Okos önkiszolgáló kassza

Feladat leírása:

A kamera látóterében található, valamilyen módon előre definiált termékek felismerése, független a termék által felvett póztól. Valós példaként a kasszáknál történő kiszolgálást veszem alapul ahol különböző előre csomagolt termékeket vásárolunk. A termékeket QR-kóddal látják el amivel a termék beazonosítható. A beadandó keretében nem QR-kód alapján azonosítja be a gép a termékeket hanem egy kamera képe alapján. Ez a módszer hatékony kiegészítő módszer lehet ha a terméken lévő QR-kód megsérül, a nem valós vagy a beolvasó nem működik. Program feladata egy képi forrás alapján eldönteni hogy milyen termékek találhatóak rajta. Minimum 6 fajta termék felismerése a cél. Akár gyorsabb is lehet a tárgyak beolvasása mert a kamera képben több tárgy is elhelyezhető, míg a QR kódot általában egyesével olvassunk be.

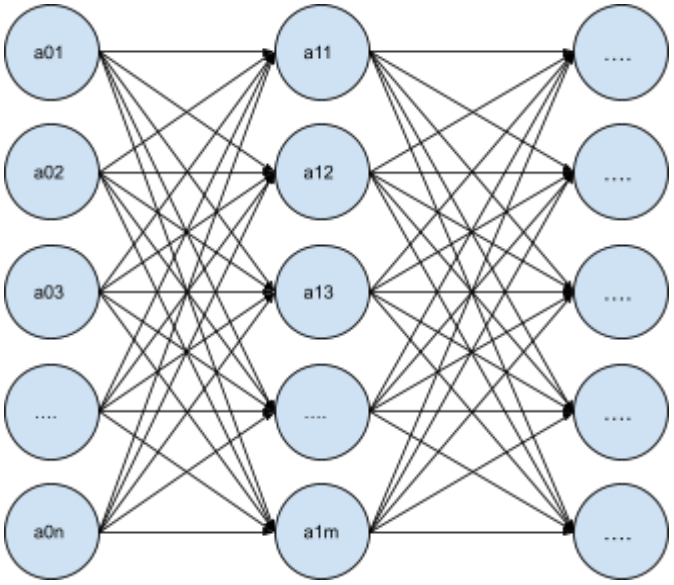
Egy kassza kezelő program létrehozása lehet a megoldás, ahol a felhasználó látja a kamera képét illetve kosarának tartalmát. Opcionálisan érdemes lehet implementálni egy forrás választó gombot, hogy több kamera képéből lehessen választani.
A termékeket a progambe előre le vannak tárolva, így csak a letárolt elemeket fogja felismerni

Elméleti háttér:

Fogalmak:

- **Mesterséges intelligencia** (Artificial Intelligence): Bármilyen gép általi reakció ami emberi intelligenciát “utánozza”. Például számítógépes játékban a karakterek mozognak, mennek a beprogramozott helyükre.
- **Gépi tanulás** (Machine learning): Mesterséges intelligencia olyan megvalósítása ahol már tanulás után dinamikusan végzi a gép a feladatokat. Például: Válós emberek játékban történő útvonalai alapján a gépi karakterek a mások által bejárt utat fogja követni (ez lehet statisztika alapú is például átlagolás).
- **Mélytanulás** (Deep Learning): Gépi tanuláson belül olyan módszer ahol neuron alapú módszerrel tanul a program (akár az emberi agy). Például: Válós emberek játékban történő útvonalait neuron alapú hálózaton keresztül végig számítva a gépi karakterek a mások által bejárt utat fogja követni (ez lehet statisztika alapú is például átlagolás).

Neurális háló:



Neurális háló magyarázat:

- Neurális hálókban a neuron (pontok) egy értékkel (számmal) rendelkeznek, mondhatni egyszerű változók.
- Első oszlop neuronjai lesznek a bemeneti adatok.
- Utolsó oszlop neuronjai lesznek a kimeneti adatok.
- Köztes oszlopokat rejtett rétegnek (hidden layer) nevezzük.

Az neuron oszlop értékeit a megelőző oszlop összes eleméből számoljuk ki (természetesen az első oszlop már adott) a következőképpen:

- l: réteg
- $a_n^{(l-1)}$: előző oszlop elemei ($a_0^{(l-1)}$ első elem, $a_1^{(l-1)}$ második elem, stb.)
- $a_m^{(l)}$: számolandó oszlop elemei ($a_0^{(l)}$ első elem, $a_1^{(l)}$ második elem, stb.)
- $b_m^{(l)}$: számolandó oszlop torzító (bias) elemei
- $w_{m,n}^{(l)}$: súlyvektor megadja melyik él milyen súllyal bírjon a számolandó sorba (oszlop sora m)
- $\sigma = f(x) = \frac{1}{1+e^{-x}}$: szigmoid függvény, általában optimalizáció miatt lecserélik egy egyenirányító (ReLU) függvényre: $ReLU(x) = \left\{ \begin{array}{ll} 0, & x<0 \\ x, & különben \end{array} \right\}$

Röviden:

$$a^{(l)} = \sigma(w^{(l)} \cdot a^{(l-1)} + b^{l})$$

Vektorosan felírva

$$\sigma\left(\begin{bmatrix} w_{0,0}^{(l)} & w_{0,1}^{(l)} & \cdots & w_{0,n}^{(l)} \\ w_{1,0}^{(l)} & w_{1,1}^{(l)} & \cdots & w_{1,n}^{(l)} \\ \vdots & \vdots & \ddots & \\ w_{m,0}^{(l)} & w_{m,1}^{(l)} & \cdots & w_{m,n}^{(l)} \end{bmatrix}\begin{bmatrix} a_0^{(l-1)} \\ a_1^{(l-1)} \\ \vdots \\ a_n^{(l-1)} \end{bmatrix} + \begin{bmatrix} b_0^{(l)} \\ b_1^{(l)} \\ \vdots \\ b_m^{(l)} \end{bmatrix}\right)$$

Egy neuronra felírva:

$$a_0^{(1)} = \sigma(w_{0,0}^{(1)} \cdot a_0^{(0)} + w_{0,1}^{(1)} \cdot a_1^{(0)} + \ \ldots + w_{0,n}^{(1)} \cdot a_n^{(0)} + b_0^{(1)})$$

Tanítás folyamata:

Neurális pontok kiszámítása adott az értékek változtatására a súlyokkal és torzító értékekkel van lehetőségünk. Ezeket a tapasztalások útján lehet meghatározni. A programnak meg kell adni bemenetet és az elvárt kimeneti értéket.

Célunk hogy a számítógépnek megmondjuk hogy mennyit rontott és hogy melyik neuronon értékén mennyit kellene javítani a jobb eredményhez. Általában ezek a lokális minimum értékek.

Végső eredmény az alábbi módon befolyásolhatjuk:

- Változtathatjuk a torzító (bias) értéket: b

- Változtathatjuk a súly értékeket: w_i
    - “Fire together, wire together” szabály : Ahol az erős neuronok (többiekhez képest magasétékűek) kapcsolódnak egy általunk növelni vagy csökkenteni való neutronhoz akkor a kisebb neutronokat kevésbé kell csökkentenünk, a nagyobb erősségű neutronok súlyaira kell összpontosítani
- Változtathatjuk az előző neuron értékét: a_i
    - Növelhetjük az értéket ha: ahol neuron pozitív azokat növeljük, ahol negatív azokat csökkentjük (módosítandó neuronokat is az azokat megelőző súlyokkal tudjuk változtatni
    - Közvetlen nem tudjuk változtatni csak súly (w_i) és torzító értékeket tudunk változtatni (b)

Végső veszteség/költség számítása:
L: utolsó réteg
$a_n^{(L)}$: kimeneti eredmény
y_n: elvárt eredmény

$$C = \sum_{i=n}^{i=0} (a_i^{(L)} - y_i)^2 = (a_1^{(L)} - y_1)^2 + (a_2^{(L)} - y_2)^2 + \ldots + (a_n^{(L)} - y_n)^2$$

Számolás menete:

Számoláshoz rengeteg módszer alkalmazható, a konkrét módszereket általában a feladathoz igazítják és optimalizálják.
Lényege a számolásnak hogy a költség függvény legjobban közelítsen a nullához. Egy $w + b$ elemszámú függvénynek kellene keresni a nullához közeli értékét (általában elképzelhetetlenül sok, több ezer dimenzió). $C(w_n, \ldots, w_n, b_0 \ldots b_m) \approx 0$

Jelölések:

- Parciális deriválás, jele: $\frac{\partial f}{\partial x}$
- Nabla operátor (a vektort különböző elemi mentén parciálisan deriváljuk), jele: ∇

Feladatunk:

Hagyományosan

1. Kiszámolni $\nabla C(\ldots)$
2. $-\ \nabla C(\ldots)$ irányába lépni egy értékkel (tanulási rátával)

Tovább gyorsítható ha:

1. Több “kis köteg”-re (“mini batch”) számoljuk ki a $\nabla C(\ldots)$
2. Kiszámolt ∇C értékeket átlagoljuk

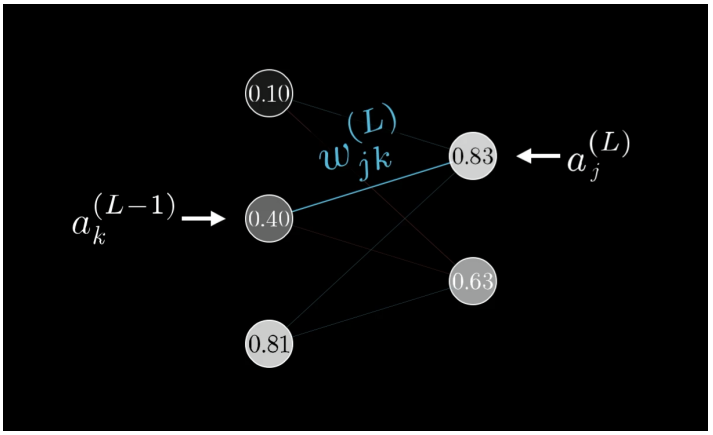
3. Átlag – ∇C értékkel lépünk a megfelelő irányba (tanulási rátával)

Teljes képlet:

Gradiens vektor:

$$\nabla C = \begin{bmatrix} \frac{\partial C}{\partial w^{(1)}_{00}} \\ \frac{\partial C}{\partial b^{(1)}_{00}} \\ \vdots \\ \frac{\partial C}{\partial w^{(L)}_{jk}} \\ \frac{\partial C}{\partial b^{(L)}_{lm}} \end{bmatrix}$$

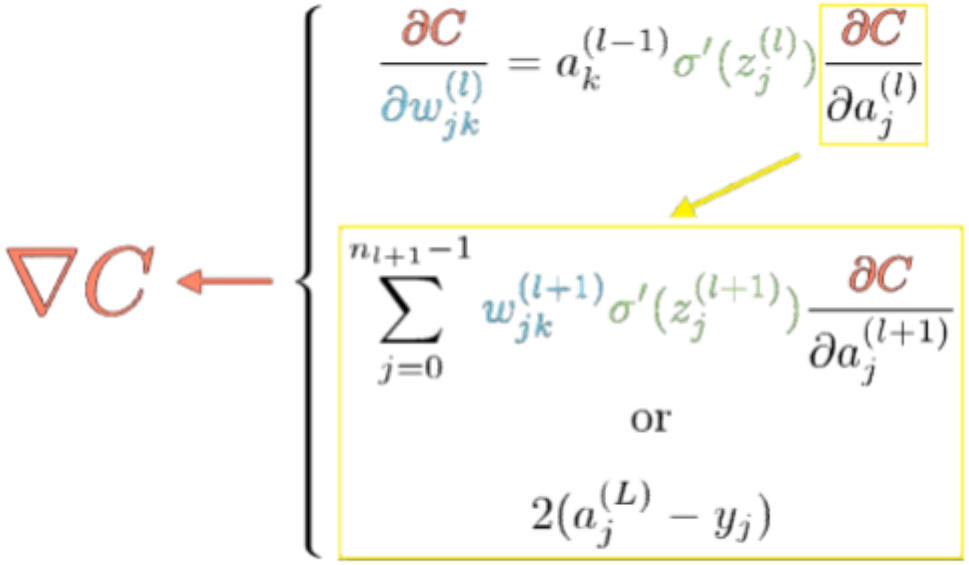
k: előző oszlopban lévő neuron
j: következő oszlopban lévő neuron



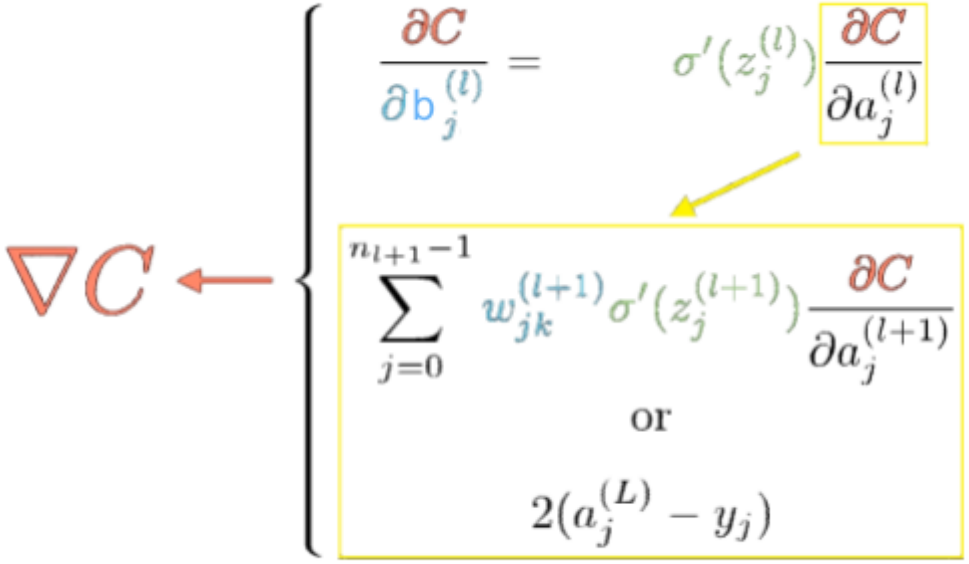
Külön z-vel jelöljük neuron értékét sigma függvény nélkül:

$$z_j^{(L)} = w_{j0}^{(L)} a_0^{(L-1)} + w_{j1}^{(L)} a_1^{(L-1)} + w_{j2}^{(L)} a_2^{(L-1)} + b_j^{(L)}$$

Súly értékének kiszámítása (láncszerűen épül fel az utolsó neuronig):

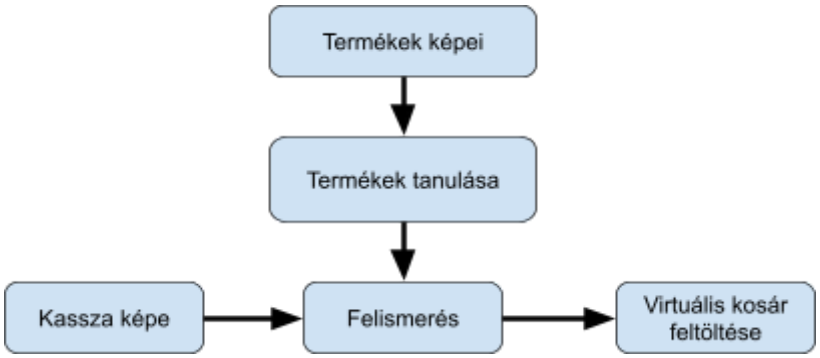


Torzító (bias) értékének kiszámítása (láncszerűen épül fel az utolsó neuronig):



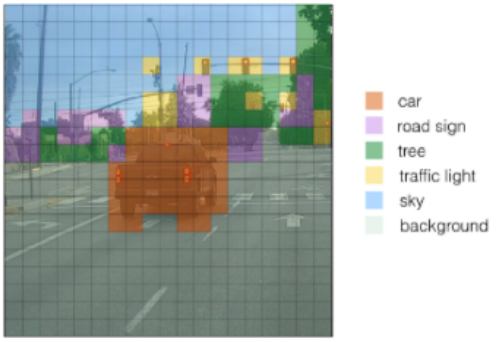
Megvalósítás:

Programozási környezetnek JavaScript-et választottam nagy kompatibilitási és hordozhatósági képessége miatt. A program “Single Page” (egylapos) applikáció lesz.
Korábban szerzett összesített adatokkal el kezdjük elemezni a külső kamera képét és amelyik terméknél elég nagy egyezőséget tapasztalunk azt berakjuk a virtuális kosárba.



Tensorflow (neutrális háló):

Az tárgyak felismerésére tensorflow keretrendszert használok. YOLO R-CCN algotimus logikáját fogja használni a képfelismerő. YOLO - You Only Look Once (egyszer nézheted meg) algoritmus egy adott képből (nem pedig képfolyamból) állapítja meg, hogy mit tartalmaz. R-CNN (Region-based Convolutional Neural Network - Régió alapú Konvolúciós Neurális hálózat) egy mély konvolúciós hálózatot jelöl ami a képet részekre bontva elemzi és a találati helyeket közelíti egymáshoz.
kép felosztása:

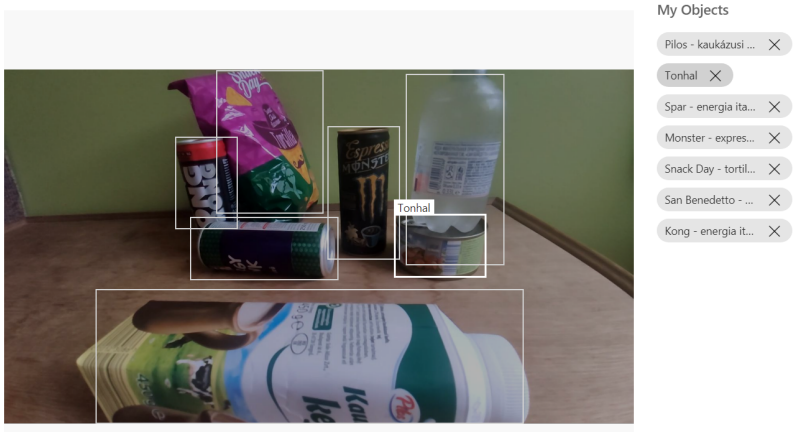


Melyik doboz mennyire tartalmazza a felismert tárgyat:

Az egymást fedő dobozok összeillesztése

Custom Vision (tanítás folyamata):

A tárgyakról sok különböző képet kell készíteni. Minél változatosabbak a képek annál többféle szituációban lesz képes felismerni a program a termékeket. Különböző napszakokban, helyen, pozícióban és környezetben is szerepeljenek a képek. A képek mennyiségére igaz, hogy minél több annál jobb de minimálisan legalább érdemes 50-60 képet egy tárgyról. Tanításhoz a Microsoft Custom Vision szolgáltatását használom, ez tanításhoz szükséges erőforrásokat és szoftvereket is biztosítja.
A tanításhoz nem csak a képeket kell biztosítanunk, hanem hogy a képen milyen objektum és hol szerepel.



Ha megvan egy kritikus szint (20-30 kép), akkor már a tanítás elkezdhető. A kezdeti tanítással a további képfelvitel sebessége növelhető mert a rendszer előre fel fogja ismerni a tárgyak egy részét.
Custom Vision-be maximális tanulásra szánt processzoridőt adhatjuk meg. A tanulás hatékonysága logaritmikus szerűen egyre csökken így nem feltétlenül fogja a program a megadott időt teljesen felhasználni (nem lenne érdemi javulás).
Tanítás végeztével a modell beállítások és a súly fájlok letölthetőek és beilleszthetőek a tensorflow.js keretrendszerbe.

Egyéb fájlok:

- cvexport.manifest: exportálás körülményeit, adatait, ellenőrző összegét tartalmazza.
- labels.txt: betanított címkéket tartalmazza
- LICENCE: mellékelt licensz leírás

Model:

metadata_properties.json

A használt algoritmus. Jelen esetben ez YOLO, leírja:

- torzító (bias) értékeket
- Kép vágási metodikát
- Méretezést
- Feldarabolási méretet (itt 512x512)
- Használt színteret (RGB8)

model.json

A tensorflow által biztosított környezet beállításait tartalmazza. Ez alapján fogja a tensorflow létrehozni a rétegeket, neuronokat, súlyokat, bemenő rétegeket, kimenő rétegeket

weights.bin

A model.json-ban hivatkozott súlyokat és torzító elemeket tartalmazza.

Saját API dokumentációja:

Kosár függvények:

- setProduct(name, count, price, unit): Termék beállítása a kosárban
    - name : string - A termék neve
    - count : integer - a termék darabszáma
    - price : integer - a termék ára (opcionális)
    - unit : string - egység (db, kg stb.) (opcionális)
    - Visszatérési érték: Ha sikerült a végrehajtás akkor igaz különben hamis
- addProduct(name, count) : Termékszám módosítása a kosárba
    - name : string - A termék neve
    - count : integer - a hozzáadandó darabszám
    - Visszatérési érték: Ha sikerült a végrehajtás akkor igaz különben hamis

Videó függvények:

- VIDEO: videó HTML objektum elérés

- async listVideo(): Kilistázza a videóforrásokat
    - Visszatérési érték: Promise objektumot ad vissza, utána igaz ha sikeres a listázás különben hamis
- async setVideo(index): beállítja a megadott videó forrást, listázás után elérhető
    - index : integer - videó forrás indexe
    - Visszatérési érték: Promise objektumot ad vissza, utána igaz ha sikeres a beállítás különben hamis
- async setVideoFile(): betölt egy kiválasztott videó fájlt és beállítja forrásként.
    - Visszatérési érték: Promise objektumot ad vissza, utána igaz ha végzett a metódus
- removeVideo(): eltávolítja az aktuális videóforrást és felszabadítja a lefoglalt memóriát
- async startVideo(): Videó első indítása (inicializálása), ellenőrzi a támogatást, betölti a függőségeket, kilistázza a videókat és a legutolsó forrásra állítja
    - Visszatérési érték: Promise objektumot ad vissza, igaz ha sikeres a betöltés különben hamis
- drawCanvas(name, x, y, width, height): Kirajzol egy dobozt az objektum nevével
    - name: string - Objektum neve
    - x:integer - X koordináta (bal felső)
    - y: integer - Y koordináta (bal felső)
    - width: integer - objektum szélessége
    - height: integer - objektum magassága
- clearCanvas(): törli az összes kijelzett objektumot

Objektum detektálás:

- async loadWorker(): tárgyfelismerés betöltése
    - Visszatérési érték: Promise objektumot ad vissza, igaz ha sikeres a betöltés különben hamis
- async detect(): tárgyfelismerés, a VIEWCART objektum feltöltése a látott tárgyakkal
    - Visszatérési érték: Promise objektumot ad vissza
- startDetection(): detektálás elindítása
- stopDetection(): detektálás leállítása

Tesztelés:

Videó forrás kiválasztása után betölthetővé válik egy előre felvett teszt videó. Videón interaktívan, éles környezethez hasonló módon lehet tesztelni.
A tanítás során a termékekhez minőségi mutatókat is lehet rendelni, ezzel megmondhatjuk a tanítás minőségét az adott termékre:

K: $képek\ száma$
$T(t)$ Termék összes megjelenése (t: termék)
$COT(t,\ k)$ Helyes találat és kategorizálás (t: termékre, k:képnél)
$CO(t,\ k)$ Helyes találat (t: termékre, k: képnél)
$F(t,\ k)$ Találat (t: termékre, k: képnél)

- Recall (újrahívás): Az összes találatból hány százalékot talál el a modell helyesen:

$$R(t) = \frac{\sum_{k=K}^{k=0} COT(t,k)}{T(t)} \times 100$$

- Precision (precizitás): Ha megtalálta helyesen objektumot a modell az mennyi esetben kategorizálta be helyesen:

$$R(t) = \frac{\sum_{k=K}^{k=0} COT(t,k)}{\sum_{k=K}^{k=0} CO\ (t,k)} \times 100$$

- Mean average precision (átlagos precizitás): Az objektum detektálás pontosságát méri:

$$P(t) = \frac{\sum_{k=K}^{k=0} CO(t,k)}{\sum_{k=K}^{k=0} F(t,k)} \times 100$$

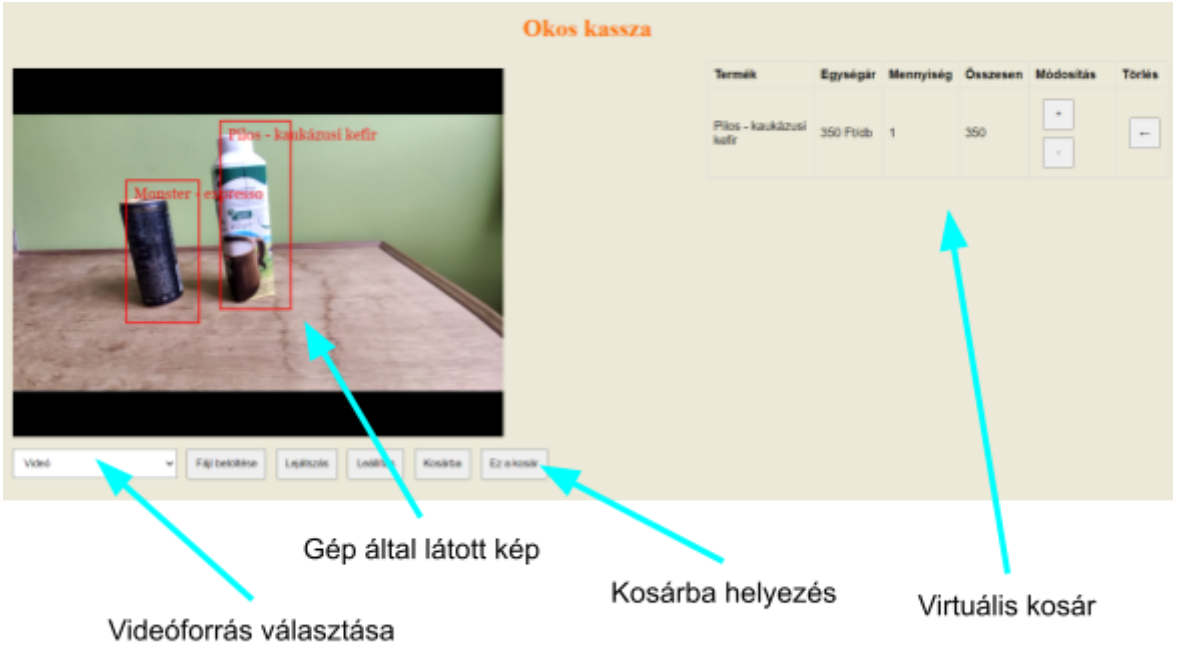
Az azonos mutatószámok átlagolásával az egész modellre is kifejezhetjük a pontosságot.

Termék	**Precizitás**	**Újrahívás**	**Átlagos Precizitás**
Kong - energia ital (piros)	100.0%	86.7%	100.0%
Monster - expresso	100.0%	86.7%	100.0%
Pilos - kaukázusi kefir 91.7%	91.7%	91.7%	98.8%
San Benedetto - ásványvíz	100.0%	75.0%	100.0%
Snack Day - tortilla (BBQ)	100.0%	63.6%	88.5%
Snack Day - tortilla	100.0%	100.0%	100.0%

Termék	Precizitás	Újrahívás	Átlagos Precizitás
(édes chili)			
Spar - energia ital (lila)	100.0%	91.7%	97.5%
Tonhal	100.0%	92.3%	100.0%
Összesen	**98.8%**	**86.7%**	**98.1%**

Felhasználói leírás:

A program betöltése után a felhasználó a kamera képét és mellette vagy alatta a kosár tartalmát láthatja. Kamera képe alatt egy legördülő menüből lehet kiválasztani a program által fogadott kép forrását.
A kép forrásból folyamatosan próbálja a program felismerni a tárgyakat (ez a gép sebességétől függően 2-6 másodperc). Az aktuálisan látott objektumokat a bekeretezve és feliratozva mutatja
A “Kosárba” gomb megnyomásával a termékeket a virtuális kosarunkhoz adjuk, “Ez a kosár” gombbal pedig csak a látható termékek lesznek a kosárba.



Irodalomjegyzék:

Képek: https://pixabay.com/hu/

YOLO:
https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e
https://towardsdatascience.com/yolo-you-only-look-once-3dbdbb608ec4
Neutrális hálózat: https://www.3blue1brown.com/lessons/neural-networks,
https://www.3blue1brown.com/lessons/gradient-descent,
https://www.3blue1brown.com/lessons/neural-network-analysis,
https://www.3blue1brown.com/lessons/backpropagation,
https://www.3blue1brown.com/lessons/backpropagation-calculus
W3 School példa: https://www.w3schools.com/ai/ai_training.asp
Tensorflow playground: https://playground.tensorflow.org/
Tensorflow.js: https://www.tensorflow.org/js/models
Custom vision:
https://learn.microsoft.com/en-us/azure/cognitive-services/Custom-Vision-Service/overview