[Külső fekete borítólap formátuma]

Széchenyi István Egyetem
Gépészmérnöki, Informatikai és Villamosmérnöki Kar
Informatika Tanszék

SZAKDOLGOZAT

Józsa Dávid
Mérnök Informatikus BSc szak

[beadás éve]

[Gerincen:] Hallgató Neve, Évszám {Titkosított}

SZAKDOLGOZAT

Kapcsolási rajz digitalizáló szoftver

Józsa Dávid

Mérnök Informatikus BSc szak

[beadás éve]

Nyilatkozat

Alulírott, Józsa Dávid (R629Q7), mérnök informatika, BSc szakos hallgató kijelentem, hogy a Kapcsolási rajz digitalizáló szoftver című szakdolgozat feladat kidolgozása a saját munkám, abban csak a megjelölt forrásokat, és a megjelölt mértékben használtam fel, az idézés szabályainak megfelelően, a hivatkozások pontos megjelölésével.

Eredményeim saját munkán, számításokon, kutatáson, valós méréseken alapulnak, és a legjobb tudásom szerint hitelesek.

Győr, [beadás dátuma] _____

hallgató

Kivonat

Kapcsolási rajz digitalizáló szoftver

A szakdolgozat célja egy szoftver fejlesztése, amely egy képről, vagy élő kameraképről képes felismerni, valamint digitalizálni egy kapcsolási rajzot úgy, hogy az utána digitálisan szerkeszthető legyen.

A dokumentum első felében beszélek a kapcsolási rajzokról, valamint a szoftver elkészítéséhez szükséges szoftverfejlesztési eszközökről. Minden fejlesztési eszköz leírásában szó lesz arról, hogy hogyan működnek, mire használthatók, valamint arról, hogy miért hasznos ezen szoftver elkészítéséhez.

A második felében a program elkészítésének folyamatáról, valamint az egyes részfeladatok megoldásáról lesz szó, úgymint a bemenetek kezelése, a kép előfeldolgozása, szegmentálása, az elemek felismerése, valamint a kimenet létrehozása.

Abstract

Circuit diagram digitizer software

The goal of the dissertation is to develop a software which is able to recognize and digitize a circuit diagram from a picure or live video stream in a way that makes it editable.

In the first half of the document I talk generally about circuit diagrams and the development tools required to make the software. In every section about the development tools it will be discussed that how they work, what they can be used for and why are they useful for the making of this project.

In the second half I write about the process of making the software and how I solved the subprocesses like the input, preprocessing the image, segmentation, recognizing the elements and creating the output.

Tartalomjegyzék

1 Bevezetés

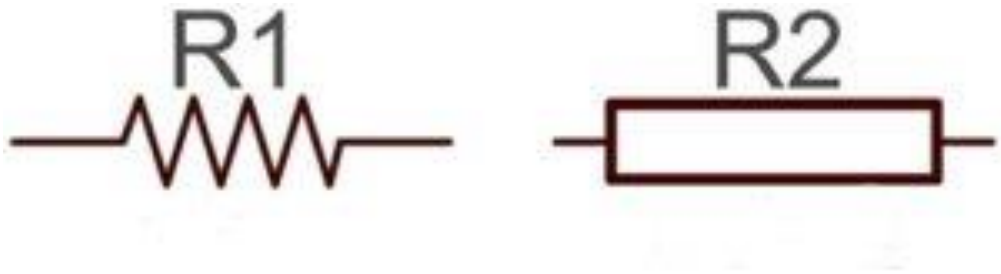
Elektronikai eszközök tervezésekor, vizsgálatakor gyakran találkozunk kapcsolási rajzokkal. Ezen rajzok, mint egy térkép, segítenek kiigazodni egy eszköz, vagy rendszer működésében, belső felépítésében. A kapcsolási rajzok egy nagyrészt egységes nyelvezetet biztosítanak, amely leírásnak köszönhetően egy adott rendszer könnyen újraépíthető, javítható. A szakdolgozat célja egy kapcsolási rajz digitalizáló szoftver elkészítése.

Kapcsolási rajzok gyakran fellelhetőek nyomtatott formában, egy kezelési útmutatóban, vagy rendszerleíró dokumentumokban. Ezek a dokumentumok régóta léteznek, azonban a számítógépes grafika fejlődésével és széleskörű terjedésével megszületett az igény a régebbi formátumú iratok, rajzok digitalizálására. A digitalizáció eleinte kézzel valósult meg. Ez a folyamat könnyíthető, valamint felgyorsítható a gépi látás segítségével. Már léteznek jól kifejlesztett technológiák a digitalizálásra, amelyek során egy nem szerkeszthető dokumentum vagy kép keletkezik, amely alkalmas az iratok megőrzésére, emberi elemzésére. Ennek egy fejlettebb változata, amikor a bemenetből több információt nyerünk ki, amely segítségével digitálisan szerkeszthetővé válik a bemeneti kép. Ebben az iratban egy ilyen automatikus képfeldolgozás folyamata van bemutatva, amely által egy bemeneti kép, vagy élő videón található nyomtatott áramköri rajzból egy digitálisan szerkeszthető file generálódik.

2 Elméleti háttér

2.1 Kapcsolási rajzok

A kapcsolási rajz egy elektromos áramkör vázlatrajza. Segítségével áttekinthető egy áramkör szerkezete, hogy milyen komponensekből áll össze, valamint azok milyen módon vannak összeköttetve. Ezeknél a rajzoknál a lényeg a komponenseken van, hogy mikből áll össze egy rendszer, hogyan vannak az adott eszközök egybekapcsolva. A komponensek elhelyezkedése általában eltér a rajzban lévőkhöz képest, azonban a kapcsolataik ugyan azok. Egy kapcsolási rajz által az abban leírt áramkör újra előállítható, valamint segít az adott áramkör megértésében, szükség esetén javításában.



1. ábra: Amerikai (bal), és európai (jobb) ellenállás szimbólumok
Forrás: https://circuitspedia.com/all-about-resistor/

A kapcsolási rajzok általában hasonló irányelveket követnek. Felépítésük, megjelenésük általában hasonló, azonban vannak stílusbéli eltérések. Ezen eltérések leginkább a komponenseket leíró szimbólumok különbségében mutatkozik meg. Létezik standard [8] a szimbólumok leírására, azonban ezt nem mindenhol követik teljesen, néhány helyen még régebbi, más standard szerinti írásmódot használnak, a szabványtól eltérő szimbólumokkal (1. ábra). Az ilyen eltérések miatt, valamint az esetleges félreértések elkerülése végett a későbbiekben meg lesz állapítva egy lista a szimbólumokról, amelyekre a szoftver fel van készítve.

2.2 A probléma bemutatása

A modern rendszerek leírására szolgáló kapcsolási rajzok általában már digitálisan eltároltak, azonban a régebbi rendszerek esetében sokszor csak a nyomtatott változat maradt meg. Egy nyomtatott kapcsolási rajz könnyen áttekinthető emberek számára, azonban nagyobb mennyiségben megnehezíti a munkát a keresés, valamint a rendszerezettség hiánya miatt. Ez akkor mutatkozik meg leginkább, amikor egy nagy rendszerről van leírás. Ilyen esetben általában a rendszert részenként, egységekre bontva kezelik, így nem egy, hanem sok, kisebb leírás és rajz keletkezik róla. Ilyen esetben a nyomtatott rajzok beazonosítása, valamint egyként kezelése nehézkes lehet. Továbbá, ha egy rendszer nem megfelelően működik, akkor a javításhoz szükséges a rendszer ismerete, amely megismeréséhez a kapcsolási rajzok segítenek. Olyan esetben, amikor sok rajz létezik egy rendszerről, a kézi keresése, valamint a részletek egyként kezelése megnehezíti, és lelassítja a munkát.

Ezen felül, ha a rendszerben változás történik, és nincs egy digitálisan szerkeszthető változat egy rajzról, akkor az egész rajzot újra el kell készíteni, az aktuális változtatások beiktatásával. Változtatások általában tervezési fázisban történnek, amikor még szerkeszthető a dokumentum, azonban változtatási igény keletkezhet később is. Ha egy tömeggyártott termék leírását vesszük alapul, igény a változtatásra keletkezhet a gyártás megkezdése után is, például, ha a termék valamilyen szempontból nem úgy működik, ahogy kéne, vagy bizonytalan a minősége a felépítése miatt, akkor a terméket újra kell tervezni, hogy kijavítsák a hibát a meglévő termékekben, vagy új sorozatot bocsátsanak ki. Egy másik példa az utólagos módosításra, ha egy régebbi termék új változatát adják ki, amelyben más komponenseket, vagy újabb funkciókat mutatnak be.

Ezen szituációk bekövetkezésekor előnyös, ha a termék belső felépítésének kapcsolási rajza elérhető szerkeszthető, valamint könnyen kereshető formában, azaz digitálisan. Amennyiben ez nem elérhető, akkor a régebbi, nyomtatott dokumentumok digitalizálására van szükség. Ez a digitalizáció eleinte kézzel valósult meg, ahol egy ember gépen újra elkészített egy már létező rajzot. Ilyen esetben hátrány volt az emberi pontatlanság, a rajz elemei nem biztos, hogy ugyanott lesznek a digitalizált változaton, mint a rajzon. Ez inkább csak a kinézetet befolyásolja, hiszen a lényeg az elemek közötti kapcsolatokban van. További hátrány, hogy ez egy relatívan lassú, valamint repetitív folyamat, és ha sok rajzot kell digitalizálni, az ember fáradékonysága hibákhoz vezet.

A gépi látás technológiák fejlődésével a hasonló digitalizációs folyamatok

automatizálhatóvá váltak. A gépi feldolgozásnak számos előnye van a kézi digitalizációhoz képest. A legnyilvánvalóbb előny a gyorsaság. A képfeldolgozás sokkal gyorsabban teljesíthető, azonban a bemeneti képek adagolása még mindig visszafoghatja az ilyen rendszereket. További előny, hogy a gép nem fárad, folyamatosan hasonló eredményeket produkál, vagy egy megfelelően konfigurált gépi tanulás megvalósítása által a felismerési hibák száma csökkenthető. Az ember általi digitalizációval szemben előnye továbbá, hogy a tudástára nagyobb lehet, valamint tetszőlegesen tovább bővíthető, azaz sokkal rövidebb idő alatt több információt képes kivonni a különféle bemenetekből.

A továbbiakban bemutatásra kerülnek azok a technológiák, valamint módszerek, amelyek felépítik a szoftvert, és lehetővé teszik a digitalizálást.

2.3 Képek előfeldolgozása

Ebben a fejezetben azokról az eszközökről lesz szó, amelyek segítségével egy bemeneti kép átalakítható, a gép számára feldolgozhatóbb formára. Ezekkel a módszerekkel még nem vonunk ki információt a bemeneti képből, csak átalakítjuk azt, hogy könnyebb legyen az információk kinyerése. A felismerés ezen szakaszában általában megtörténik a képeken található zajok, nem kívánatos elemek eltávolítása, a fontos elemek kiemelése, egyszerűsítése, elkülönítése a háttértől. Ideális esetben a képen csak a felismerés szempontjából fontosnak bizonyuló elemek maradnak.

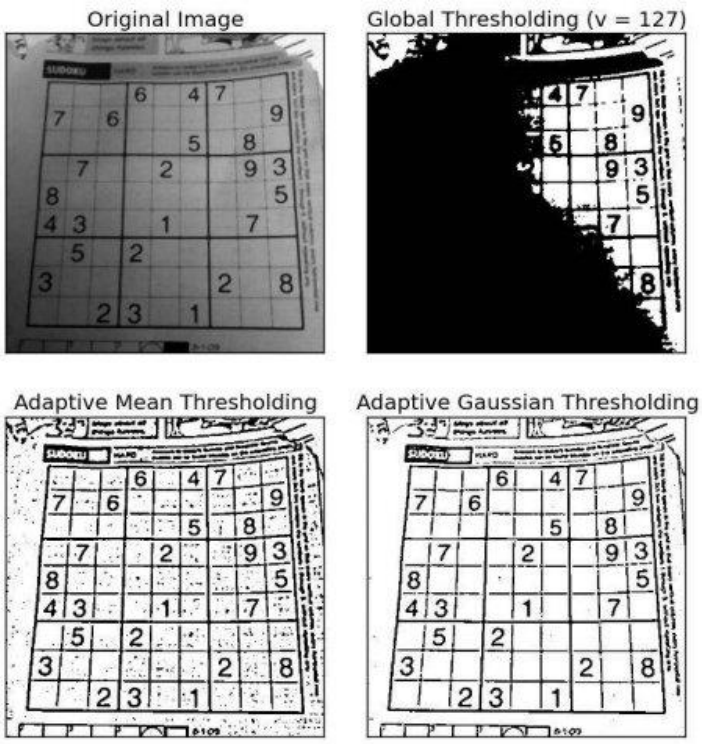
2.3.1 Thresholding

Az egyik legalapvetőbb képfeldolgozási metódus a küszöbölés (thresholding). Segítségével egy képen szétválaszthatjuk a hasznos, információt hordozó pixeleket a többi (általában háttér) pixelektől. Lefutása után az egyik részt fekete (alacsony intenzitású), a másikat fehér (magas intenzitású) pixelek fogják jellemezni, ezáltal binarizálva a képet. Azt, hogy melyik pixelekhez melyik intenzitást rendeljük, azt a küszöbölés típusa dönti el. A binarizálás előnye, hogy csökkenti az adatok komplexitását, ezáltal egyszerűbbé téve a további műveleteket. [3][7]

A leggyakoribb thresholding technikában egy paraméter, a küszöbérték (T) segítségével történik a szétválasztás. Ilyen esetben a bemeneti szürkeárnyalatos kép minden pixelje esetén, ha az a küszöbérték alatt van, akkor fekete lesz, ha felette van, akkor fehér. A metódus esetén értelemszerűen kritikus lépés a T megfelelő megválasztása. [3]

Egy másik változata az adaptive threshold, amely esetén a küszöbérték nem egy

konstans, hanem változó. Ennek az értéke egy pixel esetén a körülötte levő pixelek súlyozott átlaga alapján számítódik, mínusz egy konstans. A blokk mérete, amely megadja, hogy mekkora területen vizsgáljuk a pixelek szomszédjait, valamint a konstans paraméterként megadandó. A súlyozott átlag kiszámítása két módszer áll rendelkezésre. Az első az egyszerű átlagolás (mean thresholding), amely esetén minden pixel ugyanakkora súllyal számít az átlagba. A második a gaussian átlag (gaussian thresholding), amely esetén a pixelek a középtől való távolságuk alapján vannak súlyozva. [3]



2. ábra: Thresholding technikák
Forrás: https://docs.opencv.org/3.4/d7/d4d/tutorial_py_thresholding.html

Az adaptive threshold akkor mutat előnyt a szimpla thresholding ellenében, amikor a binarizálandó kép megvilágítása nagy mértékben egyenletlen. Ilyen esetben, ahogy a 2. ábrán is látható, az egyszerű küszöbölés esetén a kép rosszul megvilágított része olyan sötét, hogy a háttér intenzitása is áteshet a küszöbértéken, és az egész megvilágítatlan területet hasznos pixelcsoportként kategorizálja. Az adaptive változat ilyen esetben sokkal jobb eredményeket ad, mivel a megvilágított területen a pixelek a környezetükhöz képest még mindig elkülöníthetők. Ilyenkor a képtől függően keletkezhet pontszerű zaj, azonban kevesebb információt veszíthetünk el a folyamat által. [3]

2.3.2 Morfológiai transzformációk

A morfológiai transzformációk olyan műveletek, amelyek célja a képen esetlegesen jelenlévő hibák, tökéletlenségek javítása. Általában binarizált képeken alkalmazzák, de használható szürkeárnyalatos képeken is. A transzformáció a strukturális elem (structuring element) segítségével hajtja végre műveleteit. A strukturális elem egy kernel, amely alakja, mérete szabadon választható, és van egy rögzítési pontja (anchor point). Általában a kernel négyzet alakú, és a középpontja az anchor pont A kernel végig szkenneli a képet, és a műveletnek megfelelően beállítja a kiválasztott anchor pontot a megfelelő értékre. A két alapvető művelet az erózió és a dilatáció. [3]

Dilatáció (dilation) esetén, amikor a kernel végigfut a képen, minden állomásnál megkeresi a kernel által lefedett pixelek közül a legnagyobb értékűt, és az anchor pontot beállítja arra az értékre. Ennek következtében a világos területek a képen növekednek. [3]

Az erózió (erosion) a dilatáció ellentéte. Ebben az esetben nem a maximumot, hanem a minimumot keresi a kernelben, és arra állítja a pixelt. Ennek következtében nem a világos, hanem a sötét területek növekednek meg. [3]

Egy olyan képen, amelyen a háttér fekete, és a hasznos elemek fehérek, az erózió szűkíti a hasznos területeket, alakzatokat, míg a dilatáció növeli azokat. A két módszer egymás fordítottja, azonban egymás után alkalmazva a kettőt, ugyanazon a képen, nem feltétlenül adja vissza az eredeti képet. Ez megtörténhet például, ha az erózió következtében eltűnik egy 1 pixel vastagságú vonalszakasz, vagy a dilatáció következtében két elem, amelyek között kevés hely volt, egybeér ezután. Ilyen esetekben a másik módszer nem tudja visszafordítani a történteket. Ezt kihasználva a két módszer kombinálásával létrejöttek új funkciók is. Ezek a morfológiai nyitás, valamint zárás. [3]

Nyitás esetén a képen először erózió, aztán dilatáció megy végbe. Ennek következtében a korábban felvetett példán, ahol fehér az információ, és fekete a háttér, a nyitás közben az erózió eltávolítja a kis méretű elemeket, amelyeket utána a dilatáció nem tud visszanagyítani. Ennek haszna a zajok eltávolítása a bináris képről. [3]

A zárás esetén először a dilatáció megy végbe, utána az erózió. Ebben az esetben olyan szakaszok, amelyek között kevés hely van, egybeolvadnak. Ez akkor hasznos, ha van például egy vonalszakasz, amely a kép átalakítása közben sérült, azaz egy kis szakasz hiányzik belőle. Ebben az esetben az a szakasz a zárás használatával kipótolható. [3]

A morfológiai műveletek segítségével a binarizált képet át lehet formálni úgy, hogy a

különféle felismerési folyamatok könnyebben fel tudják ismerni a komponenseket a képen, ezáltal digitalizálás esetén is sokszor egy hasznos eszköz lehet.

2.3.3 Átméretezés

A vizsgált képek átméretezésére több esetben is hasznos lehet. Ha a bemeneti kép méretét csökkentjük, azáltal a rajta való műveletek végrehajtása gyorsabban lefut. Ezzel azonban vigyázni kell, mivel minden kisebbre méretezéskor információ veszik el, hiszen ugyanannyi információ leírására kevesebb hely áll rendelkezésre. [3]

Amikor egy képet lekicsinyítünk, néhány pixel a cél képen nem illeszkedik, hanem pixelek közé esik az eredeti képen, amelyek értékét nem lehet egyértelműen kikövetkeztetni. Ez a probléma nagyításkor is megjelenik, mivel vannak a felnagyított képen keletkezett új pixelek, amelyek nem csatolhatóak közvetlenül az eredeti kép egy pixeléhez. Ilyen esetekben az interpoláció befolyásolja, hogy egy adott pixel értéke hogyan legyen eldöntve. A legegyszerűbb módszer az, hogy az átméretezett képen lévő pixelekhez mindig a legközelebbi pixel értékét rendeljük, az eredeti képről. Ez a legközelebbi szomszéd (Nearest Neighbor) módszer. Egy másik megoldás (bilineáris), amikor a pixel környezetében lévő pixelek alapján dől el az értéke, méghozzá a távolságuk által vett súlyozás alapján. További megoldás, amikor az új pixelt rávetítjük az eredeti képre, és a körülötte lévő pixelek átlaga alapján dől el az új érték. A bicubic interpoláció hasonló a bilineárishoz, azonban nem csak a 4, hanem a legközelebbi 16 szomszéd alapján dönt, parciális deriválás segítségével. [3]

A feldolgozás felgyorsítása érdekében végzett kicsinyítés biztonságosabban megoldható egy már binarizált képen, amelyen az információ egyértelműen elkülönül a háttértől. Ilyen esetben még mindig probléma, hogy ha a képen kis elemek, vékony vonalak vannak jelen, mivel azok könnyen eltűnhetnek a folyamatban. Erre megoldás lehet a morfológiai dilatáció, amely segítségével megvastagíthatjuk ezen elemeket, ilyen esetben azonban vigyázni kell, mert a túl nagymértékű vastagítás esetén a nagyon közel lévő szakaszok egybefolyhatnak, amellyel szintén információt veszítünk.

2.3.4 Forgatás

Amennyiben egy képet 90° vagy annak többszöröseivel forgatunk, akkor a pixelek értéke egyértelmű, mivel ilyenkor csak a pixelek pozícióját kell változtatni. Más szögben való forgatás esetén keletkeznek olyan pixelek a cél képen, amelyeket nem lehet közvetlenül párosítani egy pixellel az eredeti képen. Ilyen esetben is megjelenik a 2.3.3 fejezetben bemutatott interpoláció. [3]

A legtöbb forgatási algoritmus lefutása után az eredeti kép részei általában levágásra kerülnek, ha nem fér bele az eredeti alakzatba. Ez elkerülhető, ha az eredeti képet behelyezzük egy nagyobb kép közepére, vagyis egy keretet adunk a képnek. Ilyen esetben a forgatás után a nagyobb keretből vágódnak le a szélek, nem az eredeti képből, ezáltal megőrizve minden hasznos elemet. [3]

2.4 Kontúrkeresés

Egy kép binárissá alakítása után a kép egyszínű hátterén, a háttérrel ellenkező színű (fekete vagy fehér) pixelcsoportok maradnak. Ezen csoportok leírására, valamint összehasonlítására szolgáló módszer a kontúrkeresés. Egy kontúr pontok listája, amely egy görbét ír le egy képen, másféleképpen értelmezhető úgy, mint egy objektumot körülvevő vonal. Egy ilyen vonalat többféleképpen le lehet írni, a két leggyakoribb leírási módszer a sokszögekkel való leírás, valamint a Freeman lánckód. [7]

A Freeman lánckód esetén egy kontúr leírása egyenes vonalszakaszok segítségével van leírva. Ezek a szakaszok felfoghatóak lépések ként, amelyek megmutatják, hogy hogyan lehet körbejárni az alakzatot. Minden ilyen lépés rámutat egy következő lépésre. Ez a rámutatás 8 féle irányba történhet (fel, le, balra, jobbra, valamint az átlós irányok). Minden irányhoz hozzá van rendelve egy szám 0-tól 7-ig. Mivel minden irányhoz van egy szám rendelve, ezért egy objektumot körülvevő irányított szakasz sorozatot leírhatunk a szakaszok irányaihoz társított számok sorozatával. Ez a sorozat a lánckód. Ezek a lánckódok könnyen tárolhatóak, valamint a lánckód ismeretében a kontúrok újra kirajzolhatóak. Emellett nagy előnye a Freeman-lánckódnak, ha a hosszúságok szabadon változtathatók, akkor skála-invariáns lesz, valamint, ha a számkódolás, amely az irányokat adja meg, forgatható, akkor az lánckód által leírt alakzat is forgatható lesz. [3][7]

A kontúrok sokféle lehetőséget biztosítanak a gépi látás terén. Kontúrokat össze lehet hasonlítani egymással, a kontúr görbéket egyszerűsítve, hozzávetőlegesen megállapítani az alakjukat, azonban ami a későbbiekben a leghasznosabb lesz, az a lehetőség a kontúrok pozíciójának, valamint a kontúrokat körbevevő téglalapok méretének megállapítására. [7]

2.5 *Hough transzformáció*

A Hough transzformáció egy olyan módszer, amely segítségével egy képen kereshetünk egyszerű formákat, mint vonal vagy kör. A legtöbbször vonalak keresésére használják. [3]

A vonalak keresése a Hough transzformáció segítségével azon az elven alapszik, hogy minden pont, ami egy bináris képen található, része lehet a képen fellelhető vonalaknak. Egy vonalat felírhatunk egy koordináta rendszerben [3]:

$$y = ax + b \tag{1}$$

Az a paraméter megadja a meredekségét a vonalnak, b pedig a metszéspontját az y tengellyel. Ebben az esetben, ha megcseréljük a paramétereket a változókkal, akkor az eredeti kép minden pontja vonalként jelenik meg az alábbi síkon [3]:

$$(a, b) \tag{2}$$

A (2) síkon a vonal képlete [3]:

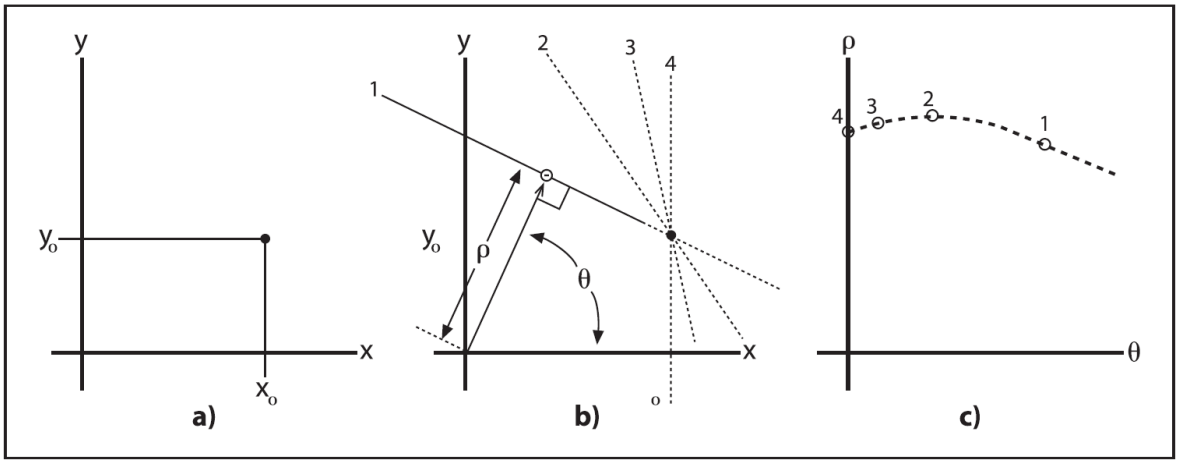
$$b = -xa + y \tag{3}$$

Ha minden jelentős képpontot átkonvertálunk erre a (2) síkra, akkor kapunk egy olyan teret, amelyben vonalak keresztezik egymás útját. Ezen a téren, ha megkeressük azokat a pontokat, ahol egy adott mennyiségű vonal keresztezi egymást, akkor az azon ponton átmenő egyenesek által képviselt pontok az eredeti síkon egy vonalat alkotnak, tehát találtunk egy vonalat. [3]

Elméletben ez így működik, azonban a gyakorlatban nem ezzel a képlettel ír le vonalakat az algoritmus, inkább a ρ és θ polárkoordináták segítségével. Ilyen esetben a következő képlet ír le egy vonalat [3]:

$$x \cos\theta + y \sin\theta = \rho \tag{4}$$

Ilyenkor ρ jelöli az origótól a vonalig vett legközelebbi távolságot, és θ jelöli a vonal és az x tengely által bezárt szöget, és ezzel a képlettel is hasonló transzformáció megy végbe, amelyre példa a 3. ábrán látható. [3]



3. ábra: Egy ponton áthaladó lehetséges vonalak transzformációja a paraméteres síkra
Forrás: *[3]*

Az imént részletezett Hough transzformáció segítségével fellelhetőek a vonalak egy képen, azonban sok számítást igényel, mivel minden nem-nulla pont esetén vizsgálódik. A számítási igény csökkentését célzó, valószínűségi Hough vonal transzformáció hasonlóképpen működik, azonban ahelyett, hogy minden lehetséges pontot megvizsgál, csak egy töredékét transzformálja át. Az ötlet mögötte az, hogy nem muszáj egy vonal minden pontját vizsgálni, mivel, ha egy vonal minden pontja után a Hough térben az intenzitás már valószínűleg úgyis túllépi azt a határt, ami szükséges a vonalnak nyilvánításhoz, akkor a pontok csak egy részének vizsgálatakor is elérthető az a határ. Ennek a feltételezésnek köszönhetően a számítási szükséglete az algoritmusnak jelentősen csökkenhet. [3]

Ebben a projektben is nagyon hasznos a vonalak felfedezése. Vonalak keresése által felfedhetőek az alakzatok közötti összeköttetések. Mivel a kapcsolatok mindenképp komponenseket kötnek össze, ezért minden vonal végén feltételezve található egy komponens. Az algoritmus találhat olyan vonalszakaszt is, amely már egy komponens része, azonban ilyen esetben keletkeznek olyan vonalak, amelyek végpontjai nem kapcsolódnak, nem illenek össze a többi vonalszakasszal, ezáltal kiszűrhetőek. Ez megnyit lehetőséget arra, hogy a megtalált vonalszakaszok összekötése által egy olyan vonalhálózat keletkezik, amelyben kimaradások vannak. Ezen kimaradások alapján feltételezhetjük, hogy a helyükön, az eredeti képen komponensek vannak jelen, amelyek a később bemutatott módszerekkel felismerhetőek.

2.6 Gépi tanulás

A gépi tanulás (ML – Machine Learning) a mesterséges intelligencia egy ága. Egy gépi tanulás algoritmus egy olyan folyamat, amely során a bemeneti adatokból úgy éri el a kívánt kimenetet, hogy annak folyamata nincs előre teljes mértékben lekódolva. Ezt úgy lehet elérni, hogy az algoritmus előre meghatározott folyamatok helyett tanulás útján jut el olyan szintre, hogy megfelelő kimenetet adhasson. [4]

A tanulás (training) ez esetben azt jelenti, hogy az algoritmusok a belső felépítésüket, ami alapján számítanak, azt folyamatosan változtatják, hogy az adott bemenetekre a megfelelő kimenetet adják. Ezt feladatok folyamatos végzésével, ismétlésével érik el. A feladatok egy előre meghatározott bemenetből és egy kívánt kimenetből állnak. Az algoritmus ezután úgy konfigurálja magát, hogy az adott bemenetre az adott kimenetet adja, valamint megpróbálja az adott feladatot általánosítani, hogy nem ugyanarra, de hasonló feladatra is a kívánt eredményt produkálja. [4]

A gépi tanulásnak több fajtája van. Az első a felügyelt tanulás (supervised learning). A felügyelt tanulás esetén minden bemeneti adathoz hozzá van rendelve az elérni kívánt kimenet, azaz fel van címkézve (labelled). Erre a legjobb példa egy osztályozó algoritmus, amely tanításakor a bemeneti adatokhoz hozzá vannak rendelve, hogy melyik osztályhoz tartoznak. [4]

Egy másik módszer a felügyelet nélküli tanulás (unsupervised learning). Ez esetén az adatokhoz nincsen hozzáfűzve címke, hogy mi is az. Ennek a tanulásnak a célja, hogy az algoritmus önmaga találjon hasonlóságokat, mintákat a bemenetek alapján. [4]

Ezen két módszer kombinációja a részben felügyelt tanulás (semi-supervised learning), amely során az adatoknak csak egy része címkézett. Ez esetben a címkézett adat segíthet a nem címkézett adatok tanításakor. [4]

A gépi tanulás által sok helyen kiváltható a repetitív emberi munka, valamint néhány téren több információt, mintát ki tud vonni adott bemeneti adatokból, ami különösen hasznos például orvosi leletek elemzésénél, ahol a kis tévedéseknek is súlyos következménye lehet. Ezen kívül a mélyreható mintakeresésnek köszönhetően a gépi látás és objektum felismerés, digitalizálás új szinteket ért el a gépi tanulás megjelenésével. Ez a fejlődés lehetővé teszi a digitalizáló szoftverek pontosságának javítását, valamint az öntanulásnak köszönhetően egy algoritmust nem kell megváltoztatni, új funkciókkal kibővíteni, ha a digitalizálni kívánt dokumentumhoz újabb felismerendő elemek kerülnek, elég csak betanítani rá az algoritmust.

Ez tűnhet hátránynak is, mivel a tanítás általában nem rövid időt vesz igénybe (projekttől függően), valamint tanítási mintákat is biztosítani kell az algoritmus számára. Ennek ellenében a felismerés logikájával nem kell foglalkozni, hiszen azt megtanítja magának az algoritmus, általában még jobb eredményeket is elérve (tanítástól függően), továbbá a felismerés nem az algoritmusok működési elvétől, és megbízhatóságától függ, hanem inkább a tanítási mintáktól, amelyek minőségét és mennyiségét általában könnyebb növelni. [4]

2.7 Neurális hálózatok

2.7.1 Általános bemutatás

Az emberi agy az egyik legösszetettebb, valamint az emberi élet szempontjából a legfontosabb biológiai szerv. Segítségével képes az ember észlelni környezetét, gondolkodni, tanulni, emlékezni, irányítani a testet, tehát lehetővé tesz mindent, amire egy ember képes. Mindezen tevékenység elvégzéséhez hatalmas számítási kapacitás, és információ-feldolgozó képességgel rendelkezik. Ezáltal nem meglepő, hogy az információs technológia megjelenése után az embereket már régóta foglalkoztatta a gondolat, hogy az agy struktúráját, valamint képességeit valamilyen módon lemásolják, újra létrehozzák azt, digitális formában. [1]

2.7.2 Felépítés

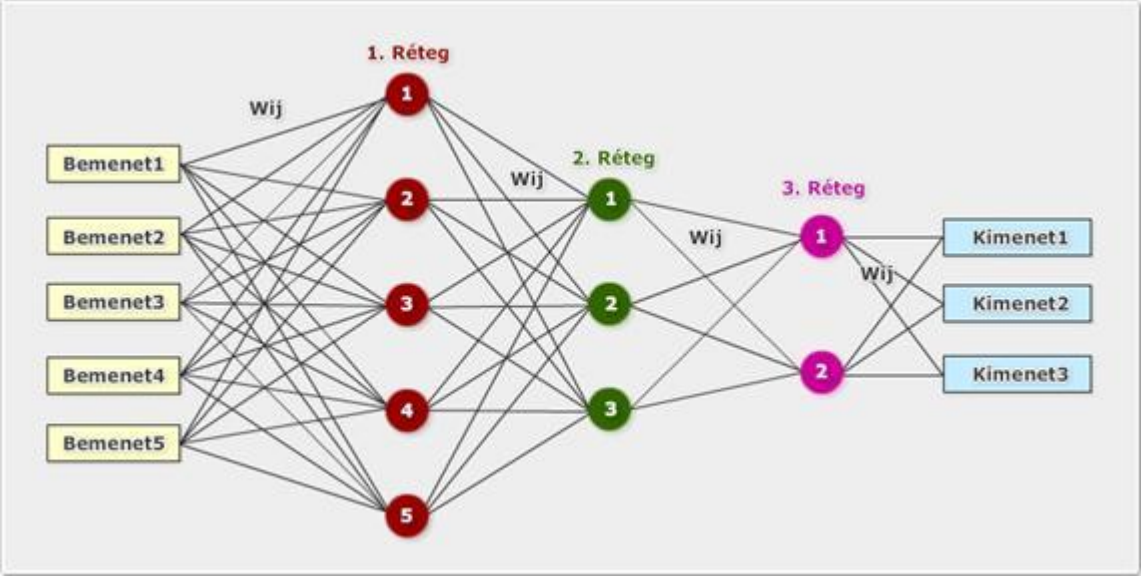
Egy agyat neuronok hálózata építi fel, amelyek faágakhoz hasonló formában kapcsolódnak egymáshoz. A neuronok egy összetett kémiai folyamat által, elektromos jelek segítségével kommunikálnak. Ezek a jelek impulzusok formájában haladnak a neuronok között. Ezt a szerkezetet, amelyet neuronok, és az azok összeköttetésük épít fel, neurális hálózatnak nevezzük. [1]

A mesterséges neurális hálózatok (ANN - Artificial Neural Network) a biológiai neurális hálózatok mintájára lett fejlesztve. Ezen hálózatok alap feldolgozó egységei a (mesterséges) neuronok. Ezek a neuronok is kommunikálnak egymással. A kommunikáció súlyozott kapcsolatok segítségével valósul meg. Egy bementet erős, azaz jelentős lesz, ha a súlyozása magas, azaz a súlyozás dönti el, hogy egy adott bemenetet mennyire jelentősen veszünk figyelembe. Emiatt a súlyok megfelelő kiosztása egy kritikus feladat a neurális hálózatoknál. Ez a feladat azonban kézzel általában nagyon komplikált, vagy szinte lehetetlen lenne megoldani. Ennél a feladatnál jelenik meg a gépi tanulás, amely által különböző algoritmusok segítségével, a tanítás során ezen súlyozások beállítása megtörténik. [1][6]

Azt, hogy egy neuron mikor aktiválódjon, azaz, hogy adjon kimenetet vagy ne, az aktivációs függvény dönti el. Egy aktivációs függvény a bemenetek súlyozott összegével, valamint további függőségek hozzáadásával dönti el, hogy az adott neuron milyen kimenetet adjon. Egy aktivációs függvény lehet lineáris, amikor a kimenet arányos a bemenettel. A másik módszer a nem lineáris függvények alkalmazása, amelyek sokkal szélesebb körben használtak. Segítségükkel könnyen tud alkalmazkodni a modell a különböző adatokhoz, azaz jobban használható a tanításhoz. A mai leggyakrabban használt aktivációs függvény a ReLu. A ReLu általában a rejtett rétegekben használt, és szinte minden konvolúciós neurális hálózat ezt használja. [5]

A neurális hálózatok neuronjait nem egyenként, hanem sok hasonló neuront egy rétegként kezelünk, amely rétegek saját funkcionalitással rendelkeznek. Egy neurális hálózat rendszerint három elkülöníthető részből tevődnek össze[1]:

1. bemeneti réteg (input layer)
2. rejtett réteg (hidden layer)
3. kimeneti réteg (output layer)



4. ábra: Neurális hálózati modell
Forrás: https://regi.tankonyvtar.hu/en/tartalom/tamop412A/2010-0017_08_meres_es_iranyitastechnika/ch06s03.html

A bemeneti réteg továbbítja a bemenetként kapott adatot a többi rétegnek. Ebben a rétegben a neuronok számát a bemeneti adat alakja határozza meg. [1]

A rejtett rétegek a bemenet és a kimeneti réteg között dolgozza fel, alakítja át az információt. A rejtett rétegek száma, és az egyes rétegekben található neuronok száma

változtatható, a használati céltól függően változik. [1]

A kimeneti réteg felépítése szintén függ a megoldandó feladat jellegétől. Osztályozás esetén a kimeneti rétegben, a kimeneti neuronok száma megegyezik a lehetséges osztályok számával, míg regresszió esetén egy van. [1]

A neurális hálózatokat, az architektúrájuk alapján 2 csoportba sorolhatjuk [6]:

1. Elsőrecsatolt (FFNN - Feed-forward neural network)
2. Visszacsatolt (RNN - Recurrent (feedback) neural network)

FFNN esetében az információáramlás egy irányba történik, a bemeneti réteg irányától a kimeneti réteg irányába. Ez az áramlás több rétegen is áthaladhat, viszont nem haladhat visszafelé, míg RNN esetén a jel visszafelé is haladhat. [6]

2.7.3 Tanítás és tesztelés

Egy neurális hálózat tanítása többféle módszerrel megvalósítható. A legfontosabb teória ilyen téren a Hebbian tanulás, amely mutatott egy koncepciót a tanítás megvalósítására. Kimondta, hogy a tanulás megoldható a neuronok közötti kapcsolatok változtatásával. Bővebben kifejtve, amikor egy neuron (A), egy másik neuront (B) ismétlődve aktivál, részt vesz az aktiválásában, akkor változtatni kell a kapcsolatokat úgy, hogy A nagyobb behatással legyen B-re. Röviden, egyszerűen átfogalmazva, ha két neuron közel egy időben aktiválódik, akkor a kettő közötti kapcsolatot meg kell erősíteni. [1][6]

A perceptron-tanulás esetén hasonló folyamat történik. Amikor kap egy bemenetet, megvizsgálja a kimenetet, és ha a kimenet nem megfelelő, akkor megváltoztatja a kapcsolatokat. A különbség a Hebbian tanulás szabályától, hogy ha egy adott bemenetre megfelelő kimenetet kapott, akkor viszont nem változtat a kapcsolatok súlyán. Egy szimpla perceptron lineáris problémák megoldására használható. Ez osztályozás esetén azt jelenti, hogy bináris osztályozó, azaz csak 2 osztályt tud egymástól megkülönböztetni. Lehet több osztály esetén is használni abban az esetben, ha az osztályok lineárisan elkülöníthetők. Ilyen esetben az algoritmus megpróbálja az egyes osztályokat elkülöníteni a többitől úgy, hogy a két lehetőség közül az egyik az adott osztály, a másik pedig az összes többi osztály, egy a mindenki ellen stílusban. [1][6]

Egy másik módszer a backpropagation (visszaterjesztéses) tanulás. Feed-forward hálózatok esetén használható, ahol a bemenet után a jel előre halad a kimenetig, majd a végén a hibákat visszaterjeszti. Ilyen esetben meg lehet tudni, hogy melyik neuron milyen mértékben járult hozzá a hibához, és aszerint lehet a kapcsolatait gyengíteni, erősíteni. [1][6]

Ahhoz, hogy egy megbízható, robosztus neurális hálózatot hozzunk létre, a megfelelő tanítási módszer alkalmazásán kívül elengedhetetlen a tanítási minták minél széleskörűbb variációja, minél karakterisztikusabb minták választása. Ennek eléréséhez a standard mintákon kívül gyakran alkalmaznak olyan mintákat, amelyekhez zaj, vagy valamilyen véletlenszerűség lett alkalmazva, ezáltal felkészítve a neurális hálózatot a gyakorlatban sokszor megjelenő zavaró tényezőkre. [1]

A gyenge tanítás értelemszerűen megbízhatatlan neurális hálózatot eredményez. Ennek elkerülése végett, a tanítást egy meghatározott számú epoch által, vagy egy meghatározott hibahatár segítségével hajtják végre. [1]

Az epoch, a neurális hálózatok tanításával kapcsolatos fogalom. Meghatározza, hogy az alkalmazott adatkészletet hányszor adagoljuk tanításra, azaz egy epoch esetén az egész adatkészlet egyszer átmegy a tanítási folyamaton. Egy adatkészlet többszörös tanítására azért van szükség, mivel gyakran ezek az adatkészletek csak limitált adattal rendelkeznek, amely egyszeri tanulásakor nem biztos, hogy megfelelő eredményeket érünk el, ezért többszörös átvezetésre van szükség. Ez működőképes, mivel minden előző tanulási ciklusban a neurális háló átkonfigurálta magát, általánosította a bemeneti adatokból kinyerhető attribútumokat, ezért az újabb átvezetéskor már máshogy tekint ugyanarra az adatkészletre. Egy epoch általában túl nagy ahhoz, hogy egyszerre a rendszerbe adagoljuk, ezért kisebb, feldolgozhatóbb egységekre szokás bontani. Ezen egységek a halmok (batches). [1]

Az epoch száma szabadon választható, azonban figyelni kell arra, hogy ne legyen túltanítva (overtraining) a hálózat. Ez akkor történik meg, ha túl sokszor tanítjuk a neurális hálózatot ugyanazokkal a mintákkal, tehát túl sok epoch lett választva, és a hálózat túlságosan a tanító mintákhoz lett szoktatva, és nem általánosítja megfelelően a bemeneti paramétereket. Ilyen esetben a tanítási adatkészleten kívüli mintákat nem megfelelően tudja beazonosítani. [1]

Az epoch számának megválasztásán kívül kritikus lépés a neuronok számának megfelelő választása. A választott rejtett rétegen belüli neuronok száma befolyásolja a hálózat tanulási, és felismerési teljesítményét, megbízhatóságát. Túl kevés neuron által nem tud elegendő jellemzőt, összefüggést kivonni a bemenetből, ami a felismerés pontatlanságához vezet. Túlságosan nagyszámú neuron esetén a tanulást megfelelően végre tudja hajtani, azonban ez a teljesítmény kárára megy, valamint itt is probléma, hogy a túl sok részlet által nem megfelelően általánosít. [1]

Az epoch, valamint a neuronok számának megválasztása nagyon fontos lépés. Erre azonban nincsenek mindig megfelelő értékek, minden alkalmazás esetén eltérő. Nagymértékben befolyásolja a neurális hálózat által kezelendő adatok formája, részletessége, egymástól való eltérése. [1]

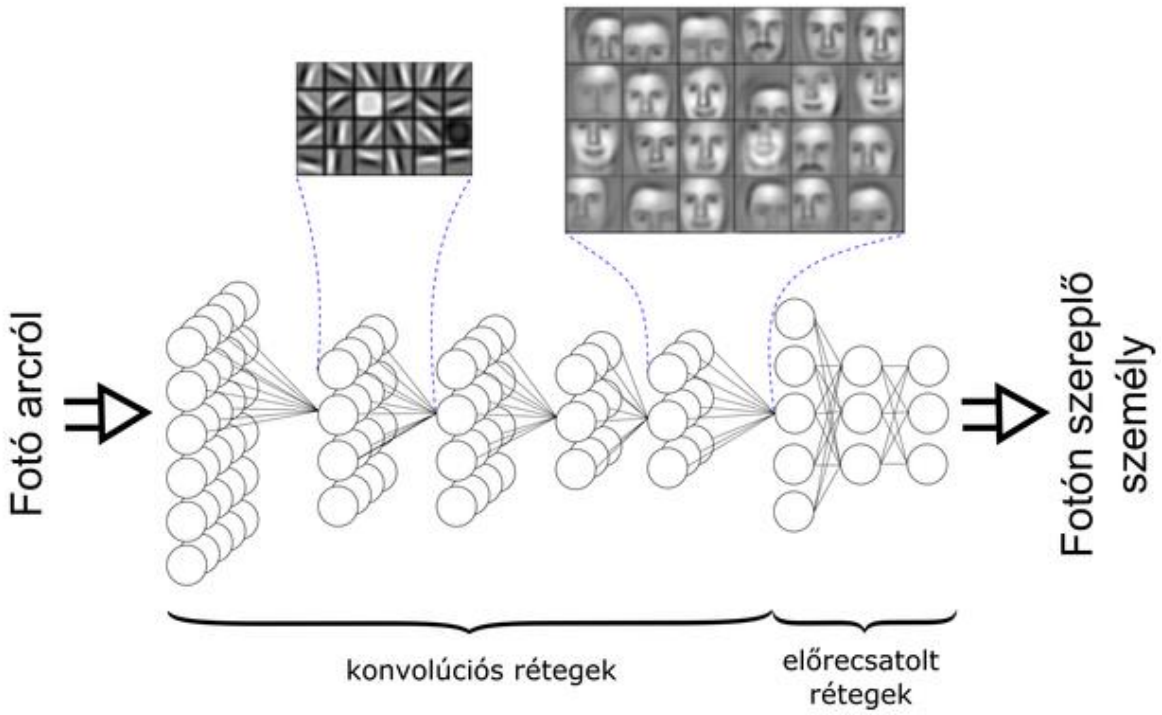
2.8 Konvolúciós neurális hálózatok

A konvolúciós neurális hálózatok (CNN – Convolutional Neural Network) a neurális hálózatoknak egy olyan típusa, amely kifejezetten nagy sikereket ért el a gépi látás területén. Használata sok téren megjelenik, amelyből az egyik legfontosabb, és leg figyelemre méltóbb az orvosi téren való felhasználása, a képes leletek elemzése. A CNN előnye, a hagyományos neruális hálózatokkal szemben a hálózatban résztvevő paraméterek csökkenése. Ezáltal nagyobb, komplexebb feladatok is megoldhatóak lettek, amelyek a klasszikus ANN segítségével csak nehezen, vagy nem volt lehetséges. Leggyakrabban képek osztályozására (image classification) van használva, és jelenleg is képek, azaz a kapcsolási rajzok szimbólumainak osztályozására lesz alkalmazva. [2]

A CNN a képek felimerésére specializált neurális hálózat, ami tipikusan 3 típusú rétegből tevődik össze [12]:

1. konvolúciós réteg (convolution layer)
2. összevonó réteg (pooling layer)
3. teljesen csatolt réteg (fully connected layer)

Az első kettő, azaz a konvolúciós és az összevonó réteg jellemző kivonást (feature extraction) végez, míg a teljesen csatolt réteg a végső kimenetet alkotja meg, ami általában osztályozás (classification). [12]



5. ábra: Konvolúciós neurális hálózat
Forrás: https://www.eletestudomany.hu/neuralis_halozatok

A konvolúciós réteg eltér az eddig ismertetett neurális hálózat rétegektől. Ez a réteg súlyozott kapcsolatok helyett szűrőket (konvolúciós szűrők – convolution filters) tartalmaz, amelyek a bemeneti kép feldolgozására valók. Ebben a rétegben numerikus tömböket (kerneleket) futtatunk végig a bemeneten, ami szintén egy numerikus tömb. A kernel végigmegy a képen, és minden lépésnél, a bemenet aktuális része, valamint a kernel között kiszámítja a két tömb elemekénti szorzatát, majd annak az összege lesz az érték a kimenet adott pozíciójában. Ez a kimenet a jellemző térkép (feature map). Ez a folyamat folytatódik, különféle kerneleket alkalmazva, míg nem érünk el egy megfelelő számú kimenetet. Az előbbiek alapján tehát minden új kernelt alkalmazva más-más jellemzőket tudunk megállapítani a bemenetből. Két fontos jellemzője a konvolúciós rétegnek a kernelek száma, valamint a mérete. A kernel mérete általában 3x3, azonban a gyakorlatban előfordul 5x5 vagy 7x7 is. A kernelek száma tetszőleges, meghatározza a vizsgálható jellemzők mennyiségét. [2][12]

A fent leírt megoldás esetén a konvolúciós lépéseknél a kernel közepe sose érinti a kép szélét, ezáltal a kimenet kisebb méretű lesz. Erre alkalmazott megoldás a padding, leggyakrabban a zero-padding. Segítségével 0 értékek lesznek fűzve a bemenet minden oldalához, ezzel megnövelve méretét. Ilyen esetben a kernel már tud a kép szélén haladni,

ezáltal a kimeneti kép ugyan akkora lesz, mint a bemenet. A módszer segítségével tehát megmarad az eredeti mérete, ezáltal több réteg is könnyen alkalmazható, mivel nem kell számolni azzal, hogy a kimenet egyre kisebb lesz. [2]

Egy másik jellemzője a konvolúciónak a lépés (stride), amely meghatározza, hogy a kernel mekkora lépéseket tegyen meg a bemeneten. Ez általában 1, mivel ilyen esetben az egész kép vizsgálva van, azonban használható egynél nagyobb is, ilyen esetben csökkentett mintavételezés (downsampling) történik. Ilyen esetben a kimenet kisebb lesz, azaz kevesebb részletet vonunk ki a bemenetből. [2]

Az összevonó (pooling) réteg segítségével csökkenthető a komplexitás a további rétegek számára. Ebben a rétegben nincs tanítható paraméter, azonban a szűrő mérete, a lépés nagysága, valamint a padding értéke megadható. [2]

A leggyakoribb pooling technika a max pooling, ami esetén a kép kis régiókra (téglalapokra) van felosztva, és minden régióból a maximum értéket adja a kimenetre. A leggyakoribb, max-pooling esetén használt szűrő méret a 2x2. Ilyen esetben az első 2x2-es régióból kimenetre adja a legnagyobb értéket, utána tovább mozdul a szűrő, mint egy kernel. Ilyenkor is tényező a lépés nagysága, amely 2 esetén egy teljes lépést teszt meg a szűrő, nincs átfedés a vizsgálatok között, ezáltal a kimenet kisebb lesz a bemenetnél. Használható 1 lépés is, ilyen esetben a méret nem változik, ez azonban nem gyakran alkalmazott, mivel a cél az egyszerűsítés. [2][12]

Pooling esetén még érdemes megjegyezni, hogy a folyamat által a pozíciós információk nem maradnak meg. Ennek következtében csak akkor alkalmazandó, amikor az információnak csak a jelenléte a fontos, a pozíciója nem. [2]

A pooling által generált kimenetet általában le van lapítva (flattened), azaz egy dimenziós szám tömbbe van átalakítva, és ezt egy vagy több teljesen csatolt, vagy más nevén dense réteghez kapcsolják. Ezek a rétegek hasonlóak a tradicionális neurális hálózatokhoz, amelyekben minden neuron hozzá van csatolva minden neuronhoz, a réteg előtti és utána levő rétegben is. Miután a jellemzők ki lettek vonva a konvolúciós rétegben, és le lettek egyszerűsítve a pooling rétegben, utána ebben a rétegben vagy rétegekben dől el a végső kimenet. A legvégső teljesen csatolt rétegben általában annyi kimenet van, amennyi osztály van osztályozás esetén, és minden kimenet megadja, hogy mekkora esély van arra, hogy a vizsgált dolog az adott osztályba tartozik. [2][12]

2.9 *Hasonló munkák*

A bemutatott módszerek segítségével megvalósítható a digitalizáló program. Ez azonban nem az egyetlen mód hasonló program készítéséhez. Ebben a fejezetben más, hasonló munkásságokat mutatok be, és hasonlítok össze az általam felvázolt megoldásokkal szemben.

2.9.1 Primitív alakzatokra alapuló felismerés

Tudhope et. al [11] által használt megoldásban a felismerő algoritmus 3 különféle primitív alakzatot keres:

1. vonal
2. görbe
3. kör

A vonalak és görbék felismerése lekövetéssel valósul meg. Ilyenkor a vonalkereső algoritmus végigmegy egy vonalon, egyik jellemzőpontjáról a másikra. A következő pontot mindig az előzőek alapján keresi, azok alapján próbálja kikövetkeztetni. Görbék esetében, a görbét egyenes vonalszakaszokkal írja le. A körök keresése sokkal kevésbé kifinomult megoldáson alapszik, amely során két, egymásra merőleges átmérő kirajzolásának sikeressége estén egy kört ismer fel a program. [11]

Miután felismert minden primitívet, megpróbál közöttük összefüggéseket találni. Ezek az összefüggések előre megírt szabályokként léteznek. A szabályok leírására egy diagramm leíró nyelvet használtak, amelyben az egyek komponensek, primitívekből való összetételét szövegesen írják le. Egy komponens leírásakor, annak minden elemét egy szám azonosít. Ahhoz, hogy felépítsenek egy alakzatot a primitívekből, a primitívek egymáshoz viszonyított kapcsolatukat szóban jellemzik, például a MEETS (találkozik), vagy ANGLE (szög) szavakkal. Ezen kívül fuzzy boolean változókat használtak, valószínűségi információk tárolásához. Ezek a változók lehetséges állapotai a következők [11]:

- NEVER (soha)
- UNLIKELY (valószínűtlen)
- MAYBE (talán)
- PROBABLY (valószínű)
- ALWAYS (mindig)

Ezek segítségével leírható, hogy adott primitívek jelenléte, kapcsolatok megléte

mennyire valószínű, és jelenléte vagy hiánya mennyire befolyásolja egy alakzat felismerését. Ha valahol talál olyan primitíveket, amelyek kapcsolatai egy szabályban leírtaknak megfelel, akkor azt elkönyveli egy felimert alakzatként. [11]

Ez a módszer már régi, azonban sikeresen megvalósít egy olyan diagram felismerő rendszert, amelyben a felismerhető elemek egy szerkeszthető adatbázisban tároltak, ezáltal változtathatóak, valamint új elemek is könnyebben hozzáadhatóak, amennyiben leírhatóak a három primitív segítségével. Legnagyobb hátránya a megoldásnak, hogy nagymértékben függ a primitívek felismerésétől. Ha egy kritikus, rosszul, vagy nem ismer fel, amely a komponens szabályában szerepel, akkor az az egy elem által már nem ismerheti fel. Ez szabályozható, hogy a gyakran tévesztett elemekhez kisebb valószínűség van rendelve, ez azonban növeli az esélyét a hamis pozitív felimeréseknek. [11]

2.9.2 Hurok alapú felismerés

Okazaki et. al [9] megoldásában a komponenseket kétféleképpen ismeri fel. Az olyan elemeken, amelyek valamilyen hurkot, zárt alakzatot tartalmaz: a képen megkeres hurkot tartalmazó elemeket, kapcsolt komponens (connected component) analízis segítségével. Ezek közül kiszűri a hamis hurkokat, vagyis azokat, amelyek a kapcsoló vonalak és alakzatok elrendezéséből keletkezett. Utána a megtalált hurkok (amelyekre karakterisztikus hurokként utal), azok alapján kategorizálja az alakzatot. Az olyan alakzatok esetén, amelyek nem tartalmaznak hurkot, a vonalak, sarkok, és vonal végpontok alapján azonosítja be. [9]

Ez a módszer a hurkot nem tartalmazó komponensek esetén összetettebb, nehezebben kezelhető megoldást alkalmaz, mivel egy új ilyen felismerendő alakzat definiálása az algoritmus számára nehézkes, mivel kézzel kell megadni, hogy az alakzatok milyen érdekpontokat tartalmaznak, és milyen elrendezésben. A hurkot tartalmazó elemek esetén a primitív hurkok alakja által beazonosítható a komponens, azonban sok változó van, ami a felismerés hibájához vezethet, például, ha nem sikeresen lettek kiszűrve a hamis hurkok, vagy a képen található zajok (amelyek az előfeldolgozás alatt is keletkezhettek) deformálhatják az alakzatokat. Ezen hibák más rendszereket is megzavarhatnak, azonban a jelenleg bemutatott CNN használatával, a széleskörű felügyelt tanítás segítségével ilyen esetekre valamilyen szinten felkészíthető a rendszer. Ezektől eltérően a hurokkeresés segítségével sok szimbólum megfelelően beazonosítható, vagy a pozíciójuk felfedhető. [9]

2.9.3 ANN alapú felismerés

Rabbani et. al [10] által egy, az ehhez hasonló megoldás kerül bemutatásra. A felismerés kézzel rajzolt szimbólumokra koncentrál, azonban nem keletkezik sok eltérés, a digitálisan létrehozott szimbólumok felismerésétől. A módszerben ANN, azaz egy mesterséges neurális hálózatot alkalmaz, mint osztályozó rendszer, felügyelt tanítással. Az osztályozás hasonló a CNN alkalmazásához, azonban ANN esetén előzetesen, a neurális hálózattól függetlenül kell a jellemző kivonást alkalmazni, azaz meg kell keresni a bemeneten azokat a pontokat, jellemzőket, amelyek alapján az osztályozás meg fog történni. CNN esetén ez a neurális hálózat konvolúciós rétegeiben történik meg, ezáltal nagyobb szabadságot adva a hálózat számára. Ez előnyösebb, hiszen a sok tanítási ciklus alatt a rendszer megtanulja, hogy melyek azok a jellemzők, amelyek leginkább befolyásolják az osztályozás sikerességét, míg a kézi jellemzők kivonásánál sok nem annyira fontos jellemző is figyelembe lehet véve, amely csökkentheti a felismerés sikerességét. [10]

2.10 Fejlesztői környezet, programcsomagok

A projekt elkészítéséhez a Visual Studio Code [13] fejlesztői környezetet használtam. A környezet előnye, hogy platformfüggetlen, kevés a helyigénye, valamint könnyen, de nagy mértékben személyre szabható, valamint többféle programozási nyelvet támogat.

A szoftver Python [14] programozási nyelven készült. A Python egy magas szintű programozási nyelv, amelynek egyszerű szintaktikája, valamint számos letölthető szoftvercsomagjai megkönnyítik a nagy projektek készítését.

A szoftver megvalósításához többféle szoftvercsomag kerül felhasználásra. Az adatok kezelését a numpy [15] programcsomag segítette, amely a tömbök kezelését támogatja. A képek beolvasását, kezelését, valamint különféle képi transzformációk elvégzését az OpenCV [16] csomag segítette. A gépi tanulás, és neurális hálózatok használatát a tensorflow [17] könyvtár, valamint az abban lévő keras API tette lehetővé.

3 Irodalomjegyzék

[1] Abraham, A.: Artificial Neural Networks, in: Handbook of Measuring System Design
Oklahoma State University, Stillwater, OK, USA, pp. 901-908, 2005.

[2] Albawi, S., Mohammed, T. A., Al-Zawi, S.: Understanding of a convolutional neural network
International Conference on Engineering and Technology (ICET), pp. 1-6, 2017.

[3] Bradski, G., Kaehler, A.: *Learning OpenCV.*
O'Reilly Media, Inc, United States of America, p. 571, 2008.

[4] El Naqa, I., Murphy, M.: Machine Learning in Radiation Oncology: Theory and Applications
Springer International Publishing, pp. 3-11, 2015

[5] Feng, J., Lu, S: Performance Analysis of Various Activation Functions in Artificial Neural Networks
Journal of Physics: Conference Series, pp. 1-6, 2019.

[6] Gupta, N.: Artificial Neural Network
International Conference on Recent Trends in Applied Sciences with Engineering Applications, vol.3, no.1, pp. 1-6, 2013.

[7] Marengoni, M., Stringhini, D.: High Level Computer Vision Using OpenCV
2011 24th SIBGRAPI Conference on Graphics, Patterns, and Images Tutorials, pp. 11-24, 2011.

[8] Nemzetközi standardok:
https://www.iec.ch/understanding-standards, letöltés ideje: 2021.03.18.

[9] Okazaki, A., Kondo, T., Mori, K., Tsunekawa, S., Kawamoto, E.: An automatic circuit diagram reader with loop-structure-based symbol recognition
IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 10, no. 3, pp. 331-341, 1988.

[10] Rabbani, M., Khoshkangini, R., Nagendraswamy, H.S., Conti, M.: Hand Drawn Optical Circuit Recognition
Procedia Computer Science, vol. 84, pp. 41-48, 2016.

[11] Tudhope, D.S., Oldfield, J.V.: A High-Level Recognizer for Schematic Diagrams
IEEE Computer Graphics and Applications, vol. 3, no. 3, pp. 33-40, 1983

[12] Yamashita, R., Nishio, M., Do, R.K.G., Togashi, K.: Convolutional neural networks: an overview and application in radiology
Insights Imaging, pp. 611–629, 2018.

[13] Del Sole, A.: Visual Studio Code Distilled: Evolved Code Editing for Windows, MacOS, and Linux.
Apress, p. 215, 2018.

[14] Van Rossum, G., Drake, F. L.: Python 3 Reference Manual.
CreateSpace, 2009.

[15] Harris, C.R., Millman, K.J., van der Walt, S.J. et. al.: Array programming with NumPy
Nature, vol. 585, no. 7825, pp. 357–362, 2020.
[16] Bradski, G.: The OpenCV Library
Dr. Dobb's Journal of Software Tools, 2000.
[17] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J. et al.: Tensorflow: A system for large-scale machine learning.
12th USENIX Symposium on Operating Systems Design and Implementation, pp. 265–283, 2016.

4 Mellékletek