Óbudai Egyetem
Neumann János Informatikai Kar
Szoftvertervezés és -fejlesztés Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Szücs Klaudia**
Törzskönyvi száma: T/006399/FI12904/N
Neptun kódja: GEL5FF

A dolgozat címe:

Pajzsmirigy csomók detektálása ultrahangos képek alapján mély konvolúciós hálózatokkal
Thyroid nodule detection based on ultrasound images with deep convolutional networks

Intézményi konzulens: Dr. Kertész Gábor
Külső konzulens:

Beadási határidő: 2023. május 15.

A záróvizsga tárgyai: Számítógép architektúrák
Szoftvertervezés és –fejlesztés specializáció

A feladat

A lakosság számottevő többsége szenved valamilyen pajzsmirigy rendellenesség által okozott betegségtől. Ezen a szervünkön lévő apró elváltozások mielőbbi észlelése is akár kulcsfontosságú lehet a megfelelő és precíz diagnosztika szempontjából.
A dolgozat célja egy olyan orvosi döntést támogató megoldás készítése, amely ultrahangos felvétel alapján képes az elváltozást detektálni. A megoldás alapja egy mély konvolúciós neurális hálózat, amely a bemenetként érkező ultrahangos kép alapján megadja, hogy a beteg milyen malignitású kategóriába tartozik a TI-RADS osztályozás szerint. A dolgozat részeként készített webalkalmazás segítségével a kezelőorvos páciensprofilokat tud létrehozni, illetve különböző statisztikai kimutatások is elérhetőek a betegségek gyakoriságáról korosztályok, vagy nemek szerinti bontásban.

A dolgozatnak tartalmaznia kell:

- A kapcsolódó szakirodalom részletes áttekintését, a kapcsolódó módszerek (úgy mint R-CNN, YOLO) részletes bemutatását;
- Hasonló problémák, hasonló módszerek bemutatását;
- Az elérhető adatok bemutatását, analízisét;
- A megoldás részletes tervét, a módszertan leírását;
- A megoldás implementációjának bemutatását, az eredmények részletes kiértékelését;
- A további kutatási-fejlesztési lehetőségek bemutatását.

Ph.

.....…….
Dr. Vámossy Zoltán
intézetigazgató

A szakdolgozat elévülésének határideje: **2025. május 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

……..
külső konzulens

..…….
intézményi konzulens

.

Tartalomjegyzék

1 A megoldandó probléma bemutatása 3
1.1 Kutatás célja . . . 3

2 Pajzsmirigy és betegségei 3
2.1 TI-RADS rendszer . . . 4

3 Irodalomkutatás 5
3.1 Neurális hálózatok áttekintése . . . 5
3.1.1 Gépi tanulás és mélytanuló algoritmusok bevezető . . . 5
3.1.2 Konvolúciós neurális hálózatok . . . 6
3.1.3 Architektúrák a probléma szempontjából . . . 9
3.2 Architektúrák áttekintése . . . 11
3.2.1 R-CNN . . . 11
3.2.2 Fast R-CNN . . . 12
3.2.3 Faster R-CNN . . . 13
3.2.4 ResNet . . . 14
3.2.5 YOLOv3 . . . 15

4 Terv megvalósításához felhasználható technológiák 18
4.1 Python . . . 18
4.2 Gép tanulási keretrendszerek . . . 18
4.2.1 Tensorflow . . . 18
4.2.2 Pytorch . . . 19
4.3 Google Colaboratory Pro (Colab) . . . 19
4.4 Webes keretrendszerek . . . 19
4.4.1 ASP.NET Core . . . 19
4.4.2 Spring Boot . . . 20
4.4.3 Frontend keretrendszerek . . . 21

5 Irodalom kutatás összegzése 22
5.1 Felhasználandó technológiák . . . 22

6 Adathalmaz bemutatása 23
6.1 Adathalmaz előfeldolgozása . . . 25

7 Rendszerterv 28
7.1 Követelmény specifikáció . . . 28
7.2 Architektúra áttekintése . . . 28

Ábrajegyzék **30**

Táblázatjegyzék **30**

Irodalomjegyzék **31**

1 A megoldandó probléma bemutatása

A lakosság számottevő többsége szenved valamilyen endokrin betegségtől, amelynek legtöbb esetben okozója valamilyen pajzsmirigy rendelleneség (lásd 1.1 fejezet). Azonban ezeknek a betegségeknek a diagnosztizálása igen nagy kihívást jelentő feladat. Általában az orvos szakképzettségén és tapasztalatán múlik. Azonban mihamarabbi pontos diagnosztika felállítása meghatározó lehet a páciens életének szempontjából.

A pajzsmirigyrák kutatása az orvosi közösség és a társadalom széleskörű aggodalmát kiváltó témává vált napjainkban. A pajzsmirigyrákok többsége pajzsmirigydaganat formájában jelenik meg, amelyet gyakran véletlenszerűen fedeznek fel a nyak képalkotó diagnosztika vizsgálata során. Ezek többsége nagy valószínűséggel ártalmatlan elváltozás, azonban a nem várt észlelések ellenére fontos, hogy minél több rosszindulatú nodolust időben azonosítsanak. Továbbá kizárják azokat, amelyek nagy valószínűséggel jó indulatúak.

A vizsgálat által készült ultrahangos képek információt tartalmaznak a pajzsmirigy csomók jellemzőiről, mint például összetétel, echogenitás, peremek, vagy akár a méret. Ezen tulajdonságok alapján a radiológusok egy szabványosított pontozási rendszer (lásd 1.2 fejezet) segítségével megállapíthatják a pajzsmirigycsomók rosszindulatúságának kockázatát. Azaz a kapott pontok alapján kategorizálható, hogy az adott csomó rosszindulatú vagy jóindulatú elváltozás.

Ahogy már említettem szabadszemmel csak kellően nagy tapasztalattal rendelkező szakorvos tudja megállapítani a betegség súlyosságát. Azonban a rohamosan fejlődő informatikai területen megjelenő mélytanulási technológiák számos új lehetőséget nyitottak az orvostudomány területén is. Számos objektumdetektálásra alkalmazható architektúra létezik, amiket megfelelően finoman hangolva nagyban segíthetik a radiológusok munkásságát is.

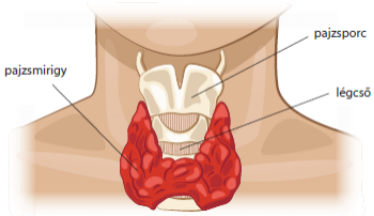
1.1 Kutatás célja

A fentebb említett betegség növekvő esetszáma miatt, dolgozatom fő célja egy olyan alkalmazás implementálása, ami képes felgyorsítani a klinikai diagnosztika létrehozást mélytanulási algoritmusok segítségével.

Emellett felhasználó számára különböző statisztikai adatok megjelenítésére, illetve szűrésére is szolgálni, a már felvett esetekből. Ezen vizualizációs funkció is hozzájárulna a precízebb lelet készítéséhez.

2 Pajzsmirigy és betegségei

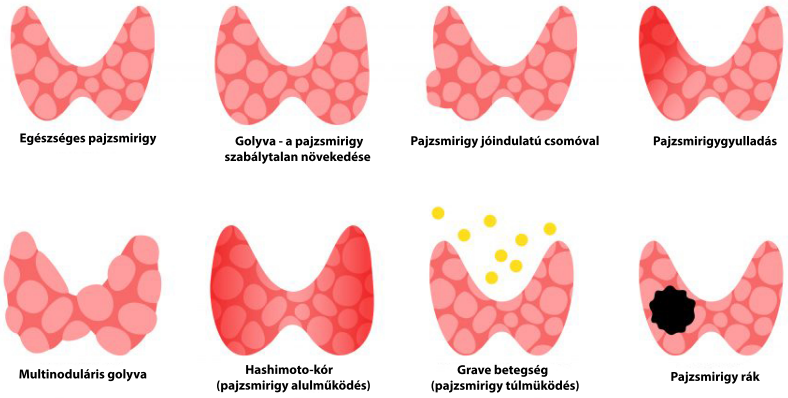
Legfontosabb belső elválasztású mirigyünk. A pajzsmirigy egy kis, pillangó alakú mirigy, amely a nyak alsó részén, a bőr és az izomrétegek alatt található.



Ábra 2.1: Pajzsmirigy elhelyezkedése

A pajzsmirigy [1] két aktív pajzsmirigyhormont termel, a levotiroxint (T4) és a trijódtironin (T3), amiknek fontos szerepe van egészséges metabolizmus fenntartásában. Ezen hormonok hatással vannak a fehérje termelődésre, egészséges testhőmérséklet fenntartására, energiatermelésre, illetve szívverés növelésével, légzés növelésével fokozzák a mitokondriumok aktivitását. Ennek eredményeképpen a pajzsmirigy működése a szervezet minden lényeges szervére hatással van.

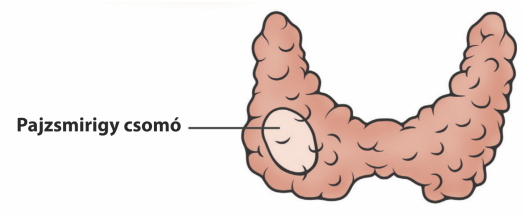
A pajzsmirigy számos különböző problémára hajlamos, amelyek közül néhány rendkívül gyakori betegségeknek mondhatóak az emberiség körében. A túl kevés pajzsmirigyhormon termelése pajzsmirigy-alulműködést, a túl sok pajzsmirigyhormon termelése pedig pajzsmirigy túlműködést okoz. Az előbbi esetben olyan állapotról van szó, amikor a pajzsmirigy alulműködik, és nem képes elegendő pajzsmirigyhormont termelni. Ezzel szemben az utóbbi esetben a pajzsmirigy túlműködik, és túl sok pajzsmirigyhormont termel.Túl működés esetén kialakulhat úgynevezett Basedow Graves-kór, ami egy autoimmun betegség, illetve alul működés esetén jellemző betegség a Hashimoto-kór.



Ábra 2.2: Pajzsmirigy betegségek

Betegségek nehezen észrevehetőek, az orvosok megannyi tényezőt, többek között klinikai értékelést, vérvizsgálatokat, képalkotó vizsgálatokat, biopsziát,és egyéb vizsgálatokat vonhatnak be a pajzsmirigybetegség diagnosztizálásába.

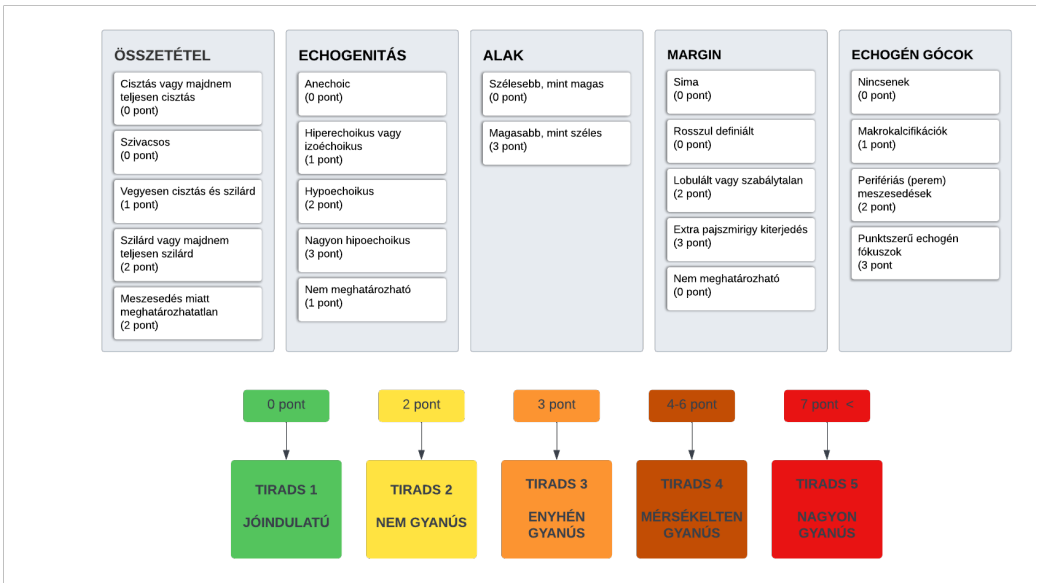
Én témám szempontjából a képalkotó vizsgálatokkal észlelhető betegségek lesznek a mérvadóak, mint például az ultrahangos képek alapján diagnosztizálható pajzsmirigy csomók. A pajzsmirigy csomók a pajzsmirigy belsejében kialakuló szilárd vagy folyadékkal teli daganatos csomók, amelyek lehetnek rosszindulatúak vagy jóindulatúak.



Ábra 2.3: Pajzsmirigy csomó elhelyezkedése

2.1 TI-RADS rendszer

Egy Amerikai Radiológiai Társaság egységes kifejezést javasolt a pajzsmirigy csomók malignitásának a meghatározására. Miszerint az adott nodulust pontszámmal értékelik a heterogenitása, echognitása, alakja, széle és echogén gócok alapján, hogy megítélhessék indokolt-e biopszia, vagy elegendő az ultrahangos követés a kezelés szempontjából. Egy göböt 5 malignitási csoportba sorolhatunk:



Ábra 2.4: TIRADS pont kalkulátor [2]

3 Irodalomkutatás

3.1 Neurális hálózatok áttekintése

3.1.1 Gépi tanulás és mélytanuló algoritmusok bevezető

A gépi tanulás az elmúlt évszázadban drámai fejlődésen ment keresztül, ami az élet minden területére kihatással van. Határozottan egyre inkább meghatározó szerepet töltenek be hétköznapi életünkben. A modern társadalom gondolkodás módját az élet számos területén képesek befolyásolni, a különböző közösségi médián megjelenő tartalmak szűrésével, illetve a személyekre szabott célzott marketing hirdetésekkel.

A mesterséges neurális hálózatok lehetővé teszik a számítógép számára, hogy összetett, bonyolult adathalmazokban ismétlődések fedezzenek fel. Ezeket a tanuló rendszerek főként képeken való objektum detektálásra, beszéd szöveggé alakítására, híreket, posztokat vagy termékeket a felhasználó érdeklődési körére szabására használják. Azonban a kezdetekben, amikor még nem voltak megfelelő erőforrások, a hagyományos gépi tanulási technikák csak korlátozott számban voltak képesek természetes adatok nyers formában történő feldolgozására. Egy darab mintafelismerő rendszer megépítéséhez mérnöki munkára és kellően nagy szakismeretre volt szükség. Azonban az elmúlt évek során rengeteg kutatás folyt a gépi tanulási technikák tervezéséről, és a tanulmány konklúziója, hogy jobb minta felismerő rendszereket lehet tervezni, ha jobban támaszkodunk az automatikus tanulásra és kevésbé a kézzel tervezett heurisztikákra. Ezt a gépi tanulás modellek időarányos fejlődése, illetve a mélytanuláshoz szükséges számítógépes infrastruktúra (hardveres és szoftveres) rohamos mértékű föllendűlése tette lehetővé.

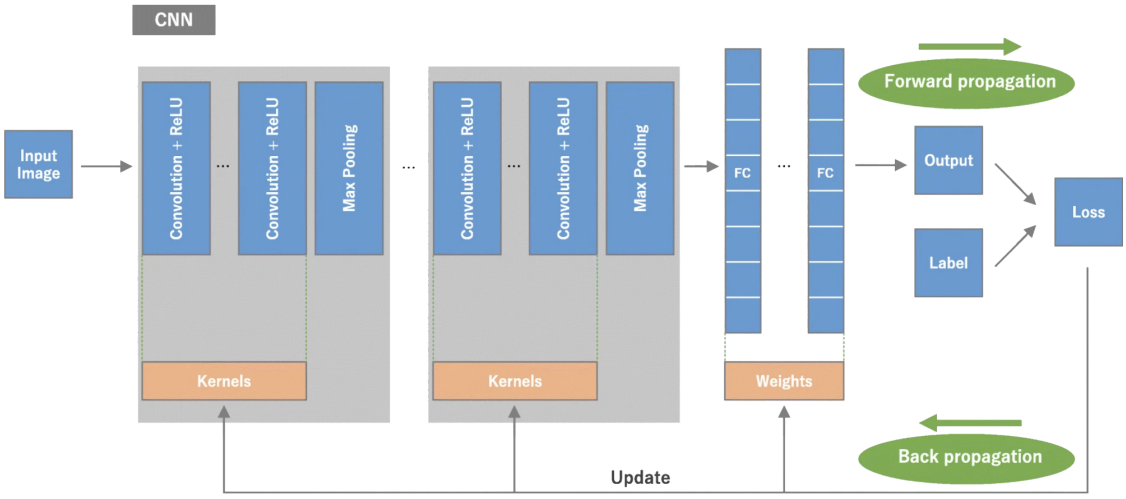
Mélytanuló algoritmusokat [3] mesterséges neurális hálókból fejlesztették ki, és mostanra legelterjedtebbek a gépi tanulás területén. Mesterséges neurális hálókkal kapcsolatos kutatások már az 1940-es években elkezdődtek. A mélytanulási modellek általában hierarchikus struktúrát alkalmaznak a rétegek összekapcsolására. Az alsó rétegek kimenete lehet egy magasabb réteg bemenete, lineáris vagy nemlineáris számítások segítségével átalakított verziója. Tehát a számítógépek a hasznos reprezentációkat és jellemzőket automatikusan, közvetlenül a nyers adatokból tanulják meg, megkerülve a kézi feldolgozású nehézkes lépést. Ezek a modellek képesek már az alacsony színtű jellemzőket magasabb szintű absztrakt jellemzőkké átalakítani. Ennek a tulajdonságnak köszönhetően mélytanulási modellek erősebbek lettek, mint a gép tanulási modellek a jellemzők reprezentálásában. Az orvosi képalkotásban a mélytanulást iránti érdeklődést leginkább a konvolúciós neurális hálózatok keltették fel. Ezekkel az algoritmusokkal lehetséges képek és más strukturált adatok hasznos reprezentációit megtanulni. Amint lehetővé vált a közvetlenül adatokból való jellemzők kinyerése, és tanításra való felhasználása, más technikákat a háttérbe szorítottak.

3.1.2 Konvolúciós neurális hálózatok

A konvolúciós hálózatok [4], vagy másnéven konvolúciós neurális hálózatok az adatok feldolgozására szolgáló neurális hálózatok egy speciális fajtája, amely ismert rácsszerű topológiával rendelkezik. Ezeket a hálózatokat úgy tervezték, hogy automatikusan és adaptívan tanulják meg a jellemzők térbeli hierarchiáit, az alacsony szintű mintáktól a magas szintű mintákig.

A CNN egy matematikai konstrukció ami jellemzően jellemzően háromféle rétegből (építő elemből) áll: konvolúciós, pooling és teljesen összekapcsolt rétegekből. Az első kettő, a konvolúciós rétegek és a pooling rétegek a jellemzők kinyerését végzik, míg a harmadik réteg a teljesen összekapcsolt réteg a kinyert jellemzőket a végső kimenetre, például a csoportosításra képezi le.

A konvolúciós rétegek kulcsszerepet játszanak az e féle hálózatok kialakításában, mivel ezek a rétegek matematikai folyamatok halmazából áll, a lineáris műveletek egy speciális típusából. A digitális képek pixelértékeit egy kétdimenziós rácsban, azaz számtömbökben tárolódnak, és egy kernelnek nevezett paraméterrácsot alkalmaznak minden egyes képi pozicióban, ami rendkívül hatékonnyá ezt a technikát, mivel egy jellemző bárhol előfordulhat a képen. Ahogy az egyik réteg a másikba táplálja a csoportosított jellemzőket, hierarchikusan és fokozatosan egyre komplexebbé vállnak. A paraméterek optimalizálási folyamatát nevezzük trainelésnek, amikor ugyanis minimalizáljuk a kimeneteket és az alapigazság-címkék közötti különbséget, többek között a backpropagation és gradiens süllyedés nevű optimalizációs algoritmus segítségével.

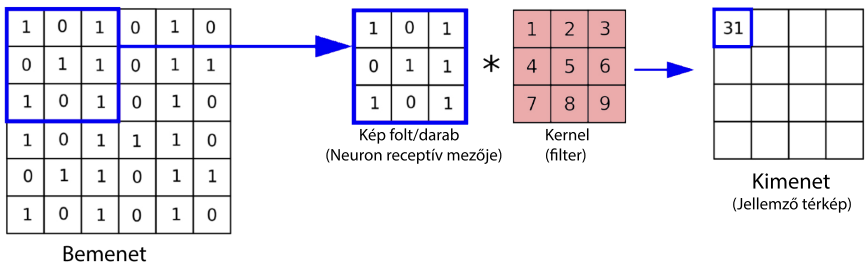


Ábra 3.1: Architektúra áttekintése
[5]

Konvolúciós neurális hálózati arhitektúrák felépítése

Felépítésük eltér a hagyományos neurális hálózatokétól. A konvolúciós neurális hálók az adatokat térbeli adatokként kezelik. Ahelyett, hogy a neuronok az előző rétegekben lévő neuronokhoz kapcsolódnának, mint a hagyományos neurális hálókban, a konvolúciós neurális hálón belüli neuronok csak a hozzájuk közeli neuronokhoz kapcsolódnak, és mindegyiknek ugyanaz a súlya. Ezen hálózatókat leginkább a konvolúciós rétegnek nevezett speciális és a pooling réteg különbözteti meg a többi neurális háló felépítésétől. Valójában ezek a konvolúciós rétegek a legfontosabb építő elemei a hálózatnak.

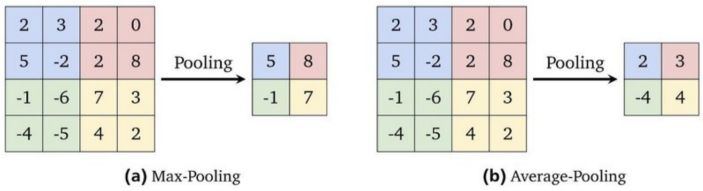
A konvolúciós réteg az első réteg, amely a különböző jellemzők kinyerésére szolgál a bemeneti képekből. Ebben a rétegben egy szűrőt vagy Kernel-módszert használunk a bemeneti kép jellemzőinek kinyerésére. Ez egy olyan folyamat, amiben vesszük a kernelnek vagy szűrőnek nevezett számmátrixunkat, majd átvezetjük a képünkön és a szűrő értékeinek megfelelően átalakítjuk azt. Ez a réteg megőrzi a pixelek közötti korrelációt a képjellemzők megtanulásával, a bemeneti adatok kis négyzeteinek felhasználásával. Lényegében egy olyan matematikai művelet, amelynek konklúziója a képmátrix és a filter segítségével egy jellemzőtérkép.



Ábra 3.2: Jellemzők kinyerése

A fentebb említett másik fontos alkotó elem a pooling rétegek. Ezen rétegnek az elsődleges célja a számítási költségek csökkentése. Ennek érdekében képes csökkenteni a jellemzőtérkép mintájának méretet a rétegek közti kapcsolatok redukálásával, így gyorsabbá téve ezeknek a feldolgozását. Azaz jellemzők általánosítására szolgáló technika, amely segít a hálózatnak a képen való elhelyezkedésüktől független jellemzők felismerésében. A pooling réteg kimenete egy összevont jellemzőtérkép lesz. Az alkalmazott módszertől függően többféle Pooling művelet létezik, max pooling, illetve average pooling. A pooling műveletek legnépszerűbb formája a max pooling, amely a bemeneti jellemzőtérképből foltok kivonásával megkeresi a köztük lévő maximális értéket, és az összes többit elveti, úgyhogy dimenziók mélységet nem változtatja. Ezzel szemben az average pooling,

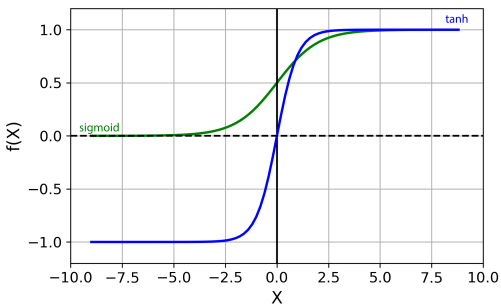
amikor a szűrő áthaladva a bemeneten kiszámítja ezen értékek átlagát, majd egy kimeneti tömbben összesíti ezeket.



Ábra 3.3: Pooling módszerek

Ezen módszerek nagy mértékben csökkentik a komplexitást, javítják a hatékonyságot, illetve korlátozzák a túlillesztés kockázatát. A konvolúciós neurális architektúra utolsó néhány rétegét a teljesen összekapcsolt rétegek alkotják. Közvetlenül a kimenet előtt helyezkednek el. Az előző rétegekben a bemeneti képek pixelértékei nem kapcsolódnak közvetlenül a kimeneti réteghez. A teljesen összekapcsolt rétegben azonban a kimeneti réteg minden egyes csomópontja közvetlenül kapcsolódik az előző réteg csomópontjaihoz. Ez a réteg a klasszifikációért felel, az előző rétegek és azok különböző szűrői által kinyert jellemzők alapján. Míg a konvolúciós és a pooling rétegek általában ReLu aktivációs függvényeket használnak, a teljesen összekapcsolt rétegek általában egy softmax aktiválási függvényt használnak a bemenetek megfelelő osztályozásához, amely 0 és 1 közötti valószínűséget eredményez.

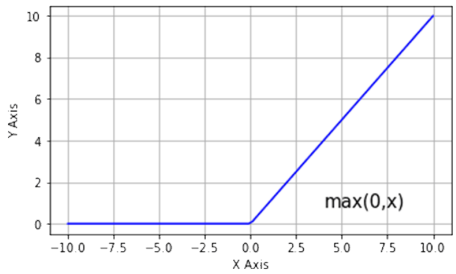
Az aktiválási függvény dönti el, hogy az adott neuron tartalmazz-e hasznos információt az előrejelzés szempontjából vagy nem. Számos általánosan használt aktiválási függvény létezik, például a ReLU, a Softmax, a tanH és a Szigmoid függvény. Mindegyik függvénynek sajátos felhasználása van. A szigmoid függvényt bináris klasszifikáció esetén szokták használni. Hiperbolikus tangens (tanH) hasonlít a szigmoid függvényre, azonban más érték tartománnyal rendelkezik.



Ábra 3.4: Szigmoid és tanH függvény összehasonlítása

A softmax függvényt gyakran használják a neurális hálózat utolsó aktiválási

függvényeként, hogy normalizálja a hálózat kimenetét a megjósolt kimeneti osztályok valószínűségi eloszlásának tekintetében. A ReLU függvény használatának előnye, hogy nem aktiválja az összes neuront egyidőben.



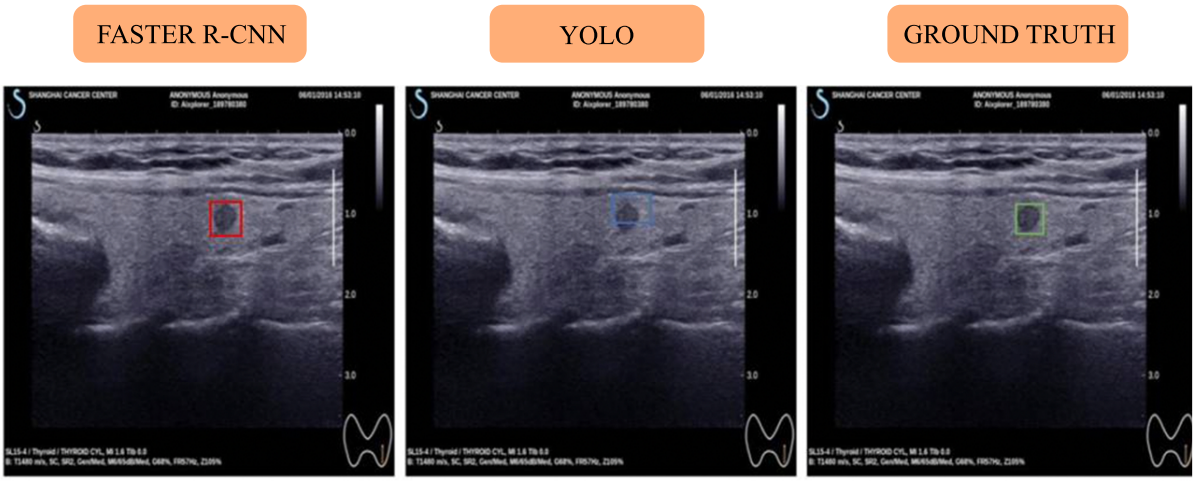
Ábra 3.5: ReLU függvény ábrázolása

3.1.3 Architektúrák a probléma szempontjából

A modernebb mély tanulás kezdetleges időszakában elkezdtek architektúrákat ötvözni. Majd a későbbi hálózati architektúrák már folyamatosan kombináltabbá váltak, hiszen korábbi architektúrák felismeréseire és ötleteire épültek. Néhány mély tanulási módszert alkalmaztak a pajzsmirigy csomók ultrahangos képein, a finom hangolt GoogleNet modell segítségével, azonban a csomók automatikus detektálása nem sikerült. Egy másik tanulmány a tovább fejlesztett Faster R-CNN modellt ajánlotta [6], amivel már sikerült elérni a nodulosok automatikus felismerését. A javasolt mélytanulási módszer más módszerekkel összevetve a legjobb igazság értékét érte el, azaz a legpontosabban képes meghatározni a csomó helyét, illetve osztályát. Azonban egy másik tanulmányban, a YOLOv3 modellel is hasonló sikereket értek el. [7]

Mindkét model a legkorszerűbb objektumdetektálási módszer. Azonban további források és a két architektúra összehasonlítása alapján kijelenthető, a megfelelően finomhangolt Faster R-CNN sokkal magasabb pontosságot képes elérni a YOLO-val szemben. [8]

A következő ábra szemlélteti a két vizsgált model tanítás utáni detektálás, alapigazsághoz való pontosságát.



Ábra 3.6: Két módszer összehasonlításának eredménye [8]

A pajzsmirigy csomók lokalizációs eredményeinek összehasonlítása a különböző módszerekkel a mAP-érték alapján.

Method	Faster R-CNN	YOLO
IoU threshold	mAP	mAP
0.50	0.68 ± 0.10	0.63 ± 0.07
0.75	0.53 ± 0.15	0.30 ± 0.11
0.90	0.36 ± 0.19	0.11 ± 0.19

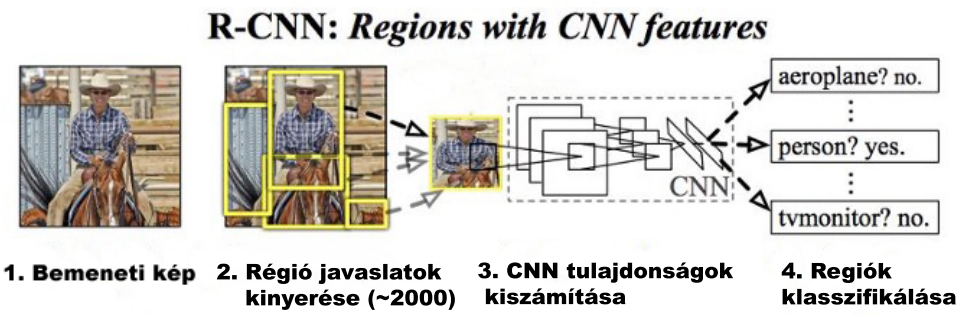
Ábra 3.7: A mAP-értékek összehasonlítása [8]

3.2 Architektúrák áttekintése

3.2.1 R-CNN

A gépi látás terjedésével, egyre nagyobb hangsúly helyeződött az objektumdetektálás témakörére. Az első igazi áttörés a területen 2014-ben következett be, az RCNN [9] publikálásával, melynek sokkal magasabb performancia szintje volt, az akkori más elterjedt objektumdetektáló metodikákkal szemben. Az addigi próbálkozások nagy komplexitású- és számításigényűek voltak, ami a detektáláshoz szükséges régiók kiszámításának költséges műveletének volt köszönhető. Bemeneti képeken külön elhelyezkedéssel és mérettel találhatóak objektumok, ebből következően a régiók meghatározásának folyamata különböző méretű, és nagyon nagy mennyiségű régió detektálását és kiértékelését igényli.

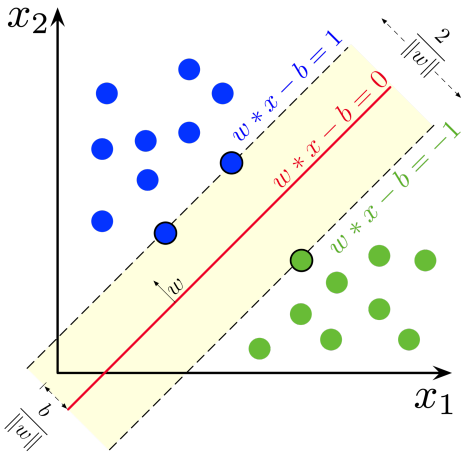
Erre a problémára ad megoldás a RCNN (Regions with CNN).Ezen architektúra három lépést tartalmazz ami a lentebb mellékelt ábrán látható.



Ábra 3.8: RCNN működése
[9]

Az RCNN pipeline első lépése a ′régiójavaslatok′ létrehozása, azaz olyan régiók létréhozása, amelyek egy objektumhoz tartoznak. Egy Szelektív Kereső algoritmus a fő működési motorja, ami úgy működik, hogy a kép olyan alszegmentációt hozza létre, amelyek egy objektumhoz tartoznak – szín, textúra, méret és alak alapján –, és a hasonló régiókat iteratív módon kombinálja objektumok kialakításához. Ezen kereső algoritmus segítségével ∼2000 darab független régiójavaslatot generál, amiken később egy konvolúciós neurális hálózat segítségével minden egyes régiójavaslathoz 4096 dimenziós jellemzővektort kapunk. Jellemzők kinyerése után, azoknak feldolgozása fog történni egy egyedi lineáris SVM (Support Vector Machine) nevű klasszifikációs módszerrel. Ez e a módszer a régiók közötti legjobb határvonal megtalálására ad megoldást, tehát probléma szempontjából besorolja egy adott osztályba, vagy jelzi annak a hiányát.

A lokalizációs teljesítmény javítása érdekében, klasszifikációs fázis után egy



Ábra 3.9: SVM mükődése

bounding-box regressziós nevű lépés kerül végrehajtásra, ami a megtalálandó objektumok köré úgynevezett határoló dobozokat helyez.

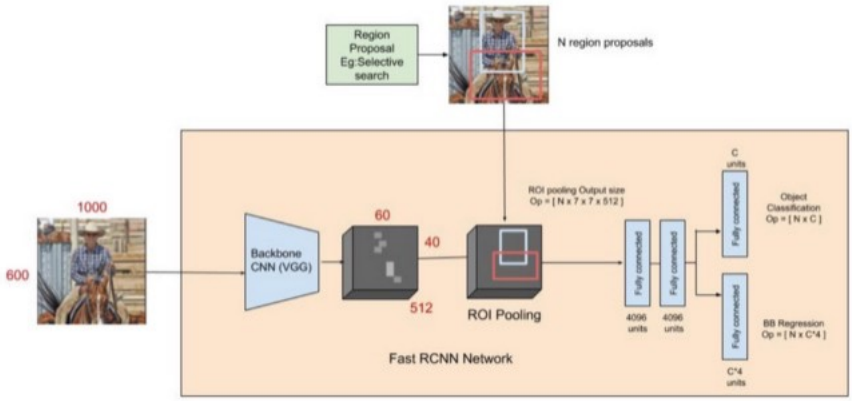
Azonban ennek a modelnek számos hátránya van.

- Ez egy többlépcsős modell, ahol minden egyes lépés egy független komponens.
- Az előre tanított CNN-ből kinyert jellemzőket a lemezen tárolja, hogy később betaníthassa az SVM-eket. Ehhez azonban nagy mennyiségű szabad tárhelyre van szükség.
- A Szelektív Kereső algoritmustól függ a régiójavaslatok létrehozása.
- Minden egyes régiójavasalatot függetlenül adunk át a neurális hálózatnak a jellemzők kinyeréséhez, ezért az R-CNN valós idejű futtatása lehetetlen.

3.2.2 Fast R-CNN

A technológia fejlődésével és számítási teljesítmény növekedésével, amelyet a 2014 óta, az R-CNN tanulmány megjelenése óta tapasztalunk, most már könnyű észrevenni az R-CNN buktatóit, amiket a fentebb került kifejtésre. Ezekre a problémákra add megoldást már a Fast R-CNN, célja, hogy egy pontosabb, gyorsabb kevésbé erőforrásigényesebb megoldást nyújtson. Míg az R-CNN alapötletében minden egyes képhez körülbelül 2000 regiójavaslatot generál, és minden egyes régiójavaslatatot betáplál a mögöttes hálózati architektúrába, ez matematikailag könnyen érthető miért is okoz próblémát, ezért a Fast R-CNN alapja, hogy a képeket csak egyszer adjuk hozzá a hálózathoz, majd a Szelektív Keresés által generált régiójavaslatokat a CNN segítségével készített Feature Map-ekre vetítjük. Ezt a folyamatot ROI (Region Of Interest) Projectionnek nevezzük. Ezután a fontos területek feltérkepézésére az SVM klasszifikálás lecserélve egy Softmax nevezetű függvényt vezettek be. Ebben az aktivációs rétegben történik az objektumok

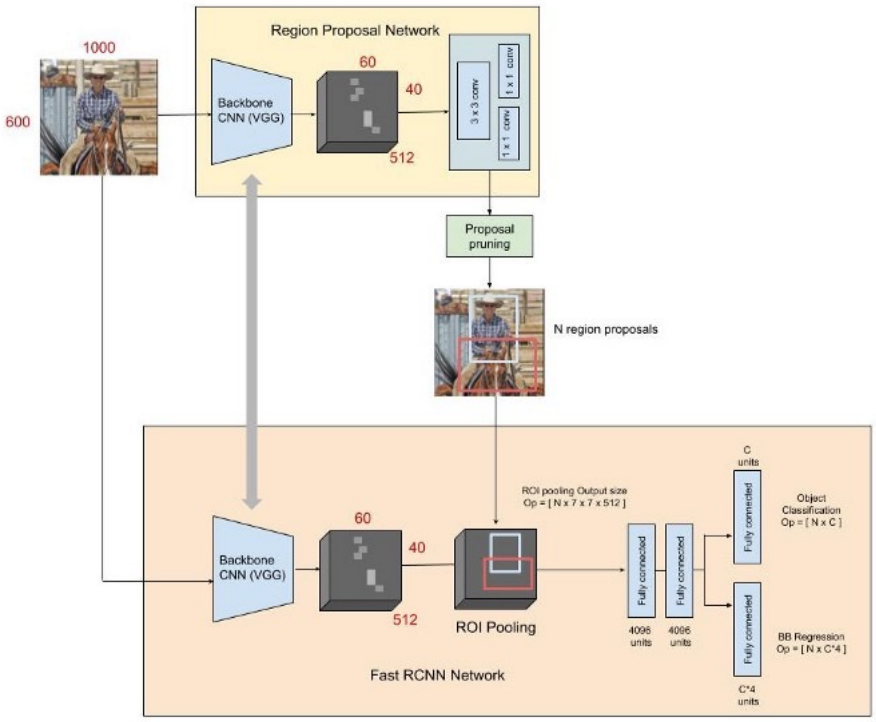
osztályainak meghatározása. A Softmax függvény célja, hogy a CNN kimenetét normalizálja, azaz nulla és egy közé sorolja be. Végül ezek a normalizált értékek megjelennek a határoló dobozokkal együtt.



Ábra 3.10: Fast RCNN működése
[9]

3.2.3 Faster R-CNN

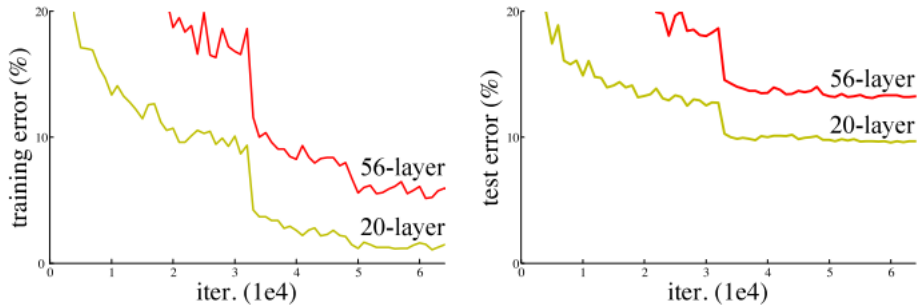
A Faster R-CNN [9] 2015-ben került publikálása, az előbbiekben ismertetett Fast R-CNN után. A Fast R-CNN már jelentős előre lépést mutatott a R-CNN architektúrák családjában, azonban a detektálás során használt szelektív keresési régiójavaslat generáló algoritmus jelentős teljesítményt igényelt. Ez az algoritmus ezért nem volt alkalmas közel valós idejű működés eléréséhez. Képenkénti $\sim$ 2000 régiójavaslatot, egy új réteg bevezetésével váltották ki, ezt nevezzük Region Proposal Networknak, amely már minimális veszteséggel dolgozott. Mivel a régiójavaslatokat egy előre tanított hálózattal generálja, most már az adott probléma paramétereire lehet szabni. Így jobb régiójavaslatokat állít elő, az olyan általános módszerekhez képest, mint a Szelektív Keresés.



Ábra 3.11: Faster RCNN működése
[9]

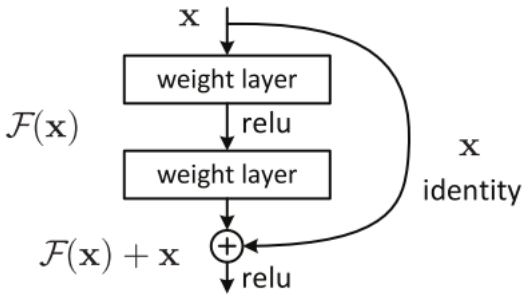
3.2.4 ResNet

A mélyebb neurális hálózatók betanítása kezdetekben nehéz volt az úgy nevezett eltűnő gradiens (vanishing gradient) probléma miatt. Ez annyit jelentett, ha egyre több réteg van a hálózatban, azaz egyre mélyebb a hálózat, akkor a derivált szorzatok értéke folyamatosan csökken. Az ismétlődő szorzás következményeképpen, a gradiens megközelíti a nullához közelei értéket, majd eltűnik. A hálózat egyre mélyebbé válásával a teljesítménye telítődik, és a tanítás folyamata leáll. Azért érdemes mélyebb, több rétegből álló hálózatok létrehozására törekedni, mert már önmagában ezzel jelentős javulásokat érhetünk el a tanítás során. A mélyebb hálózatok következménye a hosszabb tanítás szükségessége amit, ha jól konfigurálunk magával vonja a magasabb precizitást.



Ábra 3.12: A halózat mélysége és a performancia kapcsolata [10]

Ennek a problémának orvosolására úgynevezett reziduális blokkokat vezettek be. Ezekben a blokkokban a kihagyásos kapcsolatok technikának az alkalmazásával, a rétegek aktivációit úgy kapcsolja össze a további rétegekkel, hogy közben egy vagy több réteget kihagy.



Ábra 3.13: A reziduális blokk felépítése [10]

Előnye, hogy ha bármelyik réteg károsítaná az architektúra teljesítményét, a kötegnormalizálásnak (batch normalization) köszönhetően a kihagyja ezt.

E tulajdonsága miatt a ResNet a fentebb említett probléma nélküli mélytanulási hálózat megvalósítására képes.

3.2.5 YOLOv3

Az összes fentebb említett objektumdetektáló algoritmus régiók segítségével lokalizálja az objektumot a képen belül. Tehát a hálózat nem a teljes képet vizsgálja. Ehelyett a kép azon részeit vizsgálják, amelyek nagy valószínűséggel tartalmazni fogják a keresett tárgyat.

A YOLO vagy You Only Look Once egy olyan objektumfelismerő algoritmus, amely nagyban különbözik a fentebb látott régióalapú algoritmusoktól. Ezen modellek egy végponttól végpontig (end-to-end) tartó mélytanulási modellek. Félépítésükben több konvolúciós műveleteteket és pooling rétegeket tartalmaznak. Korábbi módszerek, mint például a R-CNN modellek is, több ezer hálózati

kiértékelést igényelnek ahhoz, hogy egy képre való előrejelzést készítsenek, aminek optimalizálása is jóval bonyolultabb. Míg ezek a módszerek többlépcsős folyamatok voltak, a YOLO egyesíti a különböző folyamatokat, és egyetlen neurális hálózatot használ. Ezzel futási időben teljesítmény növelést érhetünk el, azonban a YOLO nem képes felismerni a kis tárgyak csoportját vagy a szabálytalan alakú tárgyakat egy képen belül.

Egy hatékonyabb implementációja a YOLOv3 [11], amely már javított pontosságot is mutat. A modell a DarkNet-et veszi alapul, ami eredetileg 53 réteggel rendelkező neurális hálózatok képzésére szolgáló keretrendszer. A detektálás elvégzéséhez újabb 53 réteget helyeztek rá, így összesen 106 rétegű teljesen konvolúciós architektúra jött létre.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Ábra 3.14: Darknet-53 [12]

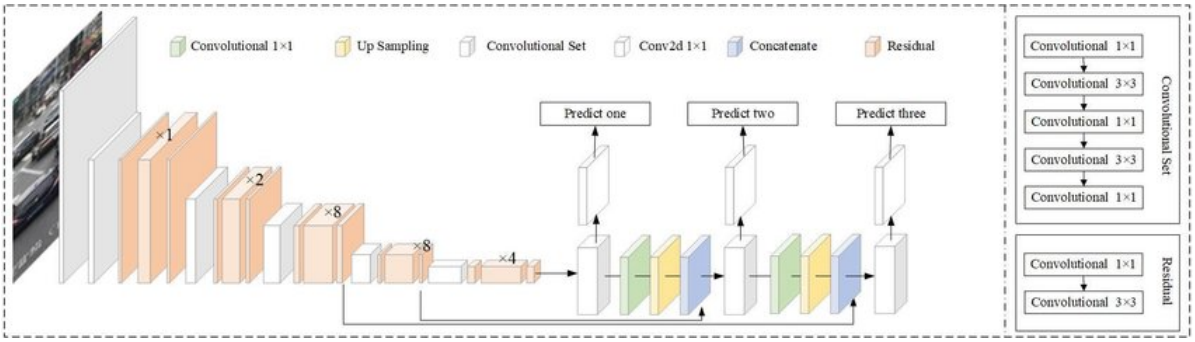
Előző verzióhoz hasonlítva azonban, a sebessége lecsökkent, de sokkal magasabb precízotással képes apróbb objektumok detektálására. Maga a detektálás egy 1x1 -es detektáló kernel segítségével történik, három különböző méretű jellemző térképen, a hálózat három különböző helyen. A detektáló kernel alapja:

$$\mathbf{1 \ x \ 1 \ x \ (B + (5 + C))},$$

ahol B, az előre megbecsülhető határoló dobozok száma. Az 5-ös szám a négy határoló doboz attribútumaira, illetve az egy objektum bizalomra, vagy másnéven bizalmi pontszámra utal (mennyire valószínű, hogy a tartalmazz objektumot, vagy nem). C pedig az osztályok száma.

Az algoritmus három skálán készít előrejelzéseket, amelyeket az adott input kép dimenzióinak 32,16, és 8 mintavételezése add meg. A YOLO v3 összesen 9 horgonyzó dobozt (anchor box) használ. Minden skálázáshoz hármat. K-Means al-

goritmus segítségével megfelelőe klaszterezéssel létrehozható a 9 horgonykocka. Az első skálához tartozik a három legnagyobb, a másodikhoz a három közepes, és az utolsóhoz a három legkisebb keret. Azaz dimenziók szerint csökkenő sorrendbe vannak rendezve. Így minden réteg a nagy, közepes vagy kis objektumok felismerésében jeleskedik.



Ábra 3.15: YOLOv3 architektúra részletei [12]

4 Terv megvalósításához felhasználható technológiák

4.1 Python

A Python [13] egy általános célú, nagyon magas szintű programozási nyelv. A nyelv tervezésekor az olvashatóságat és a programozói munka megkönnyítését helyezi előtérbe a futási sebességgel szemben. A Python többek között a funkcionális, az objektumorientált, az imperatív és a procedurális programozási paradigmákat támogatja. Úgynevezett interpretált nyelvek családjába tartozik, ami azt jelenti, hogy nincs különválasztva a forrás- és a tárgykód, azaz a megírt program nem futás előtt kerül gép kóddá való lefordításra, hanem futási időben értékelődik ki. Az előre fordított nyelveknél számottevően jobb teljesítménybeli különbég figyelhető meg, mivel a fejesztő maga tudja menedzselni a hardveres erőforrásokat. Gondolok itt memória használatra, esetleg, hogy a CPU-t mennyire terhelje le. Azonban ez egyben a hátrányául is szolgál, mivel a fordítás extra időt emészt fel, ezzel szemben ez a probléma az interpretált nyelveknél nem jelentkezik. Azonban az előre fordított nyelveken írt kódok szintaxisának validálása, a fordítás pillanatában megtörténik, míg ez a másik nyelv családnál nem figyelhető meg. Az interpretált nyelvek teljesítménybeli lemaradását az eltelt évek alatt, nagyban redukálták. Jelenleg a 3-as verziónál tart a legelőször 1991 publikált projekt. Interpretált nyelvek közül leginkább széleskörben elterjedt, ami főként az igen kiterjedt és széles körű standard könyvtárjainak köszönhet. Ezek még kiegészülnek mások által publikált nyilvános könyvtárakkal. A nyelv főbb használatos funkciók a különböző modulokban vannak. Ezeknek a moduloknak nagy részét C-ben írták meg, és beépítettek az interpreterbe, másokat Python forráskódba kell importálni. Széleskörű könyvtár készletének és keretrendszer támogatottságának köszönhetően, ez a nyelv bizonyult eddig a legmegfelelőbbnek gépi tanulás terén. Ebből következően az objektum detektáló részt ezzel fogom elkészíteni.

4.2 Gép tanulási keretrendszerek

4.2.1 Tensorflow

A Tensorflow [14] a Google által kifejlesztett nyílt forráskódú gépi tanulási keretrendszer, ami 2015-ben került publikálásra. Eredetileg a Google Brain mérnöki és kutatói csapata fejlesztette ki a gépi tanulás és mély tanulási hálózatok kutatására. Jelenleg a leginkább alkalmazott keretrendszer ezen a területen. Széleskörű, flexibilis eszközeinek és könyvtárainak köszönhető a népszerűsége, amelyek nagyban megkönnyítik a gépi tanulási alkalmazások építését, illetve telepítését. Továbbá a jó dokumentáció mellet előre tanított modellkészlettel is rendelkezik, amely jó ki-

indulópontként szolgálhat a témával idegeneknek a különböző modellek fejlesztési folyamatának megértéséhez.

Ezek mellett a Tensorflow-ba integrálva elérhető a Keras API, ami egy magas szintű, mély tanulási API, amelyet a Google fejlesztett ki a neruális halózatok hatékonyabb, gyorsabb megvalósítására.

4.2.2 Pytorch

A Pytorch [15] egy viszonylag új, Torch-on alapuló mélytanulási keretrendszer. A Facebook AI-kutatócsoportja fejlesztette ki, és publikálta 2017-ben. Ezen keretrendszer főként az egyszerűségéről, könnyű használhatóságáról, rugalmasságáról, hatékony memória használatáról és dinamikus számítási gráfjairól híres. Emellett natív érzetet kelt, ami a kódolást könnyebben kezelhetővé teszi és növeli a feldolgozási sebességet.

4.3 Google Colaboratory Pro (Colab)

A Google Research terméke, ami lehetővé teszi, hogy bárki tetszőleges python kódot írjon és futtasson a böngészőn keresztül, és különösen alkalmas gépitanuláshoz, illetve adatelemzéshez. Technikailag a Colab egy olyan hosztolt Jupyter notebook szolgáltatás, amelynek használatához nincs szükség beállításra, ugyanakkor ingyenes hozzáférést biztosít a számítási erőforrásokhoz, beleértve a CPU-kat, GPU-kat és TPU-kat is. [16]

4.4 Webes keretrendszerek

A lentebb említett két keretrendszer népszerű a webes alkalmazások terén, attól függően ki melyik program nyelvet részesíti előnyben. Aki a C# program nyelvet preferálja nagy valószínűséggel az ASP.NET Core-t fogja használni, míg a Java nyelv esetén a Spring Boot a legnépszerűbb.

4.4.1 ASP.NET Core

Az ASP.NET Core [17] a .NET platform egy újabb verziója, ami egy nyílt forráskódú és felhőoptimalizált webes alkalmazások fejlesztésére szolgáló cross-platform keretrendszer. Támogatja az MVC (Model-View-Controller) tervezési mintát.

Azaz, olyan projekt sablonokat tartalmaz, amik segítségével könnyen létre hozhatunk egy webes alkalmazást a kezdetektől, hiszen az alap projekt struktúrát le generálja a fejlesztőnek, így lerövidítve fejlesztési környezet kialakítását.

Különböző előre definiált osztályokat is tartalmaz, továbbá attribútumokat is, amik segítségével könnyen kialakítható a Controller és a View réteg közötti kommunikáció. View réteg megvalósítására számos lehetőségünk van ASP.NET-ben. Egyik

legelterjedtebb a Razor Layout, amivel dinamikus weblap készíthető olyan módon, hogy HTML szintakszisba szerver oldali kódot is beágyazhatunk. Továbbá az adatkötés is könnyen megvalósítható, a View rétegből értekező adatok automatikus hozzárendelődnek a Controller rétegben található egyes metódusok paramétereihez.

Az ASP.NET Core könnyen integrálódik a később említett közkedvelt ügyféloldali keretrendszerekkel és könyvtárakkal is, mint például az Angular-ral, React-tal és a Vue.JS-el. Ezek mellé felsorolható a Blazor keretrendszer is, ami Javascript helyett C#-t használ a felhasználói felületek készítéséhez. A Blazor alkalmazások lényegében olyan újra felhasználható komponensekből állnak, amelyeket C#, HTML és CSS segítségével implementáltak.

Az adatelérést is zökkenőmentesen implementálható az Entity Framework ORM (Object Relational Mapping) keretrendszerrel. Az így keletkezett adatbázis kontextusban hozzáférünk az egyes mezőkhöz, amiken már LINQ-t (Language Integrated Query) is használhatjuk. A LINQ lényegében a C# programozási nyelvbe integrált API-készlét, amely segítségével gördülékenyen végrehajthatunk különböző adatforrásokból történő lekérdezéseket, mint pl. Where(), OrderBy().

A token alapú JWT autentikáció és autorizáció is egyszerűen implementálható egy NuGet csomag telepítésével.

4.4.2 Spring Boot

A Spring keretrendszer [18] maga egy szintén nyílt forráskódú, vállalati szintű keretrendszer, ami Java virtuális gépen (JVM) futó önálló alkalmazások létrehozására szolgál. A Spring Boot olyan eszközöket biztosít, amely a Spring keretrendszer segítségével lehetővé teszi, hogy minimáliás konfigurációval és beállíttassál piackész webes alkalmazásokat lehessen létrehozni.

A Spring Boot keretrendszer leginkább a Dependency Injection megvalósításának módjától lett közkedvelt a mikro-szolgáltatások fejlesztése körében. A különböző objektumoknak lehetséges függőségeket definiálni, amiket később a Spring konténer injektál hozzájuk, így könnyedén létrehozhatunk lazán összekapcsolt komponensekből álló moduláris alkalmazásokat. Ezeket a függőségeket különböző annotációkkal tudjuk megjelölni, mint például @Service, @Repository, @Controller. Így ezekkel az annotációkkal ugyanúgy kialakítható a MVC tervezési mintának megfelelő struktúra, mint az előbb említett ASP.NET Core-ban.

Maga a Controller rétegben elhelyezett osztályok felépítése igen hasonlóan történik, csak másképpen nevezett annotációkkal valósítsuk meg őket.

Spring Boot-ban a Hibernate nevezetű ORM (Object Relational Mapping) keretrendszert használhatjuk. Itt nem adatbázis kontextust hozunk létre, hanem különböző tároló interfészeket, amit a Spring Data JPA-ba beépített osztályok egyikével bővítünk ki. (Ilyen lehet például a JpaRespository) Majd az így keletkezett objektumokhoz alapvető lekérdezéseket definiál nekünk a JPA, mint például find-

All(). Azonban lehetőségünk is van egyéni lekérdezésekkel is bővíteni ezeket.

A Spring Boot-ban a biztonsági konfiguráció sokkal részletesebb, mint a fentebb említett ASP.Net Core keretrendszerben. Ez az úgynevezett Spring Security dependencia egy igen jól személyre szabható megoldást kínál.

A felhasználói felület megvalósítására itt is könnyedén integrálható a fentebb említett néhány keretrendszer, mint például a React, Angular vagy a Vue.JS.

4.4.3 Frontend keretrendszerek

Az alábbi táblázatban a jelenleg piacvezető keretrendszerek összehasonlítása látható fontosabb tulajdonságaik alapján

	Angular	**React**	**Vue**
	Typescript	Javascript	Javascript
Technológia típusa	MVC keretrendszer	Javascript könyvtár	MVVM keretrendszer
Elterjedtség	Népszerű a fejlesztők között hosszab ideje	Legnépszerűbb	legnagyobb csillag GitHub-on
Tanulási görbe	Legnehezebb	Könnyeb, mint az Angular, de ugyanúgy meredekebb tanulási görbe	Legkisebb tanulási görbe, alap Javascript tudás
Legalkalmasabb	Egyoldalas alkalmazások	Minden méretű alkalmazások	Minden méretű alkalmazások
Fenntarthatóság	Nehezebb, sok új verzió, nagy különbségek	Könnyű, hibamentes	Könnyű, hibamentes

Ábra 4.1: Frontend keretrendszerek összehasonlítása
[19][20][21]

5 Irodalom kutatás összegzése

Ebben a fejezetben a fentebb részletezett irodalom kutatás eredményeit foglalom össze, azaz melyik keretrendszerek és architektúrákra esett a választásom, és azokat fogom megindokolni.

5.1 Felhasználandó technológiák

Magát a neurális hálózat tanítását, ahhoz szükséges környezet kialakítását, illetve az adatok elő feldolgozását a PyTorch keretrendszerrel fogom elvégezni. Ezen keretrendszer sokkal fejlesztő barátibb megközelítést kínál, sokkal könnyebb hiba keresési lehetőségeket tartalmaz, ami az én megközelítésemből sokkal hasznosabb, mint a vizualizációs eszközök. Illetve az alkalmazni kívánt adathalmaz, amit a későbbi fejezetben bemutatásra kerül, számos átalakítást igényel, egyszerűbb nyersadatokkal dolgozni, mint TFRecordok struktúrájába belekényszeríteni az adatokat.

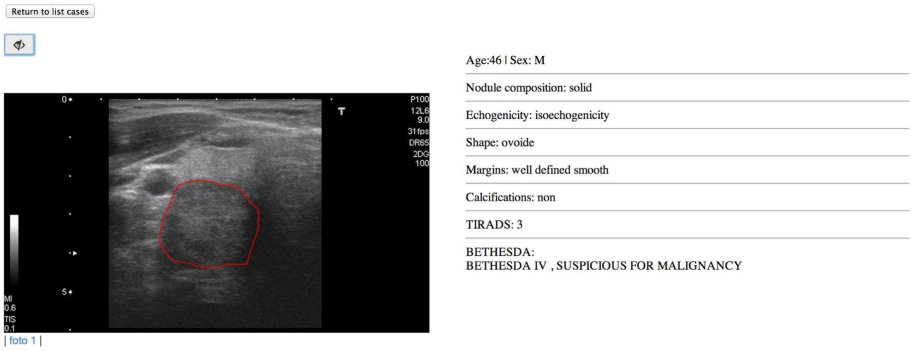
A webes alkalmazást ASP.NET Core keretrendszer fogom implementálni, mivel sokkal inkább preferálom a modernebb C# programozási nyelv használatát. Illetve a LINQ által nyújtott kényelmes lekérdezéseket. Emellett a Controller réteg megvalósítása is sokkal egyszerűbb.

A felhasználói felület megvalósítására pedig egy olyan keretrendszer szerettem volna választani, amivel könnyen lehet hibamentes, de jól bővíthető alkalmazásokat készíteni és természetesen a performanciát is szeretném figyelembe venni a felhasználói élmény miatt. Így a Vue.JS keretrendszerre esett a választásom. A Vue alkalmasabb a kisebb, kevésbé összetett alkalmazásokhoz, mint társai. Mivel szakdolgozatom egy kisebb projekt így ezt gondoltam megfelelő választásnak.

6 Adathalmaz bemutatása

Egy nyíltan hozzférhető adatbázis ami a pajzsmirigy ultrahangos képeit (Digital Database of Thyroid Ultrasound Images, DDTI [22]) tartalmazza. Az adathalmaz Kolumbia Nemzeti Egyetem által került publikálásra az alábbi linken: `http://cimalab.unal.edu.co/applications/thyroid/`

Ezen adatbázis JPEG kiterjesztésű B-modú ultrahangos képeket tartalmaz a pajzsimirgy csomókról, amely az intézet által kifejlesztett speciális annotációs eszközzel került felcímkézésre. Ez egy online Javascript rajzoló interfész, amely számos platformon működve lehetővé teszi a csomók klasszifikálását az adatbázisba feltöltött képeken.



Ábra 6.1: Annotációs eszköz bemutatása
[22]

Ezen új annotációs eszköz célja, hogy lehetővé tegye a szegmentált nodulusok és a megjegyzésekkel ellátott címkék azonnali megosztását az interneten keresztül (fentebb említett link), így folyamatosan bővíthető az adathalmaz.

A legutóbbi feljegyzés szerint az adathalmaz 428 olyan esetet tartalmaz, amely a gyanús pajzsmirigy elváltozásának teljes annotációját és diagnosztikai leírását tartalmazzák. A különböző esetek szakértő radiológusok közreműködésével a TI-RADS (lásd 1.2 fejezet) osztályozásnak megfelelően voltak végrehajtva. A lentebb látható ábra reprezentálja az egyes esetek számosságát.

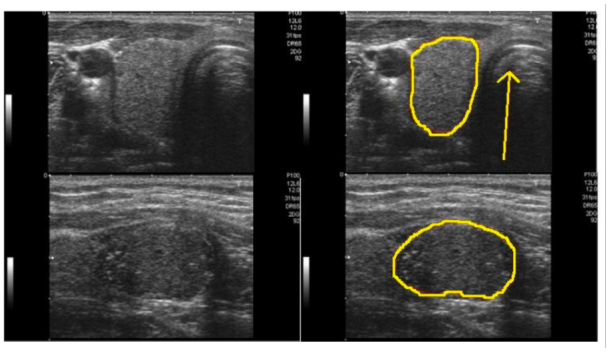
A radiológusok által felállított diagnózist ultrahang képek alapján patológusok is felülvizsgáltak és kategorizáltak őket a Bethesda rendszer [23] szerint.

A patológusok által készített leírások szintén szerepelnek XML formátumú fájlokban. Ezekben az XML-fájlokban esetenként megtalálható, hogy az adott elváltozás rosszindulatú vagy jóindulatú, emellett a betegről is számos információt szolgáltat, mint például: életkor és nem. A csomók egyéb rosszindulatúságra utaló jellemzőit is azonosították: összetétel, echogenitás, margójellemzők, mikrokalcifikációkat, kompozícióit, TI-RADS pontszám.

```
<case>
    <number>129</number>
    <age>49</age>
    <sex>F</sex>
    <composition>predominantly solid</composition>
    <echogenicity>isoechogenicity</echogenicity>
    <margins>well defined</margins>
    <calcifications>microcalcifications</calcifications>
    <tirads>4b</tirads>
    <reportbacaf></reportbacaf>
    <reporteco>Nodule 2</reporteco>
    <mark>
        <image>2</image>
        <svg>[{"points": [{"x": 214, "y": 92}, {"x": 212, "y": 82}, {"x": 208, "y": 77}, {"x": 202, "y": 73}, {"x": 186, "y": 69}, {"x": 176, "y": 65}, {"x":
            166, "y": 63}, {"x": 142, "y": 63}, {"x": 130, "y": 63}, {"x": 122, "y": 63}, {"x": 110, "y": 69}, {"x": 98, "y": 83}, {"x": 95, "y": 87}, {"x":
            91, "y": 94}, {"x": 91, "y": 101}, {"x": 109, "y": 141}, {"x": 111, "y": 143}, {"x": 117, "y": 154}, {"x": 123, "y": 156}, {"x": 147, "y": 157},
            {"x": 195, "y": 146}, {"x": 222, "y": 115}, {"x": 222, "y": 104}, {"x": 221, "y": 96}, {"x": 212, "y": 90}], "annotation": {}, "regionType":
            "freehand"}, {"points": [{"x": 407, "y": 88}, {"x": 406, "y": 82}, {"x": 398, "y": 76}, {"x": 395, "y": 69}, {"x": 390, "y": 67}, {"x": 384, "y":
            67}, {"x": 373, "y": 67}, {"x": 363, "y": 66}, {"x": 354, "y": 66}, {"x": 324, "y": 81}, {"x": 316, "y": 85}, {"x": 309, "y": 89}, {"x": 301, "y":
            97}, {"x": 298, "y": 102}, {"x": 310, "y": 129}, {"x": 318, "y": 146}, {"x": 331, "y": 149}, {"x": 336, "y": 149}, {"x": 356, "y": 157}, {"x": 361
            , "y": 159}, {"x": 387, "y": 161}, {"x": 389, "y": 161}, {"x": 396, "y": 159}, {"x": 405, "y": 150}, {"x": 413, "y": 138}, {"x": 415, "y": 131},
            {"x": 414, "y": 123}, {"x": 414, "y": 114}, {"x": 415, "y": 104}, {"x": 415, "y": 95}, {"x": 415, "y": 90}, {"x": 409, "y": 87}, {"x": 407, "y":
            86}], "annotation": {}, "regionType": "freehand"}]</svg>
    </mark>
</case>
```

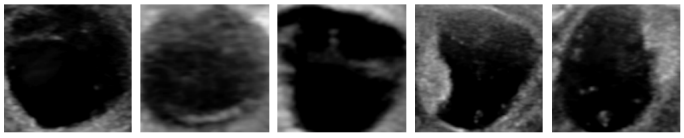

Ábra 6.2: Adathalmaz felépítése

Illetve a ′points′ tömb a szabadkézzel annotált csomók koordinátáit tartalmazza. A ábra illusztrálja a szabadkezű annotálást a fentebb említett Javascript rajzoló interfész segítségével.

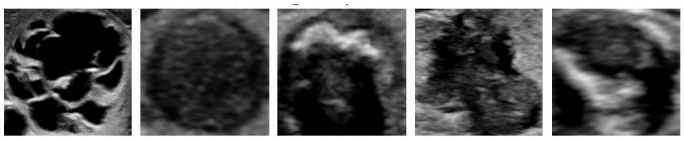


Ábra 6.3: Eredeti és annotált kép bemutatása
[22]

A következőekben néhány példát szeretnék bemutatni a pajzsmirigy jó- és rosszindulatú csomóira az adatbázisban található ultrahang képek alapján. Az első sorban lévő képek jóindulatúak, a többiek pedig rosszindulatúak.



Ábra 6.4: Jóindulatú pajzsmirigy csomó
[24]

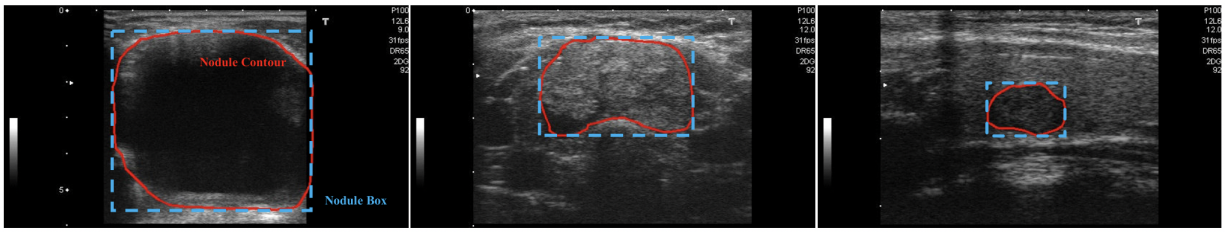


Ábra 6.5: Rosszindulatú pajzsmirigy csomó
[24]

6.1 Adathalmaz előfeldolgozása

A fentebb ismertetett adathalmazt a tanítás megkezdése előtt szükséges átalakítani. Számomra releváns adatot főként a points tömb, illetve a tirads pontszám jelent. Az utóbbi a csomó elhelyezkedéséről szolgáltat információt, míg a másik a malignitás megállapításában lesz fontos.

Azonban az előbb említett tömb, olyan pontok halmaza, ami megadja a szabadkézzel felrajzolható annotációját a csomónak. Én esetemben, úgy nevezett bounding boxok-ra, tehát egy négyszög koordinátáira lesz szükség. Ezen tömb nagyon könnyen átalakítható. A megadott koordinátákból mindig az adott tengelyen megfelelő maximum és minimum értéket megkeresve előállítható a szükséges négy pont.



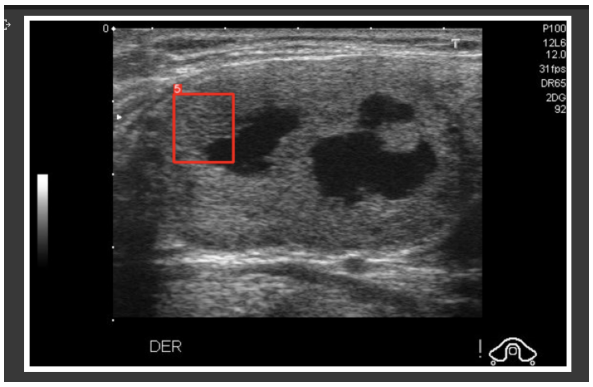
Ábra 6.6: Átalakítás illusztrációja
[24]

Majd az így keletkezett értékeket elmentettem. A további könnyebb feldolgozás érdekében az XML állományt átalakítottam JSON fájlokká.

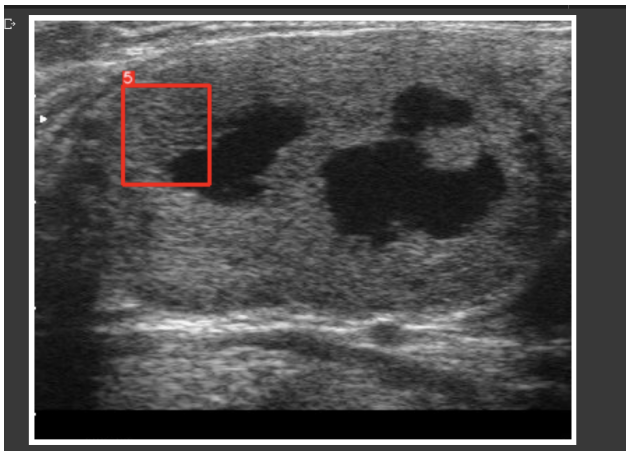
```
{
    "case_id": "162_1",
    "age": "40",
    "sex": "F",
    "composition": "solid",
    "echogenicity": "hypoechogenicity",
    "margins": "ill defined",
    "calcifications": "non",
    "tirads": "4c",
    "reportbacaf": "",
    "reporteco": "",
    "bboxes": [
        {
            "x": 367,
            "y": 78,
            "w": 84,
            "h": 95
        },
        {
            "x": 101,
            "y": 86,
            "w": 126,
            "h": 89
        }
    ]
},
```


Ábra 6.7: JSON fájl felépítése

Ezután a képek átméretezése következett. A tanításra használni kívánt képeken nagyon sok felesleges információ található. Így ezeknek levágásával kiemeltem a hasznos információkat. Az új kép méretével arányosan kell változtatni az annotációkat is.



Ábra 6.8: A kép az újraméretezés előtt



Ábra 6.9: A kép az újra méretezés után

A képen alapján jól látható, hogy az újra méretezés sikeresen végre hajtódott úgy, hogy a bounding box ugyanúgy a megfelelő objektumot jelöli. Az így keletkezett újra méretezett adatok, illetve a tanítás szemponjtából releváns adatokat szintén JSON formátumban eltároltam.


```
{
    "case_id": "88_2",
    "bboxes": [
        [
            2.0,
            0.0,
            245.0,
            209.0
        ]
    ],
    "labels": [
        4
    ]
},
```


Ábra 6.10: Az újra méretezett JSON fájl a releváns adatokkal

7 Rendszerterv

7.1 Követelmény specifikáció

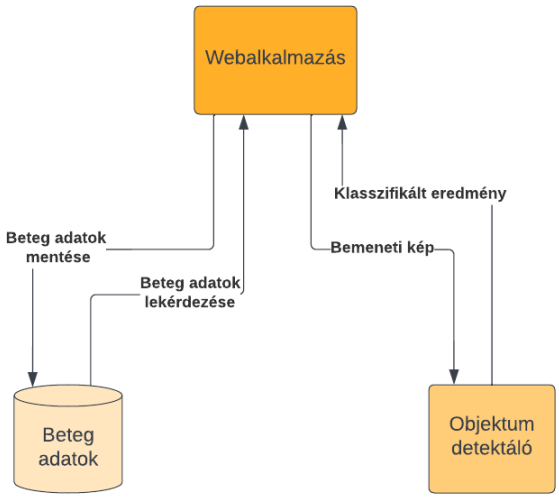
A szoftver fő funkciója, az első fejezetben említett probléma köré épül. Tehát egy bemeneti kép alapján detektálható pajzsmirigy csomó tulajdonságai alapján képesnek kell lennie eldönteni, hogy rosszindulatú vagy jóindulatú elváltozásról beszélünk. Továbbá statisztikai adatokat kell vizualizálnia a már korábban diagnosztizált betegekről.

Ehhez első sorban egy grafikus felületet kell biztosítani a felhasználó számára, ahol képes a bemeneti kép feltöltésére. A bemeneti kép minden esetben egy nyaki ultrahang felvétel, amelyen jól látható a pajzsmirigy. Emellett a szűrések elvégzéséhez egy lekérdező felület is tartalmaznia kell.

A detektálást követően, a felhasználó értesülni fog milyen malignitású az adott nodulus, és hol helyezkedik el a pajzsmirigyen.

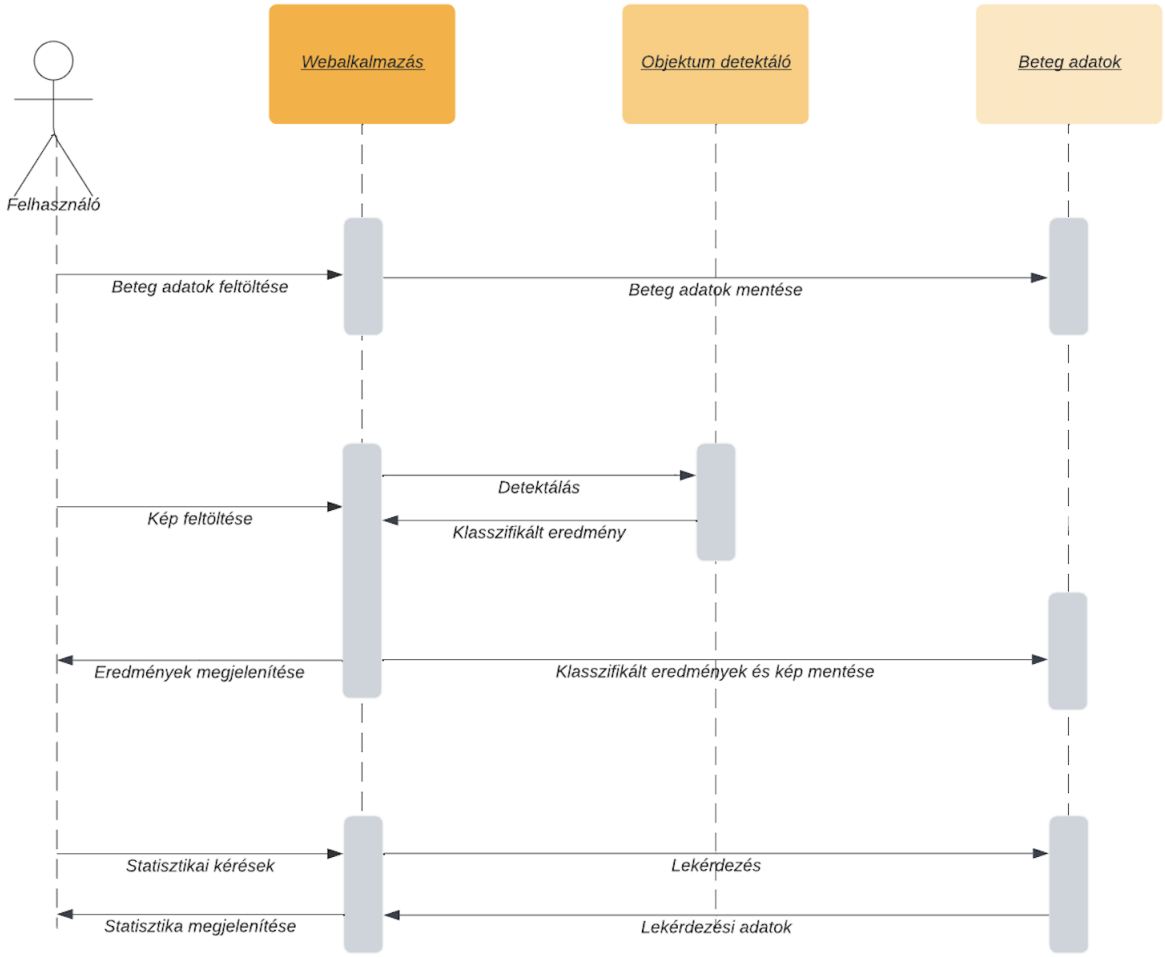
7.2 Architektúra áttekintése

Az architektúra felépítése az alábbi ábrán látható:



Ábra 7.1: Arhitektúra felépítése

A különböző folyamatok komponensek közötti megvalósítása:



Ábra 7.2: Folyamatábra

Ábrajegyzék

2.1 Pajzsmirigy elhelyezkedése 3
2.2 Pajzsmirigy betegségek 3
2.3 Pajzsmirigy csomó elhelyezkedése 4
2.4 TIRADS pont kalkulátor [2] 4
3.1 Architektúra áttekintése 6
3.2 Jellemzők kinyerése 7
3.3 Pooling módszerek 8
3.4 Szigmoid és tanH függvény összehasonlítása 8
3.5 ReLU függvény ábrázolása 9
3.6 Két módszer összehasonlításának eredménye [8] 10
3.7 A mAP-értékek összehasonlítása [8] 10
3.8 RCNN működése 11
3.9 SVM mükődése 12
3.10 Fast RCNN működése 13
3.11 Faster RCNN működése 14
3.12 A halózat mélysége és a performancia kapcsolata [10] 15
3.13 A reziduális blokk felépítése [10] 15
3.14 Darknet-53 [12] 16
3.15 YOLOv3 architektúra részletei [12] 17
4.1 Frontend keretrendszerek összehasonlítása 21
6.1 Annotációs eszköz bemutatása 23
6.2 Adathalmaz felépítése 24
6.3 Eredeti és annotált kép bemutatása 24
6.4 Jóindulatú pajzsmirigy csomó 24
6.5 Rosszindulatú pajzsmirigy csomó 25
6.6 Átalakítás illusztrációja 25
6.7 JSON fájl felépítése 26
6.8 A kép az újraméretezés előtt 26
6.9 A kép az újra méretezés után 27
6.10 Az újra méretezett JSON fájl a releváns adatokkal 27
7.1 Arhitektúra felépítése 28
7.2 Folyamatábra 29

Táblázatjegyzék

Irodalomjegyzék

[1] "Development of the thyroid gland". In: *Mikael Nilsson,Henrik Fagman* (2017). DOI: 10.1242/dev.145615.

[2] "Online TI-RADS Calculator". In: *Jimmy Tanche Wang,Tasha Ellchuk,Gary Groot ,Rob Otani* (2018). DOI: 10.4236/ojrad.2018.83020.

[3] Hinton G. LeCun Y Bengio Y. "Deep learning." In: (2015). DOI: 10.1038/nature14539.

[4] "An Introduction to Convolutional Neural Networks". In: *Keiron O'Shea, Ryan Nash* (2015). DOI: 10.48550/arXiv.1511.08458.

[5] "Convolutional neural networks: an overview and application in radiology". In: *Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, Kaori Togashi* (2018). DOI: 10.1007/s13244-018-0639-9.

[6] "Detection of Thyroid Nodules with Ultrasound Images Based on Deep Learning". In: *Xia Yu, Hongjie Wang, Liyong Ma* (2020). DOI: 10.2174/1573405615666191023104751.

[7] "Automated location of thyroid nodules in ultrasound images with improved YOLOV3 network". In: *Ling Zhang, Yan Zhuang, Zhan Hua, Lin Han, Cheng Li, Ke Chen, Yulan Peng, Jiangli Lin* (2021). DOI: 10.3233/XST-200775.

[8] "Nodule Localization in Thyroid Ultrasound Images with a Joint-Training Convolutional Neural Network". In: *Ruoyun Liu,Shichong Zhou,Yi Guo, Yuanyuan Wang, Cai Chang* (2020). DOI: 10.1007/s10278-020-00366-6.

[9] Ankita Pramanik Puja Bharati. *Deep Learning Techniques—R-CNN to Mask R-CNN: A Survey.* Advances in Intelligent Systems and Computing. Springer Singapore, 2019. ISBN: 978-981-13-9041-8.

[10] "Deep Residual Learning for Image Recognition". In: *Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun* (2015). DOI: abs/1512.03385.

[11] "YOLOv3: An Incremental Improvement". In: *Joseph Redmon, Ali Farhadi* (2018). DOI: 10.48550/arXiv.1804.02767.

[12] "Mini-YOLOv3: Real-Time Object Detector for Embedded Applications". In: *Hong-Mei Sun,Qi-Chao Mao,Yan-Bo Liu,Rui-Sheng Jia* (2019). DOI: 10.1109/ACCESS.2019.2941547.

[13] *Python.* URL: https://www.python.org/ (visited on 10/21/2022).

[14] *Tensorflow.* URL: https://www.tensorflow.org/ (visited on 10/21/2022).

[15] *Pytorch.* URL: https://pytorch.org/ (visited on 10/21/2022).

[16] *Google Colab.* URL: https://colab.research.google.com/ (visited on 10/21/2022).

[17] *ASP.NET Core*. URL: https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-6.0 (visited on 10/21/2022).

[18] *Spring Boot*. URL: https://spring.io/projects/spring-boot (visited on 10/21/2022).

[19] *Vue*. URL: https://vuejs.org/ (visited on 10/21/2022).

[20] *React*. URL: https://reactjs.org/ (visited on 10/21/2022).

[21] *Angular*. URL: https://https://angular.io/ (visited on 10/21/2022).

[22] "An open access thyroid ultrasound-image Database". In: *Lina Pedraza,Carlos Vargas, Fabián Narváez, Oscar Durán, Emma Muñoz, Eduardo Romero* (2015). DOI: 10.1117/12.2073532.

[23] "The Bethesda System for Reporting Thyroid Cytopathology: A Cytohistological Study". In: *Bakiarathana Anand,Anita Ramdas, Marie Moses Ambroise, Nirmal P. Kumar* (2020). DOI: 10.1155/2020/8095378.

[24] "Thyroid Nodule Ultrasound Image Classification Through Hybrid Feature Cropping Network". In: *RUONING SONG, LONG ZHANG ,CHUANG ZHU, JUN LIU, JIE YANG, TONG ZHANG* (2020). DOI: 10.1109/ACCESS.2020.2982767.