



ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

ORSZÁGOS TUDOMÁNYOS DIÁKKÖRI DOLGOZAT

HTML KÓD GENERÁLÁSA KÉZZEL RAJZOLT UI ELRENDEZÉSBŐL

Szerző:

Danyi Péter Márk

mérnökinformatikus BSc. szak, IV. évf.

Konzulens:

Gáspár Balázs

egyetemi gyakornok

Budapest, 2025.

ABSZTRAKT

A kutatás célja egy olyan rendszer létrehozása, mely képes kézzel rajzolt UI elrendezések-ből formázott HTML kódot generálni gépi tanulás segítségével. A rendszer felgyorsítja a felhasználói felület tervezését és elkészítését, mérsékli a megrendelő és a fejlesztők közötti, eltérő szakmából adódó félreértések számát, valamint csökkenti az ebből, illetve a programozásból adódó idővesztéseket. A rendszer képes felismerni az adott képen szereplő UI elemeket és azok relatív elhelyezkedését, majd használható, formázott HTML kód formájában visszatérni azzal.

A rendszer két fő egységből, egy erre a célra tanított, egyedi YOLOv8 neurális hálózat alapú gépi látás modellből és egy HTML kódgeneráló komponensből épül fel. A gépi látás modell a bemeneti kép előfeldolgozása után detektálja a rajzolt UI elemeket, kimeneteként a klasszifikált alakzatok osztályát és koordinátáit biztosítja. A kódgeneráló egység a kódot elkészítő algoritmus C# nyelvű implementációja, továbbá grafikus felületet biztosít a szoftver működtetéséhez és a felismert elrendezés személyre szabásához.

Az elkészített rendszer konténerizált microservice-ek összessége, melyek saját hálózaton keresztül kommunikálnak egymással. Az alkalmazás bármilyen operációs rendszeren képes működni, melyen a Docker Engine fut. A rendszer nagy pontossággal képes azonosítani a bemeneti képen található objektumokat, majd visszatérni kevés munkával véglegessé alakítható HTML kóddal.

A dolgozat a HTML kódolás gyorsítására mutat megoldást. A dolgozat rávilágít a képek megfelelő előfeldolgozásának fontosságára és különböző objektumdetektálási módszereket ismertet, ezen módszerek vizsgálata után válassza ki a YOLO-t, mint megfelelő modellt. Az egyedi tanítású modell teljesítményének és veszteségének elemzését követően bemutatja a megvalósított, használható alkalmazást.

TARTALOMJEGYZÉK

1	BEVEZETÉS	1
1.1	A probléma leírása	1
1.2	A probléma megoldása	1
1.3	A projekt összefoglalása, program használata	2
1.4	Hasonló rendszer, a Sketch2Code bemutatása	2
2	KÉPFELDOLGOZÁSHOZ KÖTHETŐ TUDOMÁNY	5
2.1	A képek előfeldolgozása	5
2.2	Háttér leválasztása	5
2.3	A papírra rajzolt elrendezés feldolgozása	6
2.4	Alkalmazott műveletek ismertetése	7
3	GÉPI TANULÁSHOZ KÖTHETŐ TUDOMÁNY	11
3.1	Gépi tanulás	11
3.2	Klasszifikáció.	12
3.3	Neurális hálózat	12
4	OBJEKTUMDETEKTÁLÁSHOZ KÖTHETŐ TUDOMÁNY	14
4.1	Nem neurális hálózat alapú objektumdetektálási módszerek	14
4.1.1	Viola-Jones módszer	14
4.1.2	Orientált gradiensek hisztogramjai az ember észleléséhez	15
4.2	Neurális hálózat alapú objektumdetektálási módszerek	16
4.2.1	Deformálható-rész modellek	16
4.2.2	Régió-CNN	18
4.3	You Only Look Once.	19
4.3.1	A YOLO tanítása	21
4.3.2	Loss függvény	21
4.4	Sebességek, modellek értékeléseinek összehasonlítása.	23

4.5	YOLOv8 gépi látás modell	24
4.6	Konklúzió	25
5	UI ELEM DETEKTÁLÓ GÉPI LÁTÁS MODELL	26
5.1	A különböző HTML elemek megkülönböztetése	26
5.2	A bemenő adathalmaz	26
5.3	A tanítás	27
5.4	A modell teljesítménye	28
5.5	A modell vesztesége	30
6	SZOFTVER ARCHITEKTÚRA.	33
6.1	A szoftver architektúrája	33
6.1.1	ASP.NET MVC Client	33
6.1.2	YOLO Computer Vision Model API.	36
6.1.3	Tárolók	36
6.2	Architektúra összefoglalása	37
7	A SZOFTVER.	38
7.1	Az alkalmazás grafikus felülete.	38
7.2	A szoftver összefoglalva	40
8	ÖSSZEFOGLALÁS	41
8.1	További célok.	42
9	RÖVIDÍTÉSEK	43
10	ÁBRAJEGYZÉK.	44
11	IRODALOMJEGYZÉK	45
12	MELLÉKLETEK.	47

1 BEVEZETÉS

1.1 A probléma leírása

Egy laikus személy számára jelentős probléma egy weboldal elkészítéséhez kellő szaktudás hiánya. Eltekintve az üzemeltetéshez szükséges ismeretektől, a probléma forrása a HTML és CSS ismeretek hiánya. Ahhoz, hogy valaki tartalomkezelő rendszer nélkül úgy alkothassa meg weblapját, ahogyan elképzelte, elengedhetetlen az előbbiekben említett kódolás ismerete. Ezen tudományok ismeretével viszonylag kevés hétköznapi ember rendelkezik, viszont annál többen szeretnének jelen lenni a weben saját elképzeléseikkel.

További gyakori probléma az informatikai szakemberek és a megrendelők közötti kommunikáció hatékonyságának hiánya. A fejlesztők és a megrendelők eltérő szakmai háttere gyakran félreértéseket okoz, mely az elvárttól eltérő végeredményt eredményezhet, továbbá hátráltatja a projekt előrehaladását.

Egy másik, fejlesztők számára gyakran felmerülő kihívás a fejlesztési idő csökkentése. Az informatikában mindenre azonnal, vagy a lehető legrövidebb időn belül van szükség, folyamatos nyomást helyezve ezzel a fejlesztőkre. A tervek, rajzolt vázlatok lekódolása munkaidőt vesz igénybe, így digitális formátumú vizuális eredmény csak később látható. Sokkal kedvezőbb, ha a tervek kódolási idejét jelentősen csökkentjük azzal, hogy a rajzot egy lépésben átültetjük webes előnézet és használható kód formájába.

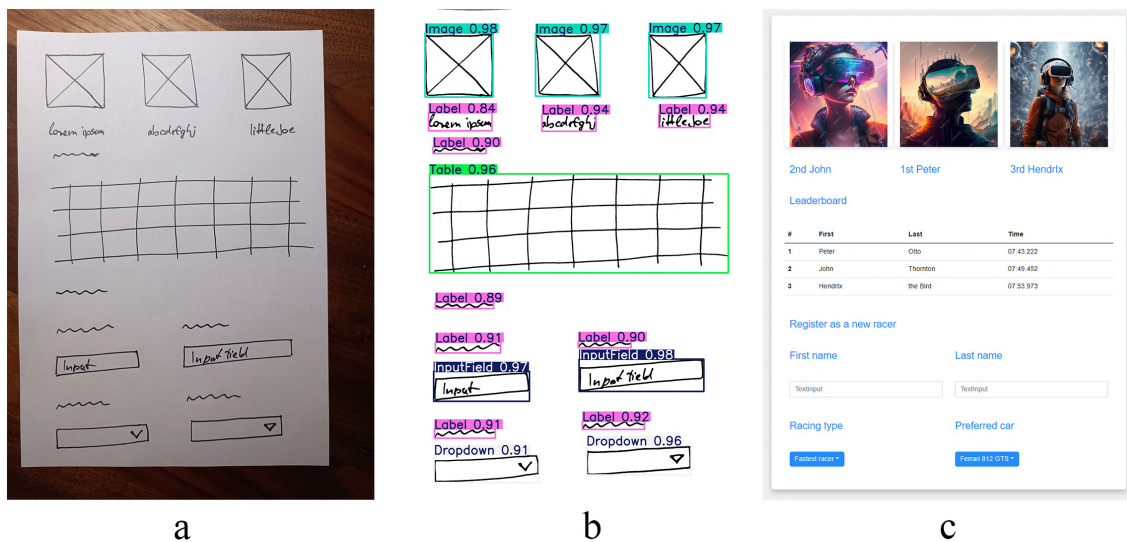
1.2 A probléma megoldása

Az előbb említett szaktudás hiánya mérsékelhető rajzolással. A programommal a fejlesztési folyamat egyszerűsítésére, felgyorsítására mutatok be megoldást. Ha elmagyarázni nem is tudjuk, hogy pontosan milyen weboldalt szeretnénk, az elvárt kinézetet könnyedén le tudjuk rajzolni. Akár saját magunk, akár egy szakember készít weboldalt, egy dolog mindenképpen kelleni fog, a kinézeti terv. Papíron meg lehet tervezni az oldal elrendezését, hogy milyen elemet hol szeretnénk látni. A elrendezést kézzel lerajzolva, vagy digitálisan illusztrálva kialakul az elvárt weboldal képe. Ennek okán szükségünk lesz egy tervrajzra, mely a weboldal megjelenését ábrázolja.

Az elkészült rajzot le kell fotózni, majd a szoftverbe betölteni, mely gépi látás segítségével digitális formában jeleníti meg a rajz webes elrendezését és biztosítja a hozzátartozó, azonnal használható HTML és CSS kódot. Ezen módszerrel szaktudás nélkül, gyorsan tudunk előrelépést elérni.

1.3 A projekt összefoglalása, program használata

A program egy rajzból, gépi látás segítségével készít használható kódot, és biztosít előnézetet. A program működéséhez szükséges képet a felhasználó személy készíti el. Szükséges egy fehér lapra, fekete színnel megrajzolni egy webes felület elrendezését. A felhasználónak úgy kell lerajzolnia, ahogyan szeretné, hogy kinézzen webes tartalma. Fontos, hogy a terv megrajzolása szabályok betartását igényli, mely szabályok az 5.1 alfejezetben vannak ismertetve. Törekedni kell arra, hogy a UI elemek és az elrendezés esztétikus, való életben használható legyen. Továbbá arányosan kell megrajzolni az elrendezést. Ezt követően a rendszer grafikus felületén keresztül fel kell tölteni a képet. A detektálás futtatása során a rendszer felismeri és klasszifikálja a különböző UI elemeket, az alakzatok koordinátái alapján pedig elkészíti a HTML kódot, majd formázatlanul és formázva visszaadja azt a felhasználónak. A biztosított kód mellett előnézetet is mutat. Emellé a felület biztosít szerkesztési lehetőséget, de a kód egyéb fejlesztőkörnyezetbe való másolására is lehetőséget ad. A folyamatot az 1.1 ábra szemlélteti.



1.1. ábra: A program ismertetése. A rajz készítése (a), a rendszer által detektált UI elemek (b), a generált kód előnézete és konfigurálási lehetősége (c).

1.4 Hasonló rendszer, a Sketch2Code bemutatása

A projekt témájával megegyező, hasonló rendszerként vállalati szinten egyedülállóan, a Microsoft Ai Lab Sketch2Code nevezetű kísérleti alkalmazását lehet megemlíteni [1]. A Sketch2Code eredetileg a kézzel felvázolt webes alkalmazás elemeit és elrendezését hivatott HTML kódban megvalósítani. A Microsoft fejlesztői észrevették a webes megjelenés lekódolásából fakadó idővesztést. Amikor egy gyors vázlatra lenne szükség, hogy

lássák hogyan is nézne ki a webes elrendezés, amelyet papíron megterveztek, minden alkalommal meg kellett várni, míg azt lekódolja egy erre szakosodott szakember, fejlesztő. Ez időbe telik, viszont üzleti szempontból fontos, hogy rövidítsék az ebből adódó eredményre várakozási időt.

Ezzel a számítógépes segítséggel nem kell külön webes tudású szakember munkájára várakozni, hiszen maga a program rövid idő alatt elkészíti a kódot.

Sketch2Code működési elve

A HTML kódba konvertálni kívánt elrendezést le kell fotózni, majd az alkalmazásba betölteni. Ezt a képet az alkalmazás továbbítja egy AI service-nek, amely a képről beolvassa a különböző lerajzolt elemeket és azok elrendezését, pozícióját, majd visszaszolgáltatja annak HTML kódját.

A rendszer magját egy egyedi gépi látás modell alkotja, amelyet rengeteg kézzel rajzolt képpel tanítottak be, hogy felismerje az alap HTML elemeket és elrendezésüket. Ennek eredményeképpen a program igazolni tudja a különböző elemek jelenlétét a betöltött képen. A HTML elem felismerő képességen kívül egy kézírás-szövegfelismerő service-t is hozzáadtak, amellyel sok esetben meg tudják határozni a képen lévő konkrét szöveget is.

A felhasználói felület és a lerajzolt elrendezésen alapuló webes nézet teljes egészében Azure infrastruktúrán fut. Azure felhőtárhelyen történik a kérés feldolgozása, innen hívja meg a program a különböző folyamatokat megfelelő sorrendben, itt történik az adattárolás. A Sketch2Code a kimeneti HTML kódon kívül képes biztosítani XAML kódot Xamarin-hoz, és UWP kódot is.

A projekt és a Sketch2Code összevetése

Habár az általam megvalósított program a Microsoft Sketch2Code-hoz hasonló cél megvalósításáért felel, a működésükben előfordulnak eltérő pontok.

A Sketch2Code felhőben működik, felülete egy webalkalmazás, így nem függ az operációs rendszertől, kizárólag böngésző és internet elérés szükséges hozzá. Előnye, hogy ezáltal távoli erőforrásokat tud elérni, ami meggyorsítja a számítási folyamatokat, viszont belátható, hogy a számítási kapacitás hiánya néhány rajz esetében nem számottevő probléma. Hátránya lehet, hogy felhővel való kapcsolat, vagy internet hiánya esetén nem elérhető az általa nyújtani kívánt szolgáltatás.

A saját szoftveremet webes környezetre terveztem, így az aktuális infrastruktúra erőforrásait tudja felhasználni az objektumdetektálás, elemzések és számítások megvalósításához. Mivel a programom konténerizált microservice-ekből áll, adott a lehetőség bár-

milyen platformra áttérni. A működési sebességet befolyásolja a számítógép hardveres teljesítőképessége, ám később látható, hogy a kisebb, lokális számítási kapacitás a felhasználói élményt nem rontja.

Míg a Sketch2Code az Azure Cloud Platform Cloud Ai Webservice Custom Vision API-t használja egy előre betanított gépi látás modellen, addig projektemben egy saját, betanított YOLOv8 gépi látás modellt használok fel. A képek tárolására Azure Blob Storage-t használ, saját programom helyi fájlrendszert. Továbbá saját programom lokális SQL adatbázisban tárolja a működéséhez szükséges további adatokat.

2 KÉPFELDOLGOZÁSHOZ KÖTHETŐ TUDOMÁNY

A program kimeneti HTML kódjának minősége szorosan összefügg a bemeneti kép minőségével, különös tekintettel annak élességére és felbontására. Az éles, magas minőségű, előfeldolgozott bemeneti képek lehetővé teszik a UI elemek és elrendezések pontosabb felismerését.

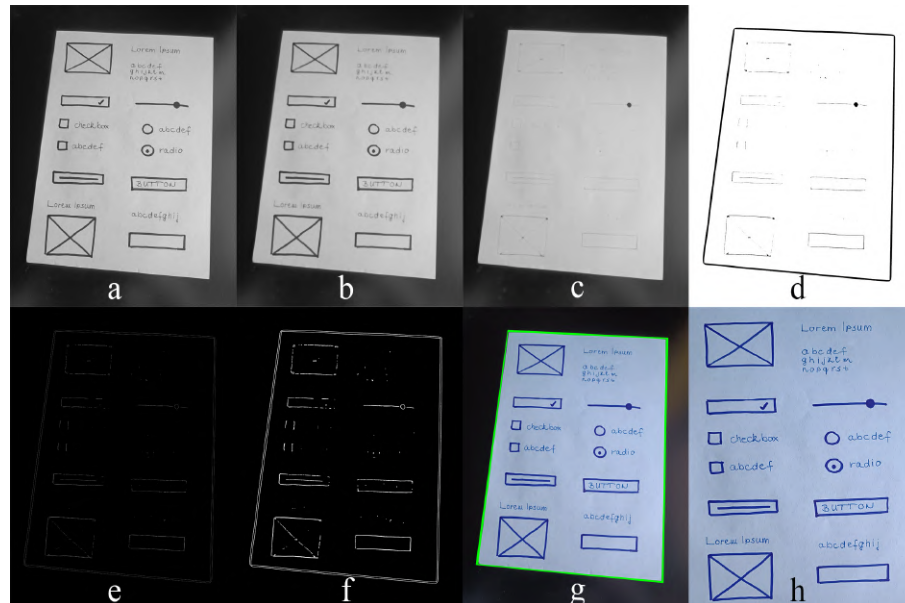
2.1 A képek előfeldolgozása

Előfeldolgozás az a folyamat, mely segítségével a képen szereplő számunkra nem releváns információ mennyisége elnyomható és csökkenthető, ugyanakkor a képen lévő fontos információ kiemelhető, felerősíthető [2]. Az előfeldolgozás célja a detektáló modell számára szükségtelen képinformáció – például színek, háttér, zaj, valamint az egyenetlen megvilágításból adódó anomáliák – minimalizálása, a releváns információ mennyiségének növelésével párhuzamban. Az előfeldolgozás műveletei között szerepelnek a geometriai transzformációk, ilyen művelet a hajlítás, a méretezés.

2.2 Háttér leválasztása

A program bemeneteként a mai mobilkamerák paramétereiből és minőségéből adódóan egy jó minőségű, éles kép várható el. A program fejlesztése során 3:4 méretarányú, 3000:4000px méretű képeket használtam. A program képfeldolgozásának első fázisában történik a képen lévő papírlap detektálása és megfelelő perspektívába hajlítása. A cél elérésére a bemenő képet szürkeárnyalatosra konvertálja, majd a zaj csökkentésére bilaterális filtert alkalmaz megfelelő kernelmérettel. A nem egyenletes megvilágításból adódó feldolgozási nehézségek elkerülése érdekében arra használ dilataciót, hogy a fehér részeket, főleg a lap széleit vastagabbá, szignifikánsabbá tegye, ugyanakkor ennek hatására a lapon található feketével rajzolt alakzatok vonalvastagságát csökkenti. Ez a folyamat jó esetben eltünteti a rajzolt alakzatokat. Mivel a rajzok tartalmazhatnak téglalap alakú UI elemet, például képet, szövegdobozt, táblázatot, az alakzatok eltüntetése előnnyel jár, ugyanis a kontúrvonalak detektálása során nem keletkezik ezen alakzatok éleit ábrázoló élszegmens, ezáltal azok nem eredményezhetnek problémát a megfelelő alakú és méretű élszegmens kiválasztása során. Ezután történik a küszöbölés megfelelően megválasztott paraméterekkel. A folyamat adaptív küszöbölést (Gauss eloszlás szerinti súlyozott átlag) használ. Ezen bináris képen egy plusz lépésként Canny éldetektálás történik, mellyel célunk a bemeneti fehér lap széleinek megtalálása. Az élek szignifikánssá tételéhez, vastagításához újra dilataciót végez. Ezen a képen keresi meg az objektumhatárokat, vagyis a kontúrokat. Az ebből kinyert információval számításokat végezve (legnagyobb területű poligon kiválasztása) történik a lap sarkaihoz tartozó koordinátáinak azonosítása. A

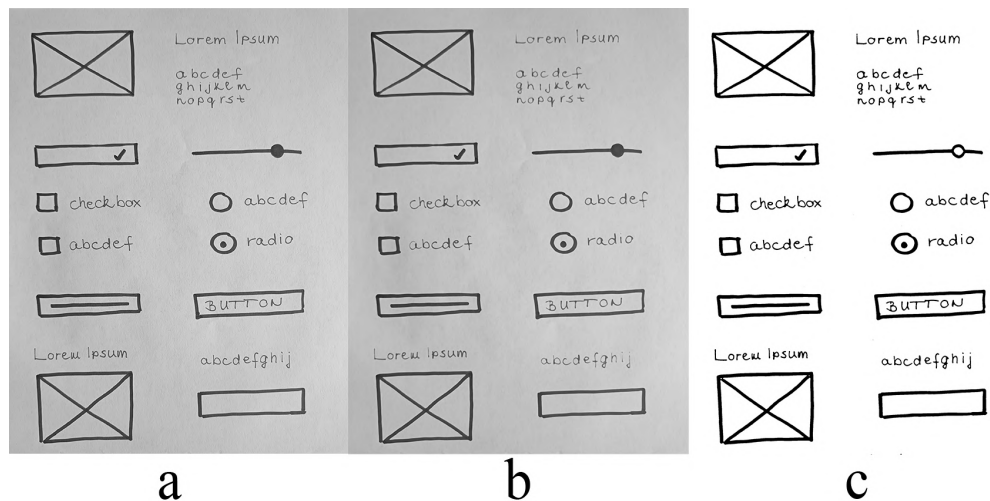
kontúrvonalakat felhasználva poligonnal közelíti az alakzatot, majd az eredeti képet felhasználva, a kontúr csúcsaiból adódó koordinátákból számítva hajlítással, majd a szegélyek minimális vágásával éri el a keresett végeredményt, vagyis a háttér nélküli, teljes méretet kitevő papírlap képét, melyen megkezdődhet az előfeldolgozás második fázisa. A folyamatot a 2.1 ábra szemlélteti. Az említett műveletek ismertetése a 2.4 fejezetben történik.



2.1. ábra: A bemeneti kép szürkeárnyalatossá alakítva (a), bilaterális filterezés (b), dilatálás (c), küszöbölés (d), Canny éldetektálás (e), dilatálás (f), kontúrvonalak detektálása (g), eredeti kép hajlítása, vágása (h).

2.3 A papírra rajzolt elrendezés feldolgozása

A program a feldolgozáshoz szürkeárnyalatossá alakítja a képet, majd megfelelő kernelű medián szűrővel csökkenti a képen szereplő zaj mennyiségét. Ilyen zajok például a papírlap rostjai. Ezután – a háttér leválasztásának folyamatában is szereplő - adaptív küszöbölést végez attól eltérő, külön erre a célra, a bemeneti képek méretéhez beállított kernelmérettel. Tapasztalat alapján, ha a kép méretének csökkentését csak a küszöbölés után végzi a program, a detektáló modell számára releváns információban gazdagabb képet kapunk. A folyamatot a 2.2 ábra szemlélteti. A következő bekezdésekben a felhasznált műveletek ismertetése következik.



2.2. ábra: A bemeneti kép szürkeárnyalatossá alakítva (a), medián filterezés (b), küszöbölés (c).

2.4 Alkalmazott műveletek ismertetése

Zaj eltávolítása

A képeket gyakran rontja néhány véletlenszerű hiba [2] – a képjel véletlenszerű változásából adódó minőségromlást általában zajnak nevezik. A zaj előfordulhat a kép rögzítése, továbbítása vagy feldolgozása során, és függhet a képtartalomtól vagy lehet attól független. A kép rögzítése során zaj a rossz fényviszonyok, rossz ISO beállítások, helytelen expozíciós idő, túl alacsony vagy túl magas hőmérséklet hatására léphet fel, továbbá a digitális képpel való munka, szerkesztés során is a képhez adódhat. A képzaj csökkentése érdekében különböző szűrők használhatók. Az előfeldolgozási folyamat során történő küszöbölés előtt nélkülözhetetlen a zaj megfelelő módon történő szűrése, a képen szereplő rajzok éleinek lehető legkisebb mértékben való elmosásával. A folyamatokban két különböző nemlineáris szűrő, a bilaterális szűrő és a medián filter szerepel.

Bilaterális szűrő

A nemlineáris bilaterális szűrőt a Gauss-féle konvolúcióhoz hasonlóan módon, egy adott lokációhoz közeli pixelek súlyozott átlagaként határozzuk meg [3]. A különbség az, hogy a bilaterális szűrő figyelembe veszi a szomszédos pixelek intenzitásbeli különbségét is, hogy az élek minél kevésbé mosódjanak el a simítás során. Ahhoz, hogy egy képpont intenzitásértéke befolyásolhasson egy másik képpontot, nem csak egy ahhoz közeli helyet kell elfoglalnia, hanem hasonló intenzitásértékkel is kell rendelkeznie. A bemeneti képeken végzendő zajcsökkentés során fontos a papír széleinek lehető legkevesbé történő elmosása, ezért erre a célra bilaterális szűrőt használ a program.

Medián szűrő

A medián szűrő egy olyan szűrő, melynek válasza az általa lefedett szomszédságban található képpontok értékeinek rendezettségén alapul [4] [330-332]. A medián szűrő egy képpont értékét az adott képpont egy előre meghatározott szomszédságában lévő intenzitás-szintek mediánjával helyettesíti. Egy helyes kernelmérettel beállított medián szűrővel a bemeneti képen jelen lévő véletlenszerű zajok túlnyomó többsége eltávolítható. Emellett kisebb elmosódást eredményez a detektálandó alakzatok körvonalain, mint a hasonló kernelméretű lineáris simító szűrők.

Dilatálás

A dilatálás egy alapvető művelet a morfológiai képfeldolgozásban. Röviden megfogalmazva a dilatáció vastagítja az adott objektumot. Alkalmazásához szükségünk van egy strukturáló elemre, mely egy olyan mátrix, melynek mérete és alakja meghatározza, hogy az adott képpont környezetében mely pixelekre alkalmazzuk a műveletet. Formája legtöbb esetben téglalap, kör vagy kereszt. A program négyzet alakú strukturáló elemet használ. Dilatálás esetén az adott pixelhez a kernelben lévő legnagyobb értékű pixelt rendeljük. Az első dilatálással a lap széleit vastagítja meg a program, továbbá a fekete rajzokat (melyek UI elemet ábrázolnak) vékonyítja, megfelelő paraméterek beállításával eltünteti. A második alkalmazásakor a Canny éldetektált képen lévő lap körvonalát teszi szignifikánssá.

Adaptív küszöbölés

A bemeneti papírlap fekete filccel rajzolt alakzatokat tartalmaz, ezért a küszöbölés előtt számított hisztogramjában megfigyelhető, hogy kettőnél több szignifikáns csúcs nem szerepel, illetve a világos pixelek irányában lévő csúcs jóval magasabb, mint a sötét. A küszöbértéket ezen tisztán bimodális hisztogram két jelentős csúcsa közötti tartományban kell keresni.

Küszöbölés során minden egyes pixelt megvizsgálunk. Ha a szürkeárnyaltos képen az adott pixel megfelel egy bizonyos kritériumnak, annak intenzitásértékét 255-re változtatjuk, ellenkező esetben 0-ra. A projektben 0-s értéket kap a detektálandó objektumhoz tartozó pixel, a háttér pedig 255-öt. Eredményként a detektáló modell számára információgazdag bináris képet kapunk.

A kép nem egyenletes megvilágításából adódóan egy olyan küszöbölő algoritmust alkalmazok, mely figyelembe veszi a változó fényviszonyokat, vagyis a küszöbértéket ennek megfelelően lokálisan változtatja. Ezen algoritmusokat adaptív algoritmusoknak ne-

vezzük. Ezen eljárások a bemeneti kép különböző területeihez automatikusan választják ki az optimális küszöbértéket. A küszöbérték az adott képpont környezetében lévő intenzitásértékek súlyozott átlaga, ahol a súlyok középpontban koncentrálódó Gauss eloszlást követnek. Eszerint az adott lokáció középpontjától való távolság növekedésével csökken a súly nagysága.

Élkeresés képeken

Az élfelismerési eljárás a képek elemzésének egyszerűsítésére szolgál azáltal, hogy drasztikusan csökkenti a feldolgozandó adatmennyiséget, ugyanakkor megőrzi az objektumhatárookra vonatkozó hasznos szerkezeti információkat. A program Canny éldetektáló algoritmusát használja [5]. Az élek megtalálása a fényesség hirtelen, szignifikáns változásán alapszik [4] [702]. Maguk az élpontok olyan pixelek, ahol az intenzitás hirtelen, nagy mértékben megváltozik. Az élszegmensek ezen összekapcsolódó élpixelek halmazai. Az éldetektorokat ezen jelenség felismerésére tervezték. A képen található rajzolt UI elem körvonala olyan élszegmens, amelyben az élszegmens két oldalán lévő háttér intenzitása vagy sokkal nagyobb, vagy sokkal kisebb, mint a vonalat alkotó pixelek intenzitása.

Helyesen kivitelezett küszöbölés eredményeként a képen szinte kizárólag a gépi látás modell számára releváns információ marad. A program ekkor használja Canny megfelelő paraméterekkel beállított éldetektorát, melynek futtatása eredményeként a lap 4 oldala folytonos, téglalap alakú élszegmenst alkot. Ugyanakkor a képen szereplő rajzok is éleket alkothatnak. A lap azonosításához szükséges megtalálni a legnagyobb téglalap alakú területet.

Kontúr detektálás

A határvonal vagy kontúr egy adott objektum körvonalát jelöli a bináris képen. A háttér eltávolításának folyamata során azt a folytonos körvonalat jelöli, mely a lapot elhatárolja a háttértől. Az erre a célra felhasznált algoritmust Suzuki és társai részletezik [6]. A határvonalak tetszőleges pontossággal közelíthetők poligonokkal. Ha egy határvonal zárt görbét alkot, akkor a közelítés abban az esetben válik maximálisan pontossá, melyben a sokszög élszegmenseinek száma megegyezik a határvonal pontjainak számával – azaz minden szomszédos pontpár egy-egy szakaszt határoz meg a sokszögben. A poligonális közelítés célja, hogy egy adott határvonal alakját minél kevesebb szakasz használatával adja vissza, megőrizve a kontúr jellegzetességeit [4] [821]. A kontúrok detektálása után ki kell választani a legnagyobb közrezárt területtel rendelkező kontúrt, mely a teljes papírlap.

Geometriai transzformációk

A háttér eltávolítása után, a kiválasztott kontúr sarok koordinátái alapján végzünk hajlítást. A hajlító függvény a sarokpontok alapján úgy alakítja át a képet, hogy a négy sarokpont egy szabályos téglalapot alkosson. Ezt követően minimális vágást szükséges végezni a kép széleiből. A papírra rajzolt elrendezés feldolgozása után a kép méretét az oldalarányok megtartásával csökkentjük. A gépi látás modell teljesítménye a pontosság és sebesség komponensekből tevődik össze. A betáplált képek méretének csökkentése alapvető fontosságú a számítási sebesség növelése és a memóriahasználat optimalizálása érdekében. A kép felbontását olyan mértékben szükséges redukálni, hogy az objektumok észlelhetősége és felismerési pontossága ne sérüljön. Mivel a modell GPU-t használ és nem valós időben szolgáltat eredményeket, ezért a lap detektálása utáni kialakított bemeneti képek (körülbelül 2000:3500px) sem okoznak jelentős sebességbeli lassulást. A rendelkezésre álló számítási kapacitás miatt elegendő visszafogott mértékben csökkenteni a kép méretét.

3 GÉPI TANULÁSHOZ KÖTHETŐ TUDOMÁNY

3.1 Gépi tanulás

A gépi tanulás [7] oly módon definiálható, mint számítási módszerek összessége, mely a megszerzett tapasztalatok alapján javítja saját teljesítményét, vagy, mint pontos előrejelzést, predikciót biztosító modell. A tapasztalat ez esetben a rendelkezésre álló, kifejezetten a modell tanítására létrehozott adathalmazra utal, melyet a tanuló modellnek biztosítanak. Ilyen, általam készített adathalmazon tanítottam be a gépi látás modellt. Fontos kiemelni, hogy a tanuló modell számára biztosított adatok mennyisége és minősége is befolyásoló tényező, minél gyorsabb és pontosabb eredményre akarunk szert tenni a detektálás során, annál több és pontosabban címkézett, minőségi adaton keresztül kell tapasztalatot szereznie.

A gépi tanulással olyan megoldás készíthető, mely a rendelkezésre bocsátott adatokból képes előre definiált szempontok alapján analízist végezni, majd ezek alapján következtetést alkotni. Már önmagában is egy komplex, számításigényes feladat, melyet egy hagyományos programmal és hardver konfigurációval nem tudunk hatékonyan elérni. A saját szoftverem esetében ilyen feladat a különböző HTML elemek felismerése, ezzel együtt annak megállapítása, hogy valójában egy számunkra releváns adat szerepel a képen, vagy sem. A gépnek meg kell tudni különböztetni a tanítás és tapasztalat alapján a különböző HTML elemeket, sőt fel is kell ismernie, vagyis meg kell értenie mi a különbség egy olyan kép között, amin szerepel a keresett elem és egy olyan kép között, amin nem szerepel.

Fontos a tanulási folyamat ellenőrzése, hogy a gép pontosan működjön, ne biztosítson hibás eredményeket, vagy ne tanuljon hibás tapasztalatok alapján. Tanulási folyamat szempontjából három csoportot különböztetünk meg. Felügyelt, felügyelet nélküli és megerősítéses tanulást. Felügyelt tanítás során ember által ellenőrzött, jóváhagyott bemeneti és kimeneti értékek állnak rendelkezésre. Az általam tanított modell számára biztosítva van a felügyelet, hiszen minden, tanító adathalmazban szereplő képhez tartozik címke, mely segítségével ellenőrizni tudja saját predikcióit a modell. Felügyelet nélküli tanításnál adott rendelkezésre álló bemenethez nem biztosított kimeneti érték helyessége. A gépnek ilyenkor szintén a bemeneti és kimeneti értékek alapján kell egy működést eszközölnie, viszont ekkor a környezetétől nem kap a működés pontosságára vonatkozó visszajelzést. A megerősítéses tanulás hasonlít leginkább az emberi tanulásra. Tanulási folyamata során pozitív vagy negatív jutalmat kap, az adott problémára nyújtott megoldására vonatkozóan. Az általam választott, későbbiekben ismertetésre kerülő modell a tanulási folyamatát súlyok használatával végzi.

3.2 Klasszifikáció

Klasszifikáció során minden egyes megtalált objektumhoz kategóriát rendel a modell. A modellem különböző HTML elemeket keres a képen, majd azonosítja az objektum HTML elem típusát.

3.3 Neurális hálózat

A neurális hálózatok [8] emberi agyi működést modellező információfeldolgozó eszközök, melyek nagy számú, párhuzamosan működő műveleti elemből (neuronból) állnak. Főbb tulajdonságai:

- Azonos, vagy hasonló típusú lokális neuron többnyire rendezett topológiájú összekapcsolt rendszeréből áll
- Rendelkezik tanulási algoritmussal, mely általában minta alapján való tanulást jelent
- Rendelkezik előhívási (megtanult információ felhasználását lehetővé tevő) algoritmussal

Tipikusan két fázisa van a neurális hálózatok működésének. Az első a tanulási fázis, mely a neurális hálózat kialakítására szolgál. E fázis során a modell számára biztosított bemeneti adatokban rejlő mintát valamilyen módon eltároljuk a hálózatban. A második fázis az előhívási fázis, mely során az első fázisban megalkotott tapasztalatot és információt használjuk fel. A tanulási fázis lassabb, több időt vesz igénybe, míg az előhívási fázis rendszerint gyors feldolgozást jelent. Az általam betanított modell esetében a tanítás eredményeit a dokumentum részletesen tárgyalja az 5. fejezetben.

A neuron

"A neuron egy több-bemenetű, egy-kimenetű eszköz, amely a bemenetek és a kimenet között általában valamilyen nemlineáris leképezést valósít meg" [8]. A neuronok rendelkezhetnek lokális memóriával, amelyben tárolni tudnak be- és kimeneti információt, valamint a korábbi, eddigi működésből következő állapotinformációt. Az aktuális kimeneti értékeket tipikusan nemlineáris függvénnyel alkotja meg, melyet aktivációs függvénynek nevezünk. Ezen aktivációs függvénnyel döntenek arról az egyes neuronok, hogy milyen kimenetet szolgáltatassanak. Ezen műveleti elemek alkotják a neurális hálózatot, melyek képesek önállóan információt feldolgozni, tanulni.

A neurális hálózat felépítése

A neuronok rétegekbe vannak rendezve. A háló különböző rétegeiben eltérő számban előforduló műveleti elemei a neuronok. Egyes rétegekben az adott réteg kimenetei lesznek a következő réteg bemenetei. A bemeneti adatok ezen háló rétegein haladnak keresztül, ez az adat feldolgozási folyamata. A bemenő adat feldolgozása során minden szint más szempont alapján értékeli az információt. Minden szint az általa kapott kimenetet biztosítja a következő szint felé. A neurális hálózatban használt összeköttetések biztosítják a jelek egyirányú terjedését. Az általam használt konvolúciós neurális hálózatot a dokumentum a 4.3 bekezdése részletezi.

4 OBJEKTUMDETEKTÁLÁSHOZ KÖTHETŐ TUDOMÁNY

Objektumdetektálás

Az objektumdetektálás lényege a képen található objektumok pozíciójának megkeresése, majd a talált elemek klasszifikálása, mely által valamely csoportba besorolható, kategorizálható, ezáltal egy bizonyos valószínűségi szinten megnevezhető a képen talált objektum.

Az objektumdetektálási módszerek és azok működése két aspektusból közelíthető meg. Egyik ilyen megközelítést követő halmazba sorolhatók a nem-neurális háló használatán alapuló modellek, másik megközelítési halmazba pedig a neurális hálón alapuló objektumdetektálási modellek tartoznak.

A következő részben tárgyalásra kerülnek ezen különböző megközelítések és módszerek.

4.1 Nem neurális hálózat alapú objektumdetektálási módszerek

4.1.1 Viola-Jones módszer

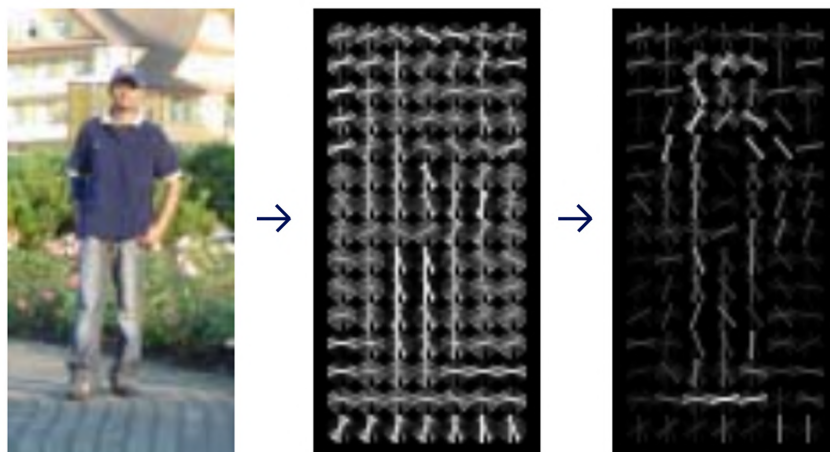
Az objektumdetektálás egyik fontos mérföldkövének tekinthető Viola-Jones valós idejű arcdetektáló rendszere [9]. Paul Viola és Michael J. Jones cikke kifejezetten az arcdetektálásról szól. A Viola-Jones robosztus, nagysebességű arcdetektáló rendszer képes arcok gyors és nagy arányban pontos detektálására. Három fontos tényezőtől áll. Az első egy új képreprezentáció, az „integrálkép” bevezetése, mely lehetővé teszi a detektor által használt jellemzők gyors kiszámítását. A második tényező egy egyszerű és hatékony, AdaBoost tanulási algoritmuson alapuló osztályozó, amely a potenciális jellemzők közül kiválasztja az osztályozás szempontjából kritikus jellemzőket. Harmadik tényező az osztályozók kaszkádban való kombinálása, mely lehetővé teszi a kép háttérének elválasztását, hogy az arcdetektálás szempontjából ígéretes területeken több számítást végezhesen. Mivel a módszer az általam felvázolt probléma hatékony megoldására nem használható, a fejezet nem tárgyalja részletesebben.

A további pontokban tárgyalt rendszerek, implementációk más nézőpontból közelítik meg az objektumok detektálását. A későbbiekben egyre közelebb kerülünk a projekt szempontjából fontos, megfontolandó rendszerekhez és a megvalósításhoz kellő objektumdetektáló modellhez.

4.1.2 Orientált gradiensek hisztogramjai az ember észleléséhez

Dalal és Triggs [10] által leírt módszer célja az emberi alakok (az eredeti cikkben főként gyalogosok) pontos és hatékony detektálása különböző környezeti feltételek mellett, beleértve a különböző megvilágítási viszonyokat és eltérő háttereket. A cikk egy olyan, jól kifejező jellemzőkészlet szükségességét tárgyalja, melyet kinyerve lehetőség nyílik zsúfoltabb, számítógép számára nehezebben értelmezhető képeken is az emberi alak detektálására. A HOG (Histograms of Oriented Gradients) jellemzőleíró módszer kiváló teljesítményt nyújt ezen területen. A HOG jellemzők kinyerése után egy bináris SVM-el (Support Vector Machine) ellenőrzik a képen szereplő objektumot, hogy az embert ábrázol, vagy sem.

Az SVM egy felügyelettel végrehajtott gépi tanulási mód. Főként lineáris klasszifikációs problémák megoldására használható. Jelen esetben a két lineárisan elkülöníthető csoport az embert ábrázoló és az embert nem ábrázoló csoport. Az SVM hozza meg a végső döntést arról, hogy az adott képen detektálható ember, vagy sem (4.1 ábra).



4.1. ábra: A képről való HOG jellemzők kinyerése, majd a pozitív és negatív SVM értékekkel súlyozott HOG leíró, az SVM megállapítja az ember jelenlétét a képen.

A képen lévő emberek különböző méretben és különböző pozícióban jelenhetnek meg, mely a teljes kép átvizsgálásának szükségességét eredményezi, vagyis a kép minden egyes pixelét érintve kell vizsgálnodni. Ezért a módszer működése csúszóablakos képfeldolgozáson alapul. Egy 64×128 pixel méretű téglalap végigiterál a teljes képen. Minden egyes iterációban az ábrán látható folyamatot hajtja végre. Ha a csúszóablakban ember detektálható, akkor annak koordinátái kinyerhetők az aktuális iteráció értékeiből számolva.

Mivel a képen eltérő méretben is szerepelhetnek emberek, a csúszóablak mérete miatt fennáll az a lehetőség, hogy túl kicsi az ablak mérete. Éppen ezért a detektálási módszer

részeként méretezéshez is folyamodni kell. Ez a folyamat képpiramis készítésével valósul meg. Az adott kép méretét csökkentjük, majd újra elvégezzük a csúszóablakos folyamatot, az eredeti méretű ablakkal. Addig csökkentjük a kép méretét, míg megfelelően képes működni a detektálás, ha már nem található ember a képen, a folyamat leáll.

4.2 Neurális hálózat alapú objektumdetektálási módszerek

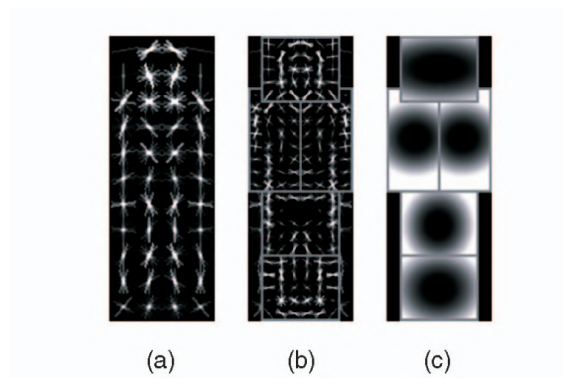
4.2.1 Deformálható-rész modellek

Az objektumfelismerés egy témakör az alapvető gépi látásban rejlő kihívások közül. A feladatot nehezíti a tény, hogy a különböző kategóriába tartozó objektumok a saját kategóriájukon belül is eltérhetnek. Lehet más a méretük, mintájuk, megjelenésük. Ahogy az előzőekben tárgyalt, nem neurális hálózaton alapuló objektumdetektálási módszerek jelentős problémaforrása volt, a neurális hálózat alapú detektálási módszereknél is közrejátsszik az emberek képen való megkeresésénél, hogy azok különböző szögben, megvilágításban, testhelyzetben és méretben helyezkedhetnek el.

A Deformable Part Models (DPM, [11]) egy olyan objektumfelismerő rendszer, amely a nagymértékben változó objektumokat több skálájú, deformálható-rész modellek keverékeinek felhasználásával reprezentálja. A deformable part kifejezés a detektálási folyamatban segítséget nyújtó részekre utal. Deformálható részeknek nevezzük egy objektumon azokat az elemeket, amelyek azonos kategóriába tartozó objektumokon belül is könnyen mutathatnak eltérést. Ilyen például egy ember felismerésének folyamata során a karok, lábak detektálása, figyelembevétele, ugyanis ezek könnyedén jelenhetnek meg eltérő helyzetben, megjelenésben, viszont nagy mértékben hozzájárulnak a képen lévő ember azonosításához, ugyanis ezen részegységek (karok, lábak, testrészek) jelenléte és megfelelő relatív elhelyezkedése ember jelenlétére utal.

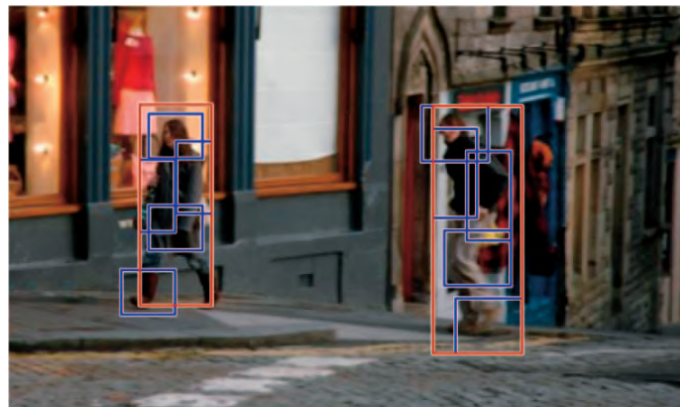
A szűrők súlyokat határoznak meg az HOG jellemzőkhöz. A 4.2 (c) ábra a pozitív súlyokat vizualizálja különböző orientációknál. A térbeli modell vizualizációja mutatja az adott rész középpontjának gyökérhez viszonyított különböző helyekre történő elhelyezkedésének költségét.

Emberek detektálását megnehezíti a más emberek által zsúfolt környezet. A DPM Dalal-Triggs rendszerét egészíti ki azzal, hogy nem kizárólagosan a teljes embert keresik a képen, hanem emellett az emberhez tartozó részeket (fej, végtagok, test) is, továbbá kiegészíti egy úgynevezett penalty score-on (büntetőpont) alapuló kiértékeléssel. Ha a folyamat az emberi alakzatot megtalálta, megvizsgálja a test részeit. Ideális esetben a testrészek közel vannak a testhez. Minél inkább megfelelő helyzetben található az adott testrész, annál kevesebb büntetőpontot ad a modell a találatnak. Értelemszerűen minél távolabbi helyen, annál több büntetőpontot ad.



4.2. ábra: Egykomponensű személymodell által kapott találatok. A modell a durva gyökérszűrő (a), számos magasabb felbontású rész szűrő (b), és az egyes részek gyökérhez viszonyított elhelyezkedésének térbeli modellje (c) határozza meg.

Ezeket a modelleket olyan diszkriminatív eljárás során tanítják be, melynek bemenete több képet, és képeken lévő objektumokról információt, azok határolódobozait (bounding box) tartalmazza. Ugyanakkor a tanítás ilyen adathalmazon a deformálható-rész modell számára problémát is hordoz magával. Mivel a határolódobozok a teljes objektumot jelölik, így az objektumon belüli külön részletek láthatatlanok maradnak a tanítás során. Képzeljünk el egy képet, melyen egy ember szerepel. A embert tartalmazó határolódoboz pontjai a címkében vannak definiálva. Tanításra ehhez hasonló módon felcímkézett képeket használunk. A képen szereplő határolódobozban a teljes ember alakzat megtalálható, viszont a különböző testrészek nincsenek külön jelölve (4.3 ábra). Ez a probléma orvosolható lenne pontosabb, részletesebb (az alakzat több részére kiterjedő) címkézéssel, viszont a nem kizárólag optimális részek használatával teljesítményromlást is eredményezhet amellet, hogy ez időigényes és költséges munka.



4.3. ábra: A detektált emberi alak és a hozzátartozó testrészek detektálása. A pirossal keretezett rész a teljes ember, a kék pedig a deformálható-rész modellek.

Bár a DPM az objektum szempontjából jelentős megjelenési változásokat is képes megragadni, egyetlen deformálható-rész modell gyakran nem elég ahhoz, hogy gazda-

gabb objektumkategóriát reprezentáljon. Ezért DPM-ben modellek keverékéből álló rendszert alkalmaznak ezen jelentősebb változások kezelésére

4.2.2 Régió-CNN

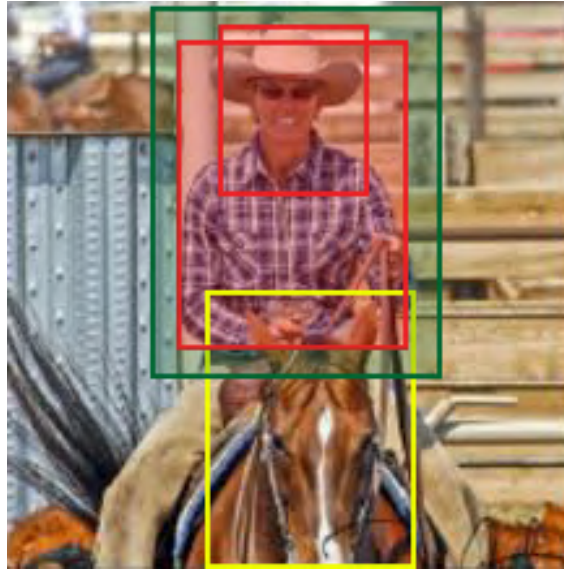
A Regions with CNN features (R-CNN, [12]) egy mélytanulási modelltípus, mely gépi látással megvalósítható feladatokra, különösen objektumfelismerésre alkalmazható. Nagy kapacitású konvolúciós neurális hálózatokat és szelektív keresést használ a képen előforduló objektumok detektálására.

Az R-CNN objektumdetektálásra használt rendszere 3 modulból áll. Az első modul kategóriafüggetlen régiójavaslatokat tesz. Ezen régiók a kép olyan részeit tartalmazzák, melyek objektumdetektálási szempontból relevánsak. A második modul egy nagyméretű konvolúciós neurális hálózat, mely minden régióból egy fix hosszúságú jellemzővektort von ki. A harmadik modul osztályspecifikus lineáris SVM-ek halmaza.

Működésekor az R-CNN a bemeneti képet különböző régiókra osztja szelektív keresés segítségével. A szelektív keresés színeken, alakzatokon, éleken, formákon és egyéb részleteken alapul. Ezáltal a képen olyan területeket talál, mely a modell számára érdekes, létfontosságú lehet. A bemeneti képen körülbelül 2000 különböző bottom-up régiót jelöl ki a modell. A bottom-up megközelítés azt jelenti, hogy a kép részeinek elemzésekor csak az adott képen szereplő információra hagyatkozik a rendszer. Ilyen információk például az élek, színek, textúrák. Az elemzés során ezeket a részleteket kombinálja a rendszer, majd ellenőrzi, hogy az adott rész megfeleltethető-e a keresett objektummal. A top-down a bottom-up-pal ellentétben már valamilyen előzetes ismereten alapszik. Ilyen előzetes ismeret lehet a tanított modelleknél egy osztály, alakzat, melyet keres a rendszer. A régiókból CNN-nel történő jellemzők kinyeréséhez ezeket a régiójavaslatokat a modellben használt CNN architektúrájából adódóan 227x227 pixel nagyságúra méretezik. Ezen méretezés mellé további műveletként társul a régió körüli 16 pixeles vastagságú külső margó hozzáadása. Utolsó lépésként a régiójavaslatokból kinyert jellemzővektorokat a megfelelő osztályhoz tartozó SVM klasszifikáló modellbe táplálják. A modell minden osztályhoz egyedi SVM-et használ. Ezen SVM-ekkel ellenőrzik a régió tartalmát. Az SVM tudja megállapítani, hogy az adott régió valóban tartalmaz-e objektumdetektálási szempontból releváns információt, és ha igen, akkor visszaadja a detektált objektum osztályát.

A Dalal-Triggs módszerrel összehasonlítva észrevehető a képen ellenőrzött régiók számából adódó különbség. Míg Dalal-Triggs esetében csúszóablakkal a létező összes 64x128 pixel nagyságú régiót és a képből alkotott képpiramis által létrejött további régiókat is ellenőrizni kellett az emberi jelenlét tényének megállapítása érdekében, addig az R-CNN ezt a rengeteg ellenőrzést kerüli el a régiójavaslatok alkalmazásával.

Felmerülhet bennünk a kérdés, hogy mi történik abban az esetben, amikor a 2000 javasolt régiónál vannak olyan területek, ahol több javasolt régió átfedésben van. Mi számít, mely régió fogja végleg kijelölni az azonosított objektumot? Az átfedésekből adódó régióduplikáció megoldására NMS-t (non-maximum suppression) használ.



4.4. ábra: A képen zöld téglalapban (ground-truth), piros téglalapokban a régiójavaslatok, sárga téglalapban pedig egy másik osztályhoz tartozó határolódoboz.

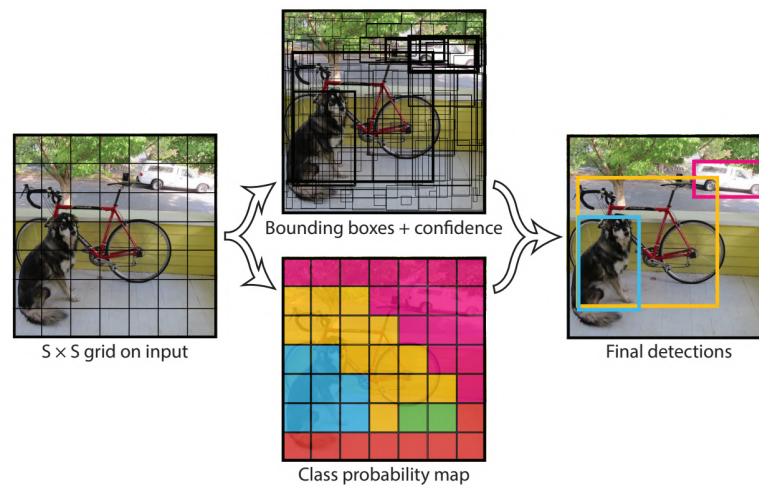
A GT (Ground Truth) az objektumokfelismerés területén, detektálható objektum azonosításban az eredeti, pontosan címkézett adatokat jelenti. Gépi látás modellek esetében a ground truth az a határolódoboz, mely legpontosabban lefedi a keresendő objektumot. Ezeket általában emberi annotátorok hozzák létre, akik manuálisan megjelölik az objektumokat a képeken. A GT-t a modellek tanulására, tesztelésére és kiértékelésre használják, hogy meghatározzák azok teljesítményét.

A tárgyalt a modell tesztelésére a tesztképen a felismerendő objektumot bejelölték a felügyelt tanítás során, ebből kifolyólag ismerjük a határolódoboz elhelyezkedését. A modell a tanulása során az NMS miatt mindig azt a régiót jelöli meg végső objektumot kijelölő határolódobozként, amely az eredeti (ground-truth) határolódobozzal a legnagyobb átfedésben van. Így jelenleg a 4.4 ábrán látható nagyobb piros téglalap lesz a megtalált objektumot kijelölő terület.

4.3 You Only Look Once

A korábbi, detektáláshoz osztályozókat használó objektumfelismerési rendszerekkel ellentétben a You Only Look Once (YOLO, [13]) az objektumfelismerést térben elválasztott határolódobozok és hozzájuk kapcsolódó osztály-valószínűségek regressziós problémája-

ként fogalmazza meg. Egy teljes képből egyetlen neurális hálózat egyetlen kiértékeléssel jósolja meg a határolódobozokat és a hozzá tartozó osztály valószínűségét. Mivel a teljes detektálási folyamat egyetlen hálózattól áll (4.6 ábra), így az teljes mértékben optimalizálható a detektálási teljesítményre. A YOLO valós idejű képfelismerés terén 45 képkocka/másodperc sebességgel tud képet feldolgozni, a hálózat kisebb verziója, a Fast YOLO pedig 155 képkocka/másodperc sebességgel. A pozitív tényezők mellett, mint az egyszerű működés és sebesség, negatív tényezőként jelenik meg a pontosság. A YOLO a képen kis méretben szereplő objektumok elhelyezkedésének meghatározásában a modell kiadásakor pontatlan.



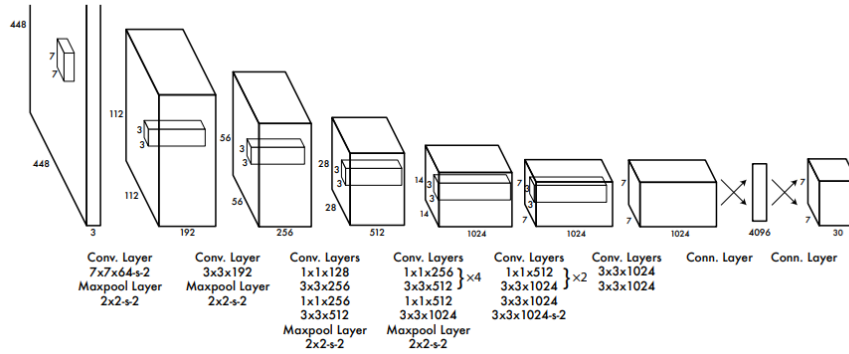
4.5. ábra: Kép $S \times S$ méretben cellákra osztása, határolódobozok és hozzájuk tartozó confidence score, osztályvalószínűségi térkép, ezek kombinációjából adódó detektált objektumok és pozíciójuk.

Kezdeti lépésként a YOLO az adott bemeneti képet $S \times S$ méretben cellákra osztja (4.5 ábra). Minden cellához hozzárendel több különböző vektort. Az első vektortípus (4.1 (a)) a határolódoboz paramétereit (x, y, w, h, C) tartalmazza. Az (x, y) koordináták a határolódoboz középpontját jelentik a rács adott cellájának körvonalához viszonyítva. A w és h a határolódoboz szélességét és magasságát adja meg a teljes képhez viszonyítva. A C (confidence score) érték, jelöli azt, hogy mennyire biztos abban, hogy milyen objektum osztályt tartalmaz az adott rács. Az efféle vektorokból mindig annyi van, ahány különböző objektumhatároló doboz középpontja helyezkedik el az adott cellán belül. Ezeket a vektorokat határolódoboz-jóslat vektoroknak nevezhetjük. A másik vektortípus (4.1 (b)) tartalmazza a cellához rendelt objektumok bizonyosságait, amennyiben az adott cella több objektumnak is tartalmazza a középpontját, több elemű a vektor. Ahelyett, hogy ezeket a vektorokat külön-külön tárolnánk, egy kimeneti vektorba helyezzük. A kimeneti vektor hossza megadható a 4.1 (c) képlettel. B az adott cellához tartozó határolódobozok száma, n a különböző detektálható objektumok osztályainak száma. A tudományos cikkben ($S = 7$), vagyis 7×7 -es rácsot, ($B = 2$) cellánkénti határolódobozt és $n = 20$ különböző

detektálható osztályt használnak. A képlet alapján $2 * 5 + 20 = 30$ elemű vektort kapunk cellánként.

$$\begin{aligned}
 & [x_1, y_1, w_1, h_1, C_1] \\
 \text{a) } & [x_2, y_2, w_2, h_2, C_2] \\
 & \dots \\
 & [x_B, y_B, w_B, h_B, C_B]
 \end{aligned}
 \quad
 \text{b) } [p(c_1), p(c_2), \dots, p(c_n)]
 \quad
 \text{c) } (B * 5 + n) \quad (4.1)$$

A YOLO architektúrájában szereplő konvolúciós neurális hálózat (4.6 ábra) 24 konvolúciós réteget tartalmaz, melyet további 2 teljesen összekötött réteg követ. A kezdetben konvolúciós rétegek a kép jellemzőit nyerik ki, míg a teljesen összekötött rétegek a kimeneti vektort állítják elő. Az architektúrában a YOLO egymást felváltva használja az 1×1 redukciós és 3×3 konvolúciós rétegeket.



4.6. ábra: A YOLO architektúra

4.3.1 A YOLO tanítása

Működése során a YOLO modell cellánként több határolódobozt jósol. A modell tanítása során viszont kizárólag egy határolódobozt alkalmaznak egy objektum jelölésére. Kizárólag egy javasolt határolódobozt tesz felelőssé a talált objektumért. Ezt a határolódobozt az alapján válassza ki, amelyik javaslatnak a legnagyobb az átfedése (IoU) a GT-vel. Ez a határolódobozt jóslók közötti szakosodáshoz vezet, mely során mindegyik prediktor egyre jobb lesz bizonyos méretek, méretarányok vagy objektumosztályok előrejelzésében.

4.3.2 Loss függvény

A YOLO a veszteség kiszámításához a klasszifikációs veszteség, lokalizációs veszteség és a bizonyosság veszteség összegét használja fel. A teljes veszteségfüggvényt a 4.5 képlet

ismerteti.

A klasszifikációs veszteség kiszámításához a predikció és a GT osztály közötti négyzetes hiba összegét használja, minden osztályt és cellát összegezve (4.2 képlet).

$$\sum_{i=0}^{S^2} 1_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \quad 1_i^{\text{obj}} = \begin{cases} 1, & \text{ha obj szerepel cella } i \text{-ben} \\ 0, & \text{egyébként} \end{cases} \quad (4.2)$$

A lokalizációs veszteség (4.3 képlet) kiszámításához a doboz-koordináta veszteséget összegzi a bizonyossági veszteséggel. A koordináta veszteség az összes cellát és összes határolódobozt összegzi minden olyan cella esetén, mely tartalmazza az i objektumot, és az i cellában lévő j -edik határolódoboz felelős az adott predikcióért. Az x és y koordináták négyzetes hibáját, a szélesség és magasság gyökének négyzetes hibáját összegzi. Továbbá a λ tényezővel tovább méretezi, amelyet a következő részben tárgyalunk.

$$\begin{aligned} & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{\text{obj}} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \\ & + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \end{aligned} \quad (4.3)$$

A bizonyossági veszteség (4.4 képlet) esetén a képlet az összes cellát és határolódobozt összegzi minden olyan cella esetén, mely tartalmaz objektumot. Ezt összegzi az összes cellával és összes határolódobozsal minden olyan cella esetén, mely nem tartalmaz objektumot, majd méretezi λ_{noobj} -al.

$$\sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \quad (4.4)$$

A teljes képlet ismeretében látható, hogy kizárólag a négyzetes hibát használja fel arra a célra, hogy megfelelő értékeket kapjunk az osztályokhoz tartozó valószínűségekre és az összes határolódoboz paraméterre. Emiatt tekintheti a YOLO az objektumdetektálást egy regressziós problémának.

A képeken sok cella esetén léphet fel az az állapot, hogy nem tartalmaz objektumot. Ez a bizonyosságot a 0 irányába tolja, gyakran felülírva az objektumokat tartalmazó cellákból származó gradienseket. Ez a modell instabilitásához vezethet. Ennek érdekében λ -val növelik a határolódoboz koordinátáinak predikciójából származó veszteséget, és csökkentik az objektumokat nem tartalmazó dobozok bizonyossági predikciójából adódó veszteséget. A cikkben a $\lambda_{obj} = 5$, a $\lambda_{noobj} = 0.5$.

$$\begin{aligned}
& \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \\
& + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
& + \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} (C_i - \hat{C}_i)^2 \\
& + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{noobj} (C_i - \hat{C}_i)^2 \\
& + \sum_{i=0}^{S^2} 1_i^{obj} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2
\end{aligned} \tag{4.5}$$

4.4 Sebességek, modellek értékeléseinek összehasonlítása

A 4.1.1 bekezdésben tárgyalt Viola-Jones módszer és a 4.1.2 bekezdésben részletezett Dalal-Triggs általi megközelítés a projekttől eltérő feladatokra használható, továbbá utóbbi, a csúszóablakos működéséből adódóan nagyobb képeken óriási számítási mennyiséget igényel, ezért a hatékonyabb neurális hálózat alapú modellek vehetők számításba.

YOLO vs DPM

A DPM csúszóablakos megközelítést alkalmaz a tárgyak felismerésére. Az objektumdetektálást külön részegységek segítségével valósítja meg. Külön nyeri ki a statikus jellemzőket, osztályozza az adott régiót és adja meg a határolódobozát. Ezzel szemben a YOLO kizárólag egy konvolúciós neurális hálózatot alkalmaz mindezen folyamatok elvégzésé-

re. A hálózat egyidejűleg végzi a jellemzők kinyerését, határolódoboz előrejelzését, az NMS-t (non-maximum suppression) és a helyzetfüggő következtetést. A statikus jellemzők helyett a hálózat a jellemzőket közvetlenül az adatok áramlása során képezi és optimalizálja. A YOLO egységesített architektúrája gyorsabb, pontosabb, mint a DPM.

YOLO vs R-CNN

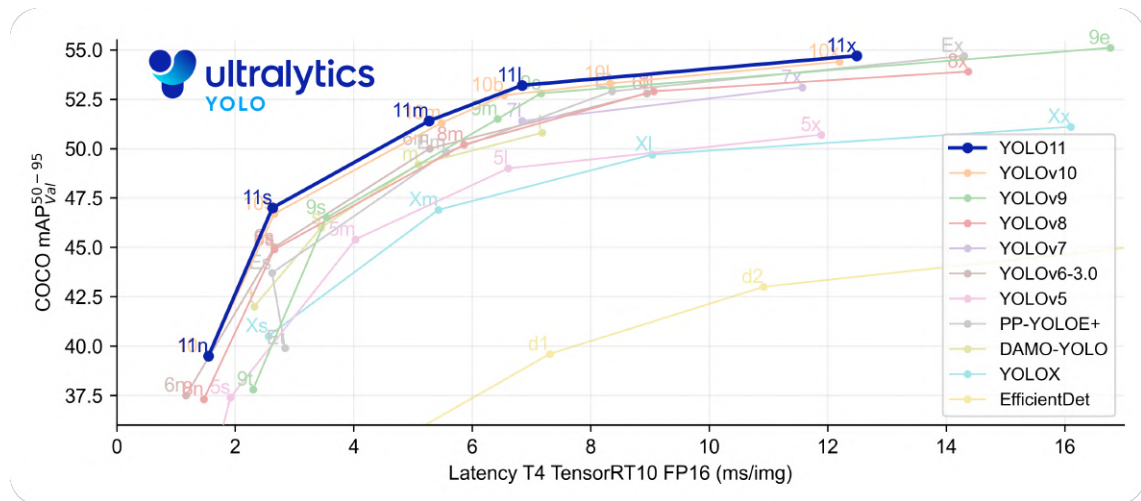
Az R-CNN csúszóablak helyett régiójavaslatokat használ a képeken lévő objektumok detektálására. A szelektív keresés potenciális határolódobozokat generál, majd egy konvolúciós háló kivonja a jellemzőket, egy SVM pontozza a határolódobozokat, egy lineáris modell pontosítja azokat, majd az NMS eltávolítja a régióduplikációkat. Minden egyes szakaszt egymástól függetlenül kell hangolni, továbbá az így kapott rendszer lassú, teszteléskor képenként több mint 40 másodpercet vesz igénybe a felismerés folyamata.

A YOLO és R-CNN rendelkezik néhány hasonlósággal. Minden egyes cella potenciális határolódobozokat javasol, és ezeket konvolúciós jellemzőkkel pontozza. A YOLO azonban térbeli korlátozásokat tesz a cellák javaslataira, ami segít csökkenteni az objektum többszörös észlelését. A YOLO sokkal kevesebb határolódobozt javasol a szelektív keresés körülbelül 2000 régiójavaslatával ellentétben. Ráadásul a YOLO az R-CNN folyamatában szereplő fázisokat egyetlen konvolúciós neurális hálózatban egyesíti.

4.5 YOLOv8 gépi látás modell

A YOLO (v1) 2015-ös megjelenése óta több személy és vállalat általi fejlesztésen ment keresztül. Az évek során folyamatosan fokozták a modell teljesítményét, javították hatékonyságát. Az Ultralytics által kiadott YOLOv8 a gépi látás alapú mesterséges intelligenciához köthető feladatok teljes skálájához támogatást nyújt, beleértve az objektumfelismerést, szegmentálást, pózbecslést, követést és osztályozást. Ez a sokoldalúság lehetővé teszi a felhasználók számára, hogy a YOLOv8 képességeit különböző alkalmazásokban és területeken használják ki, mint például UI elemek detektálására.

A YOLOv8 [14] valós idejű objektumfelismerő és képszegmentáló modell a mélytanulás és a számítógépes látás élvonalbeli fejlesztéseire épül, és magas teljesítményt nyújt a sebesség és a pontosság tekintetében. Napjainkban a YOLO11 a legfrissebb verzió, mely korábbi modellekhez viszonyított magasabb mAP50-95 teljesítménye a 4.7 ábrán látható. Látható, hogy a jelenlegi 11-es verzió nem nyújt akkora előnyt teljesítményben, hogy az általam elkészített feladatot érdemes legyen magasabb verzióra emelni.



4.7. ábra: A YOLO11 és a korábbi modellek mAP50-95 teljesítményének ábrázolása [15].

4.6 Konklúzió

Az előzőekben tárgyalt objektumdetektálási módszerekről elmondható, hogy az újabb rendszerek a régebbi módszerek gondolatmenetén indultak el, és egészítették ki őket saját, gyorsabb és hatékonyabb objektumdetektálást biztosító komponensekkel. A különböző, általam tárgyalt rendszerek YOLO-val történő összehasonlítása alapján a projekt megvalósításához a YOLO objektumdetektálási rendszer 8-as verzióját választottam. A projekt megvalósításához megfelelő egy single-shot objektumdetektáló (SSD), a YOLO. A modell tanításának menete és használata a YOLO közösség, és az Ultralytics általi támogatása miatt jól dokumentált, érthető, ami kiemelkedően fontos a fejlesztés elősegítésében. A következő fejezet részletesen tárgyalja a betanított modellt.

5 UI ELEM DETEKTÁLÓ GÉPI LÁTÁS MODELL

YOLOv8 gépi látás modell tanítása

A modell tanítása egy erre a célra fejlesztett betanítási adattár létrehozásával történt. Az adattárban szerepelnek különböző eltérő méretű, ceruzával és tollal rajzolt UI elemek (5.1 (a) ábra), továbbá ezen UI elemekből álló elrendezések.

5.1 A különböző HTML elemek megkülönböztetése

A különböző UI elemek egy kézi rajzon olykor nagyon hasonlítanak egymásra. Például fekete-fehér, rajzolt környezetben egy beviteli mező és egy legördülő menü. A hasonló alakjuk, méretük és bennük megtalálható szöveg a klasszifikációt nehezíti. Erre ad megoldást különféle, valósághoz közel álló, könnyen értelmezhető jelölések bevezetése a rajz elkészítésénél (5.1 (a) ábra). A következő felsorolás tartalmazza az osztályok neveit és ábrázolási módjait:

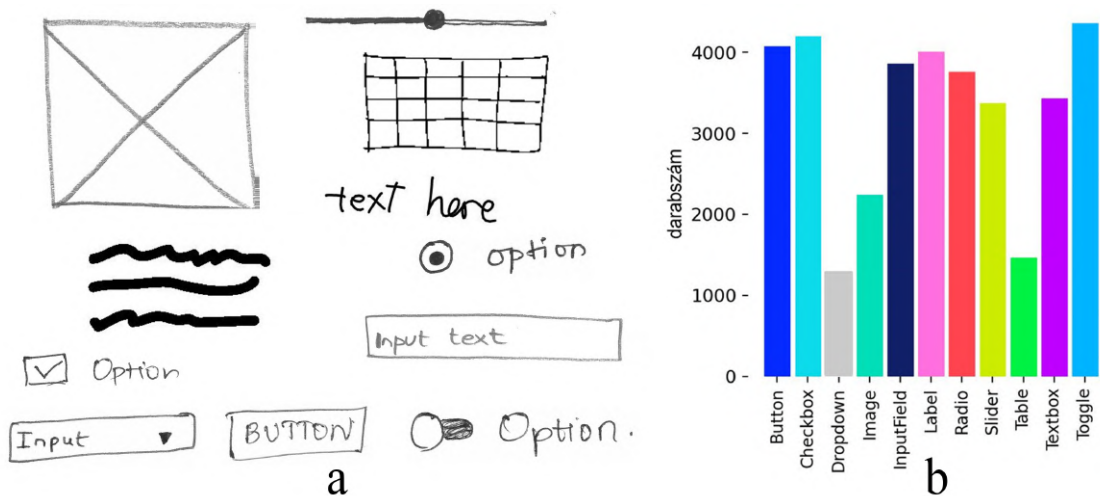
- Label: egysoros rövid szöveg, vagy hullámos vonal
- Textbox: szöveg, hullámos vonalak, vagy üres téglalap
- Button: téglalap, benne "Button" felirat
- Image: téglalap, szemközti csúcsait összekötő vonallal
- Radio: kör és benne egy pont, mellette "radio" felirat
- Checkbox: kis négyzet, mellette "checkbox" felirat
- TextInput: téglalap, benne "input" felirat
- Dropdown: téglalap, benne "input" felirat, mellette lefele mutató háromszög
- Slider: vonal, rajta egy pont
- Toggle: kapcsoló ikon
- Table: négyzetrácsos vonalak

5.2 A bemenő adathalmaz

A tanítási adathalmaz tartalmaz olyan képeket, melyeken kizárólag egy-egy UI elem található, továbbá tartalmaz olyan általam generált elrendezéseket, melyeken több UI elem is szerepel. Azok a képek, melyeken kizárólag egy UI elem szerepel, egységesen bináris képek [16]. A generált képeken a fehér háttérükről leválasztott UI elemek viszont valós, telefonnal fotózott papírlapokra vannak néhány szabály szerint elhelyezve (12.3 (a) melléklet). Ezeken a lapokon a 2.2 bekezdésben részletezett módon történt az előfeldolgozás.

Az összesen 20.000 képből álló tanítási adathalmaz 3 részre oszlik. A halmaz 75%-a training adat, 12,5%-a teszt és 12,5%-a validációs kép. A halmazokban az adott osztály

elemeit reprezentáló képek aránya megegyezik a teljes adathalmazban jelenlévő, különböző osztályú UI elemek egymáshoz viszonyított arányával. A generált képek szintén az előbb említett arányban vannak elosztva. A modell tanításához szükség van a képen szereplő, detektálandó objektumok határolódoboz koordinátáira és osztályukra. Ezen koordináták az egy UI elemet tartalmazó képeken alapértelmezetten a teljes képet jelölik. A generált képek esetében az objektumhoz tartozó koordinátákat a képek létrehozásakor számította ki az elrendezést generáló algoritmus, majd egy külön .txt kiterjesztésű fájlba mentette az adott képen szereplő összes objektum határolódobozát és osztályát. Az egyes osztályok tanítóhalmazban előforduló mennyiségét az 5.1 (b) ábra szemlélteti. Ezen pontosan elosztott, jó minőségű, annotált képeken tud megfelelően tanulni a modell.



5.1. ábra: UI elemek különböző jelölésekkel (a), elemek eloszlása a tanító adathalmazban (b).

5.3 A tanítás

A gondosan elkészített tanító adathalmazon való tanítás NVIDIA GeForce RTX 3050 4GB GDDR6 paraméterekkel rendelkező GPU-n történt. A tanulás körülbelül 20GB DDR5 RAM folytonos felhasználásával 9 órán át tartott. A tanulási folyamat egy COCO (Common Objects in Context) adathalmazon előtréningezett modellből indult ki. A COCO egy nagyméretű adathalmaz objektumdetektálási, szegmentálási és képleírás-generálási feladatokhoz [17]. A kezdő modell tulajdonságait az 5.1 táblázat szemlélteti. Egy YOLOv8s (small) modellt alapul véve, 100 tanulási cikluson keresztül zajló tanulás eredményét az 5.4 bekezdés részletezi. Az tanulási ciklus, más néven epoch a modell által a teljes tanító adathalmazon való egyszeri végighaladást jelenti, mely során a modell a saját súlyait változtatja a loss függvénynek megfelelően.

Model	Image	mAP ^{val}	Speed	Params	FLOPs
YOLOv8s	640 px	44.9	1.20 ms	11.2 M	28.6 B

5.1. táblázat: A kiinduló YOLOv8s modell teljesítménye. Image: képméret, melyen a modell tanul, mAP^{val}: mAP50-95 érték (egyetlen modellre és egyetlen méretskálára vonatkoznak a COCO val2017 adathalmazon), Speed: A100 TensorRT GPU-n mért sebesség (a COCO validációs képeken átlagolták egy Amazon EC2 P4d példány használatával), Params: modell paramétereinek száma milliókban, a modell méretét és komplexitását jelzi, FLOPs: a másodpercenkénti műveletek száma (Floating Point Operations) a modell egyetlen következtetése esetén, milliárdokban.

5.4 A modell teljesítménye

A teljesítménymérés adatai [18] a modell működésének pontosságát, megbízhatóságát és hatékonyságát értékelik. A mérési eredmény a validációs adathalmazon történő előrejelzések sikeressége alapján keletkezett. A következő metrikák több aspektusban közelítik meg a modell teljesítményét.

Precision-Recall

A 12.1 melléklet (a) grafikon ábrázolja a modell különböző küszöbértékek melletti precizitását. A precizitás megmutatja, hogy mennyire megbízható a modell a megfelelő bemeneti minták pozitívként történő osztályozásában. Az osztályonkénti precizitás értéke a helyesen klasszifikált (true positive) objektumok száma elosztva, az összes, adott osztályként klasszifikált (true positive és false positive) objektum számával (5.1 képlet). A grafikonról leolvasható, hogy a modell a label és dropdown típusú objektumok magasabb pontosságú osztályozását csak magasabb bizonyossági szinten éri el. Az ok a UI elemek ábrázolásának hasonlóságában rejlik. Labelként egy kézzel írt, rövid, egy-két szavas szöveget kell azonosítani. Viszont ilyen kinézetű objektum megjelenhet több UI elem esetén is, mint például a gombokban, dropdown menükben, input mezőkben, rádió gombokban, checkbox melletti feliratban vagy textbox esetén, emiatt még több tényezőt figyelembe kell vennie a modellnek, még jobban meg kell győződnie a talált objektum pontos klasszifikálásához. Dropdown esetében pedig az input mezőkkel való hasonlósága miatt kíván fokozott figyelmet az objektum vizsgálata. Az 5.1 (b) ábra szemlélteti a dropdown és input field-ek közti true positive és false positive eloszlást. A labelek számához képest magas az olyan objektumtípusok száma a tanítóhalmazban, melyek labelhez hasonló tulajdonsággal is rendelkeznek. A dropdown esetében pedig az input field-ek mennyisége jóval magasabb. A 12.1 melléklet (c) grafikonjáról leolvasható, hogy a többi osztályt már egész alacsony konfidenciaszinten is pontosan klasszifikálja. Ez azt mutatja, hogy

a modell tanítóhalmaza ezen objektumok esetében jó minőségűek, jól klasszifikálhatóak voltak, továbbá, hogy a modell tanulása pontos volt.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \quad (5.1)$$

A 12.1 melléklet (b) grafikon ábrázolja a modell különböző küszöbértékek melletti recall (szenzitivitás) értékét. A recall a helyesen pozitívként klasszifikált minták számának arányát jelenti az összes pozitív minta számához viszonyítva (5.2 képlet). Megmutatja, hogy a modell milyen mértékben képes észlelni a jelenlévő pozitív mintákat. A grafikonról leolvasható, hogy a modell körülbelül 0,9-es konfidencia szinten rendelkezik 0,9-es recall értékkel.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \quad (5.2)$$

A precision-recall görbe (12.1 melléklet (c)) függőleges tengelyen ábrázolja a precizitást, vízszintes tengelyen pedig a recall értéket. Különböző célokban eltérő lehet a precizitás és a szenzitivitás fontossága. Ha a cél a lehető legtöbb objektum detektálása még akkor is, ha esetleg lesz köztük néhány false positive, akkor a modellnek magas recall értékkel kell rendelkeznie. Ha a modell pontossága kulcsfontosságú még akkor is, ha emiatt nem azonosít néhány objektumot (mivel nem lépi át a meghatározott küszöbértéket az adott objektum detektálása során) akkor a pontosságot helyezzük előtérbe a szenzitivitással szemben. Ideális esetben a modell egyszerre rendelkezik magas pontossággal és magas szenzitivitás értékkel. Az általam készített modell célja az, hogy a papírlapon szereplő, detektálandó rajzok közül mindegyiket pontosan felismerje, vagyis az ideális esetet kell minél jobban megközelíteni.

F1 pontszám

Az előző bekezdésben tárgyalt ideális eset megközelítésének ellenőrzésére elemezhető a 12.1 melléklet (d) görbe, mely az F1 pontszámot ábrázolja. Az F1 pontszám a modell pontosságának értékelésére használható. Az F1 érték adott küszöbérték mellett a precizitás és a szenzitivitás harmonikus átlaga. A magas F1-pontszámok azt jelzik, hogy a precizitás és a szenzitivitás meglehetősen kiegyensúlyozott. Mivel a görbe kiszámításában harmonikus átlagot veszünk, az alacsonyabb értékű precizitás, vagy alacsonyabb szenzitivitás érték esetén jelentősen csökken az F1 pontszám. Az ábrán látható görbéből leolvasható, hogy a pontossága 0.9-es konfidenciaszinten a 0.9-es értékű. Továbbá leolvasható, hogy a legnagyobb, 0.92-es F1 érték 0.596-os konfidenciaszinten van. Ez az érték viszonylag magas, az elvárt detektálási feladatokat (megfelelő bemeneti körülmények mellett) jól teljesíti.

AP és mAP50-95

Az AP (Average Precision) a precision-recall görbe alatti területet számítja ki, és egyetlen értéket ad, mely összegzi a modell teljesítményét. Az mAP (mean Average Precision) az AP kiterjesztése, amely több objektumosztályon átlagolja az AP értékeket. A metrika átfogóan értékeli a modell teljesítményét több osztály klasszifikálásában. Fontos megemlíteni a IoU-t (Intersection over Union), mivel az mAP50 és mAP50-95 kiszámításában szerepet játszik. Az IoU egy olyan mérőszám, amely a predikált határolódoboz és a valós (ground truth) határoló doboz közötti átfedést számszerűsíti. Az IoU érték létfontosságú szerepet játszik az objektumok lokalizációjának pontosságának értékelésében. Az mAP@50-95 érték a 0.50-től 0.95-ig tartó IoU küszöbértékek közötti osztályonkénti AP átlagokat veszi, 0.05-ös lépésközönként. A 12.1 melléklet (g) görbe jeleníti meg a 100 epoch tanítás során tanítási ciklusonként mért mAP értékeket 0.50-es IoU küszöbértéknél. A 12.1 melléklet (h) görbe pedig a ciklusonként mért mAP50-95 értékeket mutatja. A modell a tanítás után mAP50 0.94-es és mAP50-95 0.93-as értékkel rendelkezik. Ez a mérőszám bemutatja, hogy a modell különböző, 0.50-0.95-ig tartó IoU küszöbértékeknél jól teljesít, mely elengedhetetlen a megfelelő HTML kódok elkészítéséhez. A 12.1 melléklet (e) görbén a 100 epoch utáni 0,923-as értékű precizitás, a 12.1 melléklet (f) görbén pedig a 100 tanulási ciklus utáni 0,926-os értékű szenzitivitás látható.

5.5 A modell vesztesége

A veszteségfüggvények döntő szerepet játszanak a modell tanulásának folyamatában [19]. Számszerűsítik, hogy a modell előrejelzései mennyire igazodnak a tényleges eredményekhez, gyakorlatilag az előrejelzési hibák költségét mérik. Irányítják a modell tanulási folyamatát, befolyásolva annak képességét, hogy pontos eredményeket produkáljon. A veszteségfüggvények az előrejelzett és a tényleges kimenetek közötti különbséget értékelik. A kisebb veszteségérték a modell jobb előrejelzéseit jelzi. Ezeket a függvényeket a képzés során olyan algoritmusok segítségével optimalizálják, mint például a Gradient Descent, amely iteratív módon állítja be a modell paramétereit a veszteség minimalizálása érdekében. A YOLO által jelenleg használt veszteségfüggvények nem teljesen publikusak, de a YOLO megalkotásáról szóló cikk alapján a veszteségfüggvényeit részletesebben, matematikai megközelítésben a 4.3.2 alfejezet tárgyalja. A következő bekezdés a tanulási fázis train és validációs adathalmazba tartozó elemein, tanulási ciklusonként mért eredményeiből ábrázolt veszteségfüggvényeit elemzi.

Box loss

A train box loss a tanító adatokon szereplő objektumok előrejelzett határolódobozai és a tényleges határolódobozok közötti eltérést méri. Minél nagyobb az előrejelzett határolódoboz és a ground truth közti átfedés (IoU), annál kisebb a box loss. Az alacsonyabb box loss a modell által megjósolt határolódobozok nagyobb pontosságát jelzi.

Class loss

A class loss biztosítja, hogy minden észlelt objektum helyesen kerüljön besorolásra az előre meghatározott osztályok valamelyikébe. Méri, hogy a modell mennyire pontosan tudja klasszifikálni az adott detektált objektumot a megfelelő osztályba. Minél alacsonyabb a class loss, annál pontosabban azonosítja a modell a megfelelő osztályokat.

Dfl loss

A Dfl loss-t (Distributed Focal Loss) a határolódoboz előrejelzéseinek finomítására használja a modell, különösen a hasonló vagy nehezen megkülönböztethető objektumok esetében. Ez a Focal Loss egy változata, mely a nehezen osztályozható példákra való összpontosítást fokozza. A kereszt-entrópia veszteségegyenletet egy olyan tényező hozzáadásával módosítja, amely csökkenti a könnyen megkülönböztethető mintákra fordított figyelmet és növeli a nehéz példákra fordítandó figyelmet. A Dfl loss változtatása segít a modellnek jobban összpontosítani a nehéz vagy kétértelmű esetekben.

A tanításról alkotott loss görbék

A 12.2 melléklet grafikonjai szemléltetik az előző bekezdésekben tárgyalt loss metrikákat. A felső sorban a tanító adathalmaz train adatain mért box loss (a), class loss (b) és dfl loss (c) görbéi, míg az alsó sorban ugyanezen típusú görbék validációs adathalmazon mért változata jelenik meg. A felső sorban ábrázolt függvényekből látszik, hogy a modell a tanulási ciklusonként egyre kisebb veszteséggel funkcionált, míg nem az utolsó epoch során elérte a végleges állapotát. Az alsó sorban ábrázolt függvények vizsgálata során észrevehető, hogy a modell a tanulási fázis 50. ciklusától kezdte el a veszteségeit a minimumra csökkenteni. A mért eredmények alapján elmondható, hogy a tanítás jól működött. A train adathalmazon mért veszteség folyamatosan csökken, ami a modell illeszkedését mutatja a tanulási adatokhoz, emellett a validációs adatokon mért veszteség is folyamatosan csökken, mely a modell új, ismeretlen mintáin való jó teljesítményét mutatja. A veszteségeket kirajzoló grafikonok alapján az overfitting (túltanulás, túlzottan

a tanítóhalmaz elemeihez való illeszkedés) jelensége nem áll fenn. A grafikonok végső veszteségértékeit az 5.2 táblázat foglalja össze.

train/cls	train/box	train/dfi	val/cls	val/box	val/dfi
0.14118	0.14602	0.80761	0.41871	0.14608	0.49317

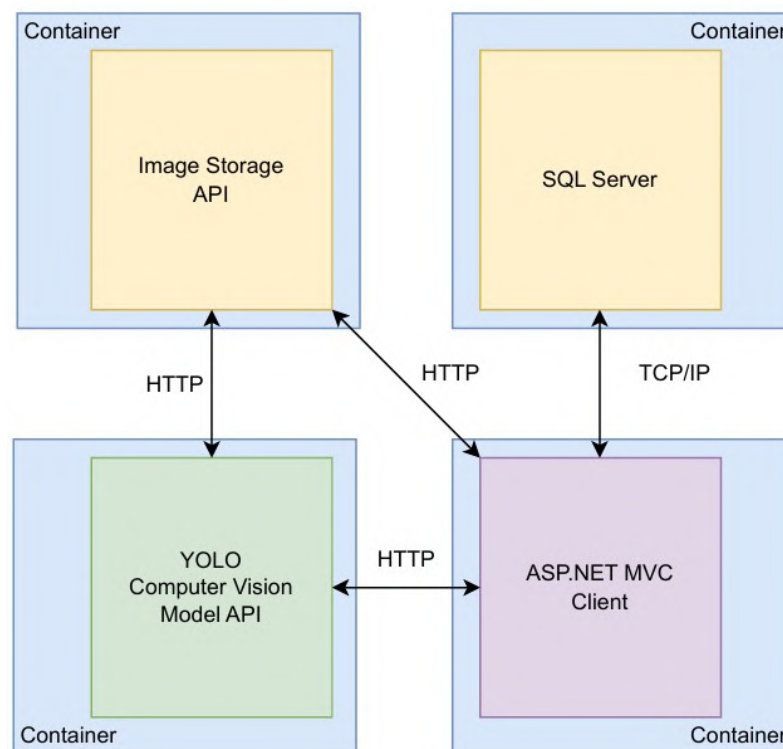
5.2. táblázat: A tanított modell veszteségei 100 epoch után.

6 SZOFTVER ARCHITEKTÚRA

Röviden összefoglalva a program lényege, hogy egy rajzról készített kép alapján képes a bemeneti elrendezést reprezentáló HTML kód generálására, amit a felhasználó már használhat is projektjében.

6.1 A szoftver architektúrája

A szoftver különböző szolgáltatásokat nyújtó, konténerizált mikroszervizekből épül fel (6.1 ábra). A mikroszervizek API-val kommunikálnak. A konténerizáció által a szoftver bármilyen operációs rendszerű eszközön működik, amelyen telepítve van a Docker Engine. A konténerek Linux alapúak.



6.1. ábra: A szoftver 4 fő egysége.

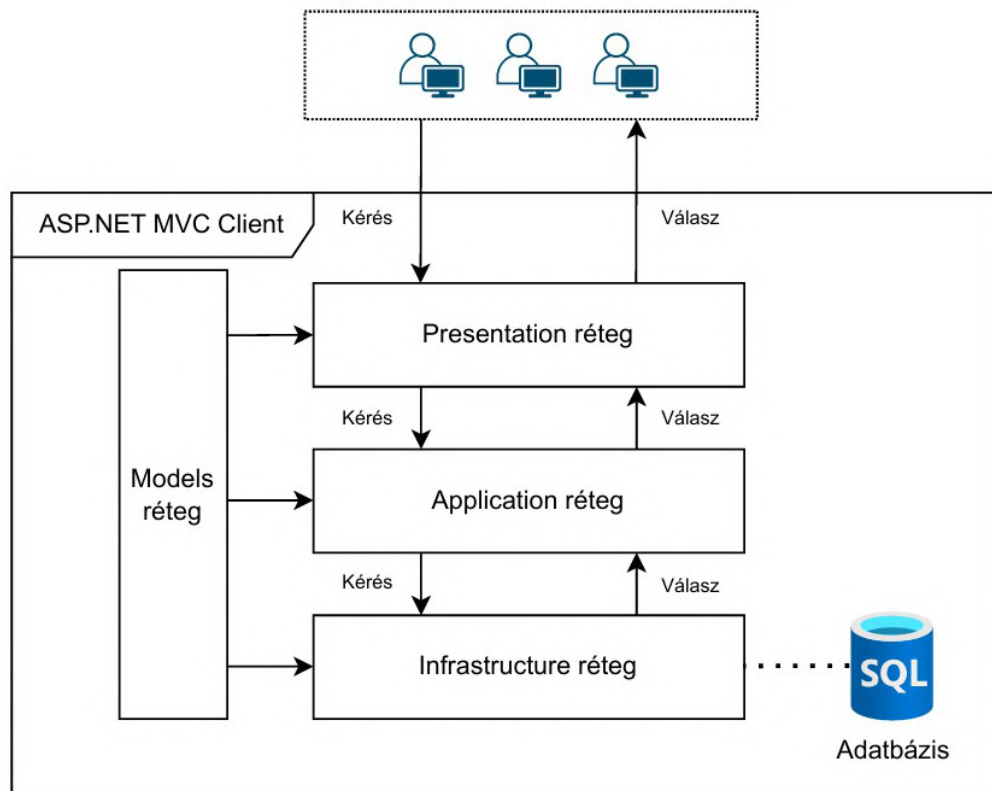
6.1.1 ASP.NET MVC Client

A felhasználói felület egy webalkalmazás. Ez biztosítja a kép feltöltésének lehetőségét, a gépi látás modell kezelését, a generált kódok és előnézet megtekintését. A konténer a prediktált határolódobozokkal ellátott képek elérésére az Image Service-szel, a detektálási adatok, sebességek és előzmények beszerzése érdekében az SQL szerverrel kommunikál.

A bemeneti képet elküldi a YOLO Computer Vision API-nak, majd a válaszból összeállítja a HTML kódot és az előnézetet. Az előnézethez személyre szabási felületet biztosít. A HTML kódot összeállító algoritmust a 6.1.1. alfejezet ismerteti.

A program struktúrája

A program egy 3 rétegű architektúrán alapul, melyet a 6.2 ábra mutat be. Az Infrastructure réteg felel az adatbázissal kapcsolatos CRUD műveletek lebonyolításáért. Az Application rétegben található a DetectService komponens, mely a detektálás megvalósításáért és a HTML kód összeállításáért felel. A Presentation réteg MVC formában valósítja meg a felhasználó számára a megjelenítést és a beavatkozási lehetőségeket. A Models réteg megosztása biztosítja a model objektumok bármely rétegben történő szabad felhasználását, úgynevezett cross-cutting concern-ként.



6.2. ábra: Az ASP.NET MVC Client rétegzett felépítése.

HTML kód generáló algoritmus

Az 1. algoritmus a program Application rétegében, DetectService komponensében megírt algoritmus működését egyszerűsítve szemlélteti. Ezt követi az algoritmus magyarázata, részletesebb ismertetése.

1. Algoritmus CreateHtml fő fázisai

Bemenet: **detections** – Dictionary (lista felismert UI elemekkel), **acceptedRange** (elfogadott tartomány értéke)

Kimenet: **htmlStructure** – string (megalkotott HTML kód)

1. Hozzuk létre a **colSorted** Dictionary-t
 2. Rendezzük a **detections** elemeit **y1** szerint növekvő sorrendben
 3. Hozzunk létre sorokat az **y1** tartomány alapján:
Csoportosítsuk az elemeket, melyek **y1** értékei az **acceptedRange** tartományon belül vannak, és adjuk hozzá a **rowSorted** listához.
 4. Minden sort rendezzünk **x1** szerint.
 5. Hozzunk létre oszlopokat:
Csoportosítsuk a sor elemeit az **x** koordináták alapján, és adjuk hozzá **colSorted** listához.
 6. Rendezzük az oszlopokat **y1** szerint a sorokban.
 7. Iteráljunk végig a **colSorted** lista elemein, majd az elemek típusa és listában elfoglalt helye szerint építsük fel a HTML kódot.
 8. Térjünk vissza a **htmlStructure** stringgel, mely a HTML kódot tartalmazza.
-

HTML kód generáló algoritmus leírása

A kódgeneráló algoritmust a szerveroldal üzleti logikai rétege tartalmazza. A Computer Vision Model API-tól válaszként kapott JSON üzenet kellő, azonosított osztályokat és határolódobozuk koordinátáit (x_1, x_2, y_1, y_2 a 12.3 melléklet (d) képen látható 4 koordináta zölddel megjelölve) tartalmazó részét Dictionary-ben tárolja. Első lépésként a Dictionary elemeit a függőleges tengely mentén úgy rendezi, hogy a UI elemek az y_1 koordinátájuk szerint fentről lefelé haladjanak az elrendezésen, vagyis a lista elejére kerül a potenciális legfelső, de legalábbis felső részen szereplő, a lista utolsó helyére pedig a legalsó részen helyet foglaló elem. Ezt követően a lista elemein végigiterál. Az iteráció célja az elemek elrendezésnek megfelelő sorába való elhelyezése (az algoritmus egy soron belüli elemekre HTML class="row" attribútumot használ, a megegyező sorban szereplő elemeket megegyező listába teszi). A 12.3 melléklet (d) képen pirossal bekeretezve látható, hogy mit értünk egy sor alatt. A sor magasságát mindig a legmagasabb elem határozza meg. A rajzokat tartalmazó előfeldolgozott lap mérete alapján számít egy, az elhelyezéshez szükséges tűréstartományt. Minden elem y_1 és y_2 koordinátáját megnézi. Az y_2 koordinátát az adott sorhoz tartozó magasságként állítja be. Ezt az értéket (vagyis a sor magasságát) akkor változtathatja meg egy másik elem, ha az ugyanabban a sorban szerepel, mint az elem, mely eddig meghatározta azt. Ennek ellenőrzésére kell a tűréstartomány, ugyanis segítségével megállapítható az, hogy az adott elem y_1 koordinátája beleesik-e a sorhoz tartozó mini-

mális és maximális érték közti tartományba. Ha a tartományban szerepel, akkor az adott sorhoz adja, ha pedig nem, akkor további ellenőrzést végez. Ekkor megállapítja, hogy az elem az adott sor eleme, annak leellenőrzésével, hogy az $y1$ koordinátája a sor magasságát adó érték és a legkisebb érték közé esik. Ha ez teljesül, akkor a sorhoz adja, egyébként új sorba sorolja az elemet. Jelenleg olyan módon rendezett listát tartalmazó listánk van, melyben a lista elemei az egy sorba tartozó elemek. Ezután az $x1$ koordináták alapján sorrendbe rendezi őket. Ezt követően a sorokban kiszámításra kerülnek az oszlopok, és az oszlopok sorai, ha vannak. A számítás az elemek szélességének felét adó koordinátával történik. Ha ez a koordináta beleesik egy másik elem $x1$ és $x2$ által nyitott tartományába, akkor új sorba kerül, ellenkező esetben új oszlopba. Az új oszlop a 12.3 melléklet (d) képen kékkel keretezve látható. Utolsó lépésként a sorokban szereplő oszlopok UI elemeinek $y1$ koordináta szerinti rendezése történik, hogy helyes sorrendben rendeződjének az elemek. Ebből a többdimenziós listából alkotható ki a HTML kód.

6.1.2 YOLO Computer Vision Model API

A szoftver objektumdetektáló része, Python nyelven implementálva. Ezen egység tartalmazza a gépi látás modellt, amely segítségével információt nyer ki a képről. Itt történik a betöltött kép feldolgozása, vagyis háttér és szükségtelen részletek eltávolítása, majd az eredményül kapott kép modell általi vizsgálata. Ezen megállapított információkat a kliensről érkezett API kérés válaszként JSON üzenetben továbbítja a kliens felé. Az általa kinyert adatok alapján történik a HTML kód generálása.

6.1.3 Tárolók

Image Storage API

Egy API-n keresztül használható fájlrendszer Python nyelven implementálva, amely a bemeneti és módosított képek tárolásáért felel. Ebből a konténerből érhető el a kliens számára a betöltött, előfeldolgozott és a detektált kép.

SQL Server

A relációs adatbázisban tárolja a rendszer a YOLO Computer Vision mikroszerviztől válaszul kapott információt. Ezen információk közé tartozik az előfeldolgozás sebessége, detektálás sebessége, detektált alakzatok osztálya és koordinátái, illetve a válaszból összeállított formázatlan HTML kód.

6.2 Architektúra összefoglalása

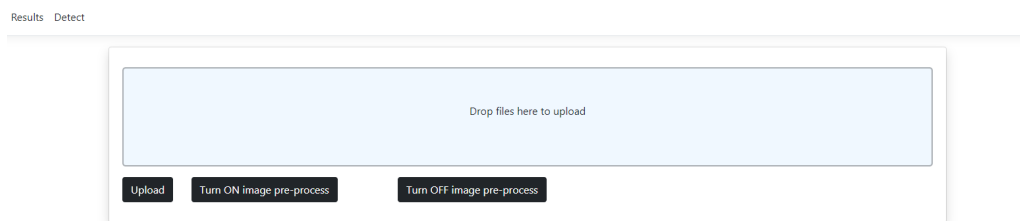
A külön microservice-ek által megvalósított architektúra követi a maximális szeparáltság elvét. A teljes program összes funkciója szét van választva, mely az egységek logikai szétválasztásának eredménye. Ennek köszönhetően bármelyik modul cserélhető, és nincs programnyelvhez kötve. A kliens szoftver nagyon egyszerű architektúráját a felhasználási lehetőségeinek szűk tartománya okozza. Mivel kevés, egymástól jól elszeparálható funkcióval rendelkezik, nem indokolt bonyolultabb architektúrát követni. A monolitikus felépítés az ehhez hasonló kis kódbázissal rendelkező alkalmazásoknál könnyen kezelhető.

7 A SZOFTVER

A fejezet a dolgozatban ismertetett, kutatás alapján megvalósított szoftvert mutatja be. A program a rendszerterv szerint épül fel, és az ott tárgyalt egységek valósítják meg a működést.

7.1 Az alkalmazás grafikus felülete

A grafikus felületen keresztül (7.1 ábra) feltölthető a kézzel rajzolt elrendezés. A drag and drop felületre húzva, majd ott elengedve, vagy a kék területre kattintással lehet a bemeneti képfájlt feltöltésre kiválasztani. Itt lehet beállítani, hogy történjen-e előfeldolgozás, vagy sem. Az "Upload" gombra kattintva elkezdődik a feltöltés, majd elindul a detektálás folyamata.



7.1. ábra: A webalkalmazás feltöltő felülete és gombjai.

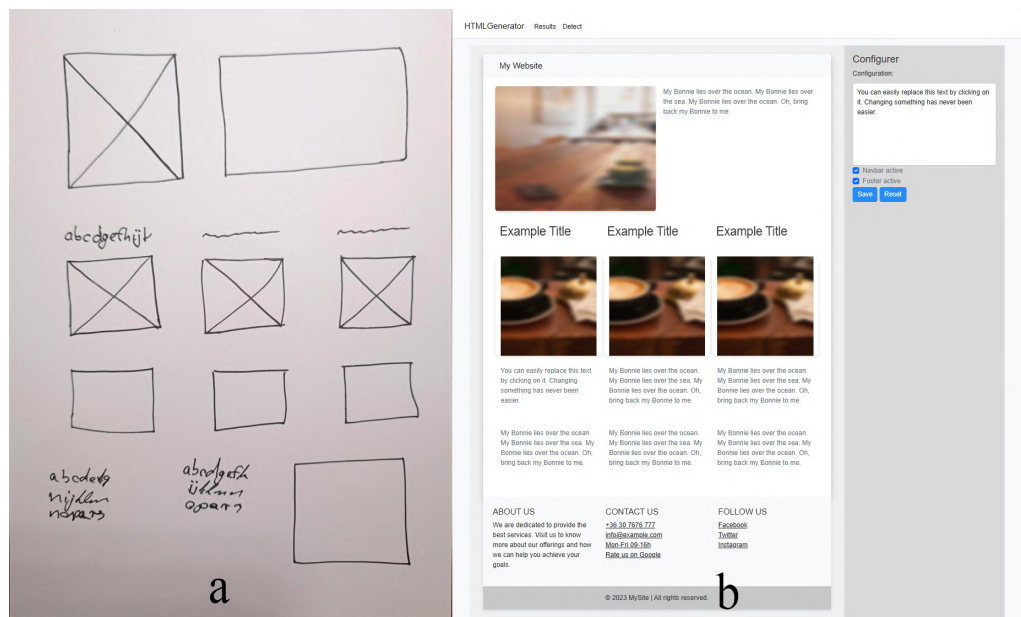
A detektálás eredménye

A program a detektálás eredményeként a HTML kód formázatlan és formázott változatát jeleníti meg (12.4 melléklet). A kódok egy gombnyomással vágólapra másolhatók. A webalkalmazás a HTML kódok mellett a "Get Preview" gombra kattintva biztosítja az felismert elrendezés előnézetét és konfigurálási lehetőségét (7.2 ábra). A felület navigációs sávjában elérhető "Results" (7.3 ábra) menüpontban pedig az eddig feltöltött képek alapján történt előfeldolgozásról, detektálásról találhatunk bővebb információt. Ezen információk közé tartozik a képek folyamatonkénti előnézete, a feldolgozás és detektálás ideje milliszekundumban. Továbbá innen is elérhető a korábbi detektálások eredménye, és a konfigurációs nézet is megnyitható.

Az előnézet és konfigurálási lehetőség

Az előnézetre való navigáláskor megjelenik az elrendezés, melyet alap szinten személyre szabhatunk. Az előnézeten adott elemre kattintva változtatható annak tartalma. A template képek és szövegek saját tartalomra cserélhetők. A textbox-ban szereplő szöveg cseréje

érdekében rá kell kattintani a cserélendő szövegre, majd az alap konfigurációs panelen elvégezni a módosítást. Header és footer hozzáadására is van lehetőség, mely tartalmát szintén személyre lehet szabni a konfigurációs felületen keresztül. A 7.2 ábrán látható egy teszt bemenet felismerése és digitális UI felületté alakítása. Az ábra (b) képen látható a bemeneti rajzból generált, személyreszabható elrendezés. A kép elemek véletlenszerű, homályosított képekkel vannak helyettesítve, a szövegek pedig példa szövegekkel. A felhasználó módosításával ezek egyszerűen változtathatók. Ha elkészült a felhasználó, egyszerűen lementi a kódot, és egy szerverre feltöltve már meg is jelenítheti azt a világhálón. A 12.5 és 12.6 mellékletek további példákat ismertetnek.



7.2. ábra: A bemeneti kép (a), az elrendezés előnézete és konfigurációs felülete (b).

Az előzmények oldal

Az előzmények oldal biztosít a szoftver működési folyamatába a képen végzett műveletekhez vizuális betekintést. Jól szemlélteti az egyes folyamatok eredményeit, a különböző fázisban bemutatott képek átláthatóságot biztosítanak. A Predicted címmel ellátott képről leolvashatók a detektálási bizonyosságok, továbbá vizuálisan észlelhető a modell teljesítménye. A 7.3 ábra jobb oldalán található információk pedig egyes mérőszámokat jelzik ki. A kép neve és detektálási ideje mellett az előfeldolgozási időt (mely tartalmazza a képek adattárba való mentését) és a detektáláshoz szükséges időt mutatja. A Show Preview gomb az elrendezés gyors előnézetét mutatja, a Configure gomb előző bekezdésben tárgyalt konfigurációs felületre navigál, a Show Code pedig a formázott HTML kódot mutatja, másolási lehetőséggel.

8 ÖSSZEFOGLALÁS

A dolgozatban kézzel rajzolt UI elrendezések rövid idő alatt HTML kóddá való átalkítására készítettem megoldást. A lerajzolt UI elrendezést a szoftver gépi látás modell használatával felhasználásra kész, webszerverre feltölthető kódként adja vissza. A szoftver bármilyen operációs rendszeren egyszerűen és gyorsan használható, melyen telepítve van a Docker Engine. Egyszerűségéből adódóan szaktudás nélkül is könnyen vezérelhető.

A megvalósítás során először a dolgozat témájához közel álló módszerek után kutattam, majd ezeket összevetve saját ötletemmel határoztam el, hogy a megoldást gépi látás bevonásával valósítom meg.

A dolgozat tartalmi sorrendje megegyezik az elkészített szoftver által végrehajtott lépések sorrendjével, vagyis a képfeldolgozás, gépi látással való detektálás, kódgenerálás sorrenddel. Ám első feladatként az objektumdetektáláshoz köthető tudományok közelebbi vizsgálata volt szükséges. A nem neurális hálózat alapú és a neurális hálózat alapú objektumdetektálási módszerek vizsgálata során választottam ki a YOLO-t, mint a célra legalkalmasabb gépi látás modellt. A modell számára szükséges tanító adathalmaz és bemeneti képek megfelelő előkészítésére a képfeldolgozás területét, mint szoftver szempontjából kulcsfontossággal bíró területet vizsgáltam. Elsőként az ehhez köthető, a projekt szempontjából fontos előfeldolgozási lehetőségeket vizsgáltam meg, majd kiválasztottam a műveleteket, melyekkel a gépi látás modell számára elő lehet készíteni a felhasználó által biztosított bemeneti képet.

A tanításhoz létrehoztam egy 20.000 képből álló címkézett tanító adathalmazt, ezen 100 tanítási cikluson keresztül tanítottam a modellt. A tanítás eredménye egy detektálásra jól használható, megfelelő szabályok betartásával rajzolt elrendezéseket nagy pontossággal felismerő modell lett.

Ezt követte a szoftver külön konténerizált egységeinek létrehozása. Implementálásra került a modell API-n keresztül vezérelhető változata, a kódgeneráló algoritmus, majd a szoftverhez szükséges vezérlő felület. Továbbá az előzmények tárolására létrehoztam a perzisztáló egységeket, az SQL adatbázist és a File Service-t. Eredményül egy működő rendszert kaptam, mely a projekt célját megvalósítja.

8.1 További célok

A további célok között a modell pontosítása és a képfeldolgozás magasabb minőségre való emelése szerepel, amelyek kulcsfontosságúak a rendszer hatékonyságának és megbízhatóságának fejlesztésében.

A pontosítás érdekében egy nagyobb, változatosabb adathalmazra van szükség, mely a modell számára újabb ismeretek megszerzését teszi lehetővé. Ez a fejlesztés biztosítja a modell általános felismerési képességének javulását, megfelelő tanítási paraméterek mellett növeli a precizitását és a klasszifikáció helyességét. A jelenleg használt YOLO small modell helyett indokolt egy nagyobb kapacitású, modernebb YOLO modell alkalmazása. Az újabb, nagyobb méretű modellek gyorsabbak és pontosabbak, ezen tulajdonságok fontosak a hatékonyság fokozásának szempontjából.

A képfeldolgozás sebessége és minősége fokozható (ahol lehetséges) GPU-ra való átállással, esetlegesen új képfeldolgozási algoritmusok felhasználásával.

Továbbá új célként szerepel a funkciók bővítése, az alap konfigurációs felület fejlesztése, pontosítása. A jelenlegi konfigurációs felület egy alap módosítási lehetőséget biztosít a kiválasztott elemekhez, a későbbiekben a funkcióinak fejlesztése elengedhetetlen a felhasználói élmény fokozása érdekében. A konfiguráció részeként akár egy további, gépi tanulással kiegészített egység is szerepet kaphatna, mely a szélesebb skálájú személyre szabhatóságot biztosítaná, különböző opciókat biztosítana az elrendezések kinézetéhez.

9 RÖVIDÍTÉSEK

AI	Artificial Intelligence
API	Application Programming Interface
CMS	Content Management System
CRUD	Create, Read, Update, Delete
CSS	Cascading Stylesheet
DFL	Distributed Focal Loss
DPM	Deformable Part Models
GT	Ground Truth
HOG	Histogram of Oriented Gradients
HTML	Hypertext Markup Language
IoU	Intersection over Union
JSON	JavaScript Object Notation
MVC	Model-View-Controller
NMS	Non-Maximum Suppression
R-CNN	Regions with Convolutional Neural Networks
SQL	Structured Query Language
SVM	Support Vector Machine
UI	User Interface
UWP	Universal Windows Platform
XAML	Extensible Application Markup Language
YOLO	You Only Look Once
mAP	Mean Average Precision
SSD	Single Shot Multibox Detector

10 ÁBRAJEGYZÉK

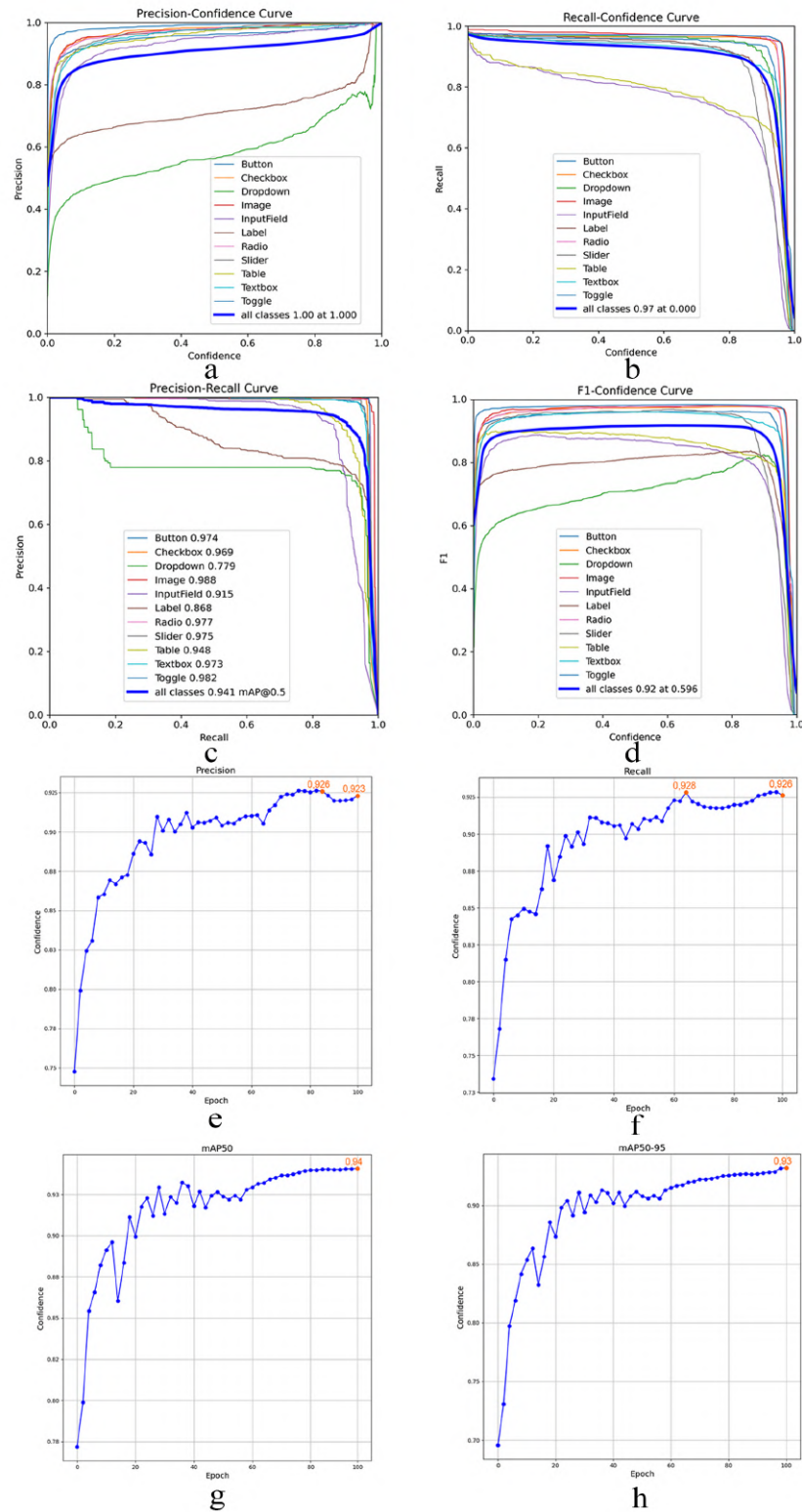
1.1. A program ismertetése	2
2.1. Háttér leválasztása	6
2.2. Papírra rajzolt elrendezés feldolgozása	7
4.1. HOG jellemzők kinyerése	15
4.2. Egykomponensű személymodell találatai	17
4.3. A detektált emberi alak és hozzátartozó részei	17
4.4. Régiójavaslatok	19
4.5. YOLO detektálási folyamat	20
4.6. A YOLO architektúra	21
4.7. YOLO11 teljesítmény	25
5.1. UI elemek és eloszlásuk a tanító adathalmazban	27
6.1. A szoftver 4 fő egysége	33
6.2. A kliens rétegzett felépítése	34
7.1. A webalkalmazás feltöltő felülete	38
7.2. Elrendezés előnézete	39
7.3. Előzmények	40
12.1. Modell teljesítménye	47
12.2. Modell vesztesége	48
12.3. Program különböző fázisai	49
12.4. Szolgáltatott kód	50
12.5. Elrendezés példák I.	51
12.6. Elrendezés példák II.	52

11 IRODALOMJEGYZÉK

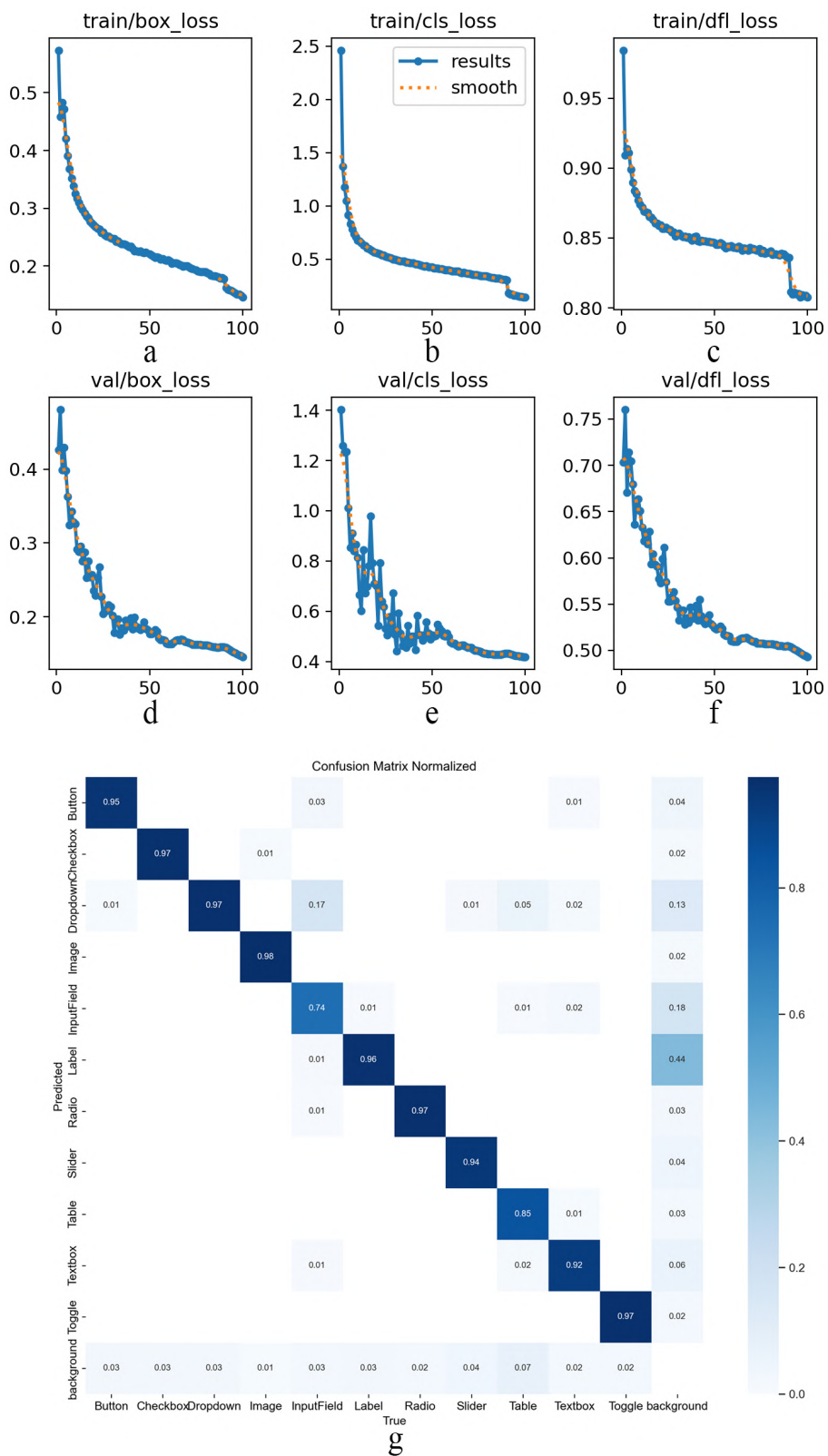
- [1] M. A. Lab. „Sketch2Code”. (2024. ápr. 15.), cím: <https://www.microsoft.com/en-us/ai/ai-lab-sketch2code> (elérés dátuma 2024. 04. 15.).
- [2] M. Sonka, V. Hlavac és R. Boyle, *Image Processing, Analysis, and Machine Vision* (Fourth Edition sor.). Cengage Learning, 2008, 106. old.
- [3] P. Kornprobst, J. Tumblin és F. Durand, „Bilateral Filtering: Theory and Applications”, *Foundations and Trends in Computer Graphics and Vision*, 4. évf., 1–74. old., 2009. jan. DOI: 10.1561/06000000020.
- [4] R. C. Gonzalez és R. E. Woods, *Digital Image Processing* (Fourth Edition sor.). Pearson Education, 2007, 701–702. old.
- [5] J. Canny, „A Computational Approach to Edge Detection”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8 évf., 6. sz., 679–698. old., 1986. DOI: 10.1109/TPAMI.1986.4767851.
- [6] S. Suzuki és K. be, „Topological structural analysis of digitized binary images by border following”, *Computer Vision, Graphics, and Image Processing*, 30. évf., 1. sz., 32–46. old., 1985, ISSN: 0734-189X. DOI: [https://doi.org/10.1016/0734-189X\(85\)90016-7](https://doi.org/10.1016/0734-189X(85)90016-7).
- [7] M. Mohri, A. Rostamizadeh és A. Talwakar, *Foundations of Machine Learning* (second edition sor.). MIT Press, 2018, 1. old.
- [8] M. Altrichter, G. Horváth, B. Pataki, G. Strausz és J. Valyon, *Neurális hálózatok*. Panem, 2006, 1–2. old.
- [9] P. Viola és M. Jones, „Rapid object detection using a boosted cascade of simple features”, *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, 1. köt., 2001, I–I. old. DOI: 10.1109/CVPR.2001.990517.
- [10] N. Dalal és B. Triggs, „Histograms of oriented gradients for human detection”, *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*, 1. köt., 2005, 886–893 vol. 1. DOI: 10.1109/CVPR.2005.177.
- [11] P. F. Felzenszwalb, R. B. Girshick, D. McAllester és D. Ramanan, „Object Detection with Discriminatively Trained Part-Based Models”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32. évf., 9. sz., 1627–1645. old., 2010. DOI: 10.1109/TPAMI.2009.167.

- [12] R. Girshick, J. Donahue, T. Darrell és J. Malik, „Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation”, *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, 580–587. old. DOI: 10.1109/CVPR.2014.81.
- [13] J. Redmon, S. Divvala, R. Girshick és A. Farhadi, „You Only Look Once: Unified, Real-Time Object Detection”, *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, 779–788. old. DOI: 10.1109/CVPR.2016.91.
- [14] Ultralytics. „Yolov8”. (2024. ápr. 15.), cím: <https://docs.ultralytics.com/> (elérés dátuma 2024. 04. 15.).
- [15] Ultralytics. „YOLO11 Performance”. (2024. nov. 5.), cím: <https://docs.ultralytics.com/models/yolo11> (elérés dátuma 2024. 11. 05.).
- [16] V. Pandian. „UISketch dataset”. (2024. máj. 15.), cím: <https://www.kaggle.com/datasets/vinothpandian/uisketch>.
- [17] T.-Y. Lin, M. Maire, S. Belongie és tsai., „Microsoft coco: Common objects in context”, *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, Springer, 2014, 740–755. old.
- [18] Ultralytics. „YOLO Performance Metrics”. (2024. nov. 1.), cím: <https://docs.ultralytics.com/guides/yolo-performance-metrics> (elérés dátuma 2024. 11. 01.).
- [19] Ultralytics. „YOLO Loss Metrics”. (2024. nov. 1.), cím: <https://www.ultralytics.com/glossary/loss-function> (elérés dátuma 2024. 11. 01.).

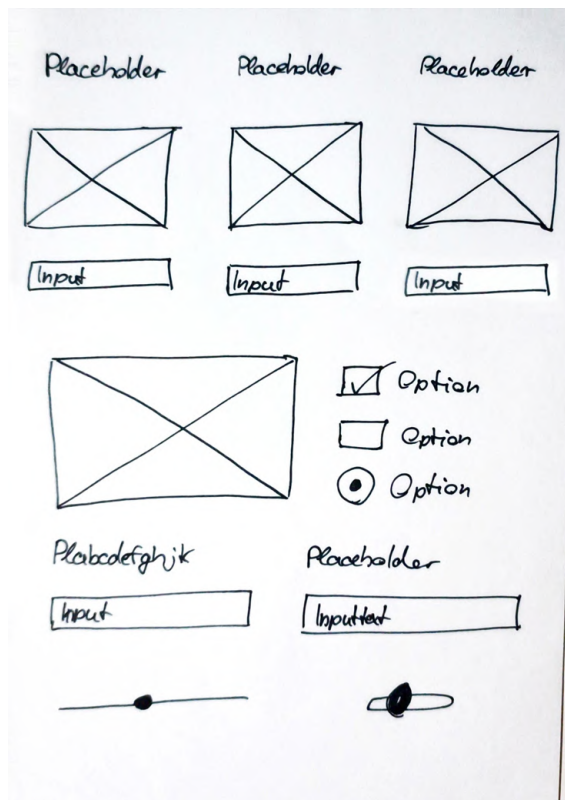
12 MELLÉKLETEK



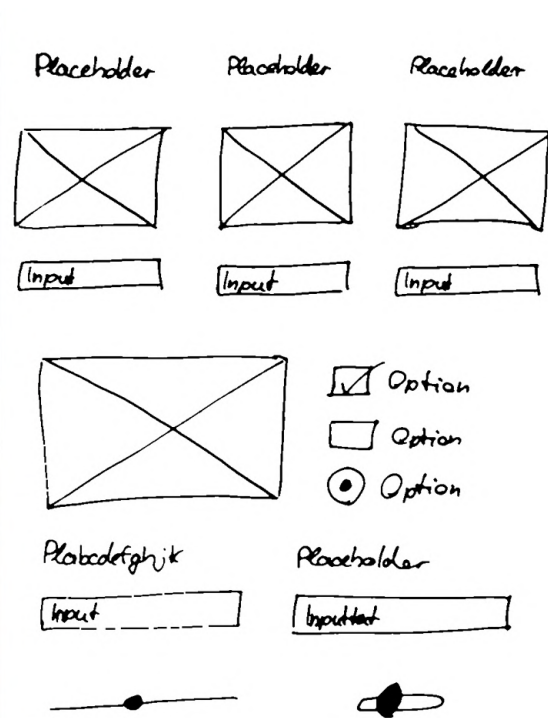
12.1. ábra: A modell 100 epoch utáni teljesítménye.



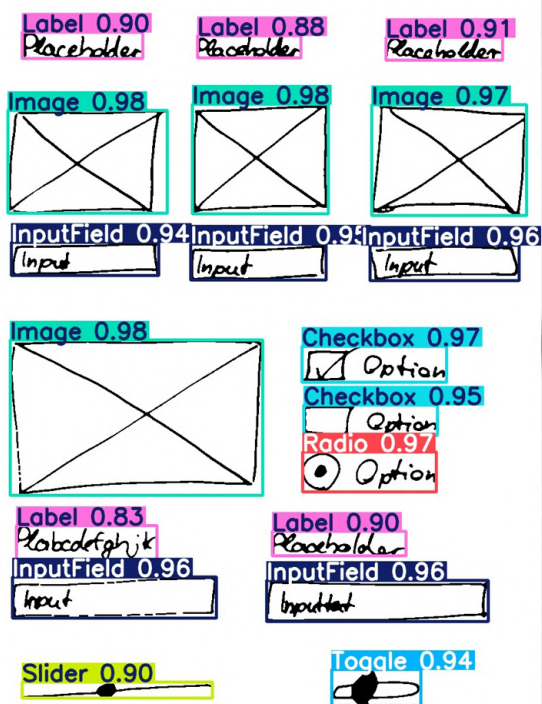
12.2. ábra: A tanítás közbeni, tanítóhalmazon keletkezett train/box_loss (a), train/class_loss (b), train/dfl_loss (c), majd a validációs adathalmazon keletkezett val/box_loss (d), val/class_loss (e), val/dfl_loss (f) és a normalizált confusion mátrix (g).



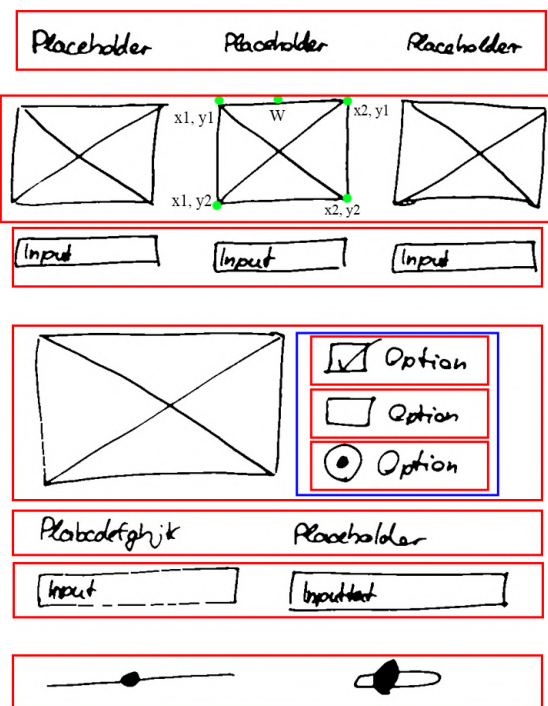
a



b

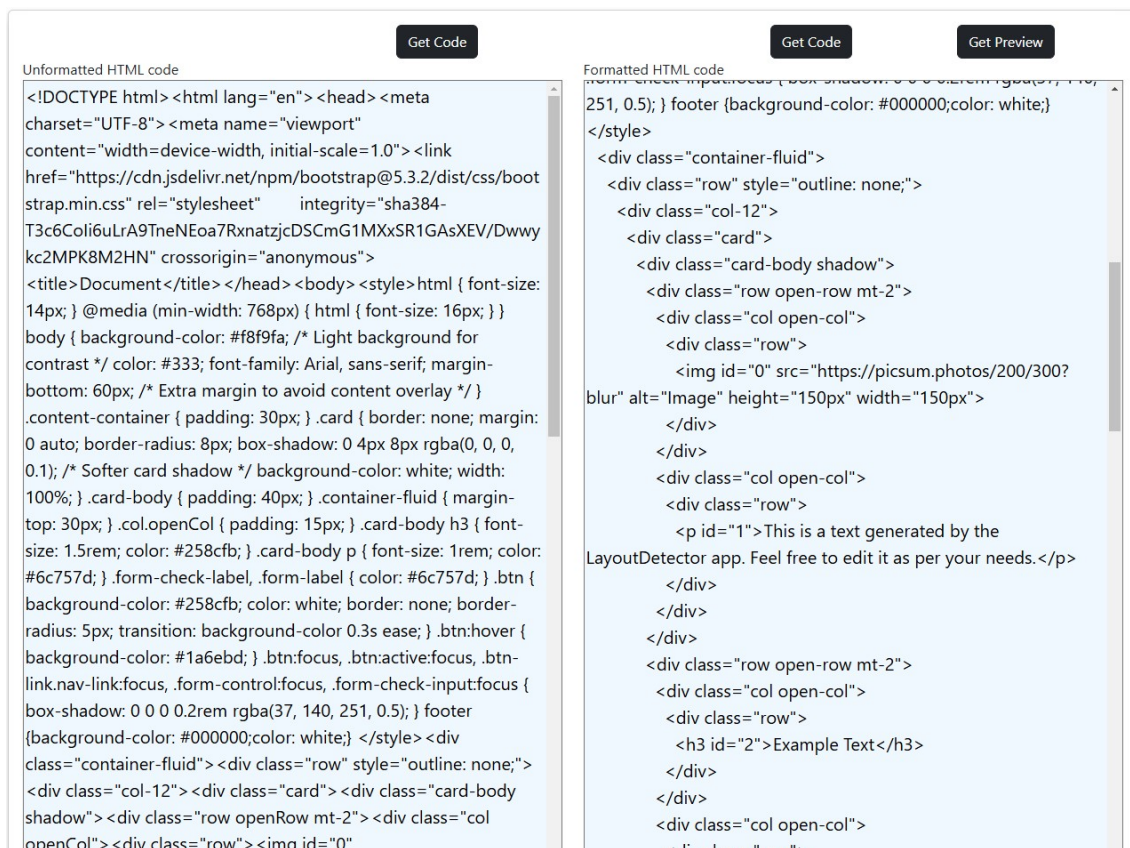


c

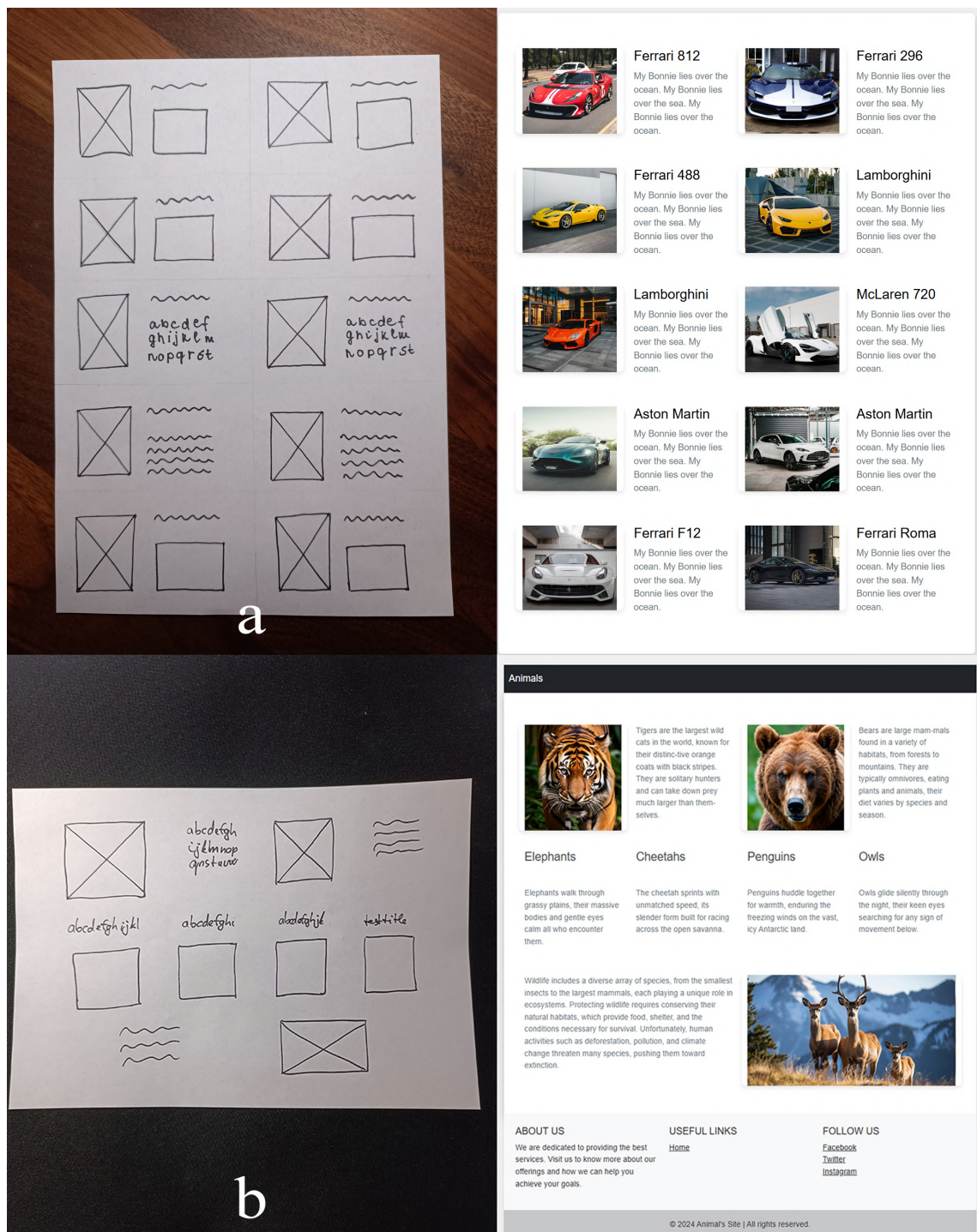


d

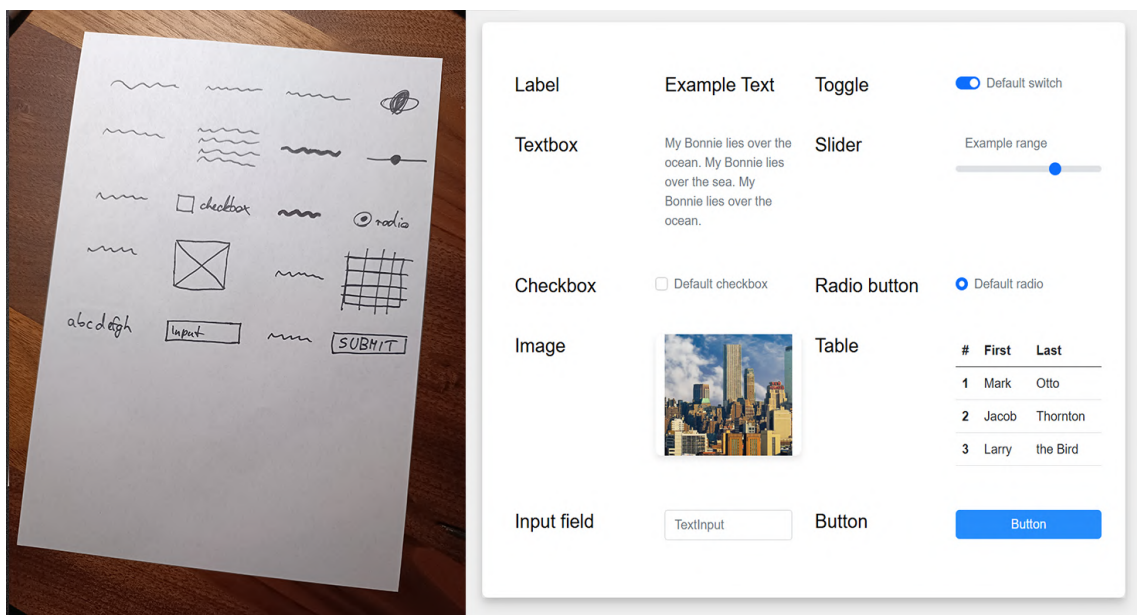
12.3. ábra: A program bemenet feldolgozásának különböző fázisai. A bemeneti kép (a), előfeldolgozott kép (b), detektált kép (c), a sorok és oszlopok megalkotásához vizuális segédlet (d).



12.4. ábra: A formázatlan és formázott kód, a feltöltött kép alapján



12.5. ábra: Bal oldalon a bemeneti kép, majd jobb oldalon egy webshop layout (a), alatta egy blog layout headerrel, footerrel (b).



12.6. ábra: Bal oldalon a bemeneti kép, jobb oldalon egy blog layout, mely ismerteti a program által felismerhető UI elemeket.