



Procesare imagini. Numărare obiecte în industrie.

Proiect Sincretic 1
Sisteme cu microcontrolere și IOT industriale

Varatic Alexandru

Crețu Vlad-Sebastian

Melinte Dan

Ursu Dumitru

Profesor de curs :

Conf.dr.ing. Adrian Korodi

Timișoara

<2023>

Cuprins

1. Introducere
2. Alegerea Modelului
3. Descrierea Modelului EfficientDet
4. Dataset-ul
5. Antrenamentul Modelului
6. Arhitectura Hardware (Schema tehnică) + Mediul fizic
7. Interfața grafică de utilizator
8. Schema funcțională
9. Bibliografie/Referințe

1. Introducere

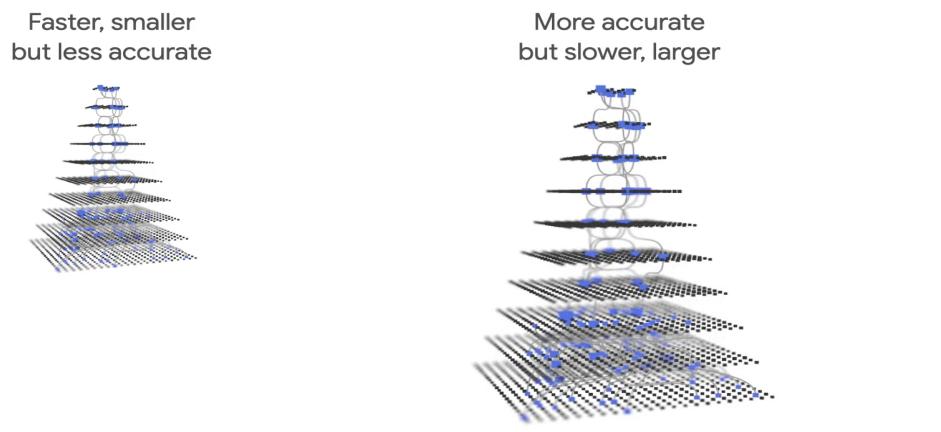
Scopul proiectului este realizarea unei aplicații IIoT prin care Raspberry Pi va fi capabil să detecteze și să numere obiecte de o anumită clasă (cărți de joc).

Detectarea se va face pe baza rețelelor neuronale, utilizând modelul antrenat cu ajutorul bibliotecii Tensorflow, dezvoltată de Google.

Programul va avea o interfață capabilă să valideze sau nu numărul de obiecte detectate, cadrele validate vor fi stocate într-un folder cu scop de a fi transmise ca confirmare. În caz de eroare este necesara depanarea de către utilizator, astfel impune reluarea procesului de detectare și numărare a obiectelor.

2. Alegerea Modelului

Cele 3 mărimi pentru măsurarea performanței unui model neuronal sunt: latența, dimensiunea, precizia. Astfel, precizia este direct proporțională cu dimensiunea și latența. În Machine Learning există mereu un trade-off între viteza de inferență și acuratețe/precizie. Pentru a îmbunătăți acuratețea modelului, avem nevoie de un model mai mare, care de obicei rulează mai greu.



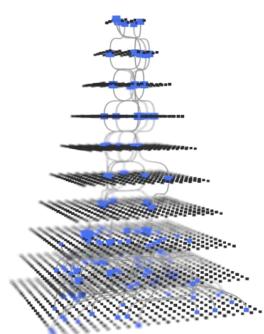
În contextul în care dorim să rulăm modelul pe microcalculatorul Raspberry Pi Model 3, care dispune de resurse computaționale limitate (Quad Core 1.2GHz Broadcom BCM2837 64bit CPU, 1GB RAM), dimensiunea modelului și resursele necesare computării sunt doi factori critici.

Din fericire, comitetele și echipele de cercetători lucrează continuu pentru descoperirea noilor arhitecturi de modele cu acuratețe ridicată, păstrând dimensiunea și viteza la un nivel rezonabil, pentru a putea fi implementate în producție și IoT.

EfficientDet-Lite este un exemplu bun în acest caz : o familie de arhitecturi neuronale, descoperită de către o echipă de cercetare de la Google prin intermediul *Neural Architecture Search* (tehnică bazată pe AutoML), în anul 2020. Ideea de bază a fost descoperirea de noi arhitecturi printr-un proces iterativ de construire a modelelor propuse, antrenare și evaluare, cu respectarea anumitor constrângeri de dimensiune și viteză, maximizând acuratețea.

EfficientDet-Lite: SoTA object detection for edge devices

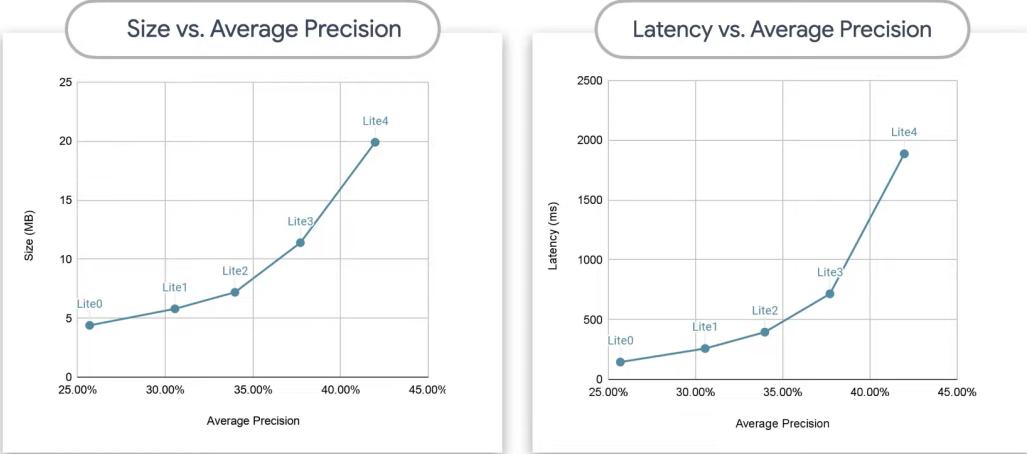
Controller: proposes ML models



Train & evaluate models

many times
Iterate to find the most accurate model





Rezultatele au fost obținute pe baza dataset-ului COCO (2017).

Model architecture	Size(MB)*	Latency(ms)**	Average Precision***
EfficientDet-Lite0	4.4	146	25.69%
EfficientDet-Lite1	5.8	259	30.55%
EfficientDet-Lite2	7.2	396	33.97%
EfficientDet-Lite3	11.4	716	37.70%
EfficientDet-Lite4	19.9	1886	41.96%

↑ Faster
Smaller
Less accurate

↓ Slower
Larger
More accurate

* Size of the integer quantized models.

** Average latency measured on Raspberry Pi 4 using 4 threads on CPU running the integer quantized model.

*** Average Precision is the mAP (mean Average Precision) on the COCO 2017 validation dataset.

Modelul de rețea neuronală ales este **EfficientDet-Lite-0**. Motivul alegerii acestui model se rezumă la : dimensiunea mică a modelului (4.4 MB), viteza de procesare mare, respectiv latență mică (146 ms), și precizie medie rezonabilă (25.69%).

3. Descrierea Modelului *EfficientDet*

Dezvoltator: Google

Atribute de bază : dimensiune mică a modelului, viteză mare de procesare, acuratețe mare

Structură funcțională :
(clasică pt. Object Detector)

- 1) *Efficient Backbone*
- 2) *BiFPN Feature Network*
- 3) *Detection Head/Network*

1) *Efficient Backbone*

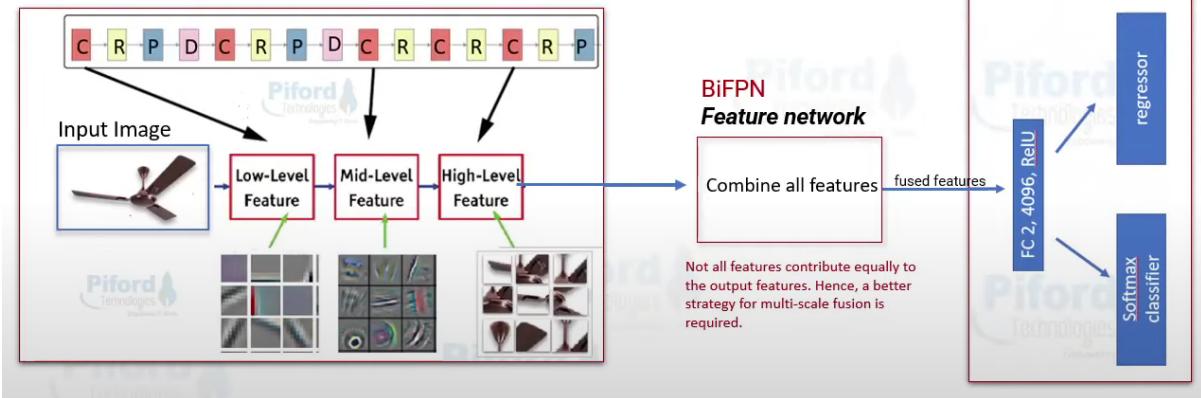
- responsabil de extragerea caracteristicilor : low-level (linii, margini), mid-level (curbe), high-level (forme abstracte), în funcție de cantitatea de informație conținută și dimensionalitate
- constituie din multiple stack-uri convoluționale
- C - *Convolution Layer* (extragă caracteristicile locale din imagini, returnează *feature maps*-uri pe baza filtrelor aplicate)
- R - *Rectified Linear Unit* (*Relu* - funcție de activare)
- P - *Pooling Layer* (reduce dimensionalitatea *feature maps*-urilor prin aplicarea unei operații suplimentare : min, max, avg)
- D - *Dropout Layer* (anulează cu o rată prestabilită intrările, pentru a preveni *overfitting*-ul)

2) *BiFPN Feature Network*

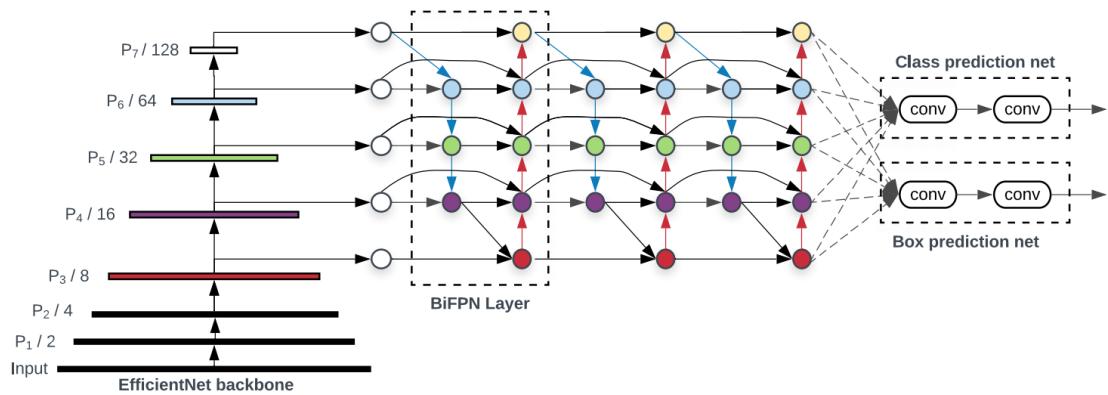
- *bidirectional feature pyramid network*
- colectează caracteristicile high-level, le fuzionează, le transmite mai departe către *Detection Head*
- multi-scale fusion (P3-P7), rezoluții diferite (+ponderate) a caracteristicilor de intrare

3) *Detection Head/Network*

- FC/Dense Layer
- Softmax Classifier - responsabil de clasificarea obiectului detectat
- Regressor - responsabil de coordonatele geometrice ale box-ului obiectului detectat

EfficientNet (Backbone – feature Extractor)


Arhitectură simplificată a modelului

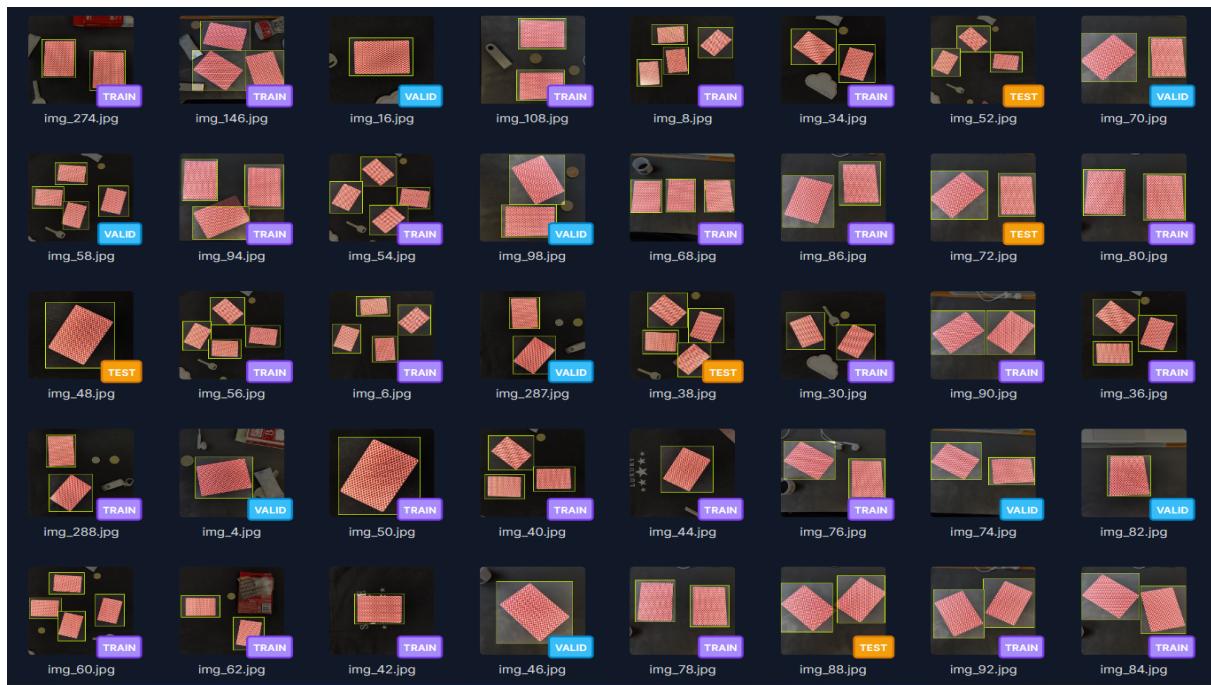


Arhitectură reală a modelului

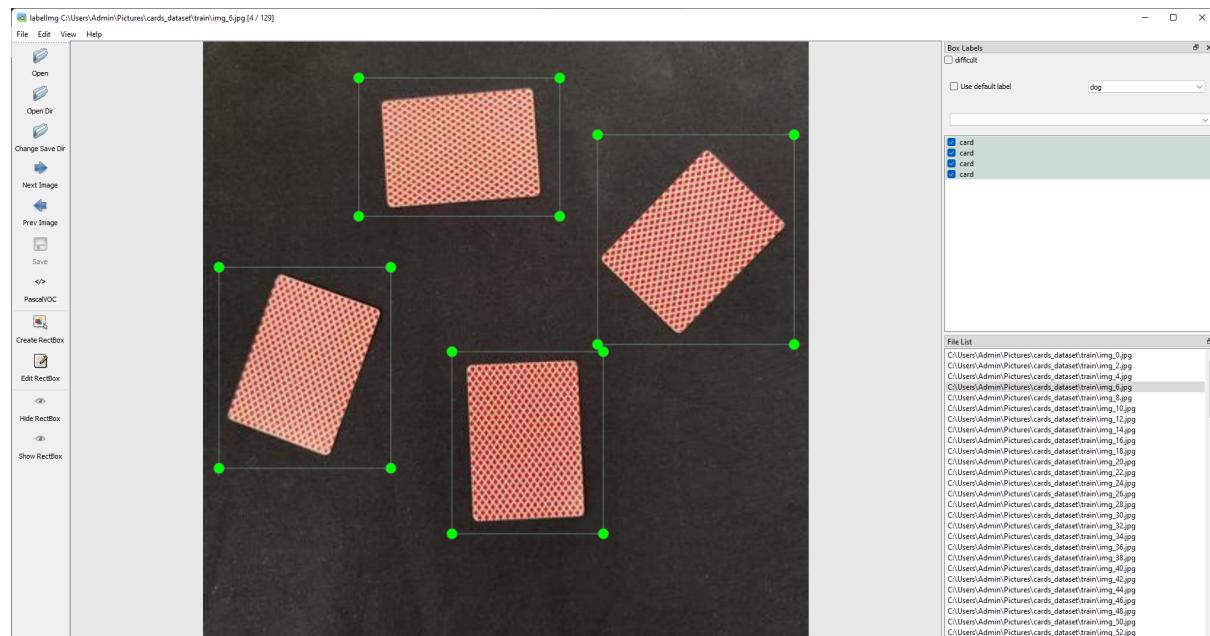
4. Dataset-ul

Dataset-ul reprezintă datele pe baza cărora este antrenat modelul neuronal. Acesta este alcătuit din **Features** (date de intrare, imaginile în cazul nostru) și **Labels** (date de ieșire, clasele obiectelor detectate și coordonatele acestora).

Dataset-ul este unul propriu. Conține 161 de poze cu cărți de joc situate pe partea posterioară. Fundalul este unul negru, pentru obținerea unui contrast mai bun. Preprocesarea datelor a constat în redimensionarea imaginilor din dimensiunea originală (3000x3000) în cea așteptată de model (320x320).



Pentru adnotarea imaginilor s-a folosit software-ul **LabelImg**, care simplifică procesul punând la dispoziția utilizatorului o interfață grafică.



Adnotarea imaginilor constă în marcarea/identificarea pe fiecare imagine a claselor de obiecte existente și a poziției geometrice a acestora în cadrul imaginilor. Acest lucru se face prin plasarea obiectelor într-un box/chenar *RectBox*, la care se calculează automat coordonatele în pixeli a 2 puncte extreme (xmin, ymin, xmax, ymax).

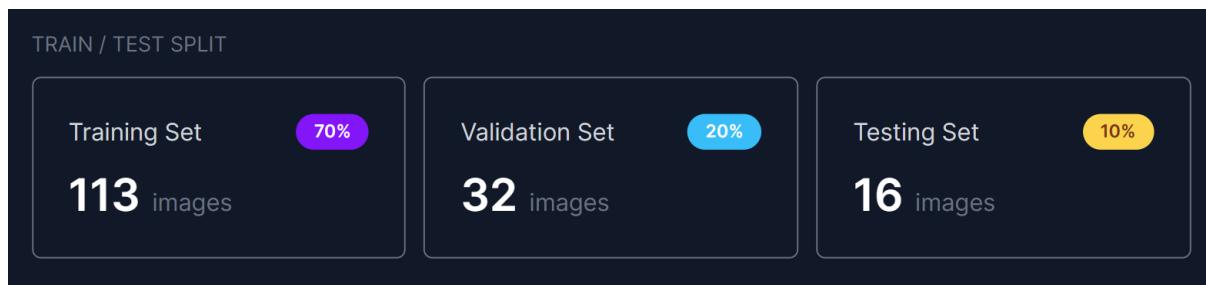
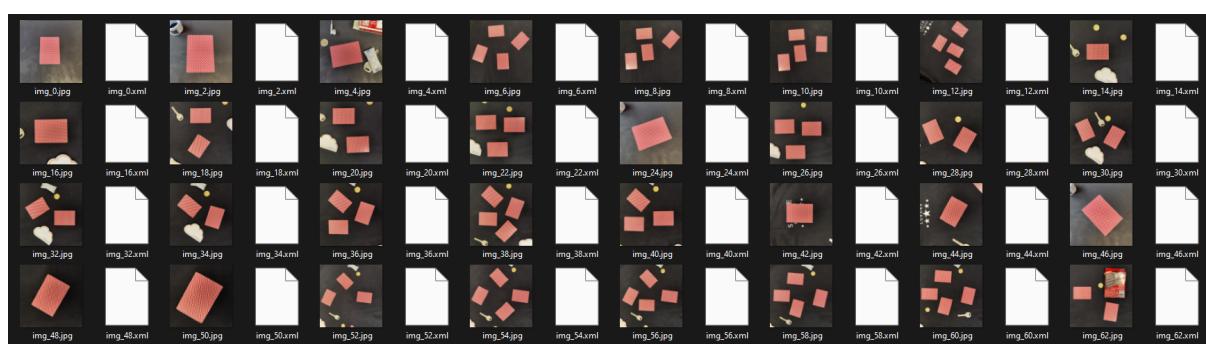
Pentru reprezentarea datelor din dataset s-a folosit formatul ***Pascal VOC***, care salvează fișierele cu adnotări în format XML. Fiecărui fișier imagine îi corespunde un fișier XML, ce conține toate informațiile utile despre imaginea dată.

```

▼<annotation>
  <folder>train</folder>
  <filename>img_0.jpg</filename>
  <path>C:\Users\Admin\Pictures\cards_dataset\train\img_0.jpg</path>
  ▼<source>
    <database>Unknown</database>
  </source>
  ▼<size>
    <width>416</width>
    <height>416</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  ▼<object>
    <name>card</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    ▼<bndbox>
      <xmin>124</xmin>
      <ymin>104</ymin>
      <xmax>275</xmax>
      <ymax>302</ymax>
    </bndbox>
  </object>
</annotation>

```

Rezultatul obținut este următorul, iar datele au fost segregate în 3 subseturi după principiul 70% : 20 % : 10 % = train : validate : test, pentru o eficiență maximă.



5. Antrenamentul Modelului

Pentru antrenamentul modelului am folosit modulul / librăria **TensorFlow Lite Model Maker**. Modulul simplifică procesul de antrenament a unui model TensorFlow Lite folosind un dataset personalizat/custom. Acesta folosește **Transfer Learning** (reutilizarea sau transferul de informații din sarcinile învățate anterior, pentru învățarea de noi sarcini).

Ca mediu de antrenare am folosit platforma **Google Colab**, ce oferă posibilitatea de rularea a unui runtime pe o mașină virtuală, optimizat pentru Data Science și Machine Learning.

Pentru antrenament a fost alocată o placă video specializată - **NVIDIA Tesla T4**.

```
! nvidia-smi
Mon Jan 9 20:19:56 2023
+-----+
| NVIDIA-SMI 460.32.03      Driver Version: 460.32.03      CUDA Version: 11.2      |
|-----+-----+-----+-----+-----+-----+-----+-----+
| GPU  Name      Persistence-M| Bus-Id      Disp.A  | Volatile Uncorr. ECC  | | | | | |
| Fan  Temp  Perf  Pwr:Usage/Cap| Memory-Usage | GPU-Util  Compute M.  |
| |          |          |              |      |          |          |      |
|-----+-----+-----+-----+-----+-----+-----+-----+
| 0  Tesla T4      Off  | 00000000:00:04.0 Off |                  0 | |
| N/A   46C     P0    27W /  70W |            310MiB / 15109MiB |      0%     Default |
|                               |              |          |      N/A |
+-----+-----+-----+-----+-----+-----+-----+-----+
```

Adițional am utilizat API-ul public *tflite_model_maker*, pentru interacțiunea cu librăria principală, împreună cu modulele *object_detector* (API pentru a antrena un model pentru object detection) și *model_spec* (API pentru a obține modelul pre-antrenat dorit - *efficientdet_lite0*).



Layer (type)	Output Shape	Param #
keras_layer (KerasLayer)	multiple	3234464
class_net/class-predict (SeparableConv2D)	multiple	1746
box_net/box-predict (SeparableConv2D)	multiple	2916
Total params:	3,239,126	
Trainable params:	3,191,990	
Non-trainable params:	47,136	

Un scurt cuprins al modelului *efficientdet_lite0*

```
▶ print(type(train_data))
<class 'tensorflow_examples.lite.model_maker.core.data_util.object_detector_dataloader.DataLoader'>
```

În continuare se folosește metoda *create* din *object_detector* pentru a începe procesul de antrenare a modelului, iar instantierea rezultată o salvăm într-o variabilă separată - *model*.

```
▶ # Train the TensorFlow model with the training data
model = object_detector.create(train_data, model_spec=spec, batch_size=8, train_whole_model=True, validation_data=val_data)
```

Dimensiunea batch-ului este de 8, iar numărul de epoch-uri îl păstrăm pe cel prestabilit (50). Procesul de antrenament a durat 364.664s (6 minute).

```
Epoch 40/50
[-----] - 6s 392ms/step - det_loss: 0.1976 - cls_loss: 0.1303 - box_loss: 0.0013 - reg_l2_loss: 0.0631 - loss: 0.2687 - learning_rate: 9.0857e-04 - gradient_norm: 1.3127 - val_det_loss: 0.2138 - val_cls_loss: 0.1612
Epoch 41/50
[-----] - 5s 392ms/step - det_loss: 0.1893 - cls_loss: 0.1247 - box_loss: 0.0013 - reg_l2_loss: 0.0631 - loss: 0.2524 - learning_rate: 7.2560e-04 - gradient_norm: 1.3188 - val_det_loss: 0.1995 - val_cls_loss: 0.1588
Epoch 42/50
[-----] - 5s 331ms/step - det_loss: 0.1750 - cls_loss: 0.1157 - box_loss: 0.0012 - reg_l2_loss: 0.0631 - loss: 0.2381 - learning_rate: 5.6837e-04 - gradient_norm: 1.2077 - val_det_loss: 0.2022 - val_cls_loss: 0.1591
Epoch 43/50
[-----] - 5s 334ms/step - det_loss: 0.1846 - cls_loss: 0.1226 - box_loss: 0.0012 - reg_l2_loss: 0.0631 - loss: 0.2477 - learning_rate: 4.2926e-04 - gradient_norm: 1.2462 - val_det_loss: 0.2059 - val_cls_loss: 0.1612
Epoch 44/50
[-----] - 5s 334ms/step - det_loss: 0.1886 - cls_loss: 0.1213 - box_loss: 0.0013 - reg_l2_loss: 0.0631 - loss: 0.2516 - learning_rate: 3.0894e-04 - gradient_norm: 1.2834 - val_det_loss: 0.2020 - val_cls_loss: 0.1587
Epoch 45/50
[-----] - 7s 440ms/step - det_loss: 0.1847 - cls_loss: 0.1237 - box_loss: 0.0012 - reg_l2_loss: 0.0631 - loss: 0.2478 - learning_rate: 2.0789e-04 - gradient_norm: 1.2696 - val_det_loss: 0.2104 - val_cls_loss: 0.1606
Epoch 46/50
[-----] - 5s 329ms/step - det_loss: 0.1880 - cls_loss: 0.1265 - box_loss: 0.0012 - reg_l2_loss: 0.0631 - loss: 0.2510 - learning_rate: 1.6653e-04 - gradient_norm: 1.2117 - val_det_loss: 0.2098 - val_cls_loss: 0.1615
Epoch 47/50
[-----] - 5s 330ms/step - det_loss: 0.1852 - cls_loss: 0.1232 - box_loss: 0.0012 - reg_l2_loss: 0.0631 - loss: 0.2483 - learning_rate: 6.5195e-05 - gradient_norm: 1.2496 - val_det_loss: 0.2056 - val_cls_loss: 0.1612
Epoch 48/50
[-----] - 5s 332ms/step - det_loss: 0.1803 - cls_loss: 0.1217 - box_loss: 0.0012 - reg_l2_loss: 0.0631 - loss: 0.2434 - learning_rate: 2.4141e-05 - gradient_norm: 1.1865 - val_det_loss: 0.2048 - val_cls_loss: 0.1612
Epoch 49/50
[-----] - 5s 331ms/step - det_loss: 0.1708 - cls_loss: 0.1148 - box_loss: 0.0011 - reg_l2_loss: 0.0631 - loss: 0.2339 - learning_rate: 3.5342e-06 - gradient_norm: 1.2422 - val_det_loss: 0.2048 - val_cls_loss: 0.1612
Epoch 50/50
[-----] - 7s 433ms/step - det_loss: 0.1851 - cls_loss: 0.1232 - box_loss: 0.0012 - reg_l2_loss: 0.0631 - loss: 0.2482 - learning_rate: 3.4587e-06 - gradient_norm: 1.5173 - val_det_loss: 0.2044 - val_cls_loss: 0.1614
```

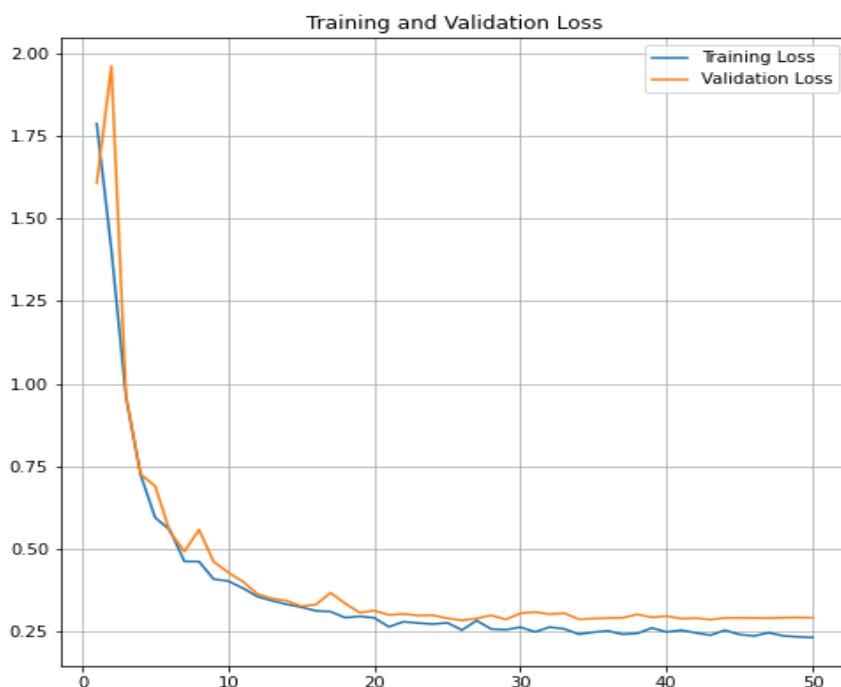
În urma evaluării modelului pe dataset-ul de test cu metoda *evaluate*, obținem următoarele rezultate.

```
# Evaluate the model with the test data
model.evaluate(test_data)

1/1 [=====] - 3s 3s/step

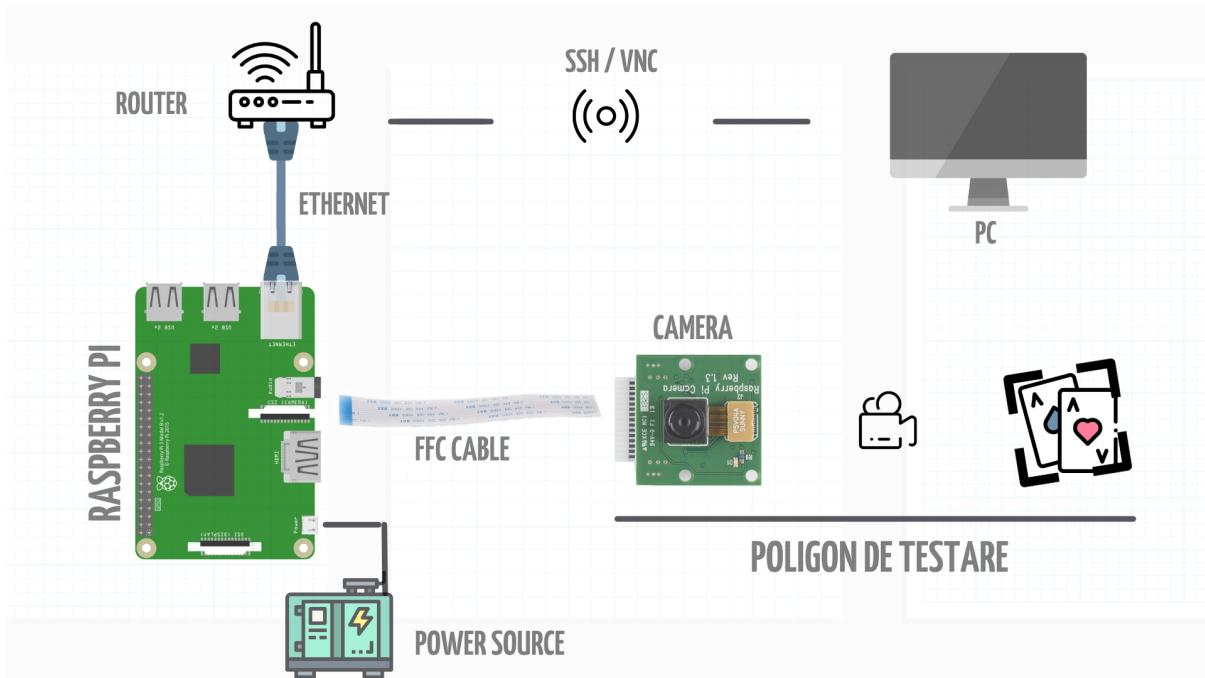
{'AP': 0.8598807,
'AP50': 1.0,
'AP75': 1.0,
'APs': -1.0,
'APm': -1.0,
'AP1': 0.8598807,
'ARmax1': 0.34615386,
'ARmax10': 0.8923077,
'ARmax100': 0.8923077,
'ARs': -1.0,
'ARm': -1.0,
'AR1': 0.8923077,
'AP_card': 0.8598807}
```

Valoarea AP-ului (*Average Precision*) este 0.8598807, ce reprezintă un scor rezonabil.



Graficul evoluției funcției Loss (exprimă diferența dintre output-ul prezis și cel corect) pentru *Training* și pentru *Validation* ne confirmă succesul antrenamentului. În final obținem valori de aproximativ 0.25 pentru ambele caracteristici.

6. Arhitectura Hardware (Schema tehnică) + Mediul fizic

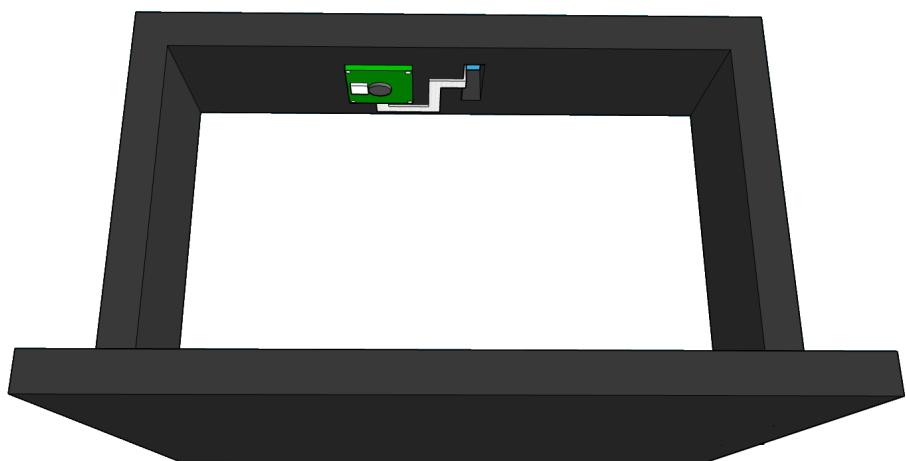
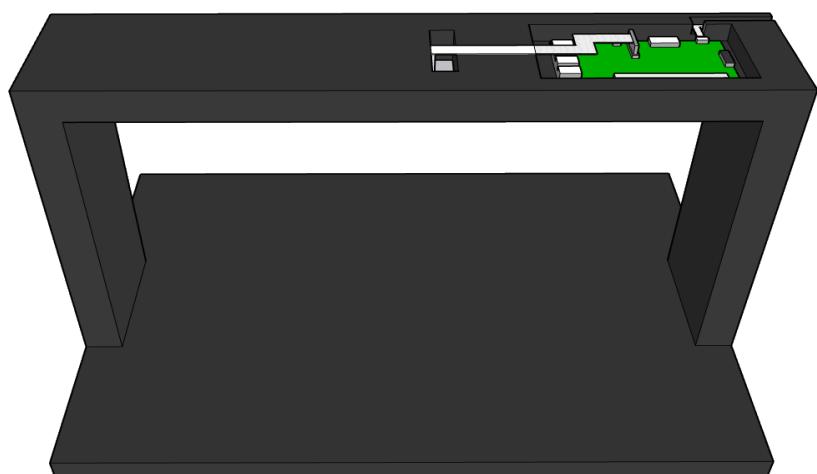


Arhitectura hardware a proiectului include câteva componente de bază:

- **Raspberry Pi Model 3** (SBC - Single Board Computer, pe o singură placă de circuit cu microprocesor, memorie, I/O, și alte funcționalități ale unui microcalculator)
- **Raspberry Pi Camera Module 2** (modul camerei pentru capturi foto/video, dotat cu un senzor de 8 MP Sony IMX219, se conectează la Raspberry Pi printr-un cablu ribbon de 15 cm la portul CSI-2, este folosit pentru captura obiectelor în timp real)
- **PC** (folosit pentru remote access la *Raspberry Pi* print suitele software *VNC Server - VNC Viewer*, printr-o conexiune de tip *SSH*)

Ca mediu fizic dispunem de un poligon de testare, care este o construcție proprie din plăci de polietilenă expandată. Poligonul are o platformă orizontală și o rampă în formă de pod (*bridge*). Design-ul poligonului a fost realizat în aplicația *SketchUp*.

Pe platformă se așeză obiectele care urmează să fie detectate. Imaginea acestora este captată în timp real sub formă de stream de către modulul camerei, transmisă către *Raspberry Pi*, care preia și prelucrează feed-ul.



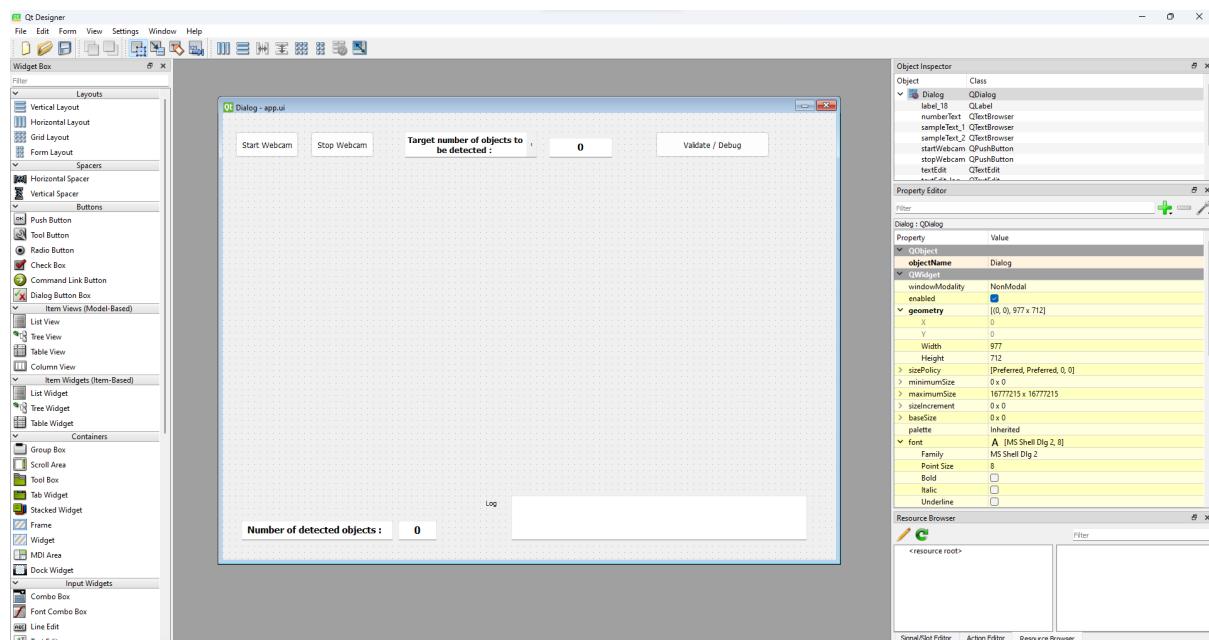
7. Interfață grafică de utilizator

Pentru crearea interfeței grafice de utilizator (**GUI**), am optat pentru **PyQt** - un *binding* pentru limbajul Python cu Qt - un set de librării și utilități C++ ce oferă API-uri de nivel înalt pentru dezvoltarea interfețelor grafice.

Versiunea modulului folosit este **PyQt5**.

Modulele secundare utilizate : **QtGui**, **QtWidgets**, **QtCore**, **uic**.

Pentru crearea interfeței de utilizator am folosit software-ul **Qt Designer**, o utilitară pentru crearea rapidă și confortabilă a interfețelor, ce pune la dispoziție widget-uri din framework-ul **Qt GUI**. Aceasta oferă o interfață simplă cu *drag-and-drop* pentru plasarea diverselor componente de dialog, precum : butoane, câmpuri de text, casete combinate, etc.



Elementul central/principal al interfeței este un label/placeholder (**QLabel**) pentru captura video care va fi afișată în aplicație (**videoLabel**). Acesta are dimensiunea de 640x480 px.

Interfața creată include o serie de butoane de comandă (**QPushButton**) : un buton de **start** (**startWebcam**) al camerei video, unul de **stop** (**stopWebcam**), și unul de **validare sau debug** (**valButton**).

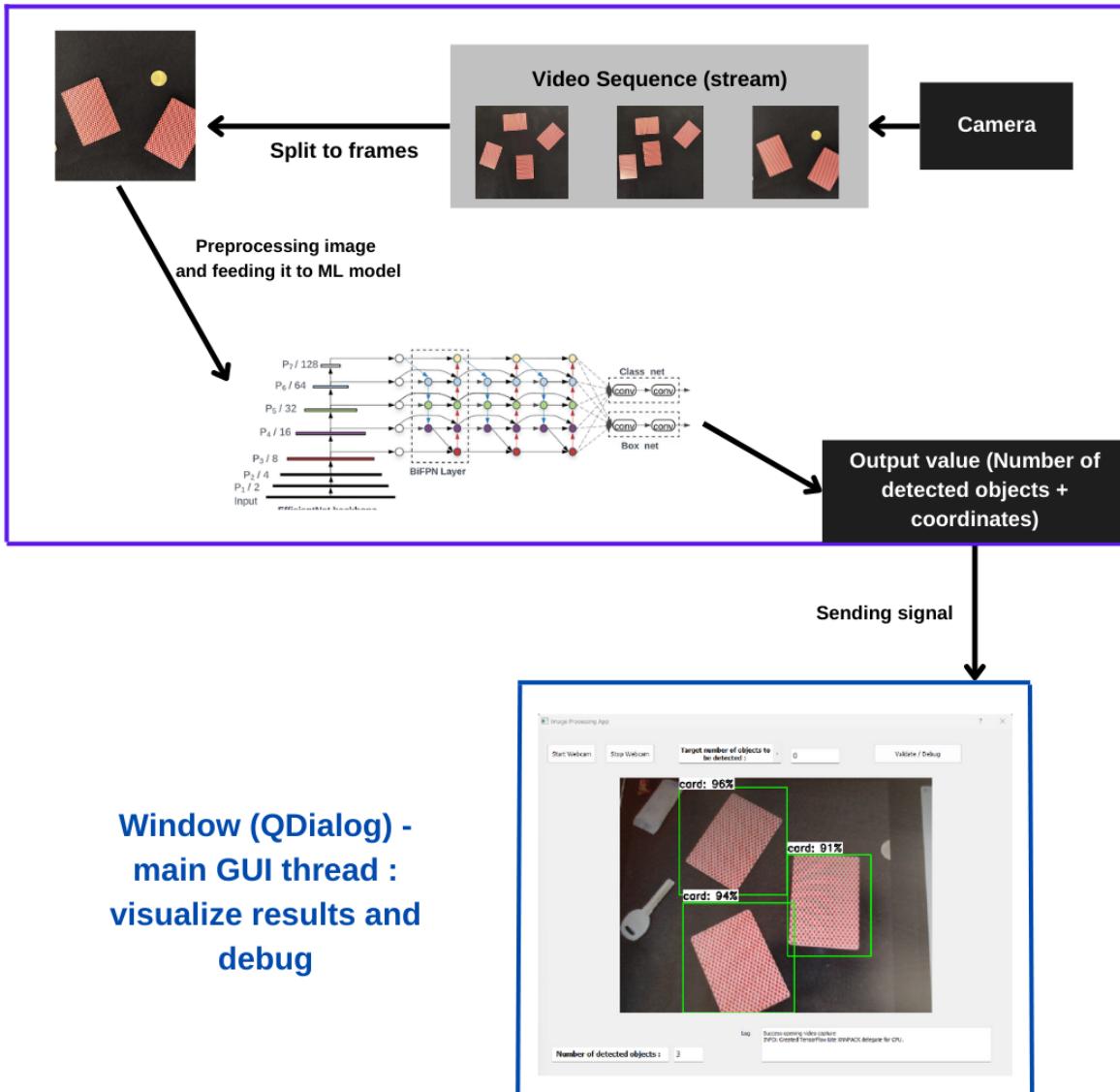
De asemenea sunt prezente 2 casete de text : una editabilă (**QTextEdit**) de către utilizator și alta nu (**QTextBrowser**).

În caseta needitabilă (**numberText**) se afișează numărul de obiecte detectate, iar în cea editabilă (**targetEdit**) - utilizatorul poate introduce un număr de obiecte țintă/target.

În partea inferioară se regăsește și o fereastră pentru debug/depanare, în care sunt afișate mesaje informative despre procesele în derulare. Aceasta e constituită dintr-o casetă text editabilă (***QTextEdit*** : *logEdit*).

8. Schema funcțională

Worker1 (QThread) : retrieves image & processes it & sends signal back



Funcțional, programul este alcătuit dintr-un **thread principal**, în care rulează fereastra de GUI împreună cu toate elementele de interfață, și un **thread secundar**, care răspunde de captarea și procesarea imaginilor.

Pentru captarea imaginilor, preprocesarea și postprocesarea lor s-a folosit modulul **OpenCV**.

Pentru **multithreading** s-a folosit clasa **QThread** din **PyQt5**.

Pentru comunicarea între thread-uri s-a folosit mecanismul **signals/slots**.

Worker1 (Qthread) :

Thread-ul secundar este o instanță a clasei **Worker1**, ce moștenește clasa **QThread**.

Se declară semnalul **ImageUpdate**, instanță a clasei **pyqtSignal**.

În interiorul metodei **run** al lui **Worker1** se încarcă modelul TFLite prin instantierea interpretorului (**tf.lite.Interpreter**), se obțin detaliile formatului de intrare și ieșire ale modelului prin metodele **get_input_details** și **get_output_details**.

Se instantiază clasa **VideoCapture** din **OpenCV**, după care se citește (cu metoda **read**) și procesează continuu, fiecare frame într-o buclă infinită, atât timp cât este activ thread-ul.

Frame-ul este **preprocesat** (convertit în RGB, redimensionat, pus în batch), feed-uit ca intrare modelului, care este invocat/rulat prin metoda **invoke**.

Rezultatele despre obiectele detectate în urma rulării inferenței sunt preluate cu metoda **get_tensor**, și salvate în 3 variabile: **boxes**, **classes**, **scores**.

Pentru fiecare obiect din lista cu scoruri, care depășește un prag minim de încredere (**minimum confidence threshold**), sunt normalizeze coordonatele casetei de încadrare, ilustrat/imprimat un dreptunghi pe frame (cu metoda **rectangle**), și o etichetă/label cu denumirea clasei obiectului (cu metoda **putText**) (partea de **postprocesare**).

Este creat un obiect de tip **QImage** din frame-ul procesat/prelucrat.

La finalul buclei se emite semnalul **ImageUpdate**.

Acest semnal este trimis din thread-ul secundar către thread-ul principal după fiecare captare și procesare de frame. Odată cu el, sunt trimise următoarele date utile :

- **coords** : o listă cu coordonatele casetelor de încadrare (**bounding box**) pentru fiecare obiect detectat
- **frame** : imaginea frame-ului procesat (pentru validare), obiect de tip **numpy.ndarray**
- **resized** : imaginea frame-ului procesat (pentru afișare), obiect de tip **QImage**
- **n** : numărul (int) de obiecte detectate

Window (QDialog) :

Thread-ul principal este o instanță a clasei personalizate **Window**, care moștenește clasa **QDialog**.

Inițializarea clasei Window presupune încărcarea interfeței (fișierul .ui), inițializarea variabilelor și a thread-ului secundar **Worker1**, conectarea evenimentelor de apăsare a butoanelor cu funcțiile corespunzătoare (**Start**, **CancelFeed**, **Validate**), conectarea semnalului (**ImageUpdate**) cu funcția sa slot (**ImageUpdateSlot**).

Funcția slot **ImageUpdateSlot** este funcția de bază. Ea este apelată de fiecare dată când thread-ul principal recepționează semnalul **ImageUpdate** din thread-ul secundar.

Funcția realizează 2 sarcini :

- imprimă/actualizează (cu metoda **setPixmap**) pe **videoLabel** frame-ul procesat recepționat
- implementează logica de validare : dacă este apăsat butonul **Validate/Debug**, se verifică dacă numărul de obiecte detectate coincide cu cel întă/target;

în caz de adevăr se validează rezultatul \Leftrightarrow se realizează o captură de ecran și se salvează în folderul *captures/*

în caz contrar se face debug/depanare \Leftrightarrow se crează un fișier de log (.log) în folderul */logs*, cu informații utile despre detectare (dată, numere, coordonate)

9. Bibliografie/Referințe

- <https://github.com/RajKKapadia/Object-Detection-Generalised-Youtube>
- <https://youtu.be/-ZyFYniGUsw>
- https://www.tensorflow.org/lite/models/modify/model_maker
- <https://arxiv.org/pdf/1611.07791>
- <https://www.sciencedirect.com/science/article/pii/S1877705812025684>
- https://www.researchgate.net/profile/Ayush-Jain-23/publication/331421347_Real_Time_Object_Detection_and_Tracking_Using_Deep_Learning_and_OpenCV/links/5d70a32f299bf1cb8088576c/Real-Time-Object-Detection-and-Tracking-Using-Deep-Learning-and-OpenCV.pdf
- [Module: tf.keras | TensorFlow v2.11.0](#)
- [Convolutional Neural Network \(CNN\) | TensorFlow Core](#)
- <https://youtu.be/tPYj3fJGjk>
- <https://youtu.be/OsA3zH5NKYc>
- <https://youtu.be/twtBcfonSyE>
- <https://youtu.be/oXIwWbU8I2o>
- [Guide | TensorFlow Core](#)
- https://www.tensorflow.org/lite/api_docs/python/tflite_model_maker
- <https://github.com/heartexlabs/labelImg>
- <https://www.sketchup.com/>
- <https://colab.research.google.com/>
- <https://arxiv.org/pdf/1911.09070.pdf>
- <https://tfhub.dev/tensorflow/efficientdet/lite0/detection/1>
- <https://universe.roboflow.com/upt/card-detection-nzzvt/dataset/1>
- <https://github.com/varaticalexandru/object-detection-project>