

OTDK DOLGOZAT

ÉTELFELISMERÉS NEURÁLIS HÁLÓZAT SEGÍTSÉGÉVEL MOBILESZKÖZÖN

Szerző(k):

Kiss Ámon

Óbudai Egyetem, mérnökinformatikus
BSc. szak, IV. évf.

Konzulens(ek):

Kovács András

Tanszéki mérnök

Szekció:

Műszaki Tudományi

Budapest, 2023.

Tartalomjegyzék

1	Bevezetés, a megoldandó feladat megfogalmazása	3
2	Irodalomkutatás	5
2.1	Hasonló termékek bemutatása	5
2.2	Neurális hálózatok	9
2.3	Konvolúciós neurális hálózat	15
2.4	Létező neurális hálózatok képek feldolgozásához	19
2.5	Keretrendszerek neurális hálózatokhoz	28
2.6	Mobil keretrendszerek	30
3	A rendszer terve	32
3.1	Specifikáció	32
3.2	Felhasználni kívánt technológiák	33
3.3	A rendszer fő komponensei	35
3.4	Felhasználási esetek	36
4	Fejlesztés folyamata	38
4.1	Tanítóminták beszerzése és felosztása	38
4.2	YOLOv4 tiny hálózat tanítása és értékelése	40
4.3	Futószalag szerű klasszifikációs hálózategyüttes tanítása és értékelése	44
4.4	Eredmények értékelése	54
5	Felhasználói dokumentáció	58
6	Összehasonlítás már meglévő megoldásokkal	61
7	Összefoglalás	62
7.1	Továbbfejlesztési lehetőségek	63
	Ábrajegyzék	64
	Táblázatok	64
	Egyenletek	65
	Irodalomjegyzék	66

1 Bevezetés, a megoldandó feladat megfogalmazása

Napjainkban elterjedt az élet minőséget javító applikációk használata, gondoljunk csak az otthoni edzést segítő vagy a táplálkozást nyomon követő alkalmazásokra. Számos különböző megoldás létezik az életminőség javítása érdekében. Mindegyikben kardinális kérdésként jelenik meg a használatához szükséges idő, amelyet az alkalmazás használatára kell szánni. Minél könnyebb és gyorsabb használni egy alkalmazást, annál több embernek nyerheti el a tetszését, mert nem kell sok időt eltölteni a dokumentálással. Különösen igaz ez az ételeket naplózó alkalmazásokra. Ilyen népszerű applikáció például a Yazio, CalorieMama és a Bitesnap. Ezek segítségével könnyebben tudunk tudatosan, egészségesen táplálkozni vagy hosszútávú célokat kitűzni testsúlyunk sikeres csökkentése, vagy épp növelése érdekében. Számos embernek lehet kulcsfontosságú a tápanyagok követése, gondoljunk csak a cukorbetegségekre vagy a magas koleszterinszint kialakulásának egyik okára, a túl sok telített zsír bevitelére. A WHO adatai szerint 2000 és 2016 között a cukorbetegség száma 5%-kal emelkedett és a jövőt tekintve sem úgy tűnik, hogy ez a szám csökkenni fog. Éppen ezért fontos lenne sok embernek egy olyan alkalmazás, amely az ételek dokumentálását felgyorsítja és nem kell hozzá semmilyen speciális eszköz csak egy mobiltelefon, ami rendelkezik kamerával.

A legtöbb hasonló témával foglalkozó program működése egy adatbázison alapul, amiben rengeteg különböző étel szerepel és a felhasználónak kell kiválasztania azt, amit elfogyaszt. Kevés olyan applikáció van, amely ezen túlmutat és képes felismerni azt, hogy mi van a tányérunkon, mit fogunk enni, és milyen összetevőket tartalmaz. Az egyik legnagyobb előnye ennek a funkciónak, hogy sokkal kényelmesebbé és gyorsabbá tudja tenni a dokumentálás folyamatát. Minél kényelmesebb, gyorsabb a funkció, annál több emberhez juthat el egy ilyen program és válhat hasznossá sok ember számára.

Probléma lehet az ételfelismeréssel, hogy a tradicionális magyar ételek (mint például a gulyás, kürtőskalács vagy pörkölt) nem elterjedtek külföldön, így ezeket nem ismeri fel a rendszer, nem tudja, hogy milyen ételt próbálunk meg felismertetni, így azt sem, hogy milyen tápanyagok találhatóak bennük. Másik általános probléma, hogy ha van képfelismerés a szoftverben, akkor az csak a pro verzióval érhető el, ami fizetős és enélkül a funkció nélkül egy sima naplózó alkalmazást kapunk.

Projektünk célja egy olyan ingyenes program elkészítése neurális hálózatok használatával, amely képes valós időben detektálni a különböző, különösképp a magyar ételeket, megbecsülni a bennük lévő makrotápanyagok mennyiségét egységnyi adagra, 100 gramm, majd ezeket naplózni is tudja. A megszerzett információk segítségével statisztikákat tudunk készíteni étkezési szokásunkról. Egyéni célok és limitek beállításával pedig ellenőrizni tudjuk, hogy miből mennyit eszünk. A projekten belül összehasonlításra kerül az egy neurális hálózatot használó és a többszintű, futószalag szerű klasszifikációs neurális hálózat a feladat megoldására.

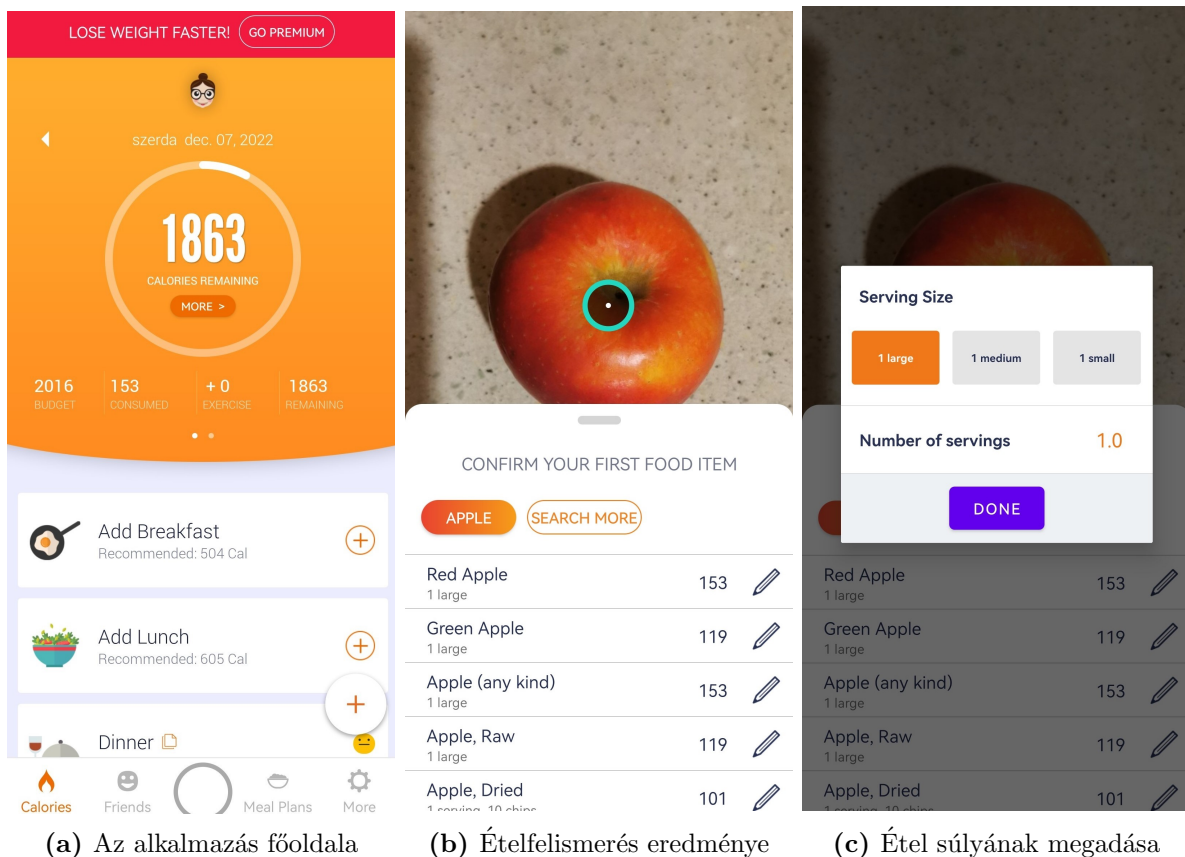
2 Irodalomkutatás

2.1 Hasonló termékek bemutatása

2.1.1 CalorieMama

A CalorieMama egy olyan mobiltelefonos applikáció, amely képes ételek felismerésére. Az alkalmazást az Azumio fejlesztette. A cégnek több olyan programja van még, ami az egészséges életmód betartásában segít. Az alkalmazást világszerte több, mint százezren töltötték már le és sokan használják napi szinten.

Az applikáció az ételek felismeréséhez egy fizetős szolgáltatást használ, a FoodApi-t. A felismerés a telefon kamerájával készített képen történik, a felismert ételeket egy JSON fájlban küldi vissza az alkalmazásnak. Ez a folyamat nem valós időben történik, a felhasználónak el kell küldenie a képet és a feldolgozást is meg kell várnia. Amennyiben nem jól, vagy nem is képes felismerni az ételt, úgy új képet kell készítenie. Egyszerre több étel felismerése lehetséges. A válasz JSON fájlban, szerepel minden fontos információ az ételről egységnyi adagra. Az egységnyi adag itt nem pontos súlyt jelöl, hanem általános adagokat tekint súlynak, ezért minden ételnél ez más lehet. Például az almánál lehet választani kicsi, közepes, és nagy alma között, nem pedig grammra pontosan lehet azt megadni. A válasz üzenetben összesen öt lehetséges ételt ajánl fel a rendszer, ezek közül kell kiválasztani azt, amit mi valójában eszünk. A lehetőségek között a valószínűségek is szerepelnek az ételek mellett. Ez azt mutatja, hogy milyen valószínűséggel sikerült azonosítani egy ételt. Számos előnye van ennek a funkciónak, az egyik, hogy gyorsan lehet vele dokumentálni, de sokszor nem a valóságot tükrözi az elfogyasztott tápanyagtartalomra nézve. Ha nem sikeres a felismerés esetenként lassulhat is ez a folyamat. Az egyes ételekben a makrotápanyagok mellett a só, cukor, és egyéb vitaminokat is képes megbecsülni. Ezen kívül a szoftver képes még részletes, személyre szabott táplálkozási tanácsokat is adni az új vagy régi felhasználóknak. Megtalálhatóak benne különféle diétát segítő leírások is, amelyek segítenek elérni célunkat. A 2.1 ábrán látható az alkalmazás használat közben. [1]



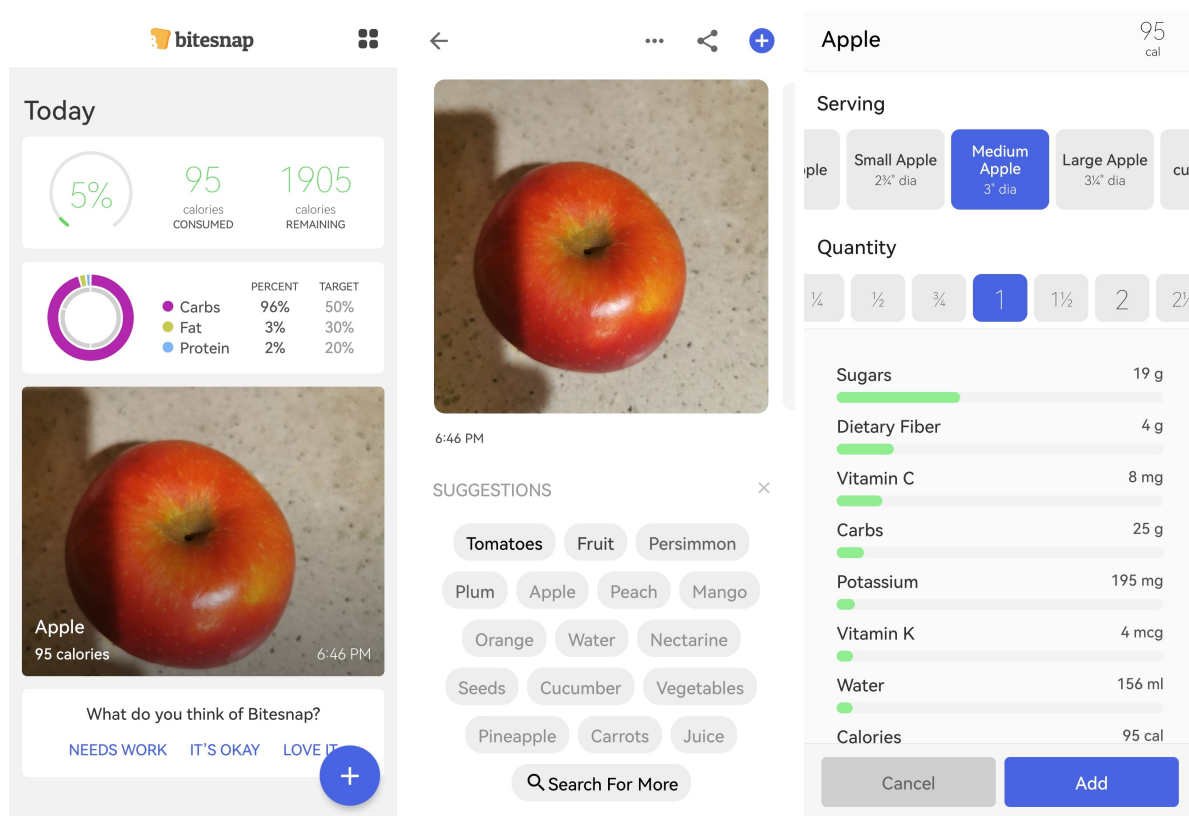
2.1. Ábra: CalorieMama alkalmazás bemutatása

2.1.2 Bitesnap

A Bitesnap, hasonlóan a CalorieMama alkalmazáshoz, egy olyan mobil applikáció, amely képes az ételek dokumentálását felgyorsítani azáltal, hogy fotók alapján ismeri fel az ételeket. Az alkalmazást a Bite AI fejlesztette ki és számos frissítés érkezett már az alkalmazáshoz. Összesen 1300 különböző ételt és italt képes megkülönböztetni az általuk fejlesztett Bite AI segítségével.

Működését tekintve is hasonló a CalorieMama alkalmazáshoz, az ételeket lehetőségünk van lefotózni, majd a kép alapján egy API híváson keresztül történik a detektálás. Az API javaslatokat ad arra, hogy milyen étel látható a képen és ezek közül tudjuk kiválasztani azt, hogy ténylegesen melyik étel van a fotón. Az általunk kiválasztott ételnek ezután lehetőségünk van megadni a méretét és mennyiségét, hogy mennyit eszünk belőle, ezek alapján naplózhatjuk azokat. Az étel méretének állításával változik a tápanyagtartalom, viszont nincs lehetőség grammra pontosan megadni a súlyt. A javaslatok között több étel is ki lehet emelve, ezek súlyának megadását külön is el lehet végezni. Amennyiben több étel van a képen és nem ismerte fel az alkalmazás, lehetőségünk van kézzel is bevenni őket. Amennyiben az élelmiszerek el vannak látva vonalkóddal, az alapján is lehet azokat naplózni. Hasznos funkció ezeken felül, hogy a nemrégiben elfogyasztott ételeket külön

kezeli és ezeket egy kattintással hozzá lehet adni az elfogyasztott ételek listájához. Ez egy tervezett, kiszámítható életmód és étrend mellett kifejezetten hasznos funkció, ami ténylegesen segít még gyorsabban dokumentálni. A galériából is lehet képet importálni és hozzáadni a rendszerhez, ha az étkezés előtt nem akarunk foglalkozni a naplózással. A 2.2 ábrán a Bitesnap alkalmazás ételfelismerésének folyamata látható. [2]



(a) Az alkalmazás főoldala

(b) Ételfelismerés eredménye

(c) Étel súlyának megadása

2.2. Ábra: Bitesnap alkalmazás bemutatása

2.1.3 Im2Calories

Hasonló megközelítést alkalmaz az Im2Calories alkalmazás is. Az alapot is itt egy konvolúciós neurális hálózat adja. A képeket, amiket használtak a hálózatok tanításához, 23 különböző étterem ételeiből gyűjtötték össze. Ezzel a megoldással azokban az éttermekben, amelyeken a hálózat tanult képes pontos eredményeket elérni, viszont nagy általánosságban mindennapi ételeket nem képes detektálni. Az adatbázis 75.000 képet tartalmaz tanításra és 25.000-t tesztelésre. Az osztályokat tekintve összesen 2517-t tartalmaz. Hátránya, hogy csak 1 ételt képes felismerni egyszerre. Az előállított képek, amiken a hálózatokat tanították szegmentációs maszkokkal látták el, vagyis az étel körvonalát határolták körbe és azzal tanították a hálózatot. Az alkalmazás eredményeit kevés kategóriával mutatják be. A hálózat a mobiltelefonon futott, tehát a detektálást

nem egy távoli szerver végzi. [3]

2.1.4 FoodTracker

A FoodTracker egy olyan mobilalkalmazás, amely képe valós időben felismerni az ételeket. Az étel felismerést konvolúciós neurális hálózat segítségével valósították meg. Eredményük közel 80%-os pontosság elérése volt és ezt adaptálták mobil eszközökre. Modellnek a YOLOv2 neurális hálózatot vették alapul és kombinálták a MobileNettel, így képesek voltak kis számításigényű és pontos hálózatot létrehozni. Az architektúra 3,5 millió paraméterből és 30 rétegből áll. Teszteléshez a UECFood100 és UECFood256 adatbázisokat vették alapul. Az előbb említett adatbázisokban 100, illetve 256 étel található. A két adatbázisban leginkább japán ételek találhatók. Az UECFood100 adatbázison 76,36%-os átlagos pontosság, a UECFood256 75,05%-os pontosságot voltak képesek elérni a kutatók. Állításuk szerint az alkalmazás képes valós, életszerű képeken is jól teljesíteni. A válaszdíót egy Google Pixel 2 Android 9.0, és egy OnePlus 5 Android 8.0-s operációs rendszerrel telepített mobiltelefonon tesztelték. Az eredmény 75ms volt. A hálózatok a mobiltelefonon futottak, nem távoli feldolgozásra küldte el a képeket. Eredményeit tekintve egy igen gyors, kis mérettel bíró alkalmazást valósítottak meg. Hátránya, hogy ugyan a tápanyagtartalmakat a felismerés után visszakapjuk és meg tudjuk nézni, viszont naplózásra nem alkalmas a rendszer. [4] [5] [6] [7] [8]

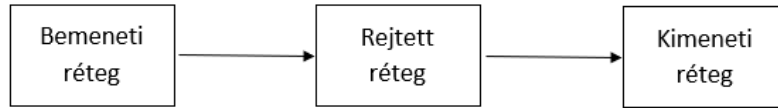
2.2 Neurális hálózatok

2.2.1 Neurális hálózatok általánosan

A projekt fejlesztésének kezdete előtt a neurális hálózatok működésének áttekintése és megértése kihagyhatatlan lépés, hiszen erre épül majd az egész alkalmazás. A neurális hálózatok olyan algoritmusok, amelyek utánózzák az emberi agy működését, hogy felismerjék a hatalmas mennyiségű adatok közti kapcsolatokat. Mindenekelőtt, hogy ez lehetővé válhasson szükség van egy folyamatra, amit tanításnak nevezzünk. Tanítás során egy tanító adatbázisból megadhatjuk a bemeneteket és az elvárt kimeneteket. Ezek alapján a neurális hálózat képes detektálni különböző szabályszerűségeket a bemenetek és kimenetek között. Lehetőség van olyan neurális hálózat létrehozására, ahol nem társítunk a bemenetekhez elvárt kimenetet. Ilyen esetben csak a hasonlóságokat érzékeli a bemenő adatok között, de nem társítja semmilyen kimenethez. A továbbiakban csak olyan neurális hálózatokkal foglalkozok, amelyek tanítása során van hozzá kapcsolva kimenet.

A neurális hálózatokat általánosan három fő részre lehet tagolni: bemeneti réteg, rejtett rétegek és kimeneti réteg. Ennek a vázlatos leírását a 2.3. ábra szemlélteti. A bemeneti rétegben összegyűjti a bemeneti adatokat a tanító adatbázisból. A kimeneti réteg feladata a kapott eredmények összesítése, és kiértékelése. A rejtett rétegek feladata a bemeneti réteg egyszerűsítése, finomítása úgy, hogy a bemenetek legfontosabb jellemzőit

tartják meg és ezek között összefüggéseket keresnek. A neurális hálók olyan problémákat is meg tudnak oldani, amiket még előtte nem láttak, feltételezve, hogy a tanítási minta és az elvárt kimenet pontos volt.



2.3. Ábra: Neurális hálózat rétegei [11]

A neurális hálózat voltaképp összekapcsolt neuronok halmaza. Neuronok közti kapcsolat iránya kétféle lehet, előrecsatolt vagy visszacsatolt. Előrecsatolt hálózatról akkor beszélünk, ha az egymást követő rétegek között egyirányú, a bemenet felől a kimenet felé irányuló kapcsolatok vannak. A hálózatot ilyenkor egy körmentes gráffal ábrázolhatjuk, ahol a csúcsok egy neuront reprezentálnak. Utóbbi esetben a rétegek között vannak visszacsatolások, több irányú kapcsolatok is létrejöhetnek, gráfja kört is tartalmaz. Egy neuron, egy matematikai függvény, amelynek szüksége van bemenetre, amivel számolni tud. A matematika függvény a következőképpen írható le:

$$f(x) = \gamma\left(\sum_{i=1}^n w_i x_i + b\right), \quad (1)$$

ahol,

γ valamilyen aktivációs függvény,

w_i a súlymátrix i . elme,

x_i az i . bemenet,

b pedig az eltolás mértéke.

A súlymátrixot megadhatjuk kézzel, vagy megtaníttathatjuk a hálózatnak úgy, hogy a bemeneti adatokkal kiszámítjuk az eredményt és megnézzük, hogy mekkora különbség van az elvárt kimenet és a becsült érték között. Utóbbi esetben a kezdeti súlymátrix értékei szabadon választhatóak lehetnek. Ezt a különbséget veszteségnek nevezzük. A veszteség értéknek a minimalizálásával tudjuk elérni, hogy optimális súlymátrixot kapjunk, amit a későbbiekben tudunk használni olyan bemenő adatokra, amelyekkel a rendszer nem találkozott a tanítás során.

A tanítás kezdetekor, az előbb említett súlymátrix nem kell, hogy véletlenszerűen legyen inicializálva. Alkalmazható az úgynevezett transfer learning, aminek lényege, hogy egy, már betanított súlyokat tartalmazó mátrixot veszünk alapul a tanításhoz. Ezzel a módszerrel egy olyan állapotból indul a tanítás, ami már valamennyire ismeri az adatokat,

tanult rajtuk. Ez csak akkor igaz, ha a tanítási adatbázis tartalmazza azokat az adatokat, vagy legalábbis egy részét, amikkel a tanítás majd végződni fog. Ezzel gyorsabb konvergencia érhető el és kisebb lehet a veszteség értéke is.

Minimalizálásra egy példa a gradient descent nevű algoritmus, ami a bemenet változása esetén mutatja meg, hogy hogyan változik a kimenet. Az algoritmus megnézi adott bemenetekre a veszteség értékét, és ezeknek az eredményeknek veszi a deriváltját. Ez alapján picit módosítja a súly értékét, amíg a veszteség 0 - vagy ehhez nagyon közeli - nem lesz. Ezzel az eljárással meghatározható az optimális értéke a súlymátrixnak. Ennek köszönhetően tud a későbbiekben értelmes kimenetet adni új bemenő adatokra is, mert a bemenő adatot a betanított súlymátrixal szorozza meg.[9] [10] [11] [12] [13]

2.2.2 Túltanulás, alultanulás, regularizáció

Előfordul, hogy a neurális hálózat a tanulási folyamat végeztével sem képes pontos eredményt adni. Ennek számos oka lehet, de a két leggyakoribb az alultanulás és a túltanulás. Az alultanulás során a neurális hálózat nem lesz képes értelmezhető, korrekt adatokat adni a bemeneti adatokhoz. Megoldásként a tanítási adatbázis bővítése célravezető lehet, általában ezzel meg is oldódik a probléma. A túltanulás egy olyan probléma, ami akkor jelentkezik, amikor a tanító adatbázis méretéhez képest túl sokáig tanítjuk a hálózatot. Következménye, hogy a tanító minta elemeit nagy pontossággal felismeri, viszont olyan bemenetek, amik nincsenek benne a tanító mintában fel sem ismeri, vagy csak nagyon kicsi pontossággal. Ennek megoldására a tanító minta növelése, vagy korai leállítási módszer lehet.

Túltanulás ellen a különböző regularizációs technikák alkalmazása is hatékony megoldást nyújt. Minél nagyobb a súlyok értéke annál nagyobb az esély a túltanulás előidézésére. Ha a súlyokat kisebb értékeken tartjuk, akkor sokkal kiegyensúlyozottabb hálózatot kapunk. A leggyakoribb regularizációs forma a négyzetes regularizáció. Ennek eredményeképpen a súlyokat négyzetre emeljük, ezzel alacsonyabban tudjuk tartani a súlyértékeket. Egy másik nagyon elterjedt regularizációs technika az úgynevezett dropout. Ennek folyamán a csomópontokat valamilyen valószínűséggel tartja meg, vagy dobja ki, vagyis az adott ciklusban nem fog a kidobott neuronokkal tovább számolni. [14] [15]

2.2.3 Aktivációs függvények

Az aktivációs függvények használata az adott problémától függ. Lehet lineáris vagy nemlineáris. Lineáris esetben a kimeneteket valamilyen lineáris függvénnyel szét lehet választani, erre tipikus példa, ha olyan kérdésre keressük a választ, aminek két kimenetele lehet. Nemlineáris esetben a kimeneteket egy bonyolult függvénnyel lehet elkülöníteni, ilyenkor olyan aktivációs függvényre van szükség, ami a részeredményt valamilyen korlátok közé szorítja.

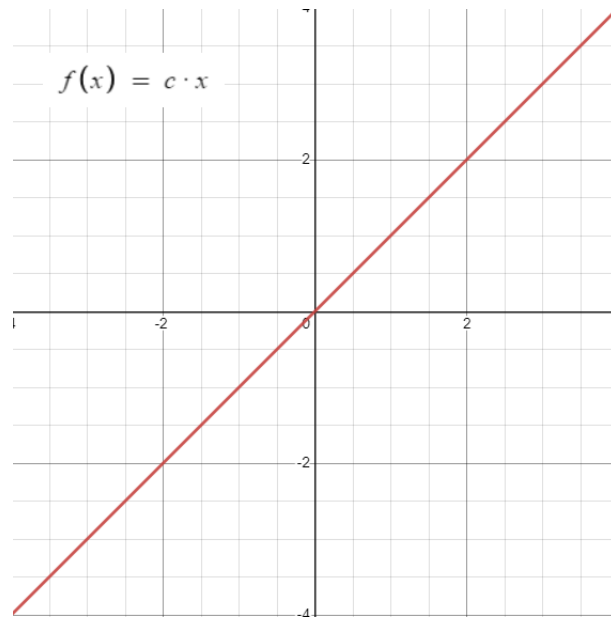
Az egyik legegyszerűbb aktivációs függvény a lineáris aktivációs függvény

$$f(x) = c * x \quad (2)$$

ahol,

c egy tetszőleges konstans.

Lineáris eredményt ad. Könnyen és gyorsan lehet deriválni, nem kerül sok időbe a műveletvégzés. Ez tipikusan olyan problémákra ad megoldást, ahol a problémateretek lineárisan szeparálhatóak. A függvény a 2.4 ábrán látható.



2.4. Ábra: Lineáris aktivációs függvény [17]

Egy másik gyakran használt aktivációs függvény a szigmoid függvény, amivel nemlineárisan szeparálható problémákat is meg tudunk oldani. Egyenlete a következő:

$$f(x) = \frac{1}{1 + e^{-x}} \quad (3)$$

A függvény folytonos, minden pontban deriválható, így nem lesz olyan érték, amire nem tud valamilyen kimenetet adni. További előnye, hogy monoton növekszik, vagyis nagy számokra nagyobb eredményt fog adni, mint a kisebbekre. Normalizál 0 és 1 közé, bármekkora bemenetet kap, mindig 0 és 1 között lesz az eredmény. Hátránya, hogy a nagyon nagy vagy nagyon kicsi számok között kicsi lesz az eltérés, ezzel nehezíti a konvergenciát. Alkalmazása főként bináris osztályozásnál jön elő.

A tangens hiperbolikus

$$f(x) = 1 - \frac{e^{-2x}}{1 + e^{-2x}} \quad (4)$$

nagyon hasonlít a szigmoid függvényhez, annyi különbséggel, hogy -1 és 1 közé képezi le az eredményt. Ez egy picivel nagyobb területre is alkalmazható, mint a szigmoid, de minden egyéb hátrányával is rendelkezik. A derivált értéknek a meghatározása több időt vesz igénybe, mint az eddigiek, tovább tart a műveletvégzés, ezzel lassítva a hálózat konvergenciáját.

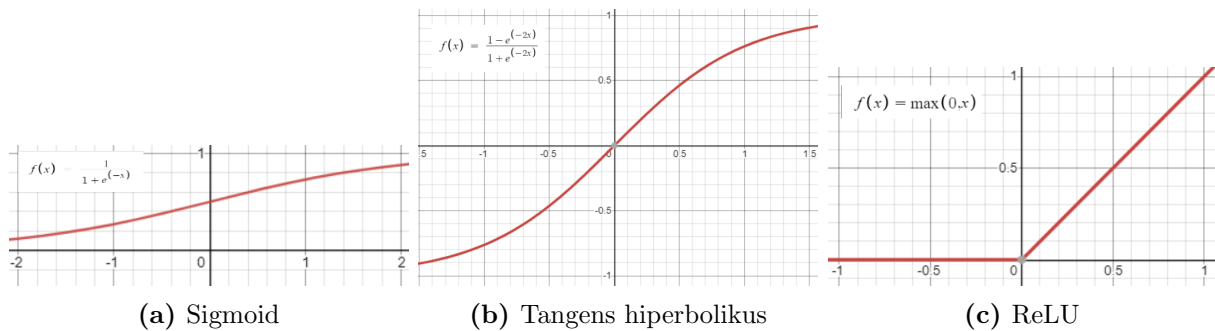
A ReLU a gyakorlatban talán az egyik legelterjedtebb aktivációs függvény. Egyenlete:

$$f(x) = \max(0, x) \quad (5)$$

ahol,

$\max$ egy maximum függvény.

Deriváltja pedig nagyon gyorsan és egyszerűen számolható, számításigénye minimális. Negatív értékekre nullát fog felvenni, ami problémát is jelenthet bizonyos helyzetekben. A függvény monoton nő és nincs eltűnő gradiens probléma, mivel a függvény a végtelenségig nő, ezzel gyorsabb konvergenciát biztosít. Hátránya, hogy a belső rétegekben használható jól és nem normalizál, amilyen nagy a bemenő érték, arányosan nagyobb lesz a kimeneti érték, feltéve persze, ha 0 fölötti bemeneteket vesszük figyelembe. Lineárisan nem szeparálható problémák akkor oldhatóak meg vele, ha több ilyen függvény van egymás után, mert ekkor az eredmény sem lesz lineáris. A 2.5 ábrán a szigmoid, tangens hiperbolikus, és a ReLU függvény van ábrázolva.

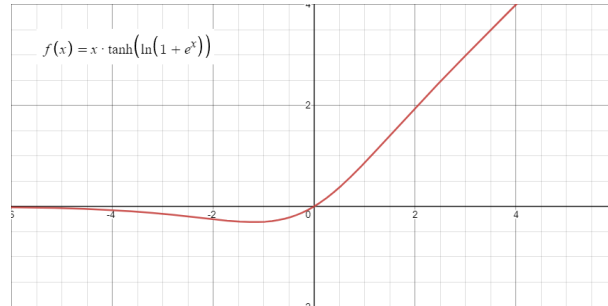


2.5. Ábra: Aktivációs függvények [17]

A Mish aktivációs függvény célja, hogy a ReLU hibáit kijavítsa. A ReLU függvény nullánál kisebb értékeinek a deriváltja is 0, ami megakadályozza az eltűnő gradiens problémát, viszont a neuronok nagy hányada nem frissül. A Mish függvény ezzel szemben nem monoton nő, és 0 alatt is képes releváns információt adni. Ezzel a számítási

igény megnő viszont, a pontosság növekszik, gyorsabb konvergenciát eredményez, ezért leginkább a mélyebb rétegekben lehet jól alkalmazni. A függvény a 2.6 ábra szemlélteti, egyenlete pedig a következő:

$$f(x) = x * \tanh(\ln(1 + e^x)) \quad (6)$$



2.6. Ábra: Mish aktivációs függvény [18]

A kimeneti rétegben leggyakrabban előforduló aktivációs függvény a softmax. Az eredményt 0 és 1 közé tudja leképezni, ami valószínűséget reprezentál a kimenetben. Máshogy fogalmazva, megadja, hogy melyik kimenetre mennyire hasonlít a bemenetként adott adat. Egyenlete a következő:

$$f_i(x) = \frac{\exp(x_i)}{\sum_{j=1}^k \exp(x_j)} \quad (7)$$

ahol,

$\exp$ az exponenciális függvény,

x_i az i . bemenet,

x_j a szumma j . eleme,

k a különböző osztályok száma.

Különbség a szigmoid és a softmax között, hogy míg előbbi bináris klasszifikációknál jól alkalmazható, ugyanis azt adja meg, hogy egy objektum milyen valószínűséggel tartozik egy adott osztályba, utóbbi leginkább több osztályú klasszifikációra használt. [16] [17] [18]

2.3 Konvolúciós neurális hálózat

A konvolúciós neurális hálózat egy vagy több konvolúciós réteggel rendelkező neuronhálózat, amelyet leginkább képfeldolgozásra, osztályozásra vagy szegmentálásra

használnak. Nevét a konvolúciós művelet miatt kapta. A következőkben a konvolúciós hálózat működését mutatom be.

2.3.1 Konvolúciós réteg

A konvolúciós réteg meghatároz egy keretet, amelyen keresztül megvizsgáljuk a kép egy részhalmazát és ezt addig ismétljük, amíg a kép végére nem érünk. A keretek paramétereizhetők, melyeknek segítségével meg tudjuk keresni a kép sajátosságait (például éleket). Ezt a keretet szűrőnek szoktuk hívni, mert a kép azon területeire fókuszál, amelyek a keresett tulajdonságot tartalmazzák. Minden szűrővel végig megyünk a kép összes pixelén és kiszámoljuk a lineáris kombinációjukat, majd meghívjuk rá az aktivációs függvényt. A szűrő feladata tehát a jellemző tulajdonságok megtalálása, detektálása, majd azok továbbadása egy következő rétegnek.

Fontosnak tartom megkülönböztetni a szűrőt és a kernelt. A kernel egy súlyokat tartalmazó mátrix, amivel megszorozzuk a bemenetet, releváns jellemzők kinyerése érdekében. A kernel egy olyan mátrix, ami súlyokat tartalmaz, méretük, és értékeik előre definiáltak. A kernelmátrix dimenziója egyenlő a bemenő kép dimenziójával, vagyis egy kétdimenziós konvolúcióban a kernelmátrix is kétdimenziós lesz. A szűrő valójában több kernel összefűzése, amiben minden kernel a bemenet egy adott csatornájához van hozzárendelve. A szűrők mindig egy dimenzióval nagyobbak, mint a kernelek. Például egy kétdimenziós konvolúcióban a szűrők háromdimenziós mátrixok. A konvolúciós réteg ilyen szűrők sokaságából áll. [19]

2.3.2 Párnázás, kitöltés

A konvolúciós réteg a bemenő kép minden pixelére alkalmazza a szűrőt és veszi a skalárszorosát, majd ezt adja tovább egy aktivációs függvénynek. A probléma ezzel az, hogy a bemenő kép mérete így mindig csökken, adatvesztés történik. Például, ha egy 7×7 -es képen konvolúciót hajtunk végre 3×3 -as szűrő mérettel, az aktivációs térkép mérete 5×5 -ös lesz. Még nagyobb probléma lehet akkor, ha a lépésköz nem egy pixel, hanem több. A lépésköz azt mondja meg, hogy mennyi pixelt lép arrébb a szűrő. A probléma megoldására több opció is rendelkezésünkre áll. Feladatuk egy keretet adni a kép köré, hogy minél kisebb legyen az adatvesztés. A keret mérete, és a lépésköz szabadon választható.

Egyik megoldásként előjöhethet a zéró kitöltés, ami a keretet nulla értékű pixelekkal tölti fel, ezáltal a kép a konvolúció után megtartja eredeti méretét. Ehhez nagyon hasonló a konstans kitöltés, ami nem nullákkal, hanem valamilyen előre megadott számértékkel egészíti ki a képet. Ezen megoldásokra egy példa a 2.7 ábrán látható.

0	0	0	0	0	0	0	0	0	0
0									0
0									0
0									0
0									0
0									0
0									0
0									0
0									0
0									0
0	0	0	0	0	0	0	0	0	0

(a) Zéró kitöltés

1	1	1	1	1	1	1	1	1	1
1									1
1									1
1									1
1									1
1									1
1									1
1									1
1									1
1									1
1	1	1	1	1	1	1	1	1	1

(b) Konstans kitöltés

2.7. Ábra: Zéró és konstans kitöltés [20]

Másik megoldásként felmerülhet még a tükrözéses kitöltés is. Itt a keret kitöltése a vizsgált pixel környezetétől függ. A keret nem nullákból vagy valamilyen előre megadott számokból fog állni, hanem a mellette lévő értékeket a sorban vagy oszlopban tükrözi a keretbe.

Még egy jó megoldás lehet az ismétléses kitöltés alkalmazása. Ez a fajta párnázás elkészíti a kép másolatát, majd megfordítja és kitölti a keretet a kapott értékekkel. A 2.8 ábrán az előbb említett két kitöltés van. [20]

9	5	4	5	6	5	7
3	2	1	2	3	2	1
6	5	4	5	6	5	4
9	8	7	8	9	8	7
3	5	4	5	6	5	1

(a) Tükrözéses kitöltés

2	1	1	2	3	3	2
2	1	1	2	3	3	2
5	4	4	5	6	6	5
8	7	7	8	9	9	8
8	7	7	8	9	9	8

(b) Ismétléses kitöltés

2.8. Ábra: Tükrözéses és Ismétléses kitöltések [20]

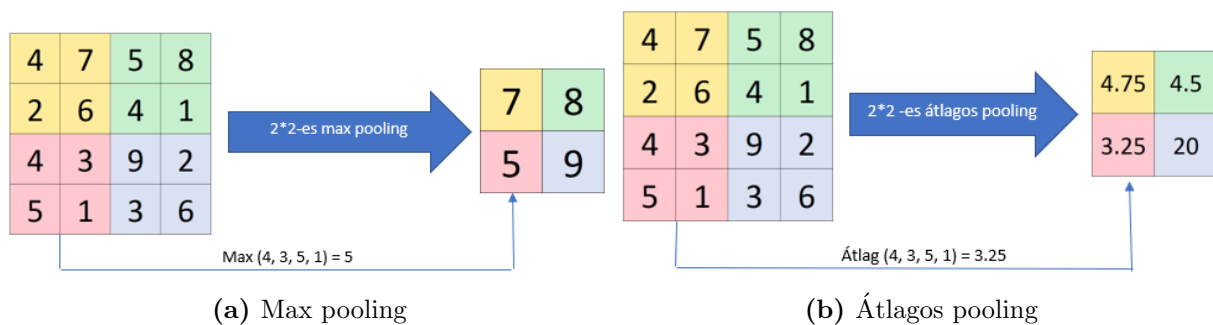
2.3.3 Pooling réteg

A konvolúciós réteg alkalmazza a szűrőket a bemenetre, így több aktivációs térkép jön létre. Az aktivációs térkép a kép jellemző tulajdonságait tartalmazza. Itt jön

képbe a pooling réteg, ami az aktivációs térképek egyszerűsítésére jött létre, amit egy mátrixszorzással ér el. Segítségével a számítások mennyisége redukálható. Mindenképp fontos tudni, hogy a pooling ugyan egyszerűsít, de adatvesztéssel jár. Túl sok pooling használata pontatlan felismerést eredményezhet. A mátrixnak kisebbnek kell lennie az aktivációs térképnél, melynek mérete és értékei előre definiáltak. Több ilyen pooling mátrix létezik, amikkel különböző egyszerűsítéseket tudunk végezni, ezek közül két népszerű az átlagos pooling és a max pooling.

A max pooling esetében a pooling méretét nézve kiválasztja a legnagyobb értéket, és ezt adja hozzá az újonnan létrejött aktivációs térképhez. Például, ha a pooling mérete 2×2 , akkor megvizsgálja a kép egy 2×2 -es részét, tehát négy pixelt, és ezekből kiválasztja a legnagyobb értékkel bírót. Ez egy olyan egyszerűsítést eredményez, aminek következtében a kép mérete $\frac{1}{4}$ -re csökken, mert minden dimenzió nagyságát megfelezzük. Például egy 20×20 -as kép 400 pixelből áll, egy 2×2 pooling mátrix alkalmazása után már csak 10×10 pixelből, összesen 100 pixelből fog állni. A művelet gyors és könnyen számítható. Előnye, hogy az élek pontosan detektálhatóak lesznek, erőteljesebben látszódnak.

Az átlagos pooling az értékek átlagát adja hozzá az aktivációs térképhez. A 2.9 ábrán a két pooling közötti különbségek láthatóak. [21]



2.9. Ábra: Pooling rétegek [21]

2.3.4 Sűrűn csatolt réteg

A konvolúciós hálózatok végén általában egy sűrűn csatolt réteg is található. A kapott eredmények itt kerülnek osztályozásra, kiértékelésre. A sűrűn csatolt réteg lényege, hogy minden neuron kapcsolatba áll minden egy réteggel tovább lévő neuronnal. Ez végzi a végső osztályozást, nem végez plusz műveleteket, a kapott eredményt adja tovább a kimenet felé.

2.4 Létező neurális hálózatok képek feldolgozásához

Mivel a projekt alapja is egy konvolúciós neurális hálózat, érdemes egy pillantást vetni a már létező neurális hálózatokra. Ugyan ezek az alapoktól indulnak, viszont érdemes őket szemügyre venni, mert rengetek jó ötlet és hasznos tapasztalat vonható le belőlük.

Az alábbi áttekintésben zömmel azokat a konvolúciós hálózatokat mutatom be, amelyek az ImageNet Large Scale Visual Recognition Challenge nevű kihívásban vettek részt. A kihívás során több száz különböző objektumot kellett megkülönböztetni. Az ehhez való adatbázis közel 1,3 millió képből állt, amihez 50.000 validációs és 100.000 tesztképet adtak. A kihívás 2010-től 2017-ig volt megrendezve minden évben. Az eredmények rohamos fejlődése miatt már nem okozott akkora kihívást a feladat - vegyük például a 2017-es eredményeket, ahol a 38 indulóból 29-en 95%-os pontosság fölötti eredményt értek el - 2018-ban már nem rendezték meg ebben a formában a kihívást a gyorsan fejlődő hálózatok miatt. [22]

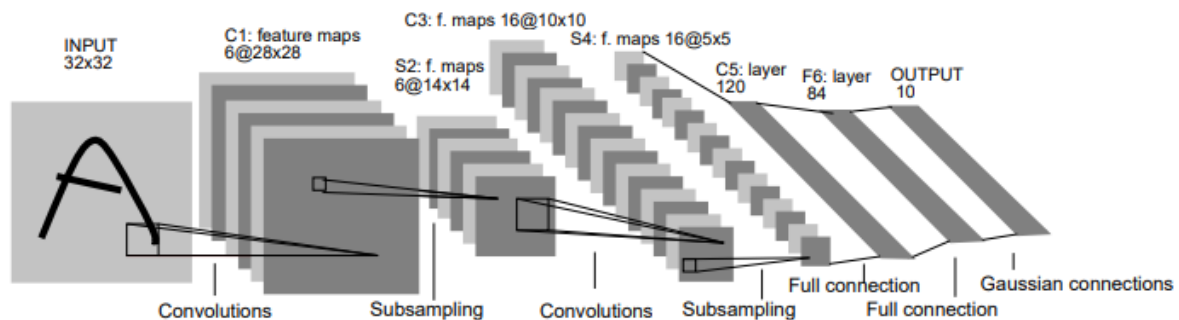
2.4.1 LeNet

A LeNet ugyan nem indult a versenyen, de mindenképp említésre méltó, mert ez volt az első képfelismerésre alkalmas konvolúciós neurális hálózat.

A LeNet-et Yann LeCun, Léon Bottou, Yoshua Bengio, és Patrick Haffner fejlesztették, 1998-ban. Kézzel írt számok detektálására volt alkalmas, ennek megfelelően a kimenet 10 elemű volt, mindegyik elem egy számot jelöl a tízes számrendszerben. Több változat is készült belőle, én ezek közül csak kettőt mutatok most be, a LeNet 5-öt és a LeNet-4 boosted-et.

A LeNet-5 architektúrája 7 rétegből áll, nem számolva a bemenetet. A 7 réteg közül 3 konvolúciós, 2 subspampling és 2 sűrűn kapcsolt rétegre bomlik.

A LeNet4 boosted annyiban különbözik társaitól, hogy három darab LeNet-4 kerül összefűzésre. Ennek a neurális hálózatnak az architektúráját a 2.10 ábra mutatja be. A LeNet 5-nek 0,95%, míg a LeNet-4 boosted-nak a hibaaránya 0,7%. [23]



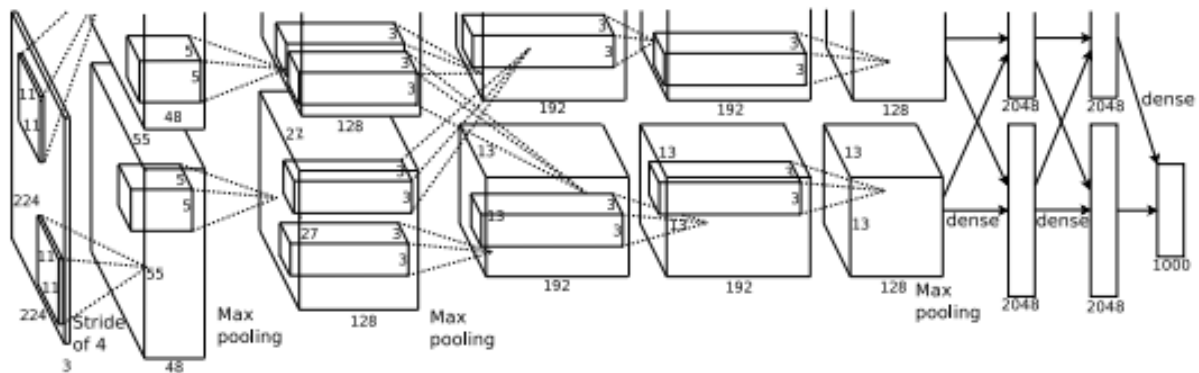
2.10. Ábra: LeNet-5 architektúra [23]

2.4.2 AlexNet

Az AlexNet-et 2012-ben fejlesztették, Alex Krizhevsky, Ilya Sutskever és Geoffrey E. Hinton nevéhez fűződik. A 2012-es ImageNet kihívás első helyezettje lett.

A konvolúciós neurális háló 60 millió paramétert és 650 ezer neuront tartalmaz. Ez feldolgozásban rengeteg időt vett volna igénybe, ezért több GPU-t használtak a feldolgozáshoz.

Összesen nyolc rétegből áll, ebből öt konvolúciós réteg és három sűrűn kapcsolt réteg. Az öt konvolúciós réteg közül három után van pooling réteg is. Aktivációs függvényként a ReLU függvényt használja mindegyik konvolúció során. A konvolúciós rétegek nagy, 11*11-es méretű szűrőket alkalmaznak. A pooling rétegek 2*2 méretűek és max pooling eljárást használnak. A sűrűn kapcsolt rétegek közül az első kettő 4096 neuronnal, az utolsó pedig 1000 darab neuronnal rendelkezik. A 2.11 ábrán az AlexNet vázlata található meg. [24]

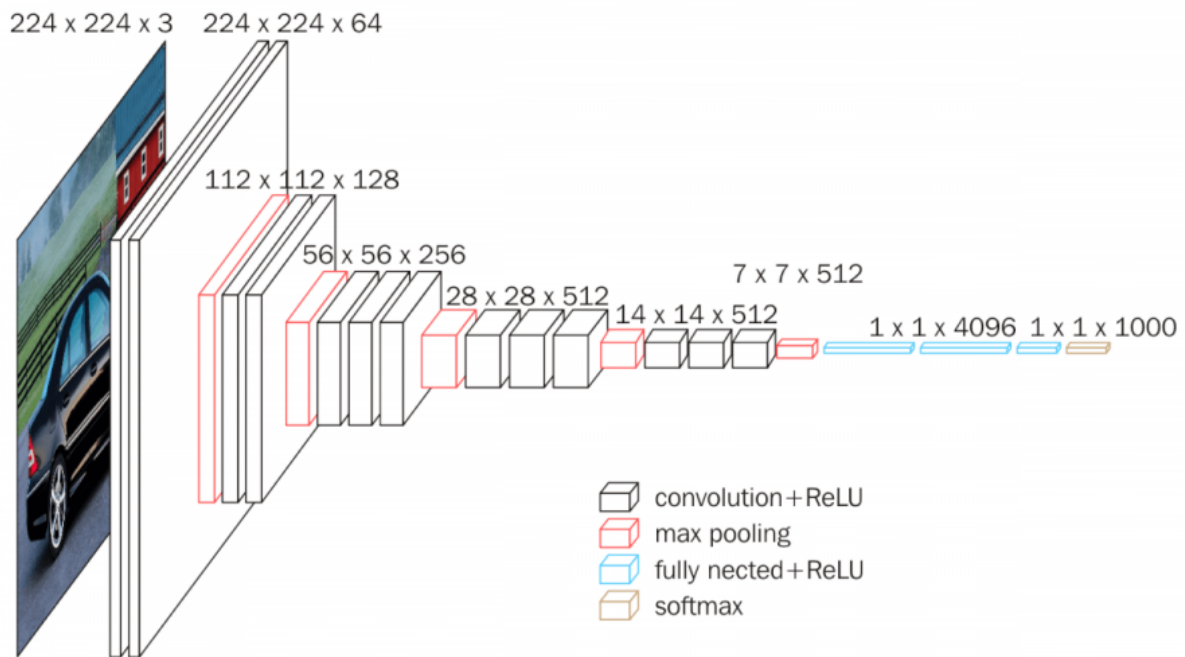


2.11. Ábra: AlexNet architektúra [24]

2.4.3 VGG-16

Alkotói Karen Simonyan és Andrew Zisserman. A projekt 2014-ben készült el és második helyezést ért el az ImageNet-en.

A VGGNet-en belül a bemenő kép mérete 224*224*3 méretű. A konvolúciós rétegekben kicsi, 3*3 méretű szűrőket használ, a lépésköz két pixel. Ez jelentősen megnövelte a paraméterek számát az eddigi hálókhoz képest, ugyanis így körülbelül 140 millió számítandó paraméter keletkezett. Poolingnak a max pooling-ot használják, aminek mérete 2*2 és két pixel a lépésköze. A konvolúciós rétegeket három sűrűn kapcsolt réteg követi, az első kettő ebből 4096 neuronnal, az utolsó pedig 1000 neuronnal rendelkezik. Minden rejtett réteg aktivációs függvénynek a ReLU függvényt használja. Összesen 16 rétegből áll. Az architektúrát a 2.12 ábra mutatja. [25]



2.12. Ábra: VGG-16 architektúra [25]

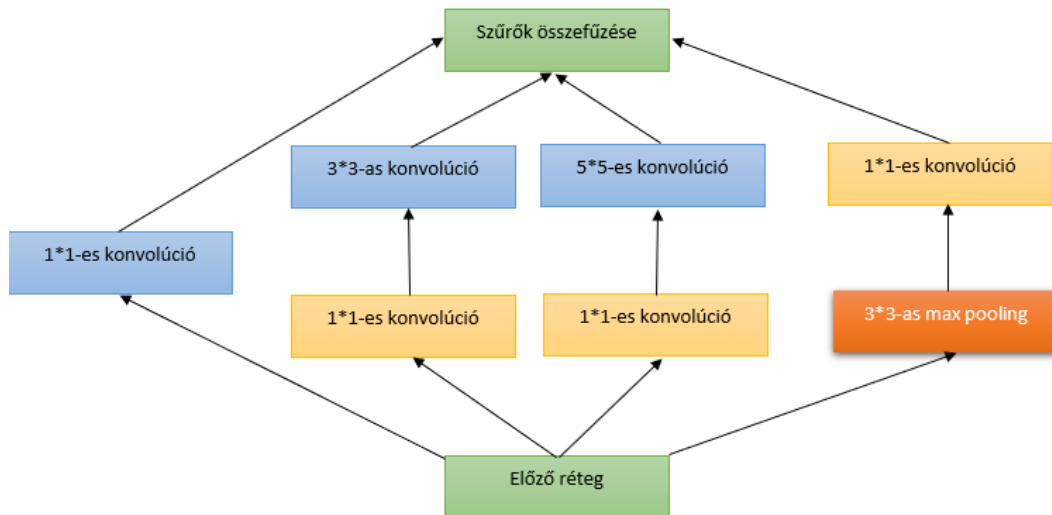
2.4.4 GoogLeNet

Az architektúrát 2014-ben fejlesztették ki, ami összesen 22 rétegből áll. A 2014-es ImageNet első helyezettje lett.

Az architektúra szembement az addig látott, korszerűnek tartott megoldásokkal. Fő különbség az inception modul alkalmazása és az átlagos pooling használata max pooling helyett. A VGG-16-hoz képest jóval kevesebb, 5 millió paraméterrel rendelkezik, az új eljárásoknak köszönhetően pontosabb lett a végeredmény, mint a versenytársaknál.

Az 1×1 -es konvolúció feladata a paraméterek csökkentése és a mélység növelése. Például vegyünk alapul egy képet, ami $14 \times 14 \times 480$ méretű. Erre alkalmazzunk 48 darab 5×5 -s szűrőt, így az eredmény 48 darab 14×14 -es méretű aktivációs térkép lesz. Ha az eredeti képre alkalmazzunk 16 darab 1×1 -es szűrőt és csak ezután alkalmazzuk a 48 darab 5×5 -s filtert a számítások mérete $(14 \times 14 \times 48) \times (5 \times 5 \times 480) = 112,9\text{M}$ -ról $(14 \times 14 \times 16) \times (1 \times 1 \times 480) + (14 \times 14 \times 48) \times (5 \times 5 \times 16) = 1,5\text{M} + 3,8\text{M} = 5,3\text{M}$ -ra csökken. Erre a konvolúcióra nem hív aktivációs függvényeket.

Az inception modult csak tanítás során használják, köztes osztályozó ágakban. A modul feladata a számítások mennyiségének csökkentése. A 3×3 -as és 5×5 -s konvolúciók előtt egy 1×1 -es konvolúció alkalmazásával érték ezt el. Ezek a rétegek segítenek az eltűnő gradiens probléma kiküszöbölésében és regularizációt is biztosítanak. A modul architektúráját a 2.13 ábra mutatja be. [26][27]

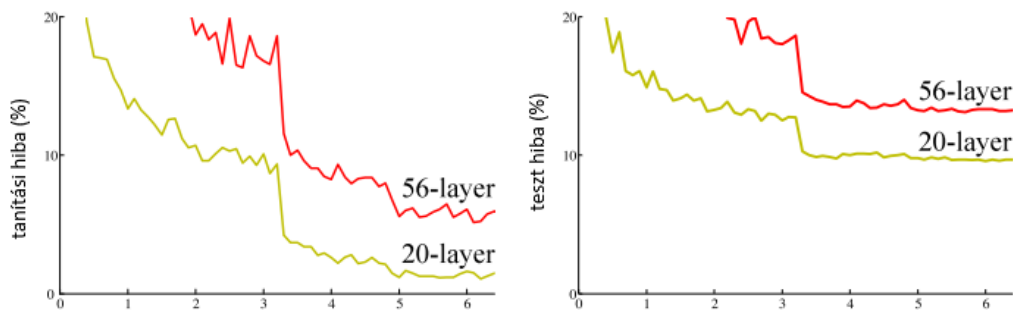


2.13. Ábra: Inception modul architektúra [27]

2.4.5 ResNet

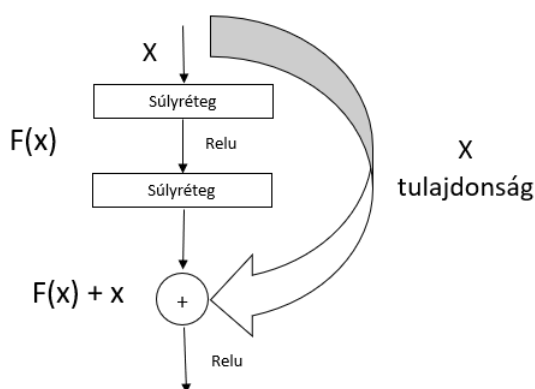
Az architektúrát 2015-ben mutatták be, és első helyezést ért el az *ImageNet*-en. Fejlesztői Kaiming He, Xiangyu Zhang, Shaoqing Ren és Jian Sun.

A többi hálózathoz képest sokkal több, egész pontosan 152 darab réteggel rendelkezik, ami nyolcszor annyi, mint ami a VGGNet-ben van. A hálózatot ma is rengetegen használják alaphoz és különböző módosításokkal több, vagy épp kevesebb réteggel. Az ötlet onnan ered, hogy az eddig megfigyelt neurális háló rétegei növekedtek és így pontosabbá váltak. Megoldás lehet a pontosság növelésére, hogy rengeteg réteget rakunk egymás után és így egy hihetetlen pontos hálót hozhatunk létre. A megoldás ilyen formában nem célravezető, ugyanis bebizonyították, hogy csak szimplán a neurális háló mélyítésével nem lehet pontosabb végeredményt elérni. A következő ábrán jól látható, hogy a 20 mélységű hálózat pontosabb volt, mint az 56 mélységű háló. A rétegek összehasonlításának eredménye a 2.14 ábrán látható.



2.14. Ábra: Mélység növelése és pontosság összefüggése [28]

Megoldásként az architektúrát változtatták, megalkották a reziduális blokkokat. Feladata az, hogy mélység növelésével, ne romoljon a pontosság. Ezt úgy érte el, hogy míg a hagyományos rétegek egymásnak a saját maguk által kiszámolt eredményeket küldik, addig a reziduális blokkon belül az előző réteg kimenetéhez hozzákerül az éppen kiszámolt új eredmény. Így megmarad az előző rétegek által szolgáltatott eredmény és a jelenleg kiszámolt eredmény is. Ezzel az eljárással sikeresen kiküszöbölték azt a problémát, hogy a mélység növelésével a pontosság is romoljon. A blokk felépítését a 2.15 ábra mutatja be. [28]



2.15. Ábra: Reziduális blokk [28]

2.4.6 Neurális hálók összehasonlítása

A hálózatok összehasonlításából jól látszik, hogy évről évre nőtt a pontosság. Ezt mutatja be a 2.1 táblázat.

Neurális háló megnevezés	Év	Helyezés	Top-1 pontosság	Top-5 pontosság
AlexNet	2012	1	63,3%	84,6%
ZFNet	2013	1	-	88,3%
VGG-16	2014	2	74,4%	91,9%
GoogleNet	2014	1	74%,8	92,2%
ResNet	2015	1	78,54%	93,34%

Tábla 2.1: Bemutatott neurális hálózatok összehasonlítása

2.4.7 YOLOv4

A YOLOv4-et Alexey Bochkovskiy, Chien-Yao Wang, Hong-Yuan Mark Liao fejlesztette ki, alapvető célja egy olyan pontos és gyors neurális hálózat megvalósítása, mely képes valós időben egyszerre több objektum detektálására. Megvalósításánál az optimális egyensúly megtalálására törekedtek a bemeneti hálózat mérete, konvolúciós rétegek száma, a paraméterek száma és a kimeneti rétegek száma között. További cél volt még úgy megalkotni a hálózatot, hogy azt saját tanító adatbázissal is könnyedén lehessen használni egy hagyományos grafikus gyorsító segítségével.

Tanítás során több eljárást alkalmaztak, hogy pontosabb végeredményt tudjanak produkálni. Az egyik ilyen eljárás a bag-of-freebies, amivel növelni lehet a tanító adatbázis méretét. A tanítási adatbázis méretének növelésével elérhető, hogy olyan helyzetekből tanuljon a hálózat, amivel nem találkozna. Ilyen eljárások közé tartoznak a geometriai torzítások a körbevágás, forgatás vagy véletlenszerű skálázás. Másik módja a fotometriai adatok módosítása, kép zajának növelése, kontraszt és fényerő növelése vagy csökkentése. Ezzel szemben a bag-of-specials módszerrel nem nő a tanító adatbázis mérete, hanem a belső folyamatok megváltoztatásával pontosítják a hálózatot.

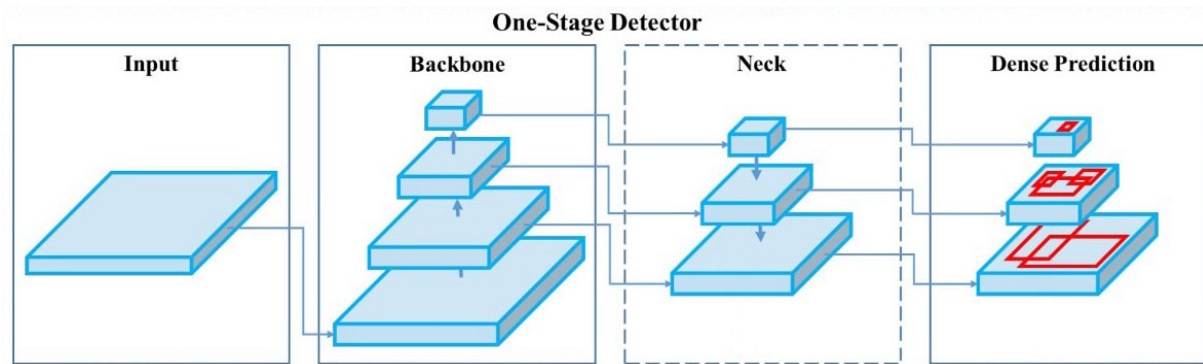
A hálózat gerincét a Darknet keretrendszer adja, melyet CUDA-C nyelven írtak és kifejezetten objektumok detektálására alkalmaznak. Ezen felül implementálja az OpenCV könyvtár egyes algoritmusait is, amik még hatékonyabbá teszik a hálózatot. Az OpenCV egy nyílt forráskódú, platformfüggetlen, ingyenesen elérhető könyvtár mely olyan algoritmusokat kínál, amik megkönnyíthetik a program fejlesztésének lépéseit.

A gerinc és a detektor összekapcsolásáért az SPP (Spatial Pyramid Pooling) és a PAN-t (Path Aggregation Network) felel. Az SPP végzi a különböző méretű kernelek egyesítését, hogy mindig ugyan olyan méretű jellemzőtérkép kerüljön kiértékelésre. A PAN-t azért választották, mert segítségével képesek pontosan megőrizni az információkat a hálózat különböző rétegei között.

A detektor réteg egy vektorból áll, amely tartalmazza a prediktált határoló doboz koordinátáit, a megbízhatósági pontszámát és a címkét. A vektorból könnyen kiolvashatóak az objektumok adatai, amiket későbbi feldolgozásra fel lehet használni. Kimenetelkor több határoló doboz is megjelenik az objektum körül, ennek oka, hogy felismerésnél pixelre pontosan nem tudja megmondani, hogy hol van egy objektum határa, mikor van vége. Megoldásként csak azt rajzolja ki, aminek a valószínűsége a legnagyobb, vagyis a legvalószínűbb, hogy abban a határoló dobozban az az objektum található.

A YOLOv4 sokkal inkább a pontosságra helyezi a hangsúlyt, mintsem a gyorsaságra. Éppen ezért jött létre egy kicsinyített változata, ami kevesebb paraméterrel és kisebb válaszidővel képes objektumokat detektálni. Az eredeti 137 réteg helyett 29 réteggel rendelkezik, átlagos pontosságát tekintve pedig 65,7% helyett 42,0%. Az átlagos pontosság mérése 0,5-s érték felett volt értelmezett, azaz, ha a hálózat 0,5-nél nagyobb értékkel találta el az objektum helyzetét és osztályát.

A hálózat felépítésének szemléltetése a 2.16. ábrán látható. [29] [30] [31] [32] [33] [34]



2.16. Ábra: YOLOv4 Architektúrája [30]

2.4.8 MobileNet

Alkotói Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam voltak. A kutatás 2016-ban készült el, ugyanebben az évben megnyerték az MS COCO Challenge-t. Céljuk egy olyan hálózat megalkotása, amely mobil eszközökön is képes jó eredményeket elérni a konkurenciához képest. A MobileNet-nek manapság számos továbbfejlesztett verziója érhető el, amely még jobban figyel a gyorsaságra és pontosabb eredményeket képes elérni, mint elődjei.

Gyorsasága a bemenő paraméterek csökkenése a mélység szerint szeparálható konvolúciónak és az 1×1 -es úgynevezett pontszerű konvolúciónak köszönhető. A mélységi szeparálás a szűrő mélység és térbeli dimenziójának szétválasztásából ered. A kutatásból kiderül, hogy az alkalmazott módszerekkel gyorsabban elvégezhetőek a szükséges műveletek.

A MobileNet továbbfejlesztett verzióinak célja a paraméterek és memória használat csökkentése és a pontosság megtartása volt a korábbi hálózatokhoz képest. A MobileNetV3-ból két fajta verzió is létezik. A kis verzió kevesebb paraméterrel rendelkezik, kisebb számításigényű eszközökre alkalmazható, viszont a pontossága csökken, míg a nagyobb verzióban a paraméterek száma ugyan több, cserébe jobb eredményeket képes elérni. A MobileNetV3 Large felépítését a 2.17 ábra mutatja be. [35] [6] [36] [37]

Input	Operator	exp size	#out	SE	NL	s
$224^2 \times 3$	conv2d	-	16	-	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	-	RE	1
$112^2 \times 16$	bneck, 3x3	64	24	-	RE	2
$56^2 \times 24$	bneck, 3x3	72	24	-	RE	1
$56^2 \times 24$	bneck, 5x5	72	40	✓	RE	2
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 3x3	240	80	-	HS	2
$14^2 \times 80$	bneck, 3x3	200	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	480	112	✓	HS	1
$14^2 \times 112$	bneck, 3x3	672	112	✓	HS	1
$14^2 \times 112$	bneck, 5x5	672	160	✓	HS	2
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	conv2d, 1x1	-	960	-	HS	1
$7^2 \times 960$	pool, 7x7	-	-	-	-	1
$1^2 \times 960$	conv2d 1x1, NBN	-	1280	-	HS	1
$1^2 \times 1280$	conv2d 1x1, NBN	-	k	-	-	1

2.17. Ábra: MobileNetV3 Large rétegei [37]

2.5 Keretrendszerek neurális hálózatokhoz

2.5.1 Tensorflow és Keras

A Tensorflow egy alacsony szintű, nyílt forráskódú keretrendszer, amit a Google fejlesztett ki, kifejezetten neurális hálózatok készítésére. A keretrendszer lehetővé teszi a fejlesztők számára, hogy adatfolyam-gráfokat készítsenek, ami lehetőséget nyújt a rendszer belső állapotainak megváltoztatására, ezáltal a nagyobb folyamatok esetén is könnyen kézben tarthatóak maradnak. Könnyű használata és átláthatósága miatt hatalmas népszerűségnek örvend, számos nagyobb cég és vállalat használja manapság. Támogatja a neurális hálók futtatását GPU-n, aminek következtében könnyen párhuzamosíthatóak a számítások. A keretrendszerhez részletes és jól követhető dokumentáció tartozik, e mellett rengeteg példát és példaprojektet mutat be különböző rendszerek alkalmazhatóságáról és értékeléséről.

A Keras egy magasabb szintű könyvtár, ami lehetővé teszi a neurális hálók létrehozását Python nyelven. A Python egyszerű és gyors implementációt biztosít, míg a C++-ban megírt alacsony szintű könyvtárak optimalizált működést biztosítanak. Nem kimondottan a Tensorflow-hoz készült, lehet használni egyéb alacsony szintű keretrendszerekhez, viszont a Tensorflow 2.0-ás verziója óta integrálva van a keretrendszerhez. Rengeteg más

magasabb szintű könyvtár készült a keretrendszerhez, de a Keras a legelterjedtebb mind közül.

A Tensorflowban készült modelleket lehetőség van átkonvertálni a Tensorflow Lite eszközcsoport segítségével, így a modellek kisebb számítási kapacitással bíró eszközökön is használhatóvá válnak. Az így konvertált Tensorflow Lite modell könnyen implementálható mobil platformra is, kis késése és mérete miatt. A modellek implementálására számos példa áll rendelkezésre Java és Kotlin nyelven egyaránt.

A modellek létrehozása és azok tesztelése egy hosszú folyamat, ennek megkönnyítésére jött létre a TensorBoard, ami vizualizálni tudja a hálózat viselkedését tanulás közben és utána visszanezézhetővé teszi azokat. Segítségével könnyen össze lehet hasonlítani az egyes modellek teljesítményét, részletesen lehet vizsgálni a modellek tulajdonságait és különböző gráfokat lehet készíteni az elkészült modellekről, melyek összesítik azok felépítését.

2.5.2 Torch

A Torch keretrendszert a Facebook készítette és hasonló népszerűségnek örvend, mint a TensorFlow. Többek között olyan alkalmazásokban használják, mint a Twitter, Salesforce és a Facebook. Leginkább kutatási körökben használják, az adatfolyam-gráf helyett az algoritmusok felépítésén, modularitásán van a nagyobb hangsúly. GPU-n való futtatásra itt is lehetőséget nyújt a keretrendszer.

Ehhez a keretrendszerhez rengeteg magas szintű könyvtárat lehet alkalmazni a fejlesztés során, ilyen például a PyTorch, GPyTorch, a BoTorch és az AllenNLP. Legelterjedtebb a PyTorch, amivel a Keras-hoz hasonlóan a neurális hálózatot Python nyelven lehet létrehozni.

Az Open Neural Network Exchange segítségével könnyen exportálható a betanított neurális háló és használható szinte minden más platformon.

2.5.3 Microsoft Cognitive Toolkit

A CNTK Microsoft által fejlesztett keretrendszer neurális hálók implementálásához. Skype, Xbox és a Cortana is ezt a keretrendszert használja. Könyvtárként is implementálható Python, C# vagy C++ programokba. Szintén támogatja a GPU-n történő futtatást. A Tensorflow-hoz hasonlóan adatfolyam-gráfok készíthetőek, így magával hozza annak minden előnyét.

Jól optimalizált erőforrás felhasználás szempontjából, nagyon gyors még a többi elterjedt keretrendszerhez képest viszonyítva is. Könnyen személyre szabható és Windows mellett Linux eszközökön is használható.

2.6 Mobil keretrendszerek

2.6.1 Android Studio

Az Android Studio egy olyan IDE, ami az IntelliJ fejlesztői környezetet használja. Sok kifejezetten előnyös funkcióval rendelkezik, ilyen például a többnyelvű fejlesztési lehetőség, refaktorálási segéd, fejlett kódanalízis, tesztelés megkönnyítése és a hibakeresés.

A két legelterjedtebb nyelv a fejlesztéshez a Kotlin, és a Java. A Kotlin egy olyan nyelv, ami kifejezetten Android alkalmazások fejlesztésére jött létre, Java nyelven. Kotlin-ban a kódot rövidebben lehet megírni, mint Java nyelven, vagyis kevesebb a hibalehetőség. A TensorFlow Lite modell segítségével könnyen át lehet ültetni ebbe a keretrendszerbe és használata is meglehetősen egyszerű és gyors. Az Android studio-ban lehetőség van a kamera elérésére beépített API használata mellett, ami nagyban megkönnyíti az ételek felismerését a telefon segítségével. Az Android app tesztelését egy emulátoron keresztül végzi. Az emulátor segítségével bármilyen Google által kiadott Android verzióra lehet fejleszteni, így széles körben elérhetővé lehet azt tenni.

2.6.2 Flutter 2

A Google tulajdonában lévő Flutter egy fejlesztői keretrendszer, népszerűsége a könnyű kezelhetőségében rejlik. Programozási nyelvként a Dart-ot használja, amit szintén a Google hozott létre 2011-ben. A Dart egy objektum orientált programozási nyelv, mint a Java, vagy a C#. A UI fejlesztés történhet kódolás nélkül is, vagyis elég megtervezni a kinézetet és a kód automatikusan generálódik le hozzá.

Szintén támogatja a Tensorflow Lite modellek használatát. A kamerarendszer elérése nehézkes lehet, nincs hozzá annyi elterjedt és jól működő API, mint a konkurenciaknál. A keretrendszer a saját renderelő motorját használja, így platformfüggetlen megoldást kínál.

2.6.3 React Native

Facebook által fejlesztett keretrendszer mobil alkalmazások készítéséhez. Programozási nyelve Java script. A Flutterhez hasonlóan ez is platformfüggetlen megoldást nyújt, vagyis Androidra és IOS-re egy időben történik a fejlesztés. Kamerarendszert biztosító API-ból rengeteg támogatott verzió van, ezzel könnyítve annak elérését. Az előre betanított neurális hálók használata ennél a keretrendszerénél sem okoz különösebb gondot.

3 A rendszer terve

3.1 Specifikáció

A kész alkalmazásnak mobil platformon kell futnia, Android eszközökön. A felhasználó a telefon kamerájának segítségével valós időben tudja megnézni, hogy milyen étel látható előtte és egy gomb segítségével azokat kilistázza számára. Akkor tekinthető felismert ételnek egy objektum, ha annak valószínűsége a hálózat szerint 0,5 fölött van, azaz a küszöbértéke nagyobb, mint 0,5. Amire kicsi valószínűséget mond a hálózat nem kerülnek bele a felismert ételek listájába. Az ételeknek nincsenek éles határai, ezért a hálózat többször is felismerheti ugyan azt az ételt. Ez azt eredményezi, hogy egy ételt több határolódobozzal ír körbe. Megoldás, hogy az ugyan azon a területen lévő elemeket egyként kezeli a rendszer. A valós időben történő felismerésre azért van szükség, mert nem biztos, hogy minden szögből pontosan ugyan azokat az ételeket tudja felismerni a program. Ha valós időben tudja a felhasználó nézni, hogy mit érzel, akkor azzal könnyedén el tudja dönteni, hogy helyesek-e a predikciók. Egyszerre több objektum felismerésére is képesnek kell lennie, mivel egy tányéron több étel is helyet kaphat. Összesen a hálózatnak egyidőben maximum 10 ételt kell felismerni, ennél több a gyakorlatban egyáltalán nem, vagy nagyon ritkán fordulhat elő. Következő lépésként a felhasználónak ki kell listázni az összes felismert ételt és az egységnyi adagban található tápanyagtartalmat is. Az ételek tápanyagtartalmait egy adatbázis tárolja, amit a neurális hálózat felismert, az adatbázisban megkeresi a hozzá tartozó adatokat és a későbbiekben ezek alapján kerülnek majd mentésre az ételek. Az ételek pontos súlyát a felhasználónak kell beírnia, majd azután az adatok elmentésre kerülnek egy adatbázisban. A mentés során az ételek összes adata és az időpontot is el kell menteni, hogy a későbbiekben lehetőség legyen nyomon követni az étkezési szokásunkat. Előfordulhat, hogy a rendszer nem képes felismerni az ételt, vagy felismeri csak rosszul, ilyen helyzetben lehetőséget kell biztosítani annak kézzel való bevitelére, törlésre is.

A betanított modelleknek a felismerni kívánt ételek között néhány népszerű és kimondottan magyar ételeknek is szerepelnie kell. Minden képhez tartozik egy .txt kiterjesztésű fájl, ami azt tartalmazza, hogy mi látható és az hol található a képen. Az első adat a fájlban egy egész szám, ami a kategóriát jelenti, a második és harmadik a középpontot jelöli, a következő jelenti az objektum szélességét az utolsó szám pedig a magasságát. Minden szám, kivéve az elsőt, 0 és 1 közé kell, hogy essen. Ezzel az eljárással sokkal pontosabbá tehető a hálózat, mivel csak az objektumot és közvetlen környezetét vizsgálja, így nem von le hamis következtetéseket, vagy tanul meg rossz összefüggéseket. A tanítóminta nagysága és a számítások bonyolultsága miatt videokártya megléte szükséges a projekt elkészítéséhez, amivel a tanítás folyamata felgyorsítható.

Szükség van egy olyan funkcióra is, ami lehetőséget kínál az összes bevitt étel megtekintésére, azok lebontására az elmúlt 31 napban. Ezek alapján egy még részletesebb

képet lehet kapni arról, hogy valóban miből mennyit fogyasztunk el. Visszamenőleg ezek változtatására nincs lehetőség, viszont az adott napon még van.

3.2 Felhasználni kívánt technológiák

Az iparág nagyon gyorsan fejlődik és évről évre egyre pontosabb hálózatok kerülnek ki a kutatók kezei alól. Széleskörű felhasználási lehetőségüknek köszönhetően, minél tovább fejlődik az iparág, annál többféle és bonyolultabb feladatok megoldására lesznek képesek. A projekt szempontjából nézve olyan neurális hálózatra lesz szükség, melyet saját tanító adatbázissal is be lehet tanítani, egy hagyományos grafikus gyorsító segítségével. Ezen felül a pontosságon és a gyorsaságon van a hangsúly a kisebb számítási kapacitással bíró mobil eszközök miatt. A követelményeknek megfelelően az ételek felismeréséért felelős konvolúciós neurális hálózatnak a YOLOv4 tiny-t választottam, ami objektumdetektálás terén magas megbízhatóságot és kis válaszidőt produkál mobil eszközökön. A hálózat keretrendszernek a Darknet-t használja. Ezt szükséges konvertálni a mobil applikációba implementálás előtt. Erre a célra Tensorflow Lite modell-t választottam, a könnyű implementálhatóság, kis méret és gyors válaszidő miatt. Az elkészült YOLOv4 tiny modellt tehát Tensorflow Lite modellé fogom konvertálni és utána azt implementálni.

A futószalag szerű neurális hálózatok felépítéséhez az első neurális hálózat a klasszifikációt végzi és ez adja tovább célzottan a felismerni kívánt objektumokat egy olyan neurális hálózatnak, ami egy kisebb részhalmazát képes felismerni az összes ételnek. A klasszifikációs célra a MobileNetV3-t és az objektum detektálásra a YOLOv4 tiny verzióját választottam. A MobileNetV3 hálózathoz a Tensorflow, Keras termékcsaládot fogom használni, ahol egy API-n keresztül elérhető a teljes modell. A tanítás során a Tensorboard segítségével részletes grafikonokat lehet készíteni az aktuális állapotról, így könnyebb a hálózatot elemezni. A MobileNetV3-t a kevés paramétere és magas pontossága miatt választottam klasszifikációs célra.

Az Android alkalmazás elkészítéséhez az Android Studio programot választottam és Java nyelven íródott az alkalmazás. Könnyen fejleszthető kezelőfelület és nagyon nagy támogatottsággal rendelkezik. Szinte minden Android eszközre egyszerűen lehet benne fejleszteni. Végso soron a Tensorflow Lite modellek egyszerű implementálási lehetősége és hatékony futtatása miatt esett erre a választásom. Továbbá a Tensorflow hivatalos honlapján rengeteg példát lehet találni az implementációra. A kamera kezelését a Camera2 API-n keresztül végzi a rendszer, mellyel könnyen lehet a telefon kameráit elérni és azokat kezelni. Az Android Studio rendelkezik Tensorflow Lite modelleket támogató könyvtárral, így ennek kezeléséhez azt használtam fel. Közvetlen tanítás után a súlyfájl nem egy Tensorflow Lite modell, így azt konvertálni kell ahhoz, hogy a telefonos applikációban is használni lehessen. A konvertáláshoz egy már megírt programot választottam, ami megkönnyítette ennek a műveletnek az elvégzését. A felismert ételek először listázásra kerülnek, majd a súlyok megadása után lehetőség van azok elmentésére.

Az egyes ételek kalóriatartalmai egy adatbázisban vannak eltárolva SQLite adatbázis könyvtár segítségével. Az adatbázis két táblát van, egyik az egységnyi adagra vonatkozó kalóriákat tartalmazza, a másik pedig a felhasználó által bevitt ételeket, a megfelelő súlyokkal és idővel. A dátumra a későbbi statisztikák elkészítése miatt van szükség. A felhasználó adatai egy másik adatbázisban kerülnek tárolásra, itt lehet megadni a napi tápanyagtartalmak mértékét is. A táblák az alábbi, 3.1 képen láthatóak. [38] [39] [40]

UserFoods		ExistingFoods		Profile	
int	id	int	id	user_name	varchar
name	varchar	name	varchar	target_kcal	int
kcal	int	kcal	int	target_carbohydrate	double
carbohydrate	double	carbohydrate	double	target_protein	double
protein	double	protein	double	target_fat	double
fat	double	fat	double	actual_weight	double
date	date			target_weight	double
weight	integer				

(a) Felhasználó által bevitt ételek vannak az adatbázisban

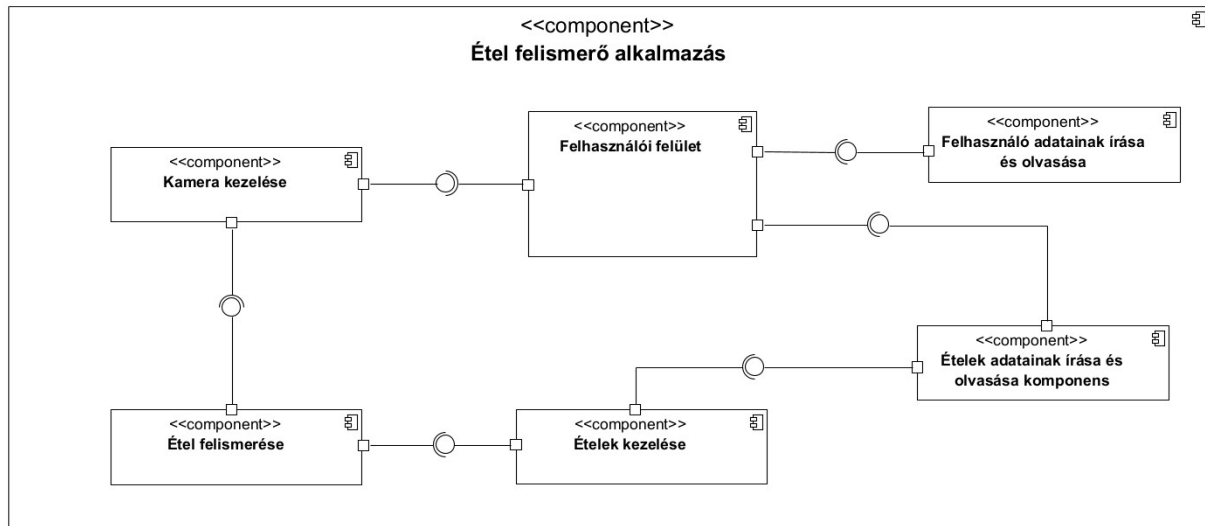
(b) Ételek, amik már benne

(c) Profil

3.1. Ábra: A rendszerben használt táblák felépítése

3.3 A rendszer fő komponensei

A rendszerspecifikációból felépíthetőek a rendszer fő komponensei, ami a 3.2 ábrán látható. Ezek függetlenül a programozási nyelvtől, keretrendszertől és neurális hálózat architektúrájától, ugyan azok. A felhasználói felület a felhasználóval való kapcsolatért felelős és ez kommunikál a többi komponenssel. A kamera kezeléséért is egy külön komponens felel, amin keresztül a neurális hálózat képes felismerni az ételeket valós időben. Felismerés után történik azok listázása, majd súlyainak beállítása. Itt lehetőség van hozzá adni, illetve törölni elemeket a listából. Amikor mentés történik a program rögzíti a bevitt adatokat egy adatbázisba. A felhasználó felület a statisztikák és a napi adatok lekérdezése miatt van kapcsolatban az adatbázist kezelő komponenssel.

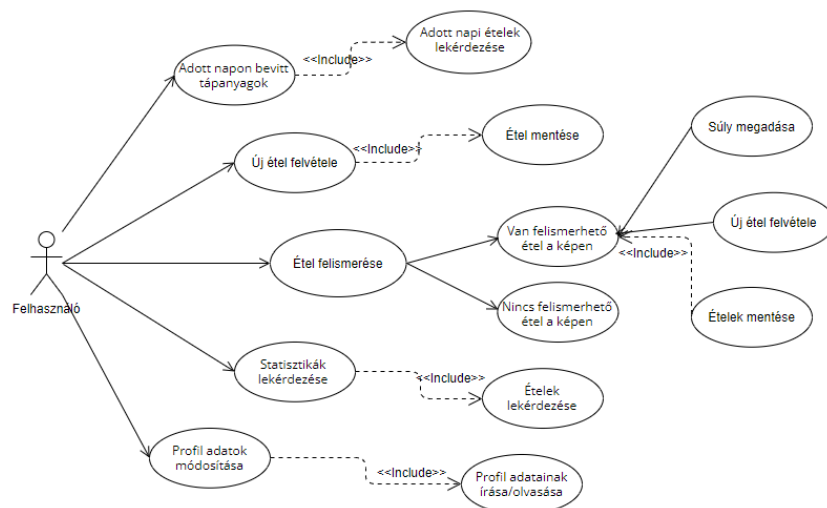


3.2. Ábra: A rendszer komponensei

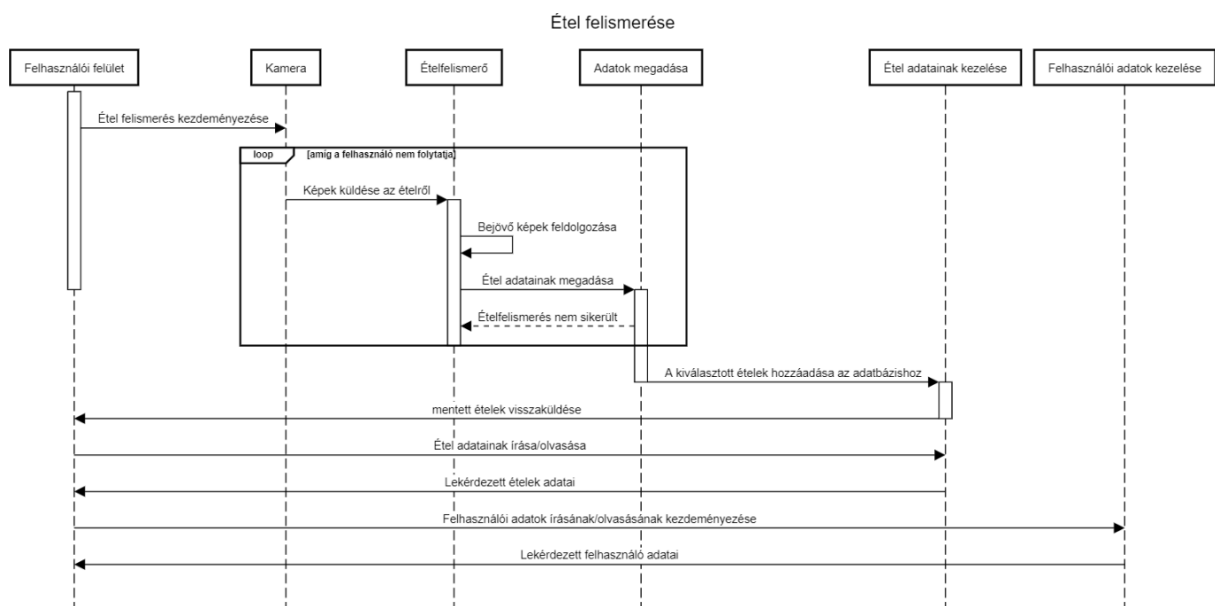
3.4 Felhasználási esetek

A felhasználási esetek leírhatóak a követelményekből. A felhasználó a programon belül öt lehetőség közül választhat. Az első az adott napi ételek megtekintése, ahol listázásra kerülnek az aznapi ételek és lehet rajtuk változtatni. A második, hogy a meglévő ételek mellé az adatbázisba új ételeket vesz fel. Ilyenkor a súlyt és az adatokat bárhogy meg lehet adni, nem csak egységnyi adagra. Ezekből tud majd a felismerést követően hozzáadni a napi elfogyasztott ételekhez. A harmadik lehetősége, hogy statisztikákat kérdez le az elmúlt 31 napból. A következő, hogy a profil adatainak szeretne módosítani és az utolsó, hogy ételt szeretne felismerni. A detektálás után dönthet, hogy mit tart meg, minek írja át a súlyát, vagy akár adhat hozzá új ételeket is. Ezek után mentheti őket.

Ezen adatok alapján készült felhasználási eset diagram a 3.3 ábrán, a szekvencia diagram a 3.4 ábrán látható.



3.3. Ábra: Felhasználási eset diagram



3.4. Ábra: Szekvencia diagram

4 Fejlesztés folyamata

4.1 Tanítóminták beszerzése és felosztása

Mielőtt a tényleges fejlesztés megkezdődne mindenképp szükség van a tanító, validációs és teszt adatbázisok előállítására. Tanításra 28817, validációra 3713, tesztelésre pedig 3416 objektum lett kiválasztva. Ez összesen 35946 objektumot jelent. A három adatbázisra bontás véletlenszerűen történt és mivel egy képen több étel is ki lehet jelölve, ezért az eloszlása az ételek számának nem egyenletes. Százalékban kifejezve a tanító adatbázis a rendelkezésre álló objektumok 80,17%-t, a validációs adatbázis a 10,33%-t, míg a teszt adatbázis a teljes adathalmaz 9,50%-t teszik ki. A validációs képek szerepe, hogy a hálózat tanítása közben képes legyen felmérni annak teljesítményét. Éppen azért a validációs adatbázisnak függetlennek kell lennie a tanítástól így ezek biztosan csak olyan képek lesznek, amiket nem használ fel tanulásra a hálózat. A teszt képek szerepe, hogy a betanult modellt olyan képeken is le tudjuk ellenőrizni, amiket még sosem látott.

Az alkalmazás témájából adódóan minél több ételt tud felismerni sikeresen a hálózat annál jobbnak mondható. Jelen esetben ez 40 különböző ételt jelent, melyek közt szerepelnek gyümölcsök, zöldségek, gyorsételek, levesek és desszertek is. A képek egy része saját, különösképpen a magyar ételek, de internetről letöltött ingyenesen felhasználható képek is helyet kaptak. A másik nagy szelete a képeknek az OpenImagesV6 adatbázisból való, ahol a képeken az egyes objektumok már határoló dobozokkal vannak ellátva. A saját képeken található objektumok körbe határolására a labelImg segédsoftvert használtam. A tanítóminták, a validációs és teszt adatok minden hálózat esetében ugyan azok. A 40 különböző étel és a felosztását a 4.2 táblázat foglalja össze. [39] [40]

Étel neve	Tanító adatbázis	Validációs adatbázis	Teszt adatbázis
Alma	611	163	165
Banán	623	87	96
Paprika	986	153	130
Brassói	410	51	53
Kenyér	505	102	102
Sajt	1354	92	104
Sütemény	1116	135	78
Uborka	605	72	74
Tojás	647	127	123
Sültkrumpli	557	68	73

Szőlő	779	119	80
Gulyásleves	405	50	51
Halászlé	431	52	51
Hamburger	565	69	61
Hot dog	573	76	52
Húsleves	405	50	51
Fagyi	546	64	59
Kolbász	648	101	72
Kürtőskalács	867	112	115
Lángos	480	64	61
Muffin	1165	104	147
Hagyma	1257	144	209
Narancs	1125	205	149
Palacsinta	680	54	60
Tészta	422	52	50
Barack	1674	188	108
Körte	842	142	95
Pizza	566	70	56
Popcorn	611	53	44
Pörkölt	403	51	50
Rántotthús	640	77	83
Rétes	655	76	94
Rizs	459	59	56
Somlói galuska	448	55	54
Eper	944	93	82
Szalonna	767	90	82
Töltöttkáposzta	720	81	74
Paradicsom	658	128	130
Gofri	691	68	65
Dinnye	977	116	77

Tábla 4.2: Tanító, validációs és teszt adatbázisok száma az egyes ételekre lebontva

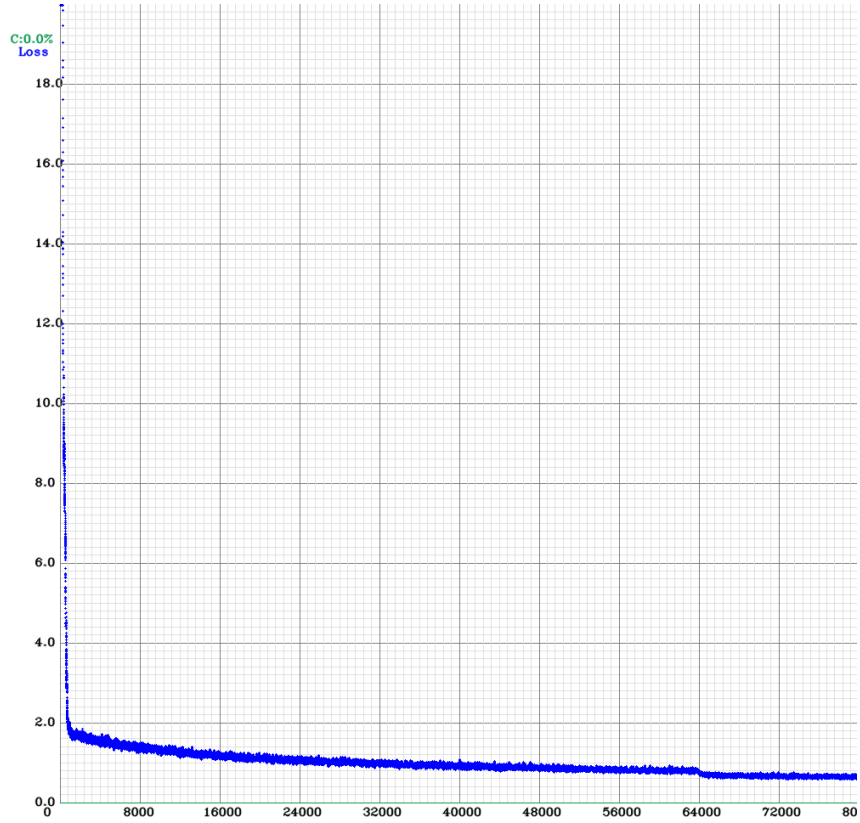
4.2 YOLOv4 tiny hálózat tanítása és értékelése

4.2.1 A tanítás folyamata

Kiinduló állapotként az MS COCO adatbázison tanult súlyfájlt választottam, a gyorsabb konvergálás eléréséhez. Felügyelt tanítás mellett történt a tanítás, azaz a hálózathoz volt elvárt kimenet kapcsolva, amik minden iterációban ugyan azok voltak. Tanítás során a hálózat konfigurációja az ajánlott beállítások mellett zajlott, 416*416-os felbontáson. Nagyobb felbontás esetén pontosabb eredményt lehet elérni felismerés terén, viszont drasztikusan csökken a sebesség, így használhatatlanná téve az alkalmazást egy jóval kisebb számítás elvégzésére képes eszközön. A végleges súlyfájl eléréséhez 80.000 iterációt futtattam, ami közben minden tíz-ezredik iteráció után elmenti az aktuális súlyfájlt. A különböző súlyok kiértékelésével megállapítható, hogy mikor volt a legpontosabb a hálózat, nem jelentkezett a túltanulás vagy alultanulás problémája. A túltanulás elkerülése érdekében amikor az átlagos veszteség nem csökken tovább érdemes azt leállítani. A tanításra egy RTX 3070 videokártyát használtam. [38]

4.2.2 Betanított hálózat értékelése

A tanítás során a hálózat elért veszteség eredményét a 4.1 ábra mutatja be. A képen jól látszik, hogy az első pár-ezer iterációig a veszteség igen magas, viszont nagyon gyors ütemben csökken, a végéhez közeledve pedig egyre inkább kisimul.



4.1. Ábra: Veszteség az iterációk függvényében

Ahhoz, hogy még pontosabb képet kapjunk a hálózat jóságát illetően érdemes több szempontból is megvizsgálni annak működését. A legelterjedtebb metrikák közé tartozik az Accuracy (pontosság), Precision (precizitás), a Recall, az IoU (intersect over Union, azaz a metszet osztva az unióval), az F1-score és az mAP (mean Average Precision) is. Kiszámításukhoz további mérőszámok ismertetése szükséges. Ennek legegyszerűbb szemléltetési módja egy 2*2-es tévesztési mátrix, amit a 4.3 táblázat szemléltet.

	Igaznak prediktált	Hamisnak prediktált
Igaz	True Positive (TP)	False Negative (FN)
Hamis	False Positive (FP)	True Negative (TN)

Tábla 4.3: Tévesztési mátrix

Az előbb említett metrikák egy része ezekből az adatokból már könnyen előállíthatóak.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \quad (8)$$

$$Precision = \frac{TP}{TP + FP} \quad (9)$$

$$Recall = \frac{TP}{TP + FN} \quad (10)$$

$$F1 - score = 2 * \frac{Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN} \quad (11)$$

Az IoU (Intersection over Union) azt vizsgálja, hogy a prediktált és a valós határoló dobozok metszete és uniója mennyire fedik egymást. Ezzel viszont lehetnek problémák ugyanis, ha mindent, ahol akár egészen kicsi metszet alakul ki úgy vesszük, hogy jó volt a prediktálás, akkor a valóságban ez azt eredményezné, hogy teljesen máshol van az objektum határoló doboza a valós objektumtól. Ennek kiküszöbölése érdekében azokat az eredményeket, ahol a felismerés helyes volt, viszont az IoU kisebb volt, mint 0,25 nem számította helyes predikciónak. A számítás alábbi egyszerű képlettel írható le.

$$IoU = \frac{Metszet}{Unio} \quad (12)$$

Az mAP (mean Average Precision) kiszámítása pedig a következőképpen történik.

$$mAP = \frac{1}{n} * \sum_{i=1}^{i=n} AP_i \quad (13)$$

ahol,

n az osztályok száma,

AP_i az i. osztály átlagos pontosságát jelöli.

Ezen metrikák mentén vizsgáltam meg a hálózat jóságát. Az eredményekből következtetni lehet a működésére és esetlegesen azt tovább is lehet tanítani a pontosabb predikciók érdekében. A hálózat eredményeit az alábbi, 4.4 és 4.5 táblázat foglalja össze, előbbi a validációs utóbbi a teszt adatbázison.

A táblázatokból jól látszik, hogy az elején rosszabb eredményekkel indul, viszont az iterációk előrehaladtával egyre jobb értékeket ér el a hálózat. A legjobb súlyfájl kiválasztásában a legnagyobb hangsúlyt az mAP értékre helyeztem, ez alapján a 80.000. iterációban elért fájl fogom használni a későbbiekben.

Iteráció	TP	FP	FN	IoU	Precision	Recall	F1 -score	mAP
10000	554	696	3159	34,71%	0,44	0,15	0,22	34,71%
20000	783	672	2930	42,68%	0,54	0,21	0,3	38,06%
30000	1093	1444	2620	32,64%	0,43	0,29	0,35	42,22%
40000	1190	1198	2523	39,20%	0,5	0,32	0,39	45,41%
50000	1431	1528	2282	37,84%	0,48	0,39	0,43	48,26%
60000	1387	1203	2326	42,34%	0,54	0,37	0,44	50,00%
70000	1699	1405	2014	44,11%	0,55	0,46	0,5	57,06%
80000	1729	1445	1984	44,02%	0,54	0,47	0,5	57,44%

Tábla 4.4: Súlyfájlok összehasonlítása a validációs adatbázison

Iteráció	TP	FP	FN	IoU	Precision	Recall	F1 -score	mAP
10000	595	721	2830	35,05%	0,45	0,17	0,25	31,63%
20000	877	694	2548	44,25%	0,56	0,26	0,35	42,37%
30000	1153	1516	2272	32,83%	0,43	0,34	0,38	45,26%
40000	1241	1232	2184	39,59%	0,5	0,36	0,42	50,55%
50000	1481	1530	1944	38,69%	0,49	0,43	0,46	54,01%
60000	1430	1242	1995	42,35%	0,54	0,42	0,47	54,89%
70000	1768	1393	1657	54,42%	0,56	0,52	0,54	61,21%
80000	1798	1433	1627	45,25%	0,56	0,52	0,54	61,85%

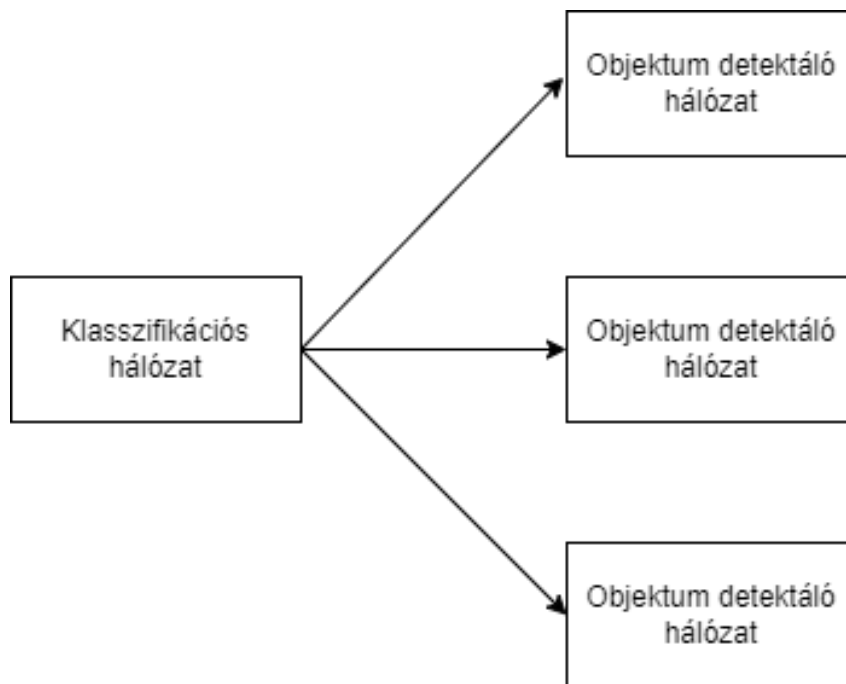
Tábla 4.5: Súlyfájlok összehasonlítása a teszt adatbázison

4.3 Futószalag szerű klasszifikációs hálózategyüttes tanítása és értékelése

4.3.1 Felépítés

A futószalag szerű klasszifikációs neurális hálózat két rétegre osztható. Az első réteg egy klasszifikációt végző hálózatot tartalmaz, ez végzi el a kép klasszifikációját és adja tovább

a második rétegnek. A második rétegben több, a probléma egészét nézve olyan hálózatok találhatóak, amik csak egy részhalmazát képesek felismerni az összes objektumnak. A következőkben bemutatott példákban egy képre adott kimenetek lehetőségeit fogom bemutatni. A feladatban a képek a kamerából érkeznek a hálózatok felé, folyamatosan, így áll elő valós időben a predikció. A hálózategyüttes felépítését a 4.2 ábra mutatja be, ahol az n - az összes osztály száma - 3 részre lett osztva és egy objektum detektáló rész $\frac{n}{3}$ objektumot képes detektálni.

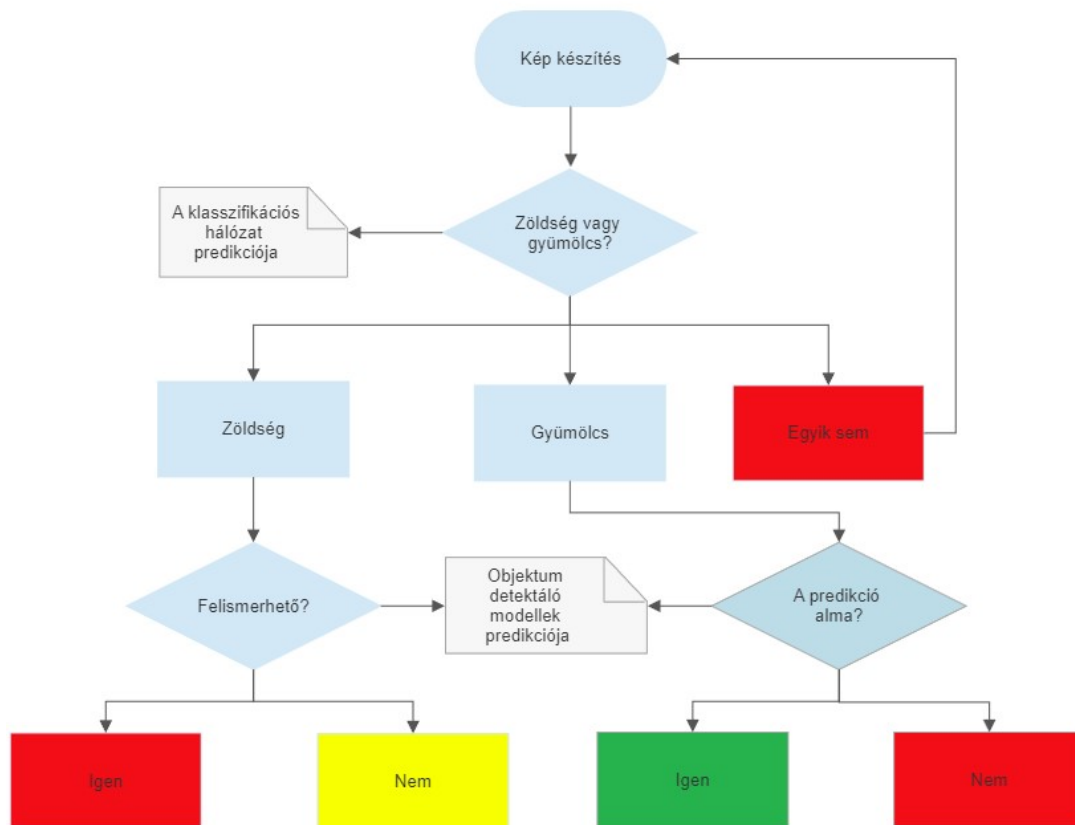


4.2. Ábra: Futószalag szerű hálózategyüttes architektúra

A feladat szempontjából az is fontos tényező, hogy ne csak egy objektum detektálókhoz kerüljön az adott kép, ha esetleg több objektum található rajta és azok különböző klasszba, azaz objektum detektáló modellhez tartoznak. Vegyünk például egy egyszerű esetet, ahol a felépítés két objektum detektáló hálózatból és előttük egy klasszifikációs hálózatból áll. A két detektáló modell közül az egyik az zöldségeket tudja felismerni, a másik pedig a gyümölcsöket. Előfordulhat az a két eset, hogy csak zöldségek, vagy csak gyümölcsök láthatóak a képen. Ez esetben, ha feltételezzük a klasszifikációs hálózat helyes működését, csak az egyik objektum detektáló modell kapja meg a képet, amin az ételek szerepelnek. Így nem kerül az a kép, amin csak zöldségek vannak, ahhoz a modellhez, ami csak gyümölcsöket tud felismerni. Ugyanez történik fordított esetben is. A valóságban viszont előfordulhat az a helyzet is, hogy a gyümölcsök és a zöldségek egy képen szerepelnek. Ebben az esetben mind a két modellnek meg kell kapnia a képet ahhoz, hogy fel tudják ismerni a fotón szereplő ételeket. Végeredményként a detektáló modellek által felismert összes objektumnak látszódnia kell.

Az előző példánál maradva, a klasszifikáció során keletkezett hibák, pontatlanságok kezelése különböző előnyökkel és hátrányokkal járnak. Előnyként jelenik meg az egy klasszba tartozó objektumok keverésének kezelése. Ha a klasszifikáció során az alma és a barack keveredik, akkor is a gyümölcsöket detektáló modell kapja meg feldolgozásra a képet. A pontatlanság ebben az esetben nem jelenik meg a feldolgozás végén, ha a detektálás során a predikció jó értéket ad. A rossz predikció lehetősége itt is fennáll, viszont a klasszifikációs pontatlanság nem játszik szerepet a későbbi predikcióban. Ez magában hordozza azt a veszélyt, hogy ha a klasszifikáció során rossz objektumok kerülnek rossz második rétegbeli modellhez. Ilyen esetben például, ha egy alma keveredik egy hagymával, a képet a zöldségeket kezelő detektáló modell kapja meg, ami nem tud almát felismerni. Ezt követően is két eset áll fenn. Vagy a detektálás során is rossz objektum kerül ki, vagy nem ismeri fel a modell az objektumot. Az előnye tehát itt is lehet az, hogy rossz predikció esetén nem kerül ki rossz objektum a detektáláskor, viszont fennáll a veszélye egy esetleges rossz predikciónak is.

További kérdésként merülhet fel még a küszöbérték megválasztása. Túl kicsi érték választása esetén nagyon sok felesleges kép fog kerülni az objektumokat detektáló modellekhez, lényegében értelmetlenné válik a klasszifikáció, mert minden második rétegbeli hálózat megkapja a képeket detektálásra. A pontosság ebben az esetben nőhet, viszont a feldolgozási idő az x szerezére növekszik, ahol x az objektum detektáló modellek számát jelöli. Ezen érték túl nagyra állítása esetén csak akkor kerülnek tovább a képek a kamerából, ha azon nagyon nagy valószínűséggel étel látható. Ez azt a veszélyt hordozza magában, hogy nagyon kevés kép, rosszabb esetben egyáltalán semmi nem fog eljutni a detektáló modellekhez és nem lesz képes ételeket felismerni. Az előbb bemutatott kimeneteli lehetőségeket a 4.3 ábra szemlélteti. A bementi kép, ahol a képen egy alma látható.



4.3. Ábra: A kimenetek lehetségei egy képre, amin alma látható

4.3.2 A tanítás folyamata a klasszifikációs hálózaton

A klasszifikációt a MobileNetV3 modell végzi. Ez a hálózat dönti el azt, hogy melyik objektum detektáló hálózatnak adja tovább a bemenő képeket a kamerából. A YOLOv4 tanításához hasonlóan a hálózat itt is csak a képen határolódobozzal kijelölt részt kapja meg mind a tanítás, mind validáció esetén, tehát tanításra és validációra ugyan annyi képe áll rendelkezésre és egy kép egy osztályba tartozhat. A tanító képeken különböző augmentációkat is végeztem, forgatás, rotáció, nagyítás vagy kicsinyítés és kontraszt állítással. Tesztelésnél egy kép több osztályba tartozhat, például lehet rajta hamburger és sültkrumpli egyszerre. Ugyan azokat a képeket használtam itt is tesztelésre, mint a YOLOv4 tiny modell tanításánál, összesen 1893 kép áll rendelkezésre, ezek pedig 2052 tesztet jelentenek.

Az alap MobileNetV3 egy Keras API-n keresztül elérhető el. Mivel egy képen egyszerre több osztályból is lehetnek ételek, ezért az utolsó sűrűn kapcsolt rétegben aktivációs függvénynek a szigmoidot alkalmaztam. Így a kimenet külön-külön minden egyes objektumra ad egy értéket, hogy az mekkora valószínűséggel tartozik az adott osztályba. Összesen négy különböző modellt tanítottam be és hasonlítottam össze a teljesítményük alapján. Mindegyik modell bemenetként egy 224*224-es képet vár és a kimenetet tekintve

négy sűrűn kapcsolt rétegekből állnak, amik közül az első 512, a második 256, a harmadik 128, és az utolsó 40 neuronnal rendelkezik. Az utolsó réteget kivéve mindegyik másik sűrűn csatolt réteg tartalmaz dropoutot is, az előbb említett sorrendben 0,5, 0,2 és 0,2 értékkel. Aktivációs függvénynek az utolsó réteg kivételével a ReLU aktivációt használtam. A tanulási rátát 0.00001-re állítottam. Túl kicsi érték megadásánál a tanítási folyamat ideje megnő és nagy veszély van arra, hogy beragad egy lokális optimumba és nem javul tovább a pontosság. Túl nagy érték választása esetén pedig túl gyorsan konvergál, akkor fennáll a veszélye, hogy átlépi az optimumot. Optimalizálónak az Adam optimalizálót használtam. A gradient descent eljárással szemben nem végig konstans tanulási rátával dolgozik, hanem azt változtatja a tanulás során. A különbséget a tanítható paraméterek száma jelenti. A tanítás összesen 300 iterációig ment, vagy addig, amíg 10 iteráció után nem történt javulás a validációs adatbázis kiértékelésekor. A modellek összesen 3.657.768 paraméterből állnak, és a különbséget a transfer learning alkalmazása és a tanítható paraméterek száma jelenti. Transfer learning használatakor az imageneten tanult modellt alkalmaztam. Teszteléskor a különböző küszöbértékeket külön teszteltem. A 4.6 és 4.7 táblázat a modellek összehasonlítását mutatja be a validációs és teszt adatbázisokon. A táblázat első két oszlopa mutatja, hogy a transfer learninget használó és a nem használó modell hogyan teljesít. Eredményként az jött ki, hogy az, amelyik nem használ transfer learninget, az jobb eredményeket ért el, ezért a harmadik és negyedik oszlopban lévő modellek tanításakor már nem használtam. [41]

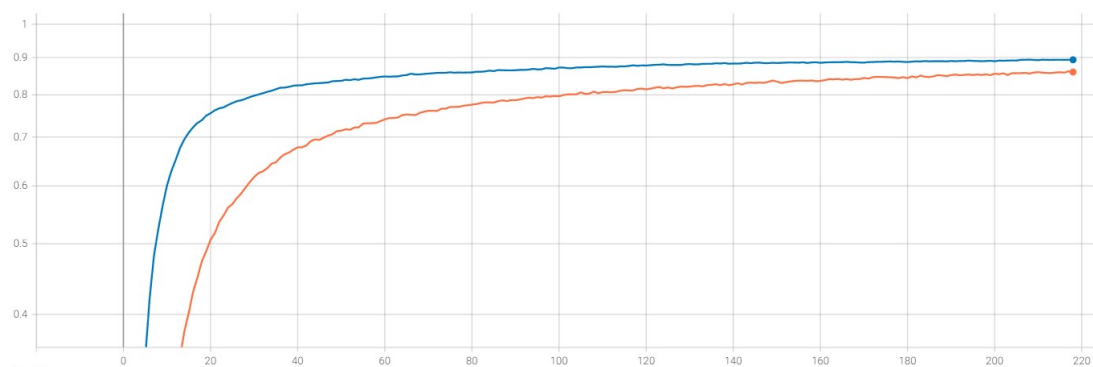
	Imagenet transfer learning	Transfer learning nélkül	Utolsó 10 réteg is tanítható paraméter	Minden paraméter tanítható
Tanítható paraméterek száma	661.416	661.416	1.432.856	3.633.368
TP	3235	3249	3344	3170
FP	478	464	369	543
FN	478	464	369	543
Accuracy	0,87	0,87	0,87	0,87
Precision	0,88	0,88	0,88	0,88
Recall	0,9	0,9	0,9	0,9
F1-score	0,85	0,85	0,85	0,85

Tábla 4.6: Betanított modellek összehasonlítása validációs adatbázison

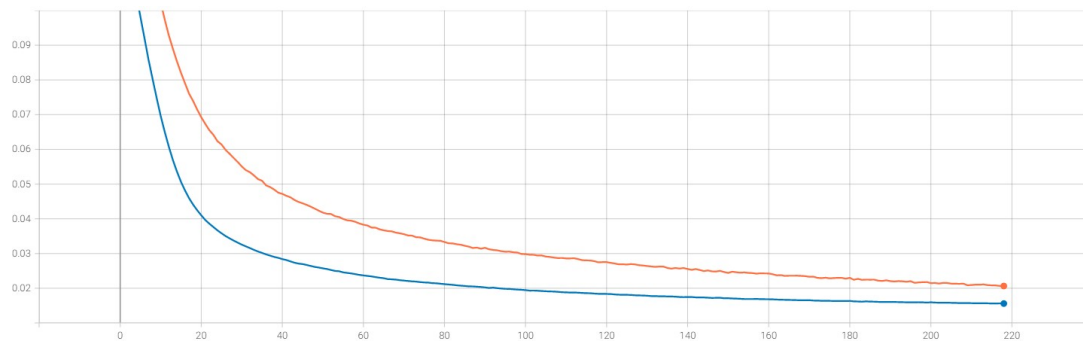
	Küszöbérték	Imagenet transfer learning	Transfer learning nélkül	Utolsó 10 réteg is tanítható paraméter	Minden paraméter tanítható
TP	0,1	1672	1693	1636	1568
FP		1499	1460	1119	1203
FN		380	359	416	484
Accuracy		0,49	0,51	0,58	0,57
Precision		0,53	0,54	0,59	0,57
Recall		0,81	0,83	0,79	0,76
F1-score		0,64	0,65	0,68	0,65
TP	0,2	1578	1591	1536	1486
FP		810	789	669	779
FN		474	461	516	566
Accuracy		0,60	0,61	0,64	0,61
Precision		0,66	0,67	0,70	0,66
Recall		0,77	0,78	0,75	0,72
F1-score		0,71	0,72	0,72	0,69
TP	0,3	1504	1528	1482	1427
FP		511	530	481	586
FN		548	524	570	625
Accuracy		0,65	0,65	0,67	0,64
Precision		0,75	0,74	0,75	0,71
Recall		0,73	0,74	0,74	0,70
F1-score		0,74	0,74	0,74	0,70
TP	0,4	1436	1446	1442	1376
FP		364	377	377	494
FN		616	606	610	676
Accuracy		0,66	0,67	0,68	0,64
Precision		0,80	0,79	0,79	0,74
Recall		0,70	0,70	0,70	0,67
F1-score		0,75	0,75	0,75	0,70
TP	0,5	1378	1387	1400	1346
FP		254	251	297	399
FN		674	665	652	706
Accuracy		0,66	0,67	0,67	0,65
Precision		0,84	0,85	0,82	0,77
Recall		0,67	0,68	0,68	0,68
F1-score		0,75	0,75	0,75	0,71

Tábla 4.7: Betanított modellek összehasonlítása a teszt adatbázison, különböző küszöbértékekkel

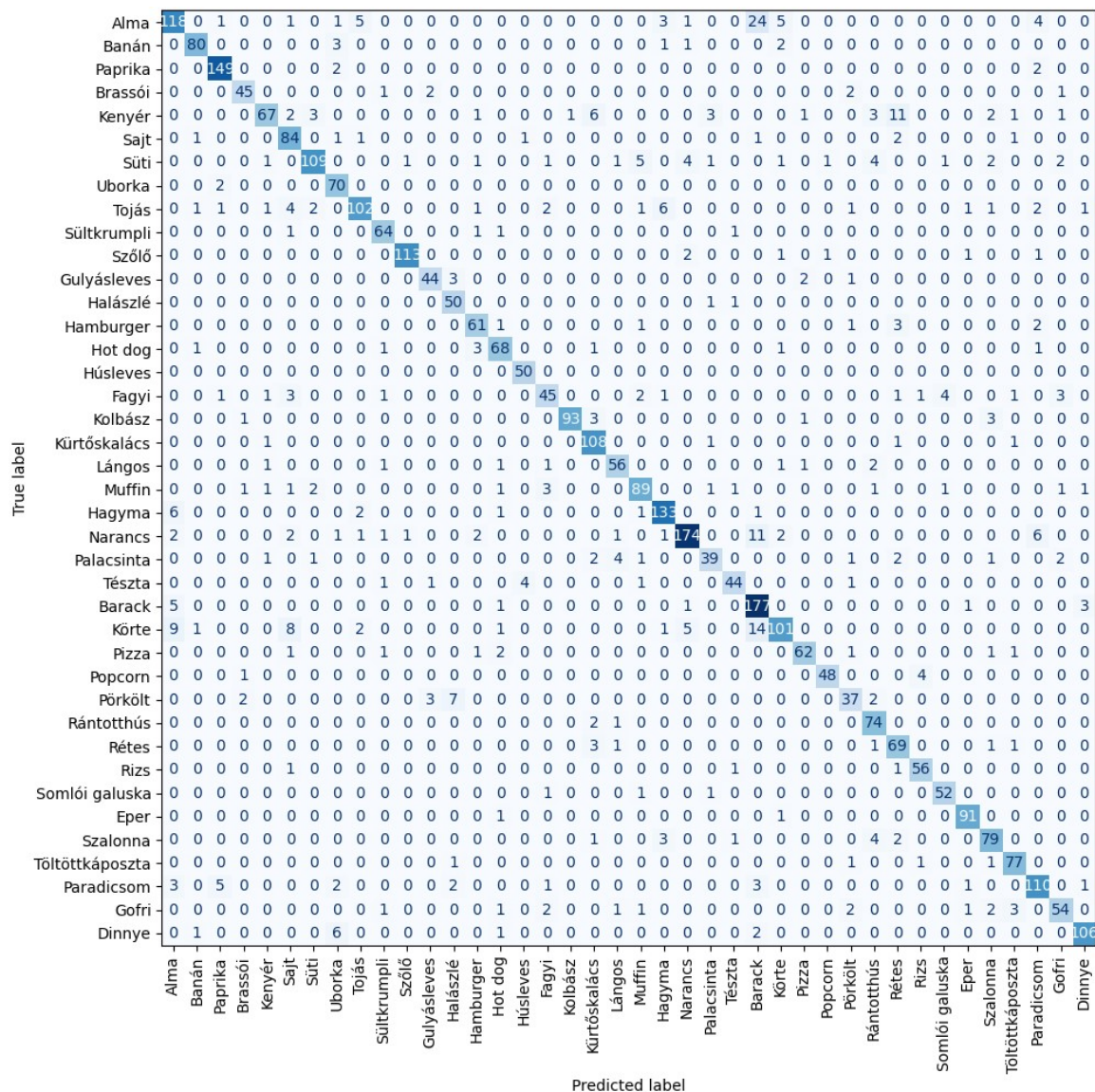
A táblázatból az következik, hogy a küszöbérték növelésével csökken a TP és FP, viszont nő a FN értéke. A feladat szempontjából olyan hálózatot lenne célszerű választani, ami a lehető legtöbb TP értékkel bír, hiszen ez alapján kerülnek a képek az egyes modellekhez. Ez esetben a 0,1-s küszöbértékkel bíró modellek valamelyike a legjobb, hiszen ott találhatóak a legjobb értékek a TP szempontjából. A probléma vele az, hogy az FP értéke is nagyon magas, nagyon sok kép kerülne olyan modellekhez, amikhez nem kéne. A másik végletben gondolkozva, azaz ahol a legkisebb az FP értéke, a 0,5-s küszöbértékű hálózatoknál található meg. A probléma itt az FN értékben keresendő, ugyanis sok objektumot nem ismer fel a hálózat. A valóságban ez azt eredményezi, hogy sok kép el sem jut a detektáló modellekhez, így végérvényesen csökkentve a felismerhető objektumok számát. Éppen ezért célszerű egy olyan metrika alapján választani, ami magában hordozza a TP, FP és FN arányokat. Ez a metrika pedig az F1-score, amit a 4.2 fejezetben részletesen bemutattem. Az F1-score értékeket vizsgálva sok modell ért el hasonló, 0,74-s és 0,75-s értéket, ezért ez után a hangsúlyt a Recall értékre helyeztem, mert ez a metrika mondja meg a TP és FN viszonyát, azaz azt, hogy hányat sikerült pontosan felismerni és hányat nem sikerült felismerni a képeken. Ezen szűkítések után azok a modellek maradtak, ahol a küszöbérték 0,3 volt. Ezek közül azt, amelyikben mindegyik paraméter tanítható volt, nem vizsgálok tovább, mert az F1-score értéke kisebb, mint a többi hálózatnak. A következőkben a TP és FN értékeket vizsgáltam meg. A TP értékének minél nagyobbának az FN értékének pedig minél kisebbnek kell lennie. Az FP értéke azért nem számít bele, mert itt még "hibázhat" a hálózat. A hiba itt azt jelenti, hogy ha azok az osztályok, amiket gyakran kever a hálózat egy objektum detektáló modellbe kerülnek, akkor a klasszifikáció pontatlansága nem befolyásolja a végeredményt. Végeredményül azt kaptam, hogy a 0,3-s küszöbértékkel bíró, transfer learninget nem használó modell adja a legjobb eredményeket, a három modell közül ennek van a legnagyobb TP és legkisebb FN értéke, ezért a továbbiakban ezt fogom használni a teszteléshez. A tanítás során a TensorBoard által készített gráfok a 4.4 ábrán láthatóak.



(a) Tanító (narancs) és validációs (kék) pontosság



4.4. Ábra: MobileNetV3 tanítás során



4.3.3 Objektum detektáló modellek felosztása, tanítása és értékelése

Az előzőekben a klasszifikációs hálózatot tanítottam be. Az eredményekből az következik, hogy több objektumot is gyakran kever a hálózat. Kiértékeléskor ezeket a felhasználó nem fogja látni, hogy a klasszifikálás során mennyit tévedett a hálózat, csak azt mutatja, amit az objektum detektáló modellek felismernek. Ennek megvan az az előnye, hogy azokat az ételek, amiket gyakran kever a hálózat, például az almát és barackot, célszerű lehet egy objektum detektáló hálózatba tenni, és úgy betanítani azt, így ha téved is a klasszifikáció során, akkor is jó detektálózhoz fog kerülni, és az pontosan felismerheti. Az adatbázisokat ebből kifolyólag négy klasszra osztottam. Az elsőbe kerültek a gyümölcsök és a zöldségek, ez összesen 12 osztály, a második klasszba a magyaros ételek kerültek, ez összesen 10 osztályt jelent. A magyaros ételek közé tartozik a brassói, gulyásleves, halászlé, húsleves, kolbász, lángos, pörkölt, rántotthús, szalonna és töltöttkáposzta. A harmadik klasszba kerültek a desszertek és egyéb édességek, ez 9 osztályt jelent. A negyedik klasszba válogattam az általánosabb ételeket: hamburger, hot dog, kenyér, sajt, tojás, sültkrumpli, tészta, pizza és rizs. Összesen ez is 9 osztályt jelent. Az adatbázisok előállítás után a tanítás az ajánlott beállítások mellett zajlott, az egyes iterációk száma pedig az osztályok száma szorozva kétezerrel volt. Az elkészült súlyokat minden ezredik iterációban mentettem és utána teszteltem. A 4.8 és 4.9 táblázatban csak a legjobban teljesítő modellek eredményeit összesítem.

Modell	TP	FP	FN	IoU	Precision	Recall	F1-score	mAP
Gyümölcsök és zöldségek	869	465	701	54,38%	0,65	0,55	0,6	59,60%
Magyaros ételek	524	169	143	63,17%	0,76	0,79	0,77	84,60%
Általános ételek	400	263	316	48,97%	0,60	0,56	0,58	61,50%
Desszertek	537	312	183	51,27%	0,63	0,75	0,68	73,85%

Tábla 4.8: Betanított modellek eredményei validációs adatbázison

Az így kapott eredményeket összevetve a 4.5 táblázatban bemutatott értékekkel azt lehet leszűrni, hogy összesen a helyes predikciók értéke 471-gyel volt magasabb a több hálózatot vizsgálva, míg az FP értéke pedig 239-cel volt kisebb. Az FN értéke 571-gyel volt kisebb. Tehát elmondható, hogy erre a problémában ugyanazokon a tesztadatokon jóval több objektumot volt képes helyesen detektálni, miközben a hibás prediktálás értéke csökkent.

Modell	TP	FP	FN	IoU	Precision	Recall	F1-score	mAP
Gyümölcsök és zöldségek	864	468	466	55,08%	0,65	0,65	0,65	66,06%
Magyaros ételek	485	191	143	59,35%	0,72	0,77	0,74	59,35%
Általános ételek	427	205	224	55,83%	0,68	0,66	0,67	76,14%
Desszertek	493	330	223	49,18%	0,60	0,69	0,64	69,33%

Tábla 4.9: Betanított modellek eredményei teszt adatbázison

4.4 Eredmények értékelése

Az már kiderült, hogy az előző 4.3 fejezetben bemutatott négy klasszra osztott hálózategyüttes jobb eredményeket ért el, mint az a hálózat, ami mind a 40 osztályt képes volt felismerni. Ez viszont még nem vezet eredményre, a klasszifikációs hálózattal együtt szükséges azokat tesztelni. A tesztelést a teszt adatbázison végeztem el. Először megvizsgáltam, hogy a klasszifikáció során milyen eredményeket ad a hálózat, az egyes teszt képekre és minden képet áthelyeztem a prediktált klasszba tartozó mappába. Így összesen négy mappa jött létre, amik tartalmazták a prediktált képeket. Egy kép itt több klasszba is tartozhat, vagyis megeshet az, hogy benne van a gyümölcs és zöldség klasszba és a desszert klasszba is. Ezek után lefuttattam a teszteket a detektáló hálózatokon. A következő 4.10 táblázat foglalja össze, hogy hogyan teljesített a két hálózat egymással szemben.

	Összes ételt tartalmazó hálózat			Hálózategyüttes		
Étel neve	AP	TP	FP	AP	TP	FP
Alma	56,07%	71	33	55,04%	69	26
Banán	29,38%	18	31	45,87%	40	32
Paprika	52,28%	70	68	76,26%	90	53
Brassói	90,42%	43	28	92,38%	45	19
Kenyér	27,67%	16	31	56,75%	32	26
Sajt	31,77%	27	65	50,26%	30	42
Sütemény	42,52%	69	101	38,85%	38	86
Uborka	42,21%	21	34	46,23%	31	30
Tojás	35,32%	43	46	69,94%	59	30
Sültkrumpli	59,07%	24	31	85,90%	36	13

Szőlő	36,15%	21	38	61,11%	43	25
Gulyásleves	88,14%	38	17	89,80%	44	21
Halászlé	85,65%	46	22	83,64%	37	22
Hamburger	69,75%	34	20	74,19%	37	9
Hot dog	57,7%4	21	28	82,97%	38	19
Húsleves	89,92%	40	22	94,20%	46	19
Fagyi	38,56%	17	13	39,28%	21	24
Kolbász	44,25%	27	44	67,30%	36	30
Kürtőskalács	75,84%	75	37	86,76%	91	29
Lángos	78,40%	41	16	88,99%	52	23
Muffin	36,93%	41	70	48,53%	83	90
Hagyma	71,44%	101	62	78,91%	145	68
Narancs	50,23%	88	42	71,36%	84	33
Palacsinta	51,55%	23	31	67,89%	36	41
Tészta	65,58%	31	21	85,39%	38	26
Barack	58,46%	113	100	77,76%	87	89
Körte	52,44%	64	24	94,11%	84	25
Pizza	66,59%	36	32	92,45%	42	12
Popcorn	71,16%	39	26	80,82%	33	19
Pörkölt	51,56%	22	29	83,97%	36	29
Rántotthús	59,74%	34	27	71,01%	44	25
Rétes	59,54%	39	37	74,23%	53	35
Rizs	94,42%	53	14	94,28%	52	20
Somlói galuska	79,25%	37	21	95,38%	50	17
Eper	42,62%	35	40	73,42%	37	42
Szalonna	63,44%	47	29	84,48%	54	25
Töltöttkáposzta	57,99%	38	27	68,04%	46	24
Paradicsom	29,16%	27	46	31,50%	33	38
Gofri	46,01%	27	15	71,34%	41	34
Dinnye	58,25%	62	34	84,12%	63	22

Tábla 4.10: A hálózatok összehasonlítása

Végeredményben a 4.2 fejezetben bemutatott hálózatnak a TP értéke 1798 volt, míg a futószalag szerű klasszifikációs neurális hálózategyüttesnek 2056 volt ez az érték. A különbség 258 a két TP érték között, ami azt jelenti, hogy a második esetben ennyivel tudott több képet felismerni, miközben az FP értéke 1433-ról 1270-re csökkent. Ez 163-mal kevesebb hibás felismerést jelent. A FN értéke a hálózategyüttesnél 854 volt, míg az összes ételt tartalmazó hálózatnál ez az érték 1627 volt, a különbség itt 773. Részletesebben vizsgálva, mindössze 7 ételnél figyelhető meg, hogy kevesebbet volt képes felismerni a hálózategyüttes. Ezeknek egy része elenyésző, azaz nagyon kicsivel - 5-nél alacsonyabb - volt kisebb a TP értéke. Számottevően a süteményből, a halászléből, a barackból és a popcornból volt kevesebb ez az érték. Az FP értékeknél 10 esetben volt több hibás felismerés a hálózategyüttesnél. A különböző metrikák összehasonlítását a 4.11 táblázat mutatja be.

Metrika	Összes ételt tartalmazó hálózat	Hálózategyüttes
TP	1798	2056
FP	1433	1270
FN	1627	854
Precision	0,56	0,61
Recall	0,52	0,70
F1-score	0,54	0,65

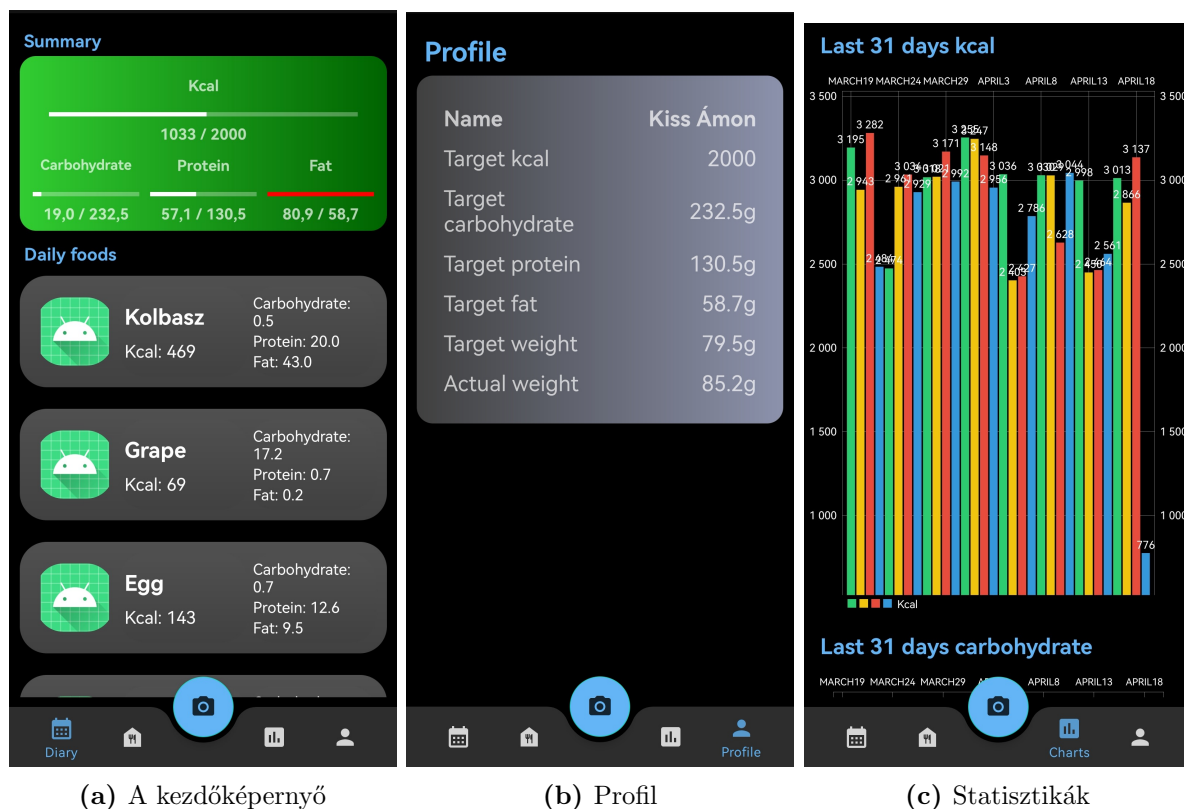
Tábla 4.11: Az összes ételt tartalmazó hálózat és a futószalag szerű klasszifikációs neurális hálózategyüttes összehasonlítása

Ezen adatok alapján arra a megállapításra jutottam, hogy a futószalag szerű klasszifikációs neurális hálózategyüttes jobb teljesítményt ért el az ételek detektálása során. A mobil applikációba tehát ezeket a hálózatokat fogom implementálni.

5 Felhasználói dokumentáció

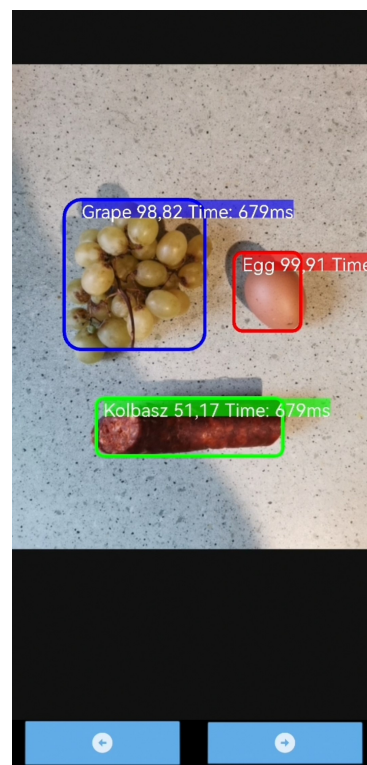
A program elindításához mindenképp egy Android operációs rendszerrel telepített eszközre van szükség. A kamera megléte nem feltétel az alkalmazás elindításához viszont ahhoz, hogy minden funkcióját használni lehessen elengedhetetlen egy jó minőségű kamera megléte az eszközben. Az alkalmazás tesztelésére egy Huawei P30 Pro készüléket használtam.

Elindításkor a kezdőképernyő fogad, ahol láthatóak az aznap elfogyasztott ételek és az azokban lévő tápanyagok összessége, előrehaladás jelzőkkel ellátva. Ha az adott nap túl sokat viszünk be valamelyik tápanyagból a program azt az előrehaladást piros színnel jelezve, hogy túllépte a beállított napi korlátot. Eltávolítani a listából egy balra vagy jobbra pöccintéssel lehetséges, változtatásra pedig a változtatni kívánt étel rábökése után van mód. A napi kalória és makrótápanyagok beállítására a profil menüpont alatt van lehetőség, itt mindenki személyre tudja szabni az alkalmazást a saját igényei szerint. Alapértelmezetten az értékek 2000 kcal, 230 gramm szénhidrát, 140 gramm fehérje és 60 gramm zsírt jelent. Az alsó menüpontok közül a második az adatbázisban szereplő ételeket listázza. Ehhez lehet hozzáadni olyan ételeket, amelyek nincsenek benne a neurális hálózat repertoárjában. Az alap 40 ételt, amit képes felismerni a hálózat nem lehet, az egyénileg hozzáadott ételek közül viszont ugyancsak egy jobbra vagy balra pöccintéssel lehet törölni. A negyedik menüpontban különböző statisztikákat láthatunk az elmúlt 31 napról. Az első négy diagramon sorban van kalória, szénhidrát, fehérje és zsír szerint az elmúlt 31 nap fogyasztása. Az utolsó kördiagram pedig a makrótápanyagok eloszlását mutatja ugyancsak az elmúlt 31 napból. 5.1 ábra a három menüpontot mutatja be.



5.1. Ábra: Főbb menüpontok bemutatása

A menüpontok közül a középsőt választva kamerán keresztül, valós időben lehet az ételeket felismerni. Egyszerre összesen 10 ételt képes felismerni az alkalmazás. Amint ez megtörtént és amennyiben sikeresen felismert egy vagy több ételt, megjelenik körülötte egy határoló doboz a képen és kiírja a valószínűségét, hogy milyen valószínűséggel az az adott objektum, amire a hálózat gondol és azt, hogy milyen étel látható a kép mely pontján. Abban az esetben, ha két határoló doboz nagyon egymáson van, érdemes lehet azt egyben kezelni. Mivel több neurális hálózat végzi a predikciót, közvetlen nem látják, hogy mit prediktált egy másik hálózat, ezért ezt célszerű lekezelni. Ez azzal az előnnyel jár, hogy egy almát a képen csak akkor fog 2 határolódobozzal körbe írni, ha a két prediktált doboz átfedése nagyobb, mint 0,6. A két doboz közül azt tartja meg, amelyiknek nagyobb a valószínűsége. Például, ha egy hagymát és almát prediktált a hálózat, előbbi 0,54 utóbbit 0,65 valószínűséggel és a határoló dobozok 0,6-nál nagyobb átfedésben vannak egymással, akkor csak az almát



5.2. Ábra: Képfelismerés

fogja kiírni a felhasználónak. Amennyiben szeretnénk továbblépni a jobb alsó gombra kell kattintani és ezek után megjelenik a felismert ételek listája. Ezek egységnyi adagra, 100 grammra, vonatkozó tápanyagtartalmai vannak feltüntetve, azokat átírni lehetséges. Ha volt olyan étel, amit nem volt képes felismerni, akkor azt kézzel is lehetőségünk van bevinni a már említett adatbázisból. Az 5.2 ábrán látható a felismerés folyamata. A time jelenti a feldolgozáshoz szükséges időtartamot, ami jelen esetben 679ms volt. A három étel három különböző objektum detektáló modell eredményéből került ki. Ha kevesebb modellhez kerül a felismerni kívánt étel ennek az időtartama lerövidül, 350-400ms egy étel felismerésénél. Természetesen a felhasználó által hozzáadott ételek is hozzáadhatóak. A mentést követően a főoldalra irányít át az alkalmazás.

6 Összehasonlítás már meglévő megoldásokkal

A 2.1 fejezetben bemutatott kipróbálható, letölthető és tesztelhető alkalmazásokkal szemben, mint a CalorieMama és Bitesnap, nem fénykép alapon valósítottam meg az ételek felismerését, hanem valós időben. Ezzel gyorsabb visszajelzést lehet adni a felhasználónak, így kényelmesebben és jobban használható. Funkciókat tekintve nem olyan széleskörű, mint az előbb említett alkalmazások, például nem lehet a galériából kiválasztott képen felismertetni az ételleket. A felismerni kívánt ételek listái nem fedik egymást, alkalmazásom kevesebb, azonban Magyarországon széles körben elterjedt ételleket képes felismerni, így regionális szinten használhatóbb, mint a konkurencia. További előny a telefon kamerájával történő felismerés. Manapság ezek az eszközök hatalmas tempóban fejlődnek a számítási kapacitásukat illetően. A válaszdíót a dolgozatban egy Huawei P30 Pro eszközön tesztetem, ami egy 2019-ben megjelent mobiltelefon. A mobiltelefonra Android 10.0 operációs rendszer volt telepítve a teszteléskor. Eredményként azt kaptam, hogy 750ms körüli időnél nem tartott tovább a felismerés. A később megjelent mobiltelefonok számítási kapacitása növekedésével ez az időtartam drasztikusan csökkenhet. Előnyt jelent az ételek pontos súlyának megadása, mivel az applikációban grammra pontosan meg lehet adni az ételek súlyát, ezért azokat pontosabban meg lehet határozni a bennük rejlő tápanyagtartalmakat.

Az Im2Calories alkalmazás képes felismerni éttermi ételleket, 2517 osztályból. Ezeket viszont nem valós időben teszi. További hátránya ezzel szemben, hogy csak 1 osztályt képes felismerni. A felismerhető osztályok nem valós, hanem éttermi ételleket képes detektálni, a valóságban azonban ez nem jól alkalmazható. A dolgozatban bemutatott alkalmazás ugyan kevesebb ételt képes felismerni, viszont azok jóval általánosabbak.

A FoodTracker már valós időben képes ételleket felismerni. A megoldása újszerű és valós adatokat tartalmaz, viszont leginkább japán ételleket képes detektálni. A gyorsasága a hálózathoz 75ms, ami rettentő gyors, jelen dolgozatban bemutatott hálózategyüttes ennek az időnek a tízszerese alatt képes felismerni az objektumokat, mobil eszközökön. A MobileNet és YOLOv2 kombinálásából a hálózat jó eredményeket ért el a tesztek során, viszont az alkalmazásban nincs lehetőség az ételleket naplózni, nincs semmilyen másik funkciója a felismerésen kívül.

7 Összefoglalás

Jelen dolgozatban a valós időben történő ételfelismerést mutattam be, Android operációs rendszerrel bíró mobil eszközön. A megoldásra alkalmas módszerek áttekintése után a rendszert megterveztem, a különböző alkalmazható technikák áttekintése után arra jutottam, hogy a feladat megoldására neurális hálózatként a YOLOv4 egy módosított változatát, a YOLOv4 tiny-t fogom használni az ételek detektálásra. Ezt követően a tanító, validációs, valamint teszt adatbázisokat előállítottam, elláttam címkékkel. Összesen 40 különböző étel szerepel az adatbázisban és ezek között több, magyar specifikus étel is szerepel. A hálózatot egy RTX 3070 videokártya segítségével tanítottam be. A tanítás során több súlyfájl is készült, amiket összehasonlítottam és kiválasztottam közülük azt, ahol a legnagyobb volt az mAP értéke.

Az előbb említett hálózatot ezek után összehasonlítottam egy futószalag szerű klasszifikációs neurális hálózategyüttessel. A klasszifikációra a MobileNetV3 modellt választottam és betanítottam összesen négyféleképpen. Tanításra, validációra és tesztre ugyan azokat az adatokat használtam. A hálózatokat összehasonlítottam különböző küszöbértékekkel és kiválasztottam közülük azt, amelyik a legjobb eredményeket érte el a feladat szempontjából.

Ezek alapján az adatbázist négy felé osztottam, és betanítottam négy objektum detektáló YOLOv4 tiny modellt. A négy hálózat eredményeit összehasonlítottam azzal a hálózattal, amiben mindegyik étel szerepelt, majd azt a következtetést vontam le, hogy jobb eredményeket képes így elérni a négy kisebb modell, mint az egy, ami mindent tartalmaz. Következő lépésként a hálózategyüttest teszteltem, majd összevettem a két lehetséges megvalósítást. Arra a megállapításra jutottam, hogy a hálózategyüttés jobb eredményeket ért el, így a mobil applikációba ezen modelleket implementáltam.

Végezetül a mobil applikáció fejlesztését valósította meg. Az alkalmazás egyes menüpontjait és funkcióit bemutattam, azokat dokumentáltam.

7.1 Továbbfejlesztési lehetőségek

A projekt fejlesztése során több olyan gondolat is felmerült bennem, amellyel tovább lehetne fejleszteni az elkészült alkalmazást. Az egyik legszembetűnőbb tény, hogy régebbi mobil eszközök szerényebb teljesítményű hardverekkel rendelkeznek és így a felismerés gyorsasága is drasztikusan csökkenhet. Megoldás lehet erre a problémára, ha nem csak telefonon, hanem egy távoli szerveren kerülnének feldolgozásra a kapott képek, így lehetővé téve annak kisebb válasz idejét.

Összesen 40 különböző étel található a felismerhető ételek között, viszont ez a szám a valóságban jóval nagyobb. Hasznos lenne még további ételek felvétele, és azokon végzett tanítás, hogy még több étel lehessen felismerhető egy felhasználó számára. A különböző elkészítési szokások miatt is több probléma felmerül a hálózattal. Nem biztos,

hogy egy pörköltöt vagy húslevest mindenki ugyan úgy készít el, további tanító adatok hozzáadásával ezen területen is lehet még csiszolni a hálózaton. A probléma megoldására még számos létező neurális háló architektúra áll rendelkezésre, ezek betanítása és tesztelése külön feladat lehet a legjobb eredmények megtalálásához.

Az applikáció funkcióinak listáját is bővíteni lehet például egy étrend tervezővel, amivel előre meg tudjuk adni azt, hogy mikor, mit fogunk enni a héten. Az alkalmazás ezek után automatikusan tudná vezetni azt, hogy mikor mit naplózzon, nekünk csak a változtatásokat kéne beírni, hogy mikor ettünk mást, mint a tervezett étrendben. További hasznos funkció lehet még az ételek gyors hozzáadása is, ami azon ételeket veszi figyelembe, hogy mit eszünk sűrűn és azokból mennyit. Ezen adatokat felhasználva egy kattintással lehetne új ételeket felvenni a naplóba. A makrotápanyagok mellett fontos szerepet játszanak az egészséges táplálkozásban a különböző mikrotápanyagok és vitaminok is. Ezen adatok hozzáadásával még több információ derülhetne ki az étkezési szokásunkról. Továbbá hasznos funkció a galériából történő kép alapján való predikció. Ez ugyan nem valós időben történik, viszont lehetőséget ad a felhasználónak arra, hogy ne kelljen étkezés előtt a dokumentálással foglalkozni. A boltban vásárolt termékek gyakran el vannak látva valamilyen vonalkóddal. Ezeket a vonalkódokat is fel lehetne használni az alkalmazásban, lényegében nem lenne szükség a felismerésre minden esetben. [3] [4]

Ábrajegyzék

2.1	CalorieMama alkalmazás bemutatása	6
2.2	Bitesnap alkalmazás bemutatása	7
2.3	Neurális hálózat rétegei [11]	9
2.4	Lineáris aktivációs függvény [17]	12
2.5	Aktivációs függvények [17]	14
2.6	Mish aktivációs függvény [18]	14
2.7	Zéró és konstans kitöltés [20]	17
2.8	Tükrözéses és Ismétléses kitöltések [20]	17
2.9	Pooling rétegek [21]	18
2.10	LeNet-5 architektúra [23]	20
2.11	AlexNet architektúra [24]	21
2.12	VGG-16 architektúra [25]	22
2.13	Inception modul architektúra [27]	23
2.14	Mélység növelése és pontosság összefüggése [28]	24
2.15	Reziduális blokk [28]	24
2.16	YOLOv4 Architektúrája [30]	26
2.17	MobileNetV3 Large rétegei [37]	28
3.1	A rendszerben használt táblák felépítése	35
3.2	A rendszer komponensei	36
3.3	Felhasználási eset diagram	37
3.4	Szekvencia diagram	37
4.1	Veszteség az iterációk függvényében	41
4.2	Futószalag szerű hálózategyüttes architektúra	45
4.3	A kimenetek lehetőségei egy képre, amin alma látható	47
4.4	MobileNetV3 tanítás során	52
4.5	Tévesztési mátrix	52
5.1	Főbb menüpontok bemutatása	59
5.2	Képfelismerés	59

Táblázatok

2.1	Bemutatott neurális hálózatok összehasonlítása	25
4.2	Tanító, validációs és teszt adatbázisok száma az egyes ételekre lebontva	40
4.3	Tévesztési mátrix	42
4.4	Súlyfájlok összehasonlítása a validációs adatbázison	43
4.5	Súlyfájlok összehasonlítása a teszt adatbázison	44
4.6	Betanított modellek összehasonlítása validációs adatbázison	49

4.7	Betanított modellek összehasonlítása a teszt adatbázison, különböző küszöbértékekkel	50
4.8	Betanított modellek eredményei validációs adatbázison	53
4.9	Betanított modellek eredményei teszt adatbázison	54
4.10	A hálózatok összehasonlítása	56
4.11	Az összes ételt tartalmazó hálózat és a futószalag szerű klasszifikációs neurális hálózategyüttes összehasonlítása	57

Egyenletek

1	Neuron, mint matematikai függvény	10
2	Lineáris aktivációs függvény	12
3	Szigmoid függvény	13
4	Tangens hiperbolikus függvény	13
5	ReLU függvény	13
6	Mish függvény	14
7	Softmax függvény	15
8	Accuracy egyenlete	42
9	Precision egyenlete	42
10	Recall egyenlete	42
11	F1-score egyenlete	42
12	IoU egyenlete	43
13	mAp egyenlete	43

Irodalomjegyzék

- [1] <https://foodai.org/> Utolsó hozzáférés dátuma: 2022.12.03
- [2] <https://bite.ai/> Utolsó hozzáférés dátuma: 2022.12.03
- [3] A. Myers, N. Johnston, V. Rathod, A. Korattikara, A. Gorban, N. Silberman, S. Guadarrama, G. Papandreou, J. Huang, K. Murphy *Im2Calories: towards an automated mobile vision food diary* 2015 IEEE International Conference on Computer Vision (ICCV), pp. 1233-1241, doi: 10.1109/ICCV.2015.146, 2015
- [4] Jianing Sun, Katarzyna Radecka, Zeljko Zilic *FoodTracker: A Real-time Food Detection Mobile Application by Deep Convolutional Neural Networks* arXiv:1909.05994v2, 16 Sep 2019
- [5] Joseph Redmon, Ali Farhadi *YOLO9000: Better, Faster, Stronger* arXiv:1612.08242v1, 25 Dec 2016
- [6] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications* arXiv:1704.04861v1, 17 Apr 2017
- [7] <http://foodcam.mobi/dataset100.html>, Utolsó hozzáférés dátuma: 2022.12.03
- [8] <http://foodcam.mobi/dataset256.html>, Utolsó hozzáférés dátuma: 2022.12.03
- [9] Simon Haykin. *Neural Networks and Learning Machines Third Edition*. Pearson Education, Inc., Upper Saddle River, New Jersey 07458, 2009
- [10] Michael Nielsen. *Neural Networks and Deep Learning*. Available: <http://neuralnetworksanddeeplearning.com/gradientdescent>
- [11] Altrichter Márta, Horváth Gábor, Pataki Béla, Strausz György, Takács Gábor, and Valyon József. *Neurális hálózatok*. Hungarian Edition Panem Könyvkiadó Kft., Budapest, 2006
- [12] Katarzyna Janocha, Wojciech Marian Czarnecki. *On Loss Functions for Deep Neural Networks in Classification*. arXiv:1702.05659v1, 18 Feb 2017
- [13] Yoshua Bengio *Deep Learning of Representations for Unsupervised and Transfer Learning* ICML-2011 Workshop on Unsupervised and Transfer learning pp. 17-36
- [14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov *Dropout: A Simple Way to Prevent Neural Networks from Overfitting* Journal of Machine Learning Research 15 (2014) 1929-1958

- [15] Xue Ying. *An Overview of Overfitting and its Solutions*. J. Phys.: Conf. Ser. 1168 022022, 2019
- [16] Siddharth Sharma, Simone Sharma, Anidhya Athaiya *ACTIVATION FUNCTIONS IN NEURAL NETWORKS* Vol. 4, Issue 12, ISSN No. 2455-2143, Pages 310-316, April 2020
- [17] Chigozie Enyinna Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall *Activation Functions: Comparison of Trends in Practice and Research for Deep Learning*. arXiv:1811.03378v1, 8 Nov 2018
- [18] Diganta Misra *Mish: A Self Regularized Non-Monotonic Activation Function*. arXiv:1908.08681v3, 13 Aug 2020
- [19] Yann LeCun, Yoshua Bengio, Geoffrey Hinton. *Deep learning*. doi:10.1038/nature14539.
- [20] Guilin Liu, Kevin J. Shih, Ting-Chun Wang, Fitsum A. Reda, Karan Sapra, Zhiding Yu, Andrew Tao, Bryan Catanzaro *Partial Convolution based Padding* arXiv:1811.11718v1, 28 Nov 2018
- [21] Hossein Gholamalinezhad, Hossein Khosravi. *Pooling Methods in Deep Neural Networks, a Review*. arXiv:1811.03378v1, 8 Nov 2018.
- [22] <http://www.image-net.org/challenges/LSVRC/> Utolsó hozzáférés dátuma: 2022.12.03
- [23] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. *Gradient-Based Learning Applied to Document Recognition*. November, 1998.
- [24] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. *ImageNet Classification with Deep Convolutional Neural Networks*. arXiv:1311.2901v3, 28 Nov 2013.
- [25] Karen Simonyan, Andrej Ziesserman. *Very Deep Convolutional Networks for Large-scale Image Ecognition*. arXiv:1409.1556v6, 10 Apr 2015.
- [26] Min Lin¹, Qiang Chen, Shuicheng Yan. *textitNetwork In Network*. arXiv:1312.4400v3, 4 Mar 2014.
- [27] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich. *Going Deeper with Convolutions*. arXiv:1409.4842v1, 17 Sep 2014
- [28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun *Deep Residual Learning for Image Recognition*. arXiv:1512.03385v1, 10 Dec 2015

- [29] Joseph Redmon *Darknet: Open Source Neural Networks in C*. 2013 - 2016 Available: <http://pjreddie.com/darknet/> Utolsó hozzáférés dátuma: 2022.12.03
- [30] Alexey Bochkovskiy, Chien-Yao Wang, Hong-Yuan Mark Liao *YOLOv4: Optimal Speed and Accuracy of Object Detection*. arXiv:2004.10934v1, 23 Apr 2020 Available: <https://github.com/AlexeyAB/darknet> Utolsó hozzáférés dátuma: 2022.12.03
- [31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun *Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition*. arXiv:1406.4729v4, 23 Apr 2015
- [32] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, Jiaya Jia *Path Aggregation Network for Instance Segmentation*. arXiv:1803.01534v4, 18 Sep 2018
- [33] Chien-Yao Wang, Hong-Yuan Mark Liao, I-Hau Yeh, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh *CSPNET: A NEW BACKBONE THAT CAN ENHANCE LEARNING CAPABILITY OF CNN*. arXiv:1911.11929v1, 27 Nov 2019
- [34] Chien-Yao Wang, Alexey Bochkovskiy, Hong-Yuan Mark Liao *Scaled-YOLOv4: Scaling Cross Stage Partial Network*. arXiv:2011.08036v2 22 Feb 2021
- [35] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, Piotr Dollar *Microsoft COCO: Common Objects in Context*. arXiv:1405.0312v3, 21 Feb 2015
- [36] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. arXiv:1801.04381v4, 21 Mar 2019
- [37] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, Hartwig Adam *Searching for MobileNetV3* arXiv:1905.02244v5, 20 Nov 2019
- [38] *tensorflow-yolov4-tflite* Available: <https://github.com/hunglc007/tensorflow-yolov4-tflite> Utolsó hozzáférés dátuma: 2022.12.03
- [39] *labelImg Tool* Available: <https://github.com/tzutalin/labelImg> Utolsó hozzáférés dátuma: 2022.12.03
- [40] *Image Download Tool* Available: <https://github.com/theAIGuysCode/Download-Google-Images> Utolsó hozzáférés dátuma: 2022.12.03
- [41] Diederik P. Kingma, Jimmy Lei Ba. *ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION*. arXiv:1412.6980v9 30 Jan 2017