**Okos önkiszolgáló kassza**

# 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 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.



1. ábra: Tárgyak felismerése a kasszánál

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

# 2. Elméleti háttér

## 2.1. 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).

## 2.2. Neurális háló



2. ábra: Neurális háló vizuális ábrázolása

Neurális háló felépítése a következő:

* Neurális hálókban a neuron (pontok) egy értéket (számmal) jelentenek, 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 (az első oszlop adott).

Neuron értékeinek számítása:

* : réteg
* : előző oszlop elemei ( első elem, második elem, stb.)
* : számolandó oszlop elemei ( első elem, második elem, stb.)
* : számolandó oszlop torzító (bias) elemei
* : súlyvektor megadja melyik él milyen súllyal bírjon a számolandó sorba (oszlop sora )
* : szigmoid függvény, általában optimalizáció miatt lecserélik egy egyenirányító (ReLU) függvényre:

Röviden:

Vektorosan felírva

Egy neuronra felírva:

## 2.3. 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:
* Változtathatjuk a súly értékeket:
  + “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:
  + 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 () és torzító értékeket tudunk változtatni ()

Végső veszteség/költség függvény pontosan megmondja hogy az algoritmus mennyit rontott. Ez önmagában nem ad sok információt, viszont különböző súly és torzító értékeket össze tudunk vetni, hogy melyik a jobb.

: utolsó réteg

: kimeneti eredmény

: elvárt eredmény

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 elemszámú függvénynek kellene keresni a nullához közeli értékét (általában elképzelhetetlenül sok, több ezer dimenzió) .

Jelölések:

* Parciális deriválás, jele:
* Nabla operátor (a vektort különböző elemi mentén parciálisan deriváljuk), jele:

Feladatunk általánosságba:

1. Kiszámolni
2. irányába lépni egy értékkel (tanulási rátával)

Tovább gyorsítható a folyamat ha:

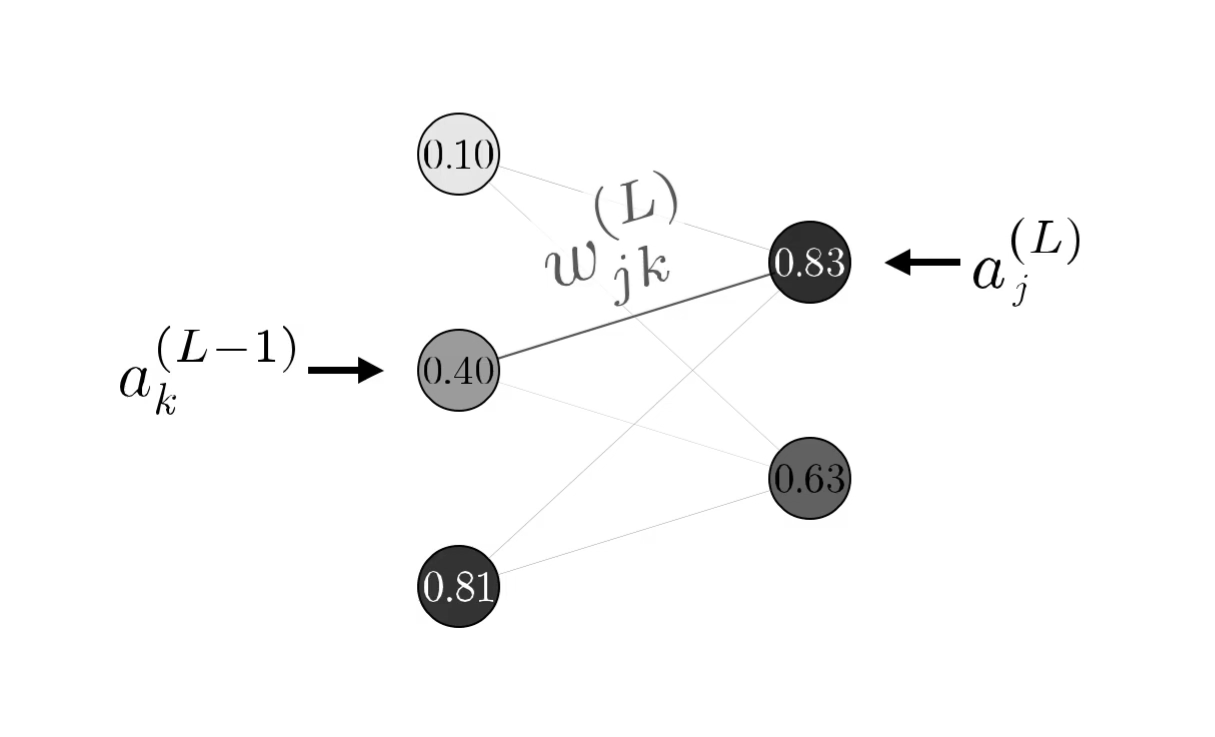
1. Több “kis köteg”-re (“mini batch”) számoljuk ki a
2. Kiszámolt értékeket átlagoljuk
3. Átlag értékkel lépünk a megfelelő irányba (tanulási rátával)

Gradiens vektor:



: előző oszlopban lévő neuron

: következő oszlopban lévő neuron



Külön z-vel jelöljük neuron értékét sigma függvény nélkül:



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



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



# 3. Megvalósítás

## 3.1. Választott környezet

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.



3. ábra: Programlogika folyamatábrája

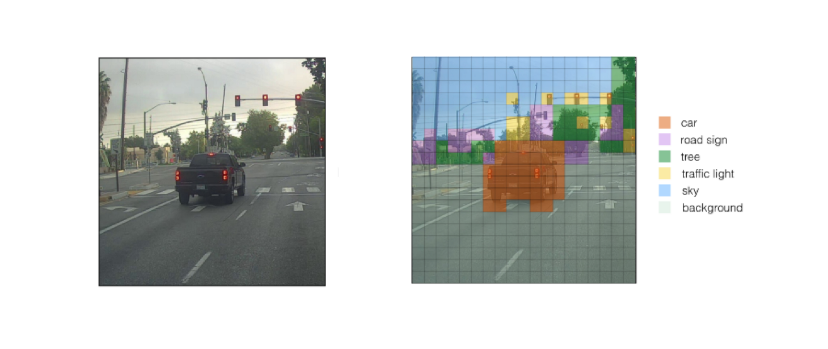
## 3.2. Tensorflow.js

A tárgyak felismerésére tensorflow keretrendszert használok. Keretrendszer függvényeket biztosít amik segítségével felépíthető saját modellünk. 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.

## 3.3. YOLO

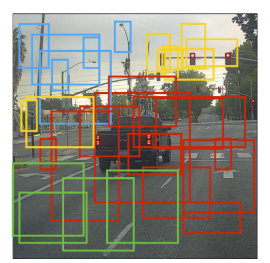
YOLO R-CCN logikáját fogja használni a képfelismerő. YOLO - You Only Look Once (egyszer nézheted meg) algoritmus egy adott képből (nem pedig képfolyamból) állapítja meg, hogy mit tartalmaz. R-CNN (Region-based Convolutional Neural Network - Régió alapú Konvolúciós Neurális hálózat) egy mély konvolúciós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

kép felosztása:



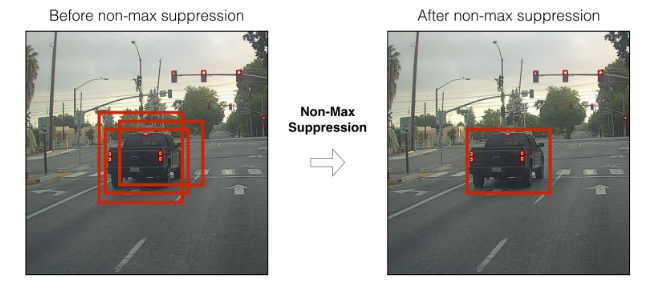
4. ábra: YOLO módszer által felosztott és kategorizált kép

Felismert kategóriák általában fedik egymást:



5. ábra: YOLO által felismert kategóriák összevonás előtt

Az egymást fedő dobozok összeillesztésével megkapjuk a tárgyat körbekeretező téglalapot.



6. ábra: YOLO felismerés összevonás előtt és utáni

1. 3 színcsatornás 416 x 416 felbontású kép lesz a bemeneti adat.
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 neutrális hálóban)
   1. kétdimenziós tömbbé alakítás: [filter\_height \* filter\_width \* in\_channels, output\_channels]
   2. új virtuális 4D tömb létrehozása: [batch, out\_height, out\_width, filter\_height \* filter\_width \* in\_channels]
   3. virtuális tömb feltöltése NHWC módszerrel (value(n, c, h, w) = n \* CHW + c \* HW + h \* W + w).
4. MaxPool alkalmazása (legnagyobb érték kiválasztása a vizsgált területből) 1 x 2 x 2 x 1-es kernel-el
5. az 2. pont ismétlése és a 3. pontban a kimeneti csatornák duplájára növelése 1024-ig
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.

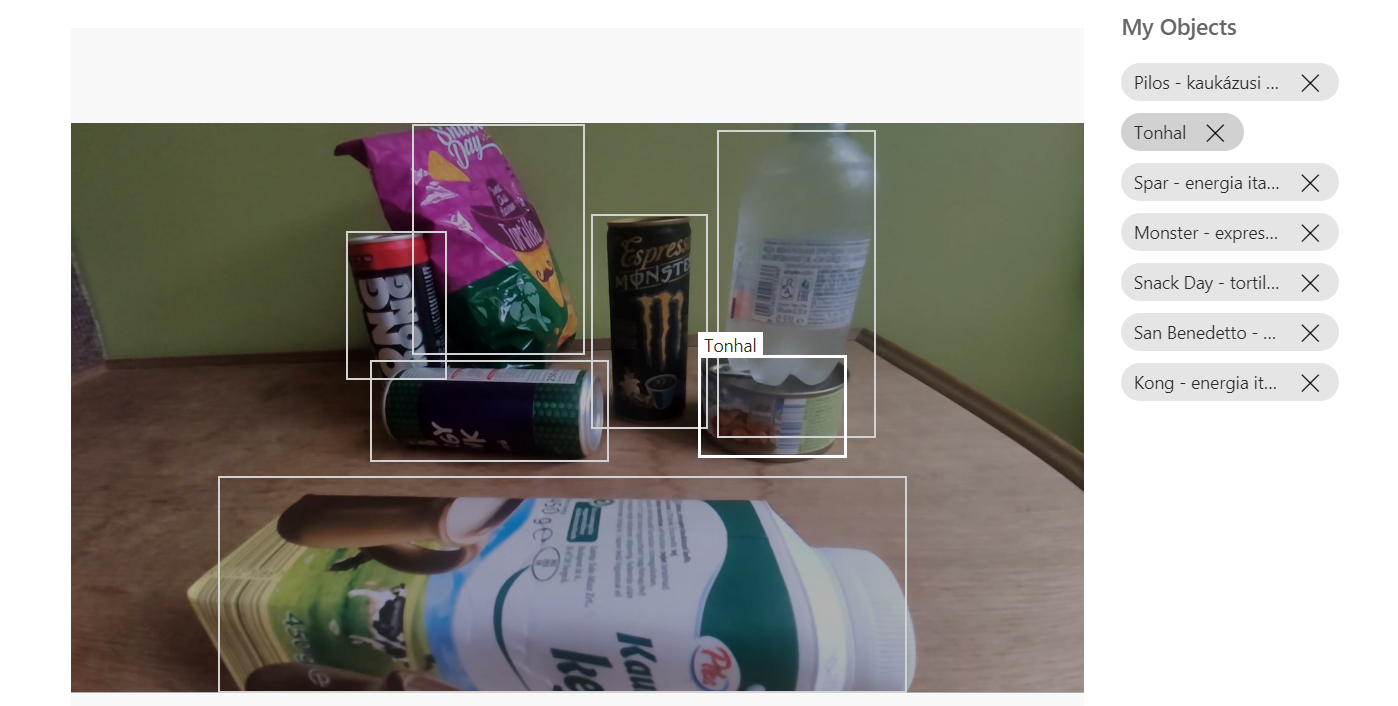


7. ábra: YOLO felismerés folyamata

## 3.4. Azure: Custom Vision

A tárgyakról sok különböző képet kell készíteni. Minél változatosabbak a képek annál többféle szituációban lesz képes felismerni a program a termékeket. Fontos, hogy 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 meg kell adni, hogy a képen milyen objektum és hol szerepel.



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

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. 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.

## 3.5. Program API

### 3.5.1. 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

### 3.5.2. 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

### 3.5.3. 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 megmondhatjuk a tanítás minőségét az adott termékre:

Termék összes megjelenése (t: termék)

Helyes találat és kategorizálás (t: termékre, k:képnél)

Helyes találat (t: termékre, k: képnél)

Találat (t: termékre, k: képnél)

* Recall (újrahívás): Az összes találatból hány százalékot talál el a modell helyesen:
* Precision (precizitás): Ha megtalálta helyesen objektumot a modell az mennyi esetben kategorizálta be helyesen:
* Mean average precision (átlagos precizitás): Az objektum detektálás pontosságát méri:

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

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 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.



8. ábra: Elkészített program működés közben

**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>