================
Total params: 976113 (3.72 MB)
Trainable params: 964001 (3.68 MB)
Non-trainable params: 12112 (47.31 KB)
_________________________________________________________________
```

6 Результати

Результати метрик від малої до великої моделі можна побачити у таблиці 9.

Висоновок: `MobileNetV3` — чудова модель, що підходь для даної задачі при цьому зберігаючи оптимізовану для мобільних систем архітектуру.

	Loss	Precision	Recall	F1	Accuracy
0	0.693166	0.493261	1.000000	0.66064984	0.493261
1	0.153329	0.935135	0.945355	0.9402174	0.940701
2	0.104167	0.988827	0.967213	0.97790056	0.978437

Рис. 9: Таблиця результатів метрик

Література

[1] Glenn Jocher та ін. *ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation*. Вер. v7.0. Листоп. 2022. DOI: 10.5281/zenodo.7347926. URL: https://doi.org/10.5281/zenodo.7347926.

[2] *Nexus AI*. https://fiber-punk.com/pages/nexusai.

[3] *Obico Spagheti Detective*. https://www.obico.io/blog/2019/08/14/3d-printing-deep-learning-training/.

[4] *Octoprint Detector2*. https://github.com/mikulash/OctoPrint-Detector2.

[5] *OctoPrint PrintWatch*. https://github.com/printpal-io/OctoPrint-Printwatch.

[6] Javier Villarroel. *3D Failures detection dataset*. https://huggingface.co/datasets/Javiai/failures-3D-print. Accessed: 2023-10-06.

[7] Javier Villarroel. *3D Failures detection model based on Yolov5*. https://huggingface.co/Javiai/3dprintfails-yolo5vs. Accessed: 2023-10-06.