SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM
MAROSVÁSÁRHELYI KAR
SZÁMÍTÁSTECHNIKA SZAK

NÖVÉNYMAGOK VALÓS IDEJŰ FELISMERÉSÉRE ÉS FIZIKAI TULAJDONSÁGAIK BECSLÉSÉRE ALKALMAS MOBIL-APPLIKÁCIÓ TERVEZÉSE ÉS KIVITELEZÉSE

Témavezető:
Dr. Brassai Sándor Tihamér,
egyetemi docens

Végzős hallgató:
Bíró Apor

2024

1. Bevezető

A Számítástechnikában, a Mesterséges Intelligencia korunkban hatalmas figyelemnek őrvend. Ugyebár az MI ezen a szakterületen már elég régóta jelen van, úgymond itt "született", de mégis csak az utóbbi pár évben ért el olyan áttöréseket - gondolok akár a GPT nyelvi modellekre, kép- és formaalkotó algoritmusokra, illetve pontos és könnyen használható képfelismerő algoritmusokra, mint például a YOLO - amelyek már nem csak a számítógéptudósok és -mérnökök figyelmét, hanem a más szakterületekben, mint például orvostudományokban, társadalomtudományokban vagy éppenséggel agrártudományokban munkálkodók figyelmét is felkeltették.

A cél egy olyan mesterséges neurális hálózatokat alkalmazó szoftver megvalósítása volt, amely valós időben, egy okostelefon kameráján keresztül képes felismerni egy képkockán lévő különböző magfajtát, ezeket osztályozni, ezek után pedig képes legyen becslést végezni ezeknek a méretére, tömegére, térfogatára illetve ebből fakadóan, az egymás közötti arányaikra. Ezt az applikációt felhasználási szempontból majd alkalmazni lehessen a mezőgazdaságban, olyas értelemben például hogy elkülönítsen több magfajtát, így ki tudja szűrni a nem egy fajhoz tartozó magokat vagy például mérleg nélkül tudjon tömeget becsülni.

Azért ezt a MI-hoz kapcsolódó témát választottam, mert amióta a Mesterséges Intelligencia tantárgy keretein belül mélyebb betekintést nyertem ennek a működésébe, történetébe, felhasználási területeibe, és a szerteágazó sok lehetőségébe, elkezdett kifejezetten érdekelni. Hirtelenjében még nem tudtam magamtól elképzelni, hogy milyen témát keressek amely érdekelne is, meg lenne benne potenciális felhasználási lehetőség, ezért témavezetőmtől kértem segítséget aki MI gyakorlat tanárom is volt. Ő tartotta a kapcsolatot a Kertészmérnöki tanszék tanáraival és egyeztetett velük, hogy milyen projekteket lehetne közösen kezdeményezni, amelyekben ők is asszisztálhatnának. Az egyik ötlet egy növényfelismerés alapján működő gyomláló-robot, a másik ötlet pedig ez a magfelismerő szoftver volt. Eredetileg a robotot akartuk megcsinálni, a tavalyi évben azt a dolgozatot is választottuk, de mivel nem sikerült elég időt szentelni azon a nyáron a tanítóhalmaz gyűjtésnek, ezért inkább ennek a projektnek fogtam neki.

Hatalmas köszönetet szeretnék mondani ezúton is vezető tanáromnak, Dr. Brassai Sándor Tihamér tanár úrnak a szakmai segítségért és útmutatásért, akinek volt türelme hozzám, akkor is amikor néha nehezen munkálkodtam, Dr. Biró-Janka Béla tanár úrnak a kertészmérnöki tanszékről,

végül de nem utolsó sorban pedig Kelemen Tamás másodéves kertészmérnök szakos hallgatónak a tanítóhalmazok elkészítésében való segédkezésért.

2. Szakirodalmi tanulmány és elméleti megalapozás

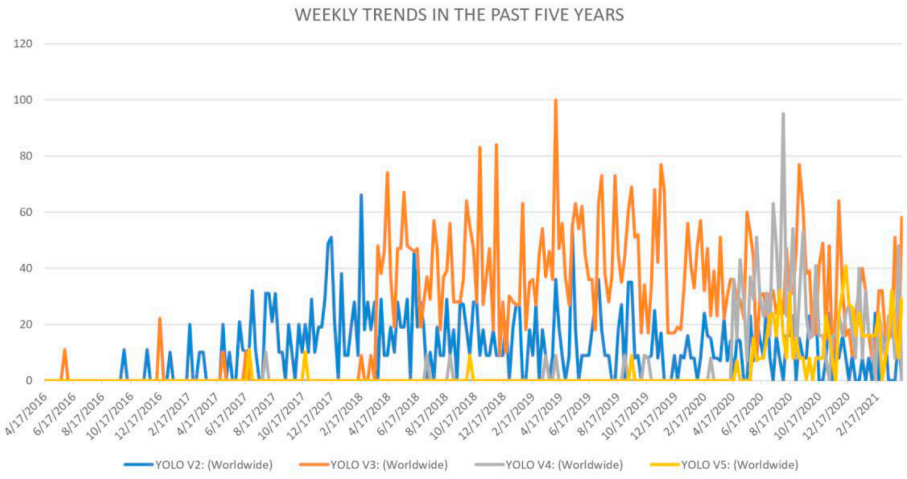
2.1. Szakirodalmi tanulmány

2.1.1. YOLO objektumdetektáló algoritmusok

Szakirodalmi tanulmányozást legelőször a nyári gyakorlatom alatt kezdtem el, ekkor még az terv az volt, hogy egy gyomnövény felismerő programot kell készítsek. Ekkor ismerkedtem meg a YOLO (You Only Look Once) algoritmussal. A YOLO az egyik legnépszerűbb objektumfelismerő algoritmus, amint a legtöbb ehhez hasonló, CNN-ek (Konvolúciós Neurális Hálózat) felhasználásával működik. Egyszerű a használata, a tanítása, az alkalmazása. Emellett az algoritmus gyors, valós idejű működést tesz lehetővé, ami azt jelenti, hogy képes az objektumok felismerésére szinte azonnali. Készítettem erre még akkor egy egyszerűbb kis programot, mely videóról tudta felismerni a Százszorszép virágait, ez bizonyította az előbb említett tulajdonságait.

Amikor végleg eldöntöttem, hogy a jelenlegi projekttel fogok foglalkozni, a képfelismerő algoritmusok magokon való alkalmazásainak tanulmányozására fektettem legelőször időt. A tanulmányok között szintén rengeteg YOLO-s projekt volt. Konkrétan a rengeteg YOLO-t alkalmazó dolgozat közül alig lehetett találni más technológiákra összpontosító tanulmányokat. Arra a következtetésre jutottam, hogy mivel már amikor legelőször kipróbáltam a YOLO-t és saját szememmel láttam, hogy milyen pontos muszáj szerepeljen a dolgozatomban. A következőkben részletesebb kutatást végeztem magáról a YOLO algoritmusról, illetve alkalmazásáról a magfelismerés terén.

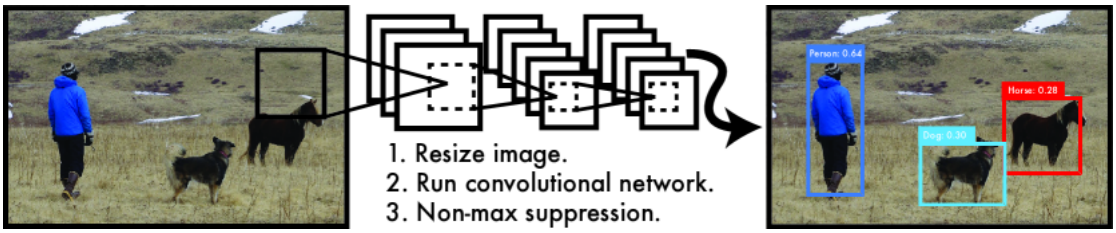
Az első YOLO 2015-ben jelent meg. Ez a verzió még nem ért el akkora ismeretséget és felhasználást mint az utódai. Az első verzióban munkálkodó CNN 24 konvolúciós réteget és utána 2 teljesen kapcsolt réteget tartalmazott, ennek volt két hibája: a pontatlan pozícionálás, illetve más akkori módszerekkel összehasonlítva az alacsonyabb visszahívási arány, a második verzióban ezeket a hibákat kijavították*[1]*. Ezeknek ellenére kezdete volt egy “dinasztiának”, amely a mai napig uralja a képfelismerés iránt érdeklődők figyelmét. A 2.1.1.1.-ik ábrán látható, hogy 2016-tól 2021-ig hogyan tornászta fel magát ez az algoritmus a csúcsra a népszerűséget tekintve*[1]*.



2.1.1.1. ábra A YOLO verziók népszerűsége[1]

Nem részletezném hosszan hogy a tanítási folyamat hogyan működik. Mint általánosan a CNN-ek tanításakor az adatok átméretezésre, illetve normalizálásra kerülnek, majd ismétlődő tanítási ciklusok során a YOLO súlyokat tanít[2]. A YOLO esetében a gyors és hatékony objektumdetektáláson van a hangsúly, ugyanis a *"You Only Look Once"* mondat egyszerűen annyit jelent, hogy egyszeri megnézése, egyszeri prediktálása egy bizonyos képkockának, ez jelentősen gyorsabb felismerési sebességhez vezet.

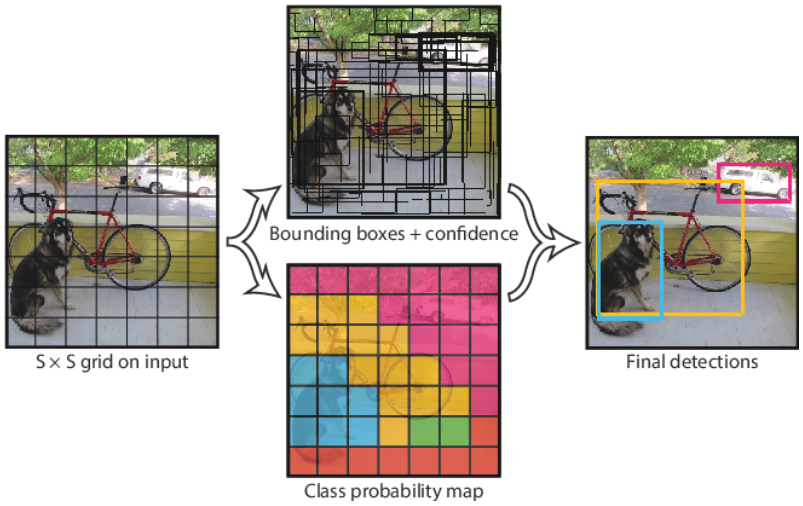
Az alábbi 2.1.1.2.-ik ábra szemlétet egy bounding box perdiktálást. Megfigyelhető az ábrán, hogy amint a képkockák végigmennek a YOLO-n, átméreteződnek, lefut rajtuk egyetlen CNN-en, végül egy non-max supression(NMS) algoritmust, amelynek a lényege hogy a duplikált, egymáson elhelyezkedő detektálásokat törli, csak a legvalószínűbbet tartja meg[2]. Eredményként pedig visszatérít detektálásonként egy bounding boxot (határoló doboz, keret), illetve hozzájuk tartozó osztályt és a detektálás konfidenciáját (pontosságának arányát)[2].



2.1.1.2. ábra A YOLO képkockánkénti működése[2]

A 2.1.1.3.-as ábrán látható, hogy mi történik egyetlen fotóval amin objektumokat kell detektálni. A YOLO felossza a bemeneti képet SxS méretű gridekre(rácsokra), minden gridben lévő

képkockára prediktál B bounding boxot, azoknak a konfidencia értékeit (azaz tartalmaz-e objektumot vagy sem), továbbá C darab osztály perdikcióját (vagyis az a bizonyos objektum ha van, akkor milyen valószínűséggel tartozik egyenként az osztályokhoz), ez a grid-based algoritmus[2]. Ezek a predikciók egy tenzorban lesznek eltárolva, aminek a méretét a 2.1.1.4.-es ábrán lévő képlettel lehet kiszámolni[2].



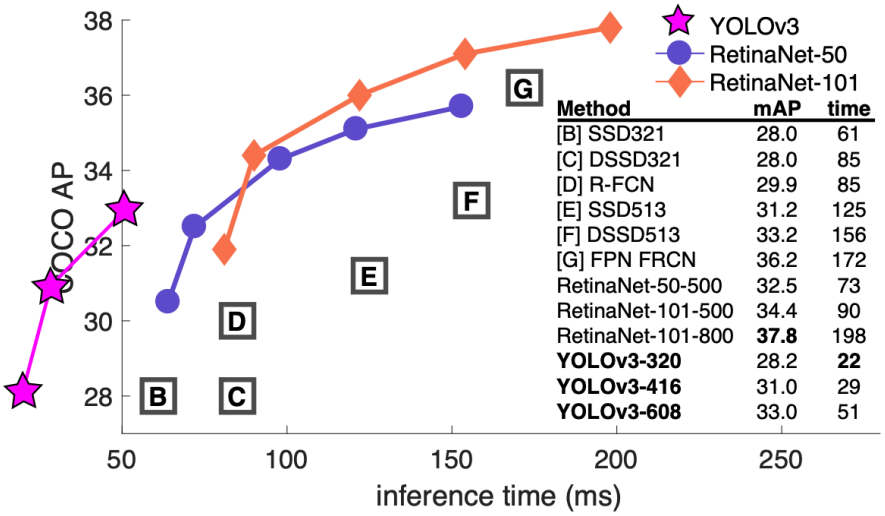
2.1.1.3. ábra A YOLO képfeldolgozása[2]

$$S \times S \times (B \times 5 + C)$$

2.1.1.4. ábra Predikciók mérete[2]

Az egyik legsikeresebb verziója a harmadik verzió (YOLO V3), amely már a fennebbi népszerűségi grafikonon is látszik, hogy kiemelkedik a többi közül. Ez a verzió nagy lépés volt az előző kettőhöz képest. Ámbár kicsit mélyebb hálót alkalmaztak ennél a verziónál, sikerült fejleszteni az alacsonyabb erőforrásigényű és hatékonyabb működést, ezek mellett ráadásul az eddigi leggyorsabb detektálási idővel rendelkezik és sokkal pontosabb az elődeihez képest[3]. A 2.1.1.5.-ik ábrán egy grafikon látható, amelyen látszik, hogy a YOLO V3 elhagyja detektálási sebességben a 2018-as kortárs hálóit, hasonló átlag pontossági metrika mellett. Látható, hogy a *"legkisebb"* 320x320-as harmadik generációs YOLO háló a leggyorsabb 22 milliszekundumos sebességgel, viszonylag kicsi 28.2 mAP-s (mean Average Precision) pontossággal[3]. A legnagyobb 608x608-as modell pedig ámbár nagyon jó 33.0 mAP-s pontosságot produkált, mégis a töredék

idejét futja a nála pontosabb vagy nagyjából megegyező pontosságú, de egyébként nagyobb RetinaNet, SSD, DSSD és FPN hálóknál[3].

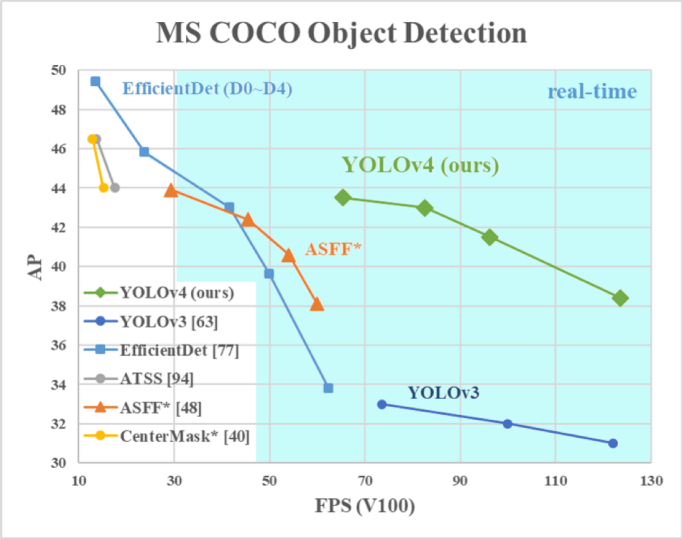


2.1.1.5. ábra YOLO V3 összehasonlítása[3]

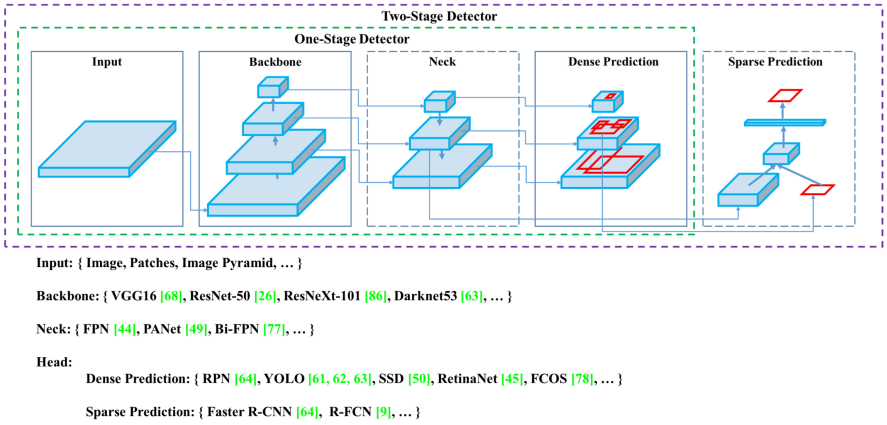
A következő verzió a YOLO V4 , amely már nem volt akkora ugrás a harmadik verzióhoz képest, kevés dolgot változtattak meg, inkább kiegészítettek részeket, például a YOLO V3 CNN-ét, a Darknet-53-at (53 konvolúciós réteg) megörökölte, épp a CSP-t (Cross Stage Partial) vezették be mellé, amely hatékonyabban kezeli az információ áramlását a hálózatban, így lett a háló neve CSPDarknet-53[1]. Ennek a fejlesztésében már nem fordítottak akkora figyelmet az alacsonyabb erőforrásigényre, ennek eredménye egy ugyan sokkal pontosabban és gyorsabban detektáló algoritmus lett, viszont a hozzáadott adat összehasonlító algoritmusok, illetve az összetettebb szoftver architektúra több számítási erőforrást igényel[1]. A YOLO V4 kb. 65 FPS-es (Framerates Per Second, Képkocka másodpercenként) sebességű valós idejű képfeldolgozással, egy NVIDIA Tesla V100-as grafikus gyorsítóval 43.5% AP-s (Average Precision) pontosságot tudott elérni. A következő 2.1.1.6.-ik ábrán az figyelhető meg, hogy a YOLO V4 kétszer gyorsabban fut, mint a EfficientDet nevű háló, mindemellett hasonló teljesítménnyel[4]. Ez a verzió az előző harmadik verzióhoz képest 10% javulást hozott az AP és 12% javulást az FPS szempontjából[4].

A modell szoftverarchitektúrája is megváltozott, ugyanis itt már beszélünk Input(bemenet), Backbone(gerinc), Neck(nyak) és Head(fej) részekről, ezek a részek egyenként magukba foglalják a hálózat bizonyos rétegeit, továbbá különböző feladatokat látnak el a bemenet feldolgozása és prediktálása, illetve a tanulás során[4]. Az Input felel a kép vagy képek helyes bemenetéért, a Backbone felel a képek jellemzőinek kinyeréséért, a Neck a Backbone egy kiegészítő része amely a

PAN (Path Aggregation Network) segítségével összefűzi a különböző mélységű feature map-eket a a Backbone-ból, végül a Head pedig a predikciók készítéséért felel*[4]*. A alábbi 2.1.1.7.-ik ábrán láthatóak a YOLO V4 modellnek a részei, megfigyelhető hogy ezeknek a részeknek az együtteséből, ezeknek sorozatának, együttes munkájuknak köszönhetően jöhet létre a Dense Prediction(sűrű predikció), majd a Dense Prediction-ből pedig a Sparse Prediction(ritka predikció) *[4]*.



2.1.1.6. ábra YOLO V4 összehasonlítása[4]



2.1.1.7. ábra YOLO V4 architektúrája[4]

A következő verzió a YOLO V5, ezt a verziót már nem részletezném hosszabban. Az ötödik verzió már nem volt akkora innováció, mint a YOLO V4, viszont itt megint visszatértek az erőforrás optimalizáció szempontba vételére*[1]*. Egy nagyon *"lightweight"* (könnyű) háló lett, ennek köszönhetően erőforrás kímélő, ellenben az elődjével*[1]*. De ami megdöbbentő az, hogy mégis vetekedik a negyedik verzió pontosságával, megint csak az eddigi legpontosabb és leggyorsabb verzió lett*[1]*. Itt adtak ki legelőször több modell méretet hozzá, ez lett a small, a

medium, a large és az xlarge, ennek következtében a modell méretének a beállítása flexibilissé vált[1].

Mostmár ugyan megjelentek újabb verziói a YOLO-nak egészen a nyolcadik verzióig, de én a YOLO V3-at, a V4-et és a V5-öt fogom csak integrálni az alkalmazásomban. Ennek az oka az, hogy egy tanulmányban - amely egy valós idejű paprika detektáló Androidos program - arra jutottak, hogy az ötödik verzió feletti YOLO-k már nem produkáltak nagyobb predikciós sebességet. Ez azért kritikus az én projektemben: egyrészt mivel nekem valós idejű felismerés a terv - ráadásul egy fényképen egyszerre sok magot, sok objektumot fog kelleni detektáljon - számít az FPS-es feldolgozás; másrészt pedig nem áll rendelkezésemre majd a teszteléseknél Androidos csúcstelefon és amúgy is az lenne a személyes elvárásom, hogy ez az alkalmazás az átlagos teljesítményű okostelefonokon is működjön. A tanulmányban az eszköz egy 2020-ban kiadott közép-felső kategóriás okostelefon, a Realme X50 Pro[5]. A 2.1.1.1.-ik táblán láthatóak ezen eszköz kamerájának a specifikációi[5]. A 2.1.1.2.-ik táblán láthatóak a paprika felismerésnek az eredményei[5]. Megfigyelhető, hogy az ötödik verzióhoz képest a hetedik és nyolcadik verziónak a sebessége majdnem a felére csökken.

Parameter	Details
Camera Type	High-resolution digital camera
Camera Make and Model	Realme AI Quad Camera
Camera Specification	12MP main with 8 MP Ultra-wide-angle
Aperture	f/1.8
Focal Length	4 mm
Exposure Time	Auto-adjusting (range: 1/100 s – 1/500 s)
ISO Sensitivity	100—800 (auto-adjusting)
Image width × height	4000 × 1800 pixels
Horizontal and Vertical resolution	72 dpi
Image Format	JPG
Environmental Conditions	Controlled greenhouse environment
Lighting Conditions	Natural and artificial lighting sources
Scene Selection Criteria	• Capsicum plants cantered in frame; Different level of occlusion and overlapping; Uniform illumination across scene; Various growth stages and angles

2.1.1.1. tábla X50 Pro kamerájának specifikációi[5]

Task	Models	Size (MB)	Max. Precision (at Confidence)	Max. Recall (at Confidence)	Max. F1 score (at Confidence)	mAP @ 0.5	Detection Speed (FPS)
Capsicum Detection	YOLOv5s	13.7	1 (0.869)	0.97(0)	0.82(0.391)	0.843	71.42
	YOLOv5l	88.4	1 (0.708)	0.98 (0)	0.80	0.77	25
	YOLOv7s	71.3	1 (0.826)	0.90(0)	0.91(0.138)	0.904	44.24
	YOLOv8s	21.5	1 (0.697)	0.98(0)	0.93 (0.146)	0.967	47.16

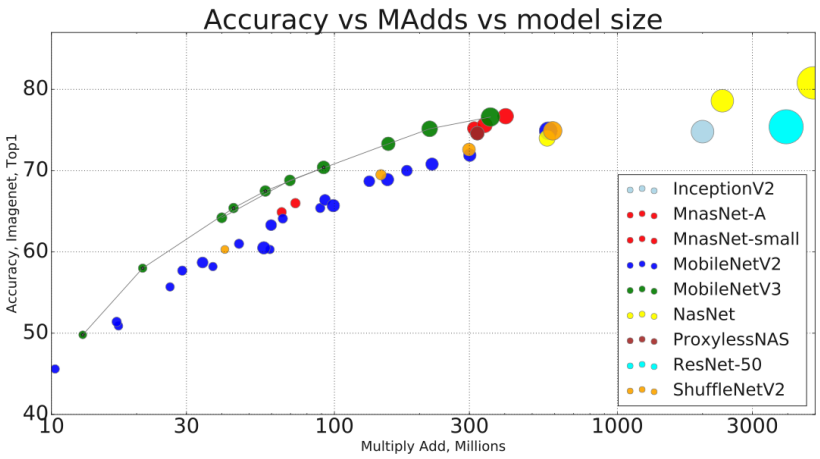
2.1.1.2. tábla Részlet a tanulmány modell eredmények táblájából[5]

2.1.2. Egyéb objektumdetektáló algoritmusok

Miután befejeztem a YOLO utáni kutatást elkezdtem keresgélni más okostelefonokra fejlesztett algoritmusok után. Felfedeztem a Faster R-CNN-t, a MobileNet-et, az MobileNet-SSD-t, az EfficientDet-et, illetve a ShuffleNet-et, a régebbi SqueezeNet-et. Ezek közül a Faster R-CNN-el készült dolgozatot sikerült találnom, de ez egy lassabb, nagyobb erőforrású igényű modell, tehát nem ideális valós idejű felhasználásra, inkább már elkészült képről történő felismerő alkalmazásokra használják*[6]*. A többi algoritmusnak már viszont alaposabban utánajártam.

A MobileNet hálózatokat a Google fejleszti és határozottan okostelefonon történő gépi-látásra vannak tervezve. Az első MobileNet-et 2017-ben adták ki, állításuk szerint amiatt, mert egy egyre csak növekvő igény van jelenleg kisebb és hatékony neurális hálózatok fejlesztésére*[7]*. Az akkori *"trendek"* azok voltak, hogy minél komplikáltabb és mélyebb hálókra összpontosítani a nagyobb pontosság elérése érdekében, viszont ezek egyre és egyre erőforrás-igényesebbek is lettek*[7]*. A legelső hálókhoz, vagyis a MobileNetV1-hez könnyű (lightweight) CNN-eket használtak, az ezeken belüli konvolúciós rétegek pedig mélységi konvolúciós műveleteket (depthwise convolution) hajtottak végre*[7]*. A mélységi konvolúció röviden annyit jelent, hogy a bemenet minden csatornájára külön-külön konvolúciós műveleteket végzünk, ellenben a hagyományos párhuzamos konvolúcióval, ahol ugye egy meghatározott szűrő halad végig a képen. Egy másik változtatás amely elősegítette az erőforrásokkal hatékonyabban *"bánást"* az a hiperparaméterek (hyper-parameters) - mint például a tanulási ráta, batch méret, tanulási ciklusok stb. - csökkentése volt*[7]*.

A második verziója a MobileNet-nek két fontos újítást hozott, ez a Linear Bottlenecks és az Inverted Residuals, ezek az információáramlásban és a hatékonyabb erőforrás kezelésben segítettek*[8]*. A legutóbbi verzió a harmadik, fontosabb újítás volt hogy megörökölte a MobileNetV2 alapján megtervezett MnasNet fejlesztéseit, mint például a lightweight attention (könnyűsúlyú figyelő) modulokat, illetve a Squeeze-and-Excite figyelő mechanizmust*[9]*. Ezenkívül az egyik legfontosabb újítás az volt, hogy a MobileNetV3 már nemlineáris Swish aktivációs függvényt használ*[7]*. A 2.1.2.1.-ik ábra azt mutatja, hogy a MobileNetV3 mennyivel lett pontosabb mint például a második verziós MobileNetV2, a MnasNet, a ShuffleNetV2 stb.



2.1.2.1. ábra MobileNetV3 összehasonlítása[9]