# Okos önkiszolgáló kassza

# Tartalomjegyzék

# 1. 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 termékek találhatóak. A termékek QR-kóddal vannak ellátva, amivel az adott termék beazonosítható. A beadandó keretében nem QR-kód, hanem a tárgy kinézete alapján fogja beazonosítani a terméket a program. Ez a módszer hatékony kiegészítő módszer lehet, ha a terméken lévő QR-kód megsérül de akár gyorsabban is elvégezhető vele több termék beolvasása. Program feladata egy kép alapján eldönteni, hogy milyen termékek találhatók a kasszapulton. Minimum 6 fajta termék felismerése a cél.


1. ábra: Tárgyak felismerése a kasszánál

Egy kassza kezelő program létrehozása lenne a célom, ahol a felhasználó látja a kamera látóterét, illetve beolvasott termékeket. Tervezek egy opcionális forrás választó gombot a megfelelő képforrás kiválasztásához.

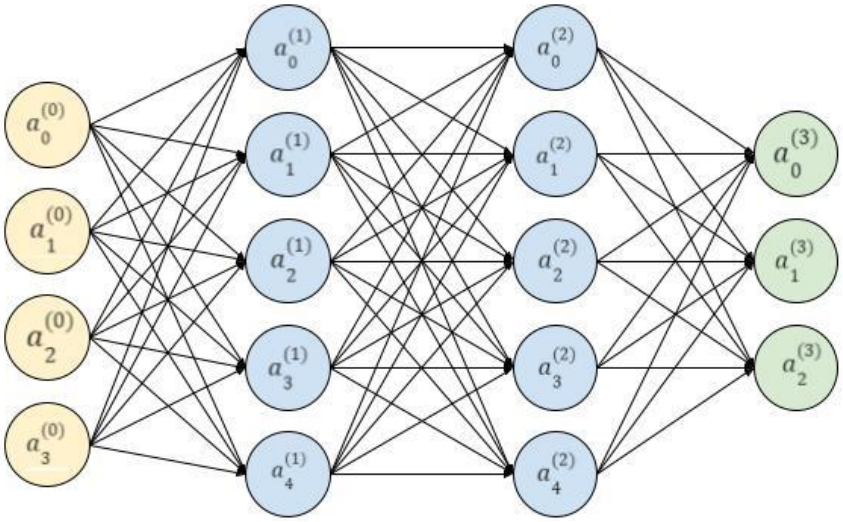
A termékek adatai a programban előre definiálva vannak, csak a definiált tárgyakat tudja felismerni a program.

# 2. Elméleti háttér

## 2.1. Fogalmak

- **Mesterséges intelligencia** (Artificial Intelligence) [1]: Bármilyen gép általi reakció ami emberi intelligenciát „utánozza”, képes saját döntést hozni a körülményeket figyelembe véve. Pl. sakkozó program, fordító program.
- **Gépi tanulás** (Machine learning) [2]: Mesterséges intelligencia olyan megvalósítása ahol a program magát tanítja adatok és tapasztalások alapján. Például: Mozgási minták keresése több ezer órányi felvételből.
- **Mélytanulás** (Deep Learning) [3] : Gépi tanuláson belül olyan módszer ahol neuron alapú a program működése (mint az emberi agy). Ez a módszer akár bonyolultabb problémák megoldására is használható. Pl. Több tárgy, helyzet azonosítása egy kép alapján.

## 2.2. Neurális háló




2. ábra: Neurális háló vizuális ábrázolása

Neurális háló felépítése a következő:

- Neurális hálókban a neuronok (pontok) egy-egy értéket (számot) jelentenek, ezeket oszlopokba rendezzük.
- 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 az azt megelőző oszlop elemeiből számoljuk ki (az első oszlop adott).

Neuron értékeinek számítása:

- $l$: 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 $m$)
- Az aktivációs függvény a példában: $\sigma = f(x) = \frac{1}{1+e^{-x}}$: szigmoid függvény lesz. Több fajta aktivációs függvény létezik, általában egyenirányító (ReLU) $ReLU(x) = \left\{ \begin{matrix} 0, & x<0 \\ x, & különben \end{matrix} \right\}$ függvényt használnak gyorsasága és egyszerűsége miatt.

Röviden a neuronok kiszámításának képlete:

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

Vektorosan felírva a neuronok kiszámításának képlete:

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

Egy neuronra felírva:

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

## 2.3. Tanítás folyamata

Neurális pontok kiszámítása adott, csak súlyokat és torzító értékeket lehet változtatni. Súlyokat és torzító értékeket tapasztalások útján lehet meghatározni. A tanítás során meg kell adni egy bemenetet és az elvárt kimeneti értékeket. Egy neuron érték tehát az alábbi módon változtatható:

- Változtatható a torzító (bias) értéket: $b$
- Változtatható a súly értékeket: $w_i$
    - „Fire together, wire together" szabály: Ahol az erős kapcsolat van neuronok (többiekhez képest erős) között akkor a gyengébb kapcsolatokat kevésbé kell csökkenteni, az erős kapcsolat súlyaira kell összpontosítani
- Változtatható az előző neuron értékét: $a_i$
    - Közvetlen nem tudjuk változtatni csak súly ($w_i$) és torzító értékeket tudunk változtatni ($b$)

Számítógépnek meg kell mondani, hogy mennyit rontott a háló és melyik neuron értékén mennyit kellene javítani a jobb eredményhez. Végső veszteség/költség függvény megmondja, hogy az algoritmus mennyit rontott. Ez önmagában nem ad sok információt, viszont különböző súly és torzító értékeket össze lehet vetni, hogy melyik ad jobb eredményt. Költség érték kiszámítási módja:

$L$: utolsó réteg

$a_n^{(L)}$: kimeneti eredmény

$y_n$: elvárt eredmény

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

A súly és torzító értékek 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 kell keresni a nullához közeli értékét (általában elképzelhetetlenül sok, több ezer dimenzió) $C(w_n,\ldots, w_n, b_0 \ldots b_m) \approx 0$.

Egyéb műveletek:

- Parciális deriválás, jele: $\frac{\partial f}{\partial x}$

- Nabla operátor (a vektort különböző elemi mentén parciálisan deriváljuk), jele: $\nabla$

A tanítás során ki kell számolni, hogy mely súly és torzító értékek kombinációja adja a legkisebb értéket a költség függvényre. A minimum érték következőképpen közelíthető:

1. Ki kell számolni a $\nabla C(\ldots)$-t
2. $-\nabla C(\ldots)$ irányába kell léptetni a súlyokat és torzító értékeket (tanulási rátával)

Tovább gyorsítható a folyamat, ha:

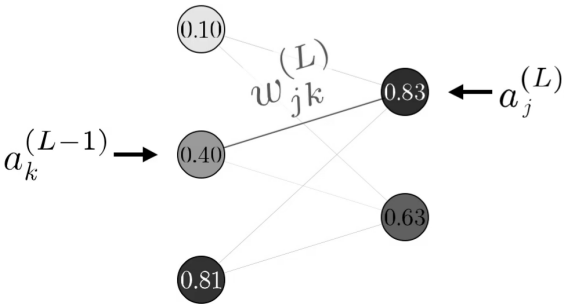
1. Több „kis köteg” -re („mini batch”) számoljuk ki a $\nabla C(\ldots)$
2. Kiszámolt $\nabla C$ értékeket átlagoljuk
3. Átlag $-\nabla C$ értékkel lépünk a megfelelő irányba (tanulási rátával)

A végső képlethez a következő rövidítéseket és jelöléseket használtam:

- z-vel rövidítem az aktivációs függvényen belüli képletet:

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

- Súlyok két réteg között helyezkednek el, vagyis az első (vagyis 0.) rétegnek nem lesznek súlyai. $L$ a réteget jelöli, $j$ a réteg neuronját, $k$ az előző réteg neuronját




3. ábra: Példa a súlyok és neuronok jelöléséhez

$\nabla C$ tartalmazza költségfüggvény súlyok és torzító értékek szerinti deriváltjait:

$$\nabla C = \begin{bmatrix} \frac{\partial C}{\partial w^{(1)}_{0\,0}} \\ \frac{\partial C}{\partial b^{(1)}_{0}} \\ \vdots \\ \frac{\partial C}{\partial w^{(L)}_{j\,k}} \\ \frac{\partial C}{\partial b^{(L)}_{j}} \end{bmatrix}$$

Súly értékek szerinti parciális derivált a következőképpen számolható ki (rekurzívan láncszerűen épül fel a képlet az utolsó rétegig) [4]:

$$\frac{\partial C}{\partial w^{(l)}_{jk}} = a^{(l-1)}_{k} \sigma'(z^{(l)}_{j}) \frac{\partial C}{\partial a^{(l)}_{j}}$$

$$\sum_{j=0}^{n_{l+1}-1} w^{(l+1)}_{jk} \sigma'(z^{(l+1)}_{j}) \frac{\partial C}{\partial a^{(l+1)}_{j}}$$

or

$$2(a^{(L)}_{j} - y_j)$$

Torzító (bias) értékek szerinti parciális derivál a következőképpen számolható ki (rekurzívan láncszerűen épül fel a képlet az utolsó rétegig):

$$\frac{\partial C}{\partial b_j^{(l)}} = \sigma'(z_j^{(l)}) \frac{\partial C}{\partial a_j^{(l)}}$$

$$\sum_{j=0}^{n_{l+1}-1} w_{jk}^{(l+1)} \sigma'(z_j^{(l+1)}) \frac{\partial C}{\partial a_j^{(l+1)}}$$

or

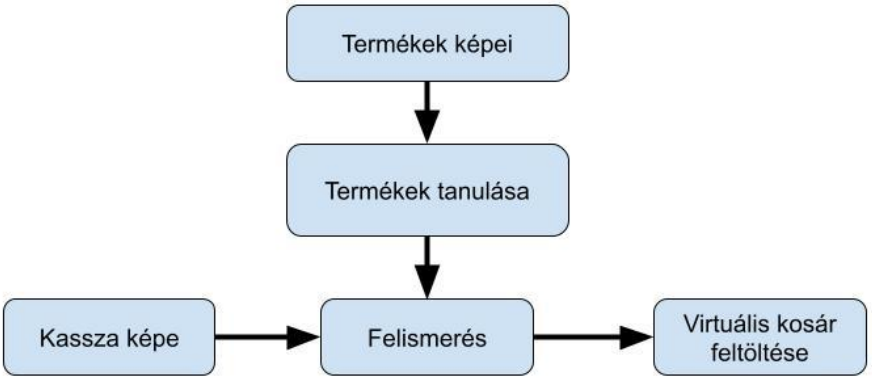
$$2(a_j^{(L)} - y_j)$$

# 3. Megvalósítás

## 3.1. Választott környezet

Programozási környezetnek JavaScript-et választottam nagy kompatibilitási és hordozhatósági képessége miatt. A program „Single Page” („egyoldalas”) applikáció lesz.

A program elemezni kezdi a kamera látóterét, majd ha elég nagy egyezőséget tapasztal valamelyik előre definiált termékkel akkor az észlelt tárgyat egy befoglaló geometriával látja el. Ha a felhasználó jóváhagyja a felismerést akkor a virtuális kosárba kerül a termék.




3. ábra: Programlogika folyamatábrája

## 3.2. Adatkészlet

A tanításhoz meg kell adni egy adatkészletet, amiből a program megismerheti a tárgyakat. Saját adatkészletet fogok használni, mert konkrét tárgyakat szeretnék felismertetni egy konkrét környezetben. A felismerendő tárgyakat a Lidl bevásárlóközpontból vettem, a kasszás pult pedig a szobámban lévő asztal lesz. A tárgyakról sok különböző képet kell készítettem. Minél változatosabbak a képek annál többféle szituációban lesz képes felismerni a program a termékeket. Fontos, hogy különböző helyen és pozícióban is szerepeljenek a képek.

4. ábra: üres „kasszapult”

5. ábra: termékek különböző helyzetekben, pozíciókban

A képkészítéshez egy Xiaomi Mi 9T telefont használtam. Készítettem pár darab 4208 széles és 3120 képpont magas (4K) képet azonban ezek túl lassan kezelhetőnek és nagyméretűnek bizonyult ezért az adatkészlet további képeit nem egyesével készítettem, hanem 1920 képpont széles és 1080 képpont magas (full hd) felbontású 30 képkocka per

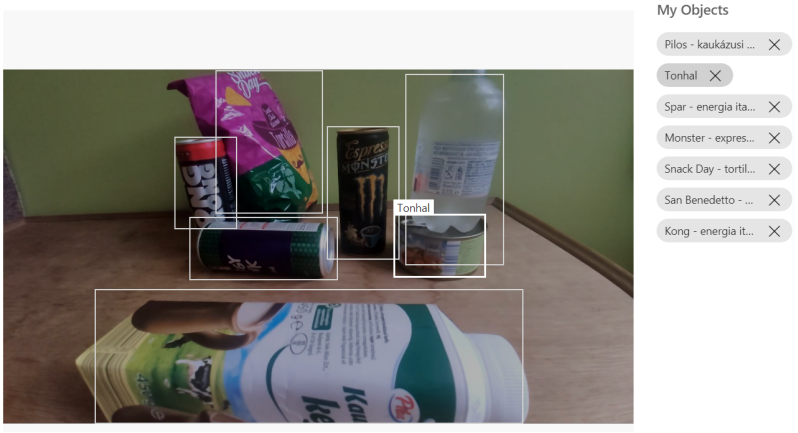
másodperc sebességű videón rögzítettem a tárgyak mozgatását, majd utólag különböző időpontokról pillanatképet készítettem. A képek készítését addig folytattam, amíg a tanítás pontossága el ben érte a 80%-ot (három részletben történt a képfeltöltés mire elértem a kívánt pontosságot). Összesen 113 kép készült, ezeken a

- Kong energiaital (piros) 65 helyen,
- Monster expresso 64 helyen,
- Pilos kaukázusi kefír 58 helyen,
- San Benedetto ásványvíz 38 helyen,
- Snack Day tortilla (BBQ) 52 helyen,
- Snack Day tortilla (édes chili) 65 helyen,
- Spar energiaital (lila) 58 helyen,
- Tonhal 61 helyen

szerepel.

## 3.3. Azure: Custom Vision

Tanításhoz a Microsoft Custom Vision [5] szolgáltatását használom, ez a tanításhoz szükséges erőforrásokat és szoftvereket is biztosítja. Meg kell adni a felismerendő objektumokat majd a feltöltött képeken be kell jelölni, hogy milyen objektum hol szerepel.




9. ábra: kép kategorizálás felület a Custom Vision-ben

Ha feltöltés és bejelölés elér egy kritikus mennyiséget (20-30 kép), akkor kezdhető a tanítás. Egy tanított modell segítségével a további képek kategorizálása felgyorsítható mert a rendszer előre fel fogja ismerni a tárgyak egy részét.

A tanítás elkezdésekor a Custom Vision-be beállíthatom a tanulásra szánt maximális processzoridőt (általában 4 óra). A tanulás hatékonysága logaritmikus szerűen nő így nem feltétlenül lesz kihasználva a megadott idő (mert nem lenne érdemi javulás).

Tanítás végeztével a modell beállítások és a súly fájlok letölthetőek és beilleszthetőek a tensorflow.js keretrendszerbe. A következő fájlok kerülnek letöltésre:

- **cvexport.manifest**: exportálás körülményeit, adatait, ellenőrző összegét tartalmazza.
- **labels.txt**: betanított címkéket tartalmazza
- **LICENCE**: mellékelt licensz leírás
- **metadata_properties.json**: A használt módszer beállításának leírása.
- **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.

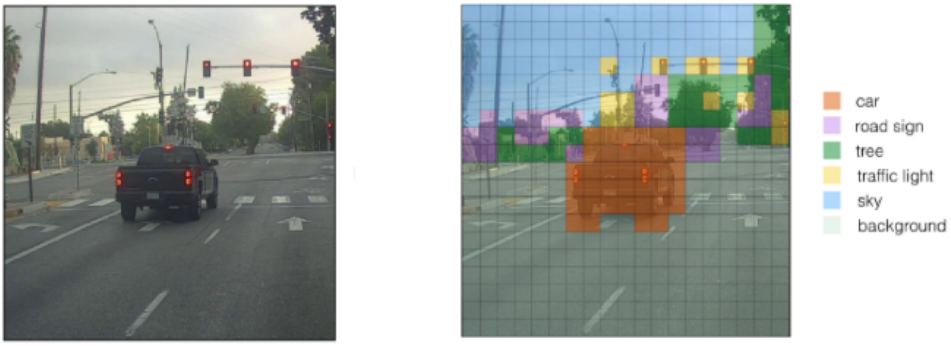
## 3.4. Tensorflow.js

A tárgyak felismeréséhez tensorflow [6] keretrendszert használok. Keretrendszer függvényeket biztosít amik, segítségével felépíthető saját modellem. A tensorflow.js és kiegészítő algoritmusai külön szálon fognak futni. A fő program a képi adatokat szolgáltatja a model pedig a képeken található információkat adja vissza egy szöveges JSON adatszerkezetben.

## 3.5. YOLO

### 3.5.1 Elmélet

YOLO R-CCN [7] logikáját fogja használni a képfelismerő. YOLO - You Only Look Once („egyszer nézheted meg”) módszer 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 A kép felosztását a 4. ábra szemlélteti. Felosztott rácsban felismert kategóriák általában fedik egymást amit a 5. ábra mutat.




4. ábra: YOLO módszer által felosztott és kategorizált kép

5. ábra: YOLO által felismert kategóriák összevonás előtt

Az egymást fedő dobozok összeillesztésével meghatározható a tárgy befoglaló geometriája.


6. ábra: YOLO felismerés összevonás előtt és utáni állapot összehasonlítása

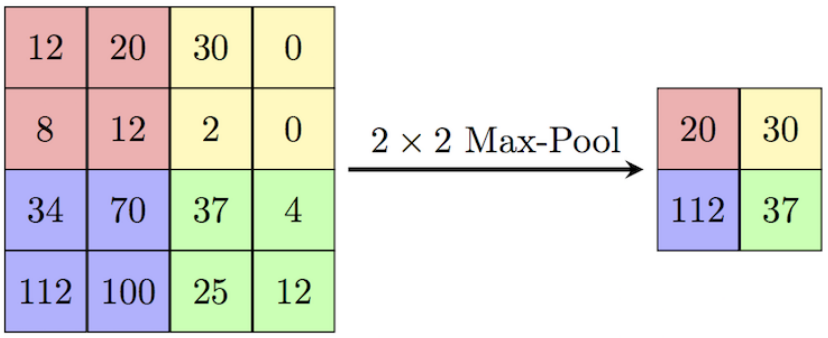
### 3.5.2 Model

A modell végső modell a következőképpen épül fel:

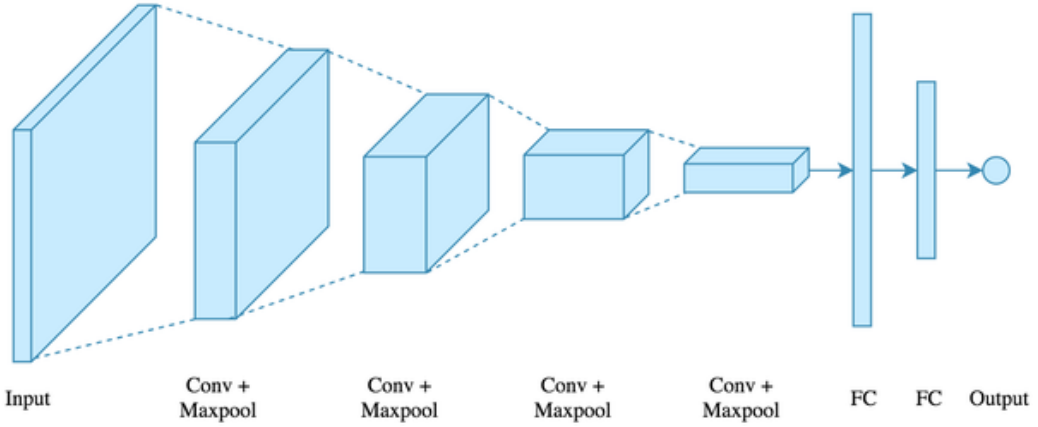
1. 3 színcsatornás $416 \times 416$ felbontású kép lesz a bemeneti adat (ha a kép nagyobb akkor feldarabolásra kerül a kép).
2. Bemenet (kép) széleinek kiegészítése felül-alul 4, jobb-bal oldalon 2 értékkel (ne legyen túlfutás)
3. 3 x 3-as kernellel konvolúció végrehajtása különböző először 3 bemeneti csatornával 16 kimeneti csatornával. (LeakyRelu függvény alkalmazása a neurális hálóban: $max(v,\ v \times 0.100000014901)$
    a. a szűrő kétdimenziós (2D) tömbbé alakítása, ahol az
        i. első dimenzió nagysága a
        $$szűrő\ magassága \times szűrő\ szélessége \times bemeneti\ csatornák$$
        ii. második dimenzió a kimeneti csatornák számával egyenlő
    b. új virtuális négydimenziós (4D) tömb létrehozása a képből ahol:
        i. első dimenzió a batch (köteg)
        ii. második dimenzió nagysága a kimeneti magasság
        iii. harmadik dimenzió nagysága a kimeneti szélesség
        iv. negyedik dimenzió
        $$szűrő\ magassága \times szűrő\ szélessége \times bemeneti\ csatornák$$
    c. virtuális tömb feltöltése NHWC [8] (bemeneti kép és szűrők szorzatösszege) módszerrel
        i. kimenet dimenziói: $b,\ i,\ j,\ k$
        ii. szűrő dimenziói: $di,\ dj,\ q$
        iii. léptetéseket tartalmazó tömb: $strides\ =\ [1,\ stride,\ stride,\ 1]$
        iv. Kimenet a bemeneti kép és szűrők szorzatösszege lesz
        $$output[b,\ i,\ j,\ k]\ = \\ sum\_\{di,\ dj,\ q\}\ input[b,\ strides[1]\ \times\ i\ +\ di,\ strides[2]\ \times\ j\ +\ dj,\ q]\ \times \\ filter[di,\ dj,\ q,\ k]$$
4. MaxPool alkalmazása (legnagyobb érték kiválasztása a vizsgált területből) 1 x 2 x 2 x 1-es kernel-el
5. A 2. pont ismétlése, de a 3. pontban a kimeneti csatornát az előző duplájára növeljük. Addig ismételjük a folyamatot míg el nem érjük 1024 kimeneti csatornát.

6. Két konvolúció keretében 1024-ről 512-re majd 512-ről 64-re csökkentjük a csatornák számát
7. Megkapjuk az egyes képrészlet milyen tárgyat tartalmaz.

Folyamatosan közelíteni fog a felismert tárgyakhoz.




7. ábra: Max-Pool szemléltetése egy példán keresztül




8. ábra: YOLO felismerés folyamata

## 3.6. Program API

### 3.6.1. 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

### 3.6.2. 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

### 3.6.3. 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

# 4. 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 megmondható a tanítás minősége 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 kifejezhető a pontosság.

1. táblázat: Model által elért eredmények a feltöltött képek alapján

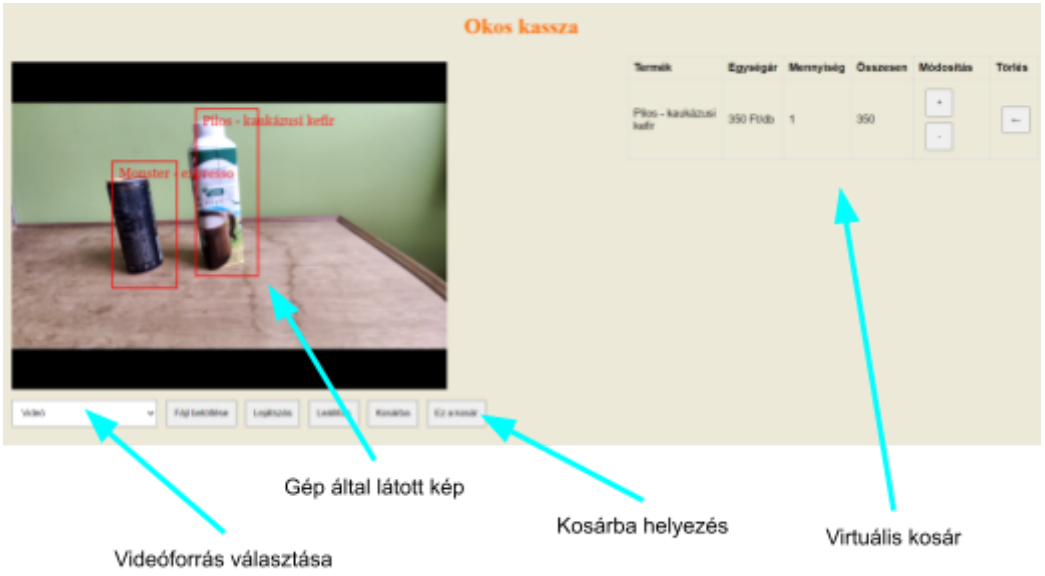
| Termék | Precizitás | Újrahívás | Átlagos Precizitás |
|---|---|---|---|
| Kong - energia ital (piros) | 100.0% | 86.7% | 100.0% |
| Monster - expresso | 100.0% | 86.7% | 100.0% |
| Pilos - kaukázusi kefir | 91.7% | 91.7% | 98.8% |
| San Benedetto - ásványvíz | 100.0% | 75.0% | 100.0% |
| Snack Day - tortilla (BBQ) | 100.0% | 63.6% | 88.5% |
| Snack Day - tortilla (édes chili) | 100.0% | 100.0% | 100.0% |
| Spar - energia ital (lila) | 100.0% | 91.7% | 97.5% |
| Tonhal | 100.0% | 92.3% | 100.0% |
| Összesen | 98.8% | 86.7% | 98.1% |

# 5. Felhasználói dokumentáció

A program futtatásához WebGL 2.0-t támogató böngésző szükséges. A betöltés 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 befoglaló geometria és felirat jelzi.

A „Kosárba” gomb megnyomásával a termékek a virtuális kosárba helyeződnek, „Ez a kosár” gombbal pedig csak a látható termékek lesznek a kosárba.




10. ábra: Elkészített program működés közben

# Irodalomjegyzék


[1] Mesterséges intelligencia https://atozofai.withgoogle.com/intl/hu/artificial-intelligence/, 2022.11.18 19:58

[2] Gépi tanulás https://atozofai.withgoogle.com/intl/hu/machine-learning/, 2022.11.18 19:58

[3] Neurális hálózatok https://atozofai.withgoogle.com/intl/hu/neural-networks/, 2022.11.18 19:58

[4] Backpropagation calculus https://www.3blue1brown.com/lessons/backpropagation-calculus, 2022.11.18 19:58

[5] Microsoft Custom Vision https://learn.microsoft.com/en-us/azure/cognitive-services/Custom-Vision-Service/overview, 2022.11.18 19:58

[6] Pang, B., Nijkamp, E., & Wu, Y. N. (2020). Deep Learning With TensorFlow: A Review. Journal of Educational and Behavioral Statistics, 45(2), 227–248. https://doi.org/10.3102/1076998619872761

[7] R. Huang, J. Pedoeem and C. Chen, "YOLO-LITE: A Real-Time Object Detection Algorithm Optimized for Non-GPU Computers," 2018 IEEE International Conference on Big Data (Big Data), 2018, pp. 2503-2510, doi: 10.1109/BigData.2018.8621865.

[8] NHWC https://www.tensorflow.org/api_docs/python/tf/nn/conv2d, 2022.11.18 19:58