

[Külső fekete borítólap formátuma]

Széchenyi István Egyetem  
Gépészmérnöki, Informatikai és Villamosmérnöki Kar  
Informatika Tanszék

# **SZAKDOLGOZAT**

**Józsa Dávid**  
Mérnök Informatikus BSc szak

2021

[Gerincen:] Hallgató Neve, Évszám {Titkosított}
---



**SZÉCHENYI  
EGYETEM**  
UNIVERSITY OF GYŐR



**INFORMATIKA  
TANSZÉK**  
DEPARTMENT OF COMPUTER SCIENCE

# **SZAKDOLGOZAT**

## **Kapcsolási rajz digitalizáló szoftver**

**Józsa Dávid**

**Mérnök Informatikus BSc szak**

**2021**

## 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 fel tud ismerni, valamint digitalizálni egy kapcsolási rajzot úgy, hogy az utána digitálisan szerkeszthető legyen. A program különböző gépi látás módszereket, valamint egy konvolúciós neurális hálózatot használ a rajz felismeréséhez.

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 használhatók, 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. A legvégén a program tesztelésének eredménye van kifejtve.

# **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 picture in a way that makes it editable. The program uses different computer vision methods and a convolutional neural network to recognize the circuit.

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. At the very end the results of the tests about the program are explained.

# Tartalomjegyzék

1	Bevezetés .....	1
2	Elméleti háttér.....	2
2.1	Kapcsolási rajzok.....	2
2.2	A probléma bemutatása.....	3
2.3	Képek előfeldolgozása.....	4
2.3.1	Thresholding.....	4
2.3.2	Morfológiai transzformációk.....	6
2.3.3	Átméretezés .....	8
2.3.4	Forgatás .....	8
2.3.5	Perspektíva transzformáció .....	9
2.4	Kontúrkeresés .....	9
2.5	Canny éldetektálás .....	10
2.6	Hough transzformáció.....	11
2.7	Gépi tanulás .....	14
2.8	Neurális hálózatok .....	15
2.8.1	Általános bemutatás.....	15
2.8.2	Felépítés.....	15
2.8.3	Tanítás és tesztelés.....	17
2.9	Konvolúciós neurális hálózatok.....	19
2.10	Hasonló munkák.....	21
2.10.1	Primitív alakzatokra alapuló felismerés .....	22
2.10.2	Hurok alapú felismerés .....	23
2.10.3	ANN alapú felismerés .....	23
2.11	Fejlesztői környezet, programcsomagok .....	24
3	Megvalósítás.....	25
3.1	Program használatának előfeltételei .....	25
3.2	Segédosztályok, konstansok .....	26
3.3	Bemenet és előfeldolgozás.....	28
3.3.1	Kép beolvasása, egyszerűsítése .....	28
3.3.2	Kép transzformációi .....	31
3.4	Komponensek és összeköttetések helyének beazonosítása .....	35
3.4.1	Vonalak felismerése, feldolgozása.....	35
3.4.2	Komponensek keresése.....	37
3.5	Komponensek felismerése, CNN.....	39

3.5.1	Komponensek feldolgozása .....	39
3.5.2	CNN.....	40
3.6	Összeköttetések azonosítása .....	42
3.7	Kimenet generálása .....	45
4	Tesztelés.....	47
4.1	Módszer .....	47
4.2	Eredmények, levont következtetések.....	49
4.2.1	Rajz feldolgozása, felismerése .....	49
4.2.2	Generált kimenet.....	52
5	Összegzés .....	57
6	Irodalomjegyzék .....	58

# 1 Bevezetés

Elektronikai eszközök tervezésekor, vizsgálatokor 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.

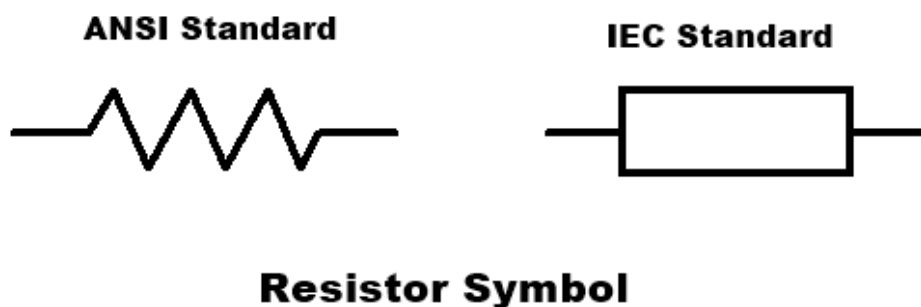
Kapcsolási rajzok gyakran fellelhetőek nyomtatott formában, egy kezelési útmutatóban, vagy rendszerleíró dokumentumokban. Ezek a dokumentumok régóta léteznek, azonban a számítógépes grafika fejlődésével és széleskörű terjedésével megszületett az igény a régebbi formátumú iratok, rajzok digitalizálására. A hasonló digitalizációs feladatokat lehet kézzel végezni, vagyis egy ember nézi a rajzot, amelyet aztán digitálisan elkészít. 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 képen 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 Különböző stílusú ellenállás szimbólumok

Forrás: [2]

A kapcsolási rajzok általában hasonló irányelveket követnek. Felépítésük, megjelenésük általában hasonló, azonban vannak stílusbeli eltérések. Ezen eltérések leginkább a komponenseket leíró szimbólumok különbségében mutatkozik meg. Létezik standard [1] 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 bocsátanak 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 tudására 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.

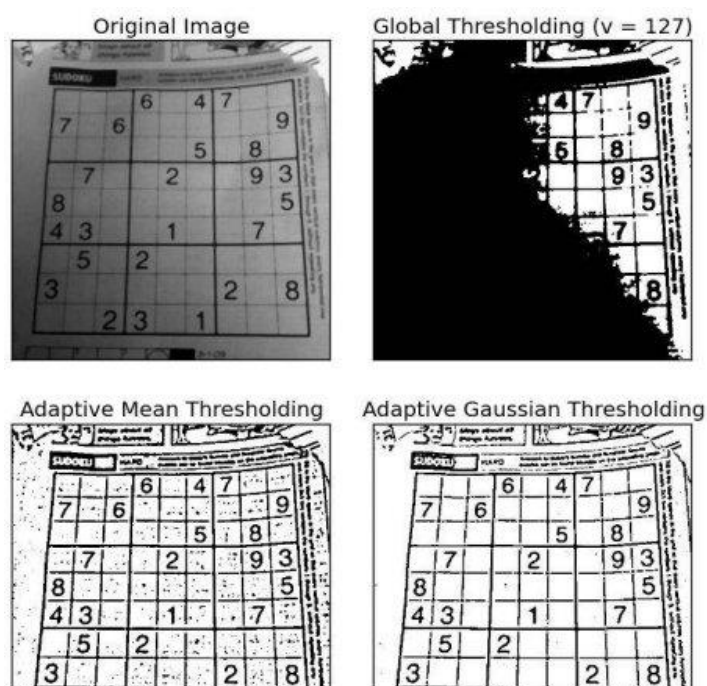
### **2.3.1 Thresholding**

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][4]

A leggyakoribb thresholding technikában egy paraméter, a küszöbérték (T) segítségével történik a szétválasztás. Ilyen esetben a bemeneti szürkeárnyaltos 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 számítható, 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

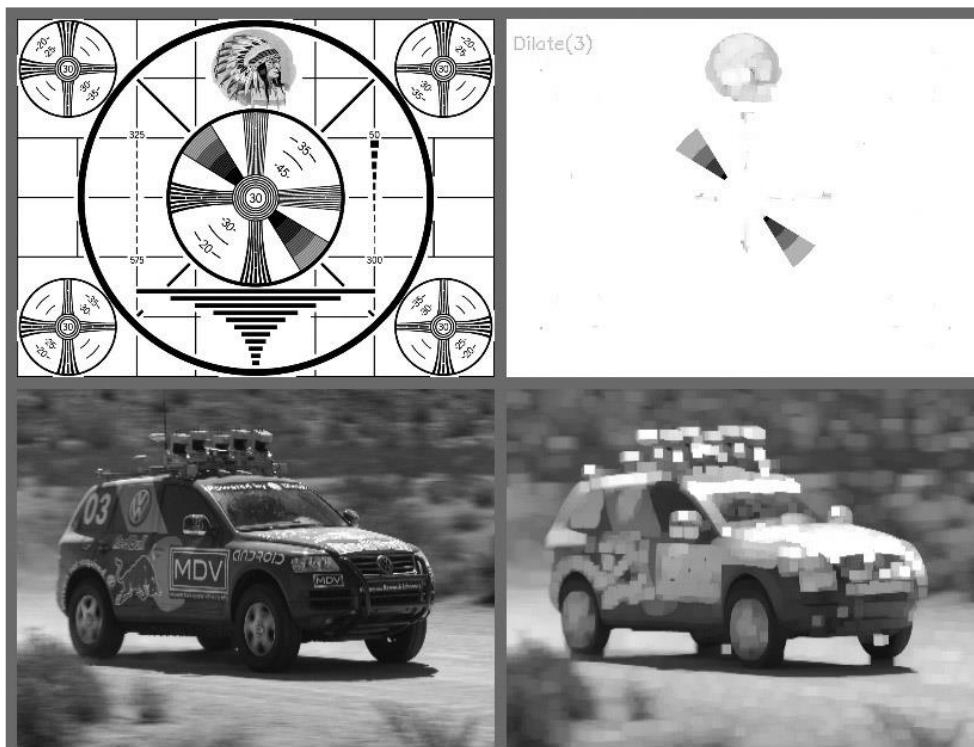
Forrás: [5]

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 szürkeárnyaltos 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 dilatació. [3]

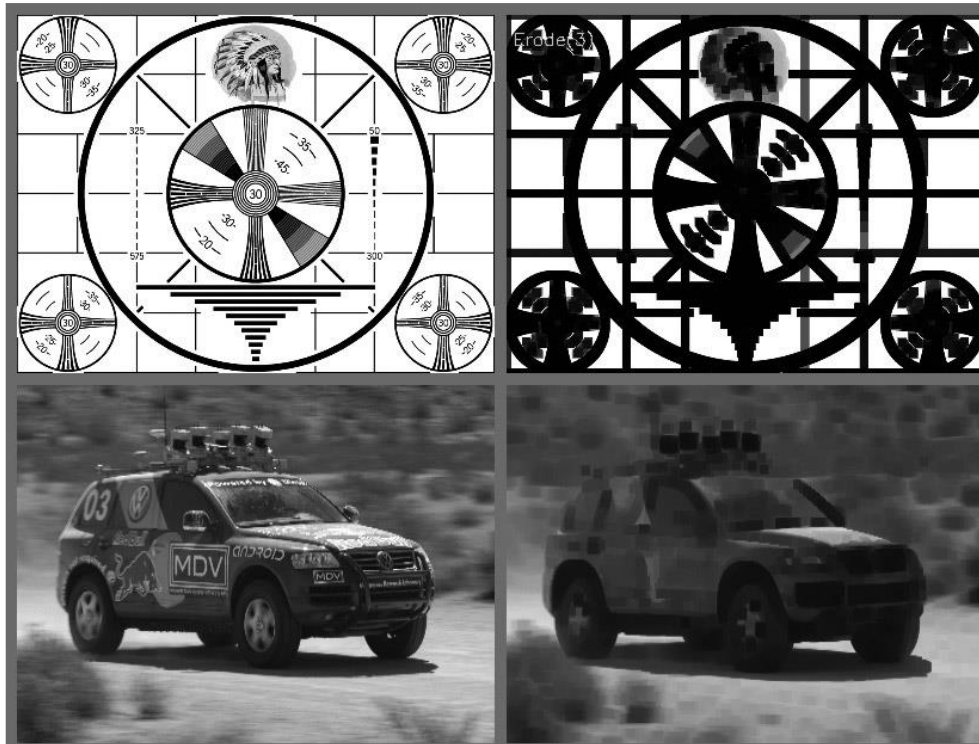
Dilatáció (dilation) esetén, amikor a kernel áthalad a képen, minden állomásnál megkeresi a kernel által lefedett pixelek közül a legnagyobb értékűt, és az anchor pontot beállítja arra az értékre. Ennek következtében a világos területek a képen növekednek. [3]



3. ábra: Dilatació (sötét területek csökkennek, világos területek növekednek)

*Forrás: [3]*

Az erózió (erosion) a dilatació 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]



4. ábra: Erózió (sötét területek növekednek, világos területek csökkennek)

*Forrás: [3]*

Egy olyan képen, amelyen a háttér fekete, és a hasznos elemek fehérek, az erózió szűkíti a hasznos területeket, alakzatokat, míg a dilatació 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 dilatació következtében két elem, amelyek között kevés hely volt, egybeér ezután. Ilyen esetekben a másik módszer nem tudja visszafordítani a történetet. Ezt kihasználva a két módszer kombinálásával létrejöttek új funkciók is. Ezek a morfológiai nyitás, valamint zárás. [3]

Nyitás esetén a képen először erózió, aztán dilatació 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 dilatació nem tud visszanagyítani. Ennek haszna a zajok eltávolítása a bináris képről. [3]

A zárás esetén először a dilatació 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 (tapasztalatom szerint) 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  $90^\circ$  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]

### **2.3.5 Perspektíva transzformáció**

A perspektíva transzformáció (perspective transform) segítségével egy képet át tudunk alakítani más, trapezoid formájúra, ezáltal úgy formálva a képet, mintha más nézőpontból látnánk azt. A gépi látás terén sokszor hasznos, főleg madárnézetű (bird's eye view) képek létrehozására. [3]

A transzformációhoz szükség van egy  $3 \times 3$ -as méretű perspektíva mátrixra. A mátrix segítségével a kép akármelyik pontját át lehet számítani az új formájú képre, egy szorzás segítségével. Ez a folyamat a homográfia, azaz egy felületen lévő pontok átszámítása egy másik felületre, esetünkben pixelek számítása egyik képről egy másik formájú képre, egyetlen mátrix segítségével. [3]

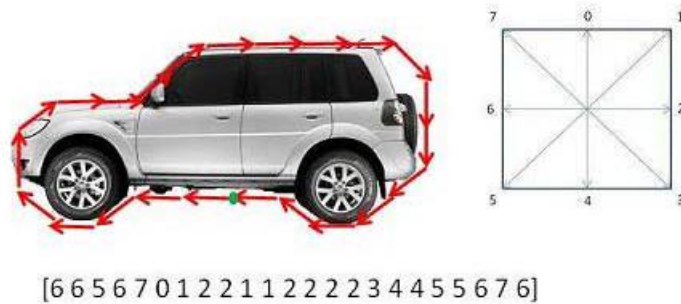
## **2.4 Kontúrkeresés**

Egy kép binárisra 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. [4]

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.



5. ábra: Freeman lánckód

*Forrás: [4]*

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][4]

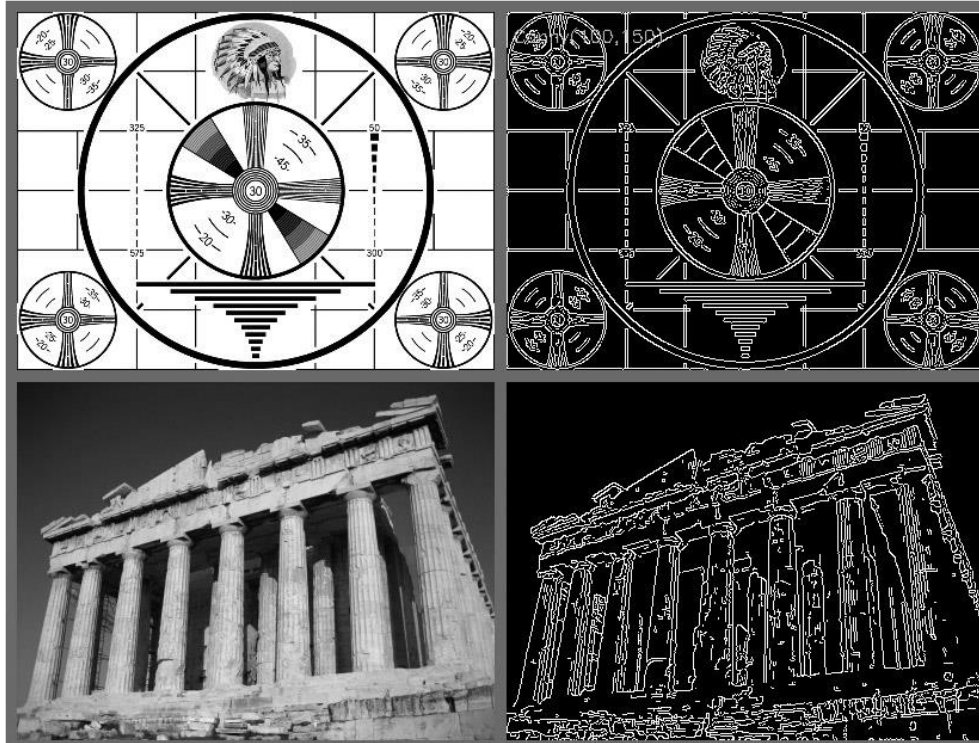
A kontúrok sokféle lehetőséget biztosítanak a gépi látás terén. Kontúrokat össze lehet hasonlítani egymással, a kontúr görbéket egyszerűsítve, hozzávetőlegesen megállapítani az alakjukat, azonban ami a későbbiekben a leghasznosabb lesz, az a lehetőség a kontúrok pozíciójának, valamint a kontúrokat körbevevő téglalapok méretének megállapítására. [4]

## 2.5 Canny éldetektálás

A canny éldetektálás széleskörben használt a gépi látás terén. A módszer első lépése a kapott kép szűrése, simítása, amelyet Gaussian szűrő segítségével hajt végre. Második lépése a kép gradiens mértékének és irányának kiszámítása, deriválás segítségével. A canny  $2 \times 2$ -es szomszédos területeket használ a számításra, így az azok közötti gradiens kiszámítható több irányban is (vízszintes, függőleges, átlós). Harmadik lépésben nem-maximum szűrés (non-maximum suppression – NMS) van alkalmazva, amely célja, hogy minden élt csak egy vonal képviseljen. [6]

A canny algoritmus következő lépése a dupla küszöbölés (duble-treshold), amely esetében van egy magas és egy alacsony küszöbérték. Amennyiben egy pixel gradiens nagysága meghaladja a magas küszöbértéket, akkor az a pixel egy él pontjaként lesz megjelölve. Ellenben, ha egy pixel gradiens értéke az alacsony határ alá esik, akkor az nem vonal elemként lesz megjelölve. A maradék pontok, amelyek a két határ közé estek,

lehetséges jelöltekként lesznek jelölve. Ezek a pontok csak akkor lesznek vonalelemként feljegyezve, amennyiben csatlakoznak más, biztosan vonalelemként jelölt pontokhoz. Ez a módszer segít a zaj hatások csökkentésében a vonalfelismerés terén. [6]



6. ábra: Canny éldetektálás  
Forrás: [3]

## 2.6 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]:

$$y = ax + b \quad (1)$$

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]:

$$(a, b) \quad (2)$$

A (2) síkon a vonal képlete [3]:

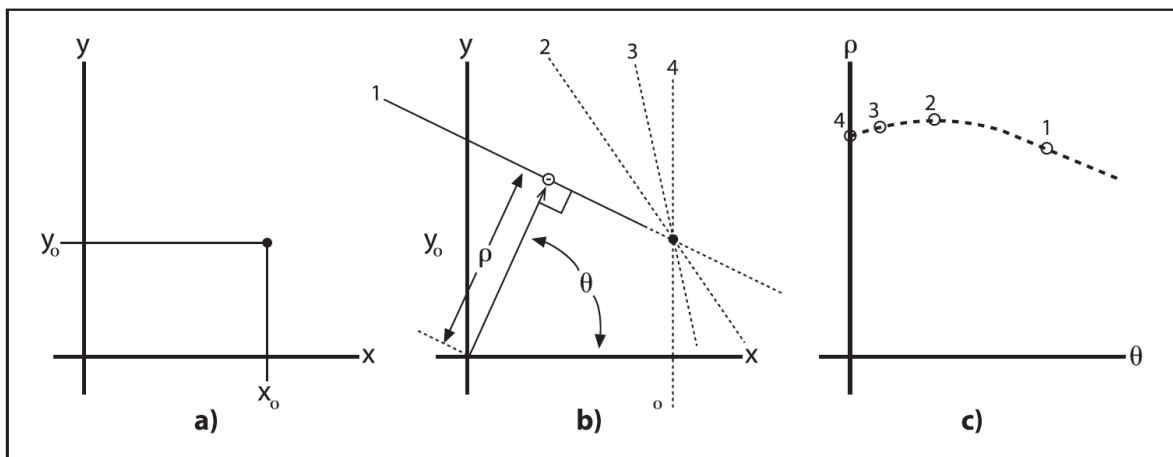
$$b = -xa + y \quad (3)$$

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  $\rho$  és  $\theta$  polárkoordináták segítségével. Ilyen esetben a következő képlet ír le egy vonalat [3]:

$$x \cos \theta + y \sin \theta = \rho \quad (4)$$

Ilyenkor  $\rho$  jelöli az origótól a vonalig vett legközelebbi távolságot, és  $\theta$  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]

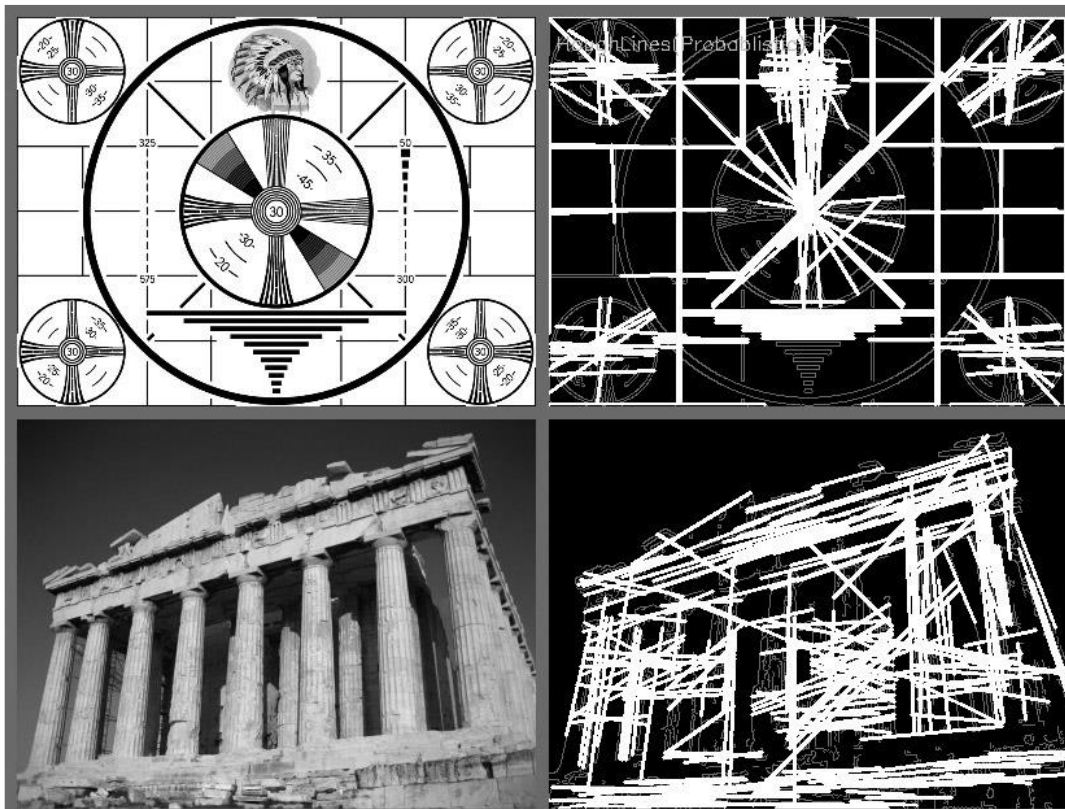


7. ábra: Egy ponton áthaladó lehetséges vonalak transzformációja a paraméteres síkra  
 Forrás: [3]

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 (8. ábra) 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 elérhető 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.



8. ábra: Valószínűségi Hough transzformáció (előzetes Canny éldetektálással)  
*Forrás: [3]*

## 2.7 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. [7]

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. [7]

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. [7]

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. [7]

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. [7]

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. [7]

## **2.8 Neurális hálózatok**

### **2.8.1 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. [8]

### **2.8.2 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. [8]

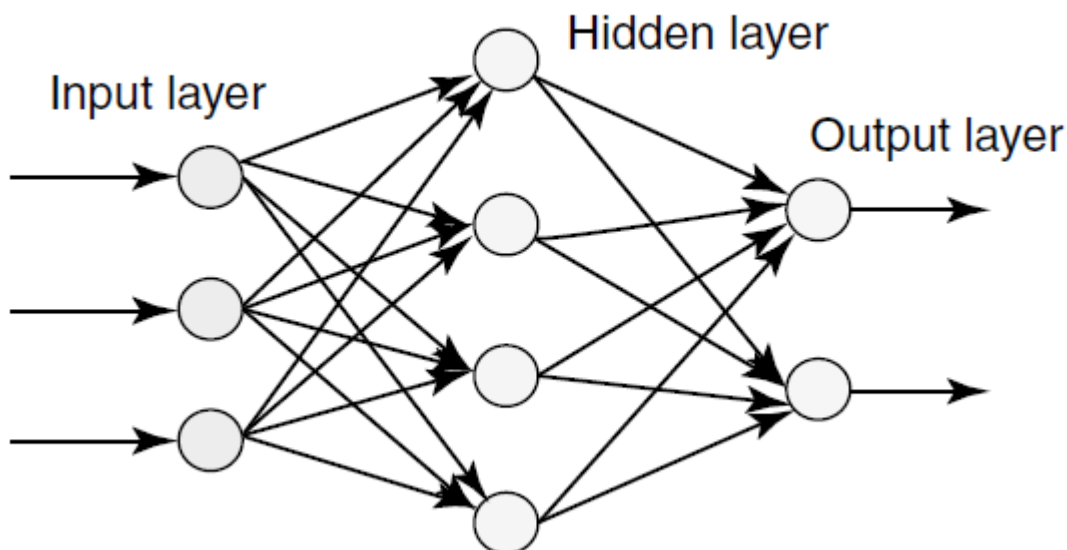
A mesterséges neurális hálózatok (ANN - Artificial Neural Network) a biológiai neurális hálózatok mintájára lett fejlesztve. Ezen hálózatok alap feldolgozó egységei a (mesterséges) neuronok. Ezek a neuronok is kommunikálnak egymással. A kommunikáció súlyozott kapcsolatok segítségével valósul meg. Egy bementet erős, azaz jelentős lesz, ha a súlyozása magas, azaz a súlyozás dönti el, hogy egy adott bemenetet mennyire jelentősen vesszünk 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. [8][21][9]

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. [10]

A neurális hálózatok neuronjait nem egyenként, hanem sok hasonló neuront egy réteggént kezelünk, amely rétegek saját funkcionalitással rendelkeznek. Egy neurális hálózat rendszerint három elkülöníthető részből tevődnek össze[8]:

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



9. ábra: Neurális hálózati modell

Forrás: [8]

A bemeneti réteg továbbítja a bemenetként kapott adatot a többi rétegnek. Ebben a rétegben a neuronok számát a bemeneti adat alakja határozza meg. [8]

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. [8]

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. [8]

A neurális hálózatokat, az architektúrájuk alapján 2 csoportba sorolhatjuk [9]:

1. Előrecsatolt (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. [9]

### **2.8.3 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. [8][9]

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. [8][9]



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. [8][9]

Ahhoz, hogy egy megbízható, robusztus 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. [8]

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. [8]

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). [8]

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. [8]

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. [8]

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. [8]

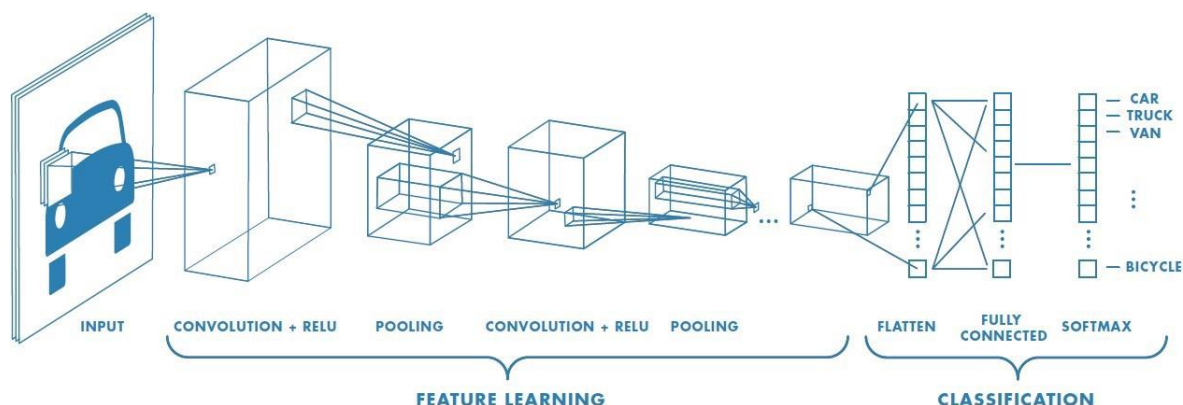
## ***2.9 Konvolúciós neurális hálózatok***

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 legfigyelemre méltóbb az orvosi téren való felhasználása, a képes leletek elemzése. A CNN előnye, a hagyományos neurális 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. [11]

A CNN a képek felismerésére specializált neurális hálózat, ami tipikusan 3 típusú rétegből tevődik össze [13]:

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). [13]



10. ábra: Konvolúciós neurális hálózat

Forrás: [12]

A konvolúciós réteg eltér az eddig ismertett 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 előbbieken 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  $3 \times 3$ , azonban a gyakorlatban előfordul  $5 \times 5$  vagy  $7 \times 7$  is. A kernelek száma tetszőleges, meghatározza a vizsgálható jellemzők mennyiségét. [11][13]

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. [11]

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. [11]

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. [11]

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  $2 \times 2$ . Ilyen esetben az első  $2 \times 2$ -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 tesz 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. [11][13]

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. [11]

A pooling által generált kimenet 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. [11][13]

## ***2.10 Hasonló munkák***

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

### 2.10.1 Primitív alakzatokra alapuló felismerés

Tudhope et. al [14] 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. [14]

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 egyes 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 [14]:

- 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 elkönnyveli egy felismert alakzatként. [14]

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 felismeréseknek. [14]

### **2.10.2 Hurok alapú felismerés**

Okazaki et. al [15] 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. [15]

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 keletkezhetnek) 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. Ezekről eltérően a hurokkeresés segítségével sok szimbólum megfelelően beazonosítható, vagy a pozíciójuk felfedhető. [15]

### **2.10.3 ANN alapú felismerés**

Rabbani et. al [16] 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. [16]

## ***2.11 Fejlesztői környezet, programcsomagok***

A projekt elkészítéséhez a Visual Studio Code [17] 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 [18] 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 [19] programcsomag segítette, amely a tömbök kezelését támogatja. A képek beolvasását, kezelését, valamint különféle képi transzformációk elvégzését az OpenCV [20] csomag segítette. A gépi tanulás, és neurális hálózatok használatát a tensorflow [21] könyvtár, valamint az abban lévő keras API tette lehetővé.

### 3 Megvalósítás

A projekt célja egy szoftver elkészítése volt, amely egy kapcsolási rajzot tartalmazó bemeneti képre egy olyan kimenetet biztosít, amely által a bemeneti kapcsolási rajz digitálisan szerkeszthetővé válik. Ez a gépi látás eszközeivel vált lehetségessé, amely segítségével elkészült egy program, amely egy bemeneti képet beolvas, feldolgoz, és legenerál egy kimeneti fájlt, amely tartalmazza a szoftver által felismert kapcsolási rajzot.

A szoftver működése, felismerési folyamata szétbontható 5 különböző részre:

1. Bemenet és előfeldolgozás
2. Komponensek és összeköttetések helyének beazonosítása
3. Komponensek felismerése, CNN
4. Összeköttetések azonosítása
5. Kimenet generálása

Ezen folyamatok főként egy programfájlban történnek meg (`circuit.py`), valamint ezt segíti néhány másik segédprogram, amelyek egy specifikus funkciót látnak el, például a konvolúciós neurális hálózat betanítását, saját képek alapján (`keras_train.py`).

#### 3.1 Program használatának előfeltételei

Az elkészült szoftver megfelelő működésének érdekében van néhány előfeltétel, aminek a felismerendő képnek meg kell felelnie.

A program olyan képekről ismer fel kapcsolási rajzokat, amelyen a kapcsolási rajz sötét színnel, a háttere pedig világos színnel van. Általános esetben, és ahogy a tesztek is készültek, ez fehér háttérű lapon, fekete színű kapcsolási rajzot jelent. A képen csak a kapcsolási rajz lehet, valamint a lap háttere. A program működőképes, ha a lap mögötti háttér világos, amennyiben nincs valami sötétebb árnyalat (más objektum, árnyék) a képen, ami bezavarhat a binarizálás során, és elronthatja a transzformációt, a felismerést.

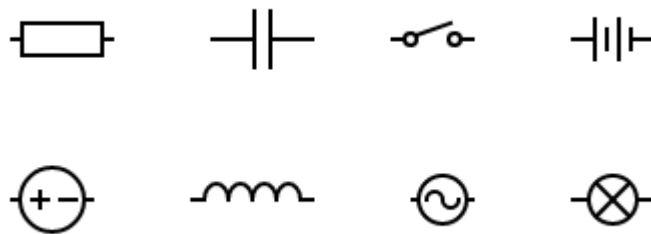
A program jelenleg jobban működik nagyobb, a tesztekénél is kifejtett felbontású képek esetében, amely több információt hordoz a kapcsolási rajzról. Kisebb felbontású képek esetén ajánlott a program eleji paraméterek változtatása.

Jelenleg 8 komponens (11. ábra) felismerésére van lehetőség, ez azonban bővíthető a konvolúciós neurális hálózat további tanításával.



A felismerhető komponensek a következők:

- ellenállás (resistor)
- kondenzátor (capacitor)
- kapcsoló (switch)
- elem (battery)
- egyenáramú áramforrás (DC source)
- tekercs (inductor)
- váltakozó áramú áramforrás (AC source)
- lámpa (lamp)



11. ábra: Felismerhető komponensek

### 3.2 *Segédosztályok, konstansok*

A program elején létre vannak hozva különböző egyszerű segédosztályok (1. kódrészlet), amelyek az adatok feldolgozásában segítenek, valamint a kód olvashatóságát javítják, a hibakeresést könnyítik. Az osztályok nem a hagyományos értelemben vett osztályok, inkább címkézett listák. Céljuk, hogy egy ilyen létrehozott elem adatait lehessen név alapján címezni, ne csak index alapján. Segítségükkel egy  $(x; y)$  koordinátával rendelkező pontot le lehet menteni úgy, hogy egy listában az első elem az  $x$ , a második az  $y$ , azonban hivatkozni rá lehet  $x$  vagy  $y$  változóként, nem csak a lista 0. vagy 1. elemeként.

Ezen kívül van egy enum is, amely célja a különböző orientációk nyilvántartása. Általa könnyen letárolható, hogy egy komponens/vonal vízszintes vagy függőleges, valamint a kód olvashatóságán is javít.

## 1. kódrészlet: Segédosztályok

```
class Orientation(Enum):
    HORIZONTAL = 0
    VERTICAL = 1

class Point(NamedTuple):
    x: int
    y: int

class Line(NamedTuple):
    p1: Point
    p2: Point

class Endpoint(NamedTuple):
    point: Point
    orientation: Orientation
```

A programban sokféle szereplő algoritmusok, függvények működéséhez sok paraméter kell. A programon való egyszerűbb szerkeszthetőség érdekében ezen paraméterek nagy része a program elején változókból vannak eltárolva (2. kódrészlet). Ezeket változtatva lehet a programot finomhangolni, felkészíteni minél több eshetőségre, vagy beállítani specifikusan egyféle eset minél hatékonyabb kezelésére. A paraméterek befolyásolják a felismerő algoritmusok, valamint az azok adatait feldolgozó algoritmusok működését, szűrők hatékonyságát, valamint a megjelenítést és a kimenetet is.

## 2. kódrészlet: Konstans paraméterek

```
POINT_SIMILARITY_COMPARE_AREA_RADIUS = 15
LINE_MIN_LENGTH = 90
LINE_PERSPECTIVE_MIN_LENGTH = 200
LINE_COUNT_CHECK_FOR_ROTATION = 10
LINE_SEARCH_ANGLE_THRESHOLD = 5
LINE_CHECK_SIMILARITY_THRESHOLD = 15
LINE_AGGREGATION_SIMILARITY_THRESHOLD = 25
COMPONENT_OTHER_ENDPOINT_SEARCH_WIDTH = 50
COMPONENT_OTHER_ENDPOINT_SEARCH_MAX_LENGTH = 350
COMPONENT_OTHER_ENDPOINT_SEARCH_MIN_LENGTH = 16
COMPONENT_MIN_BOX_SIZE = 200
COMPONENT_BOX_SIZE_OFFSET = 60
COMPONENT_PIXELS_THRESHOLD_PERCENTAGE = 6
PERSPECTIVE_IMAGE_OFFSET = 200
OUTPUT_POINT_SIMILARITY_COMPARE_AREA_RADIUS = 30
OUTPUT_SCALE = 3
PICTURE_SCALE = 20

COMPONENT_OUTPUT_SIMILARITY_THRESHOLD = OUTPUT_SCALE * 10
```

Ahhoz, hogy a program minél több eshetőségre tudjon megfelelő eredményt produkálni, néhány paraméter a későbbiekben hozzá van igazítva a kép méretéhez (3. kódrészlet). A

különböző paraméterek a kép nagyobb oldalához mérten arányosan vannak állítva, hogy különböző bemenetek esetén is tudjon működni.

### 3. kódrészlet: Paraméterek dinamikus állítása

```
biggerSide = img.shape[0] if img.shape[0] > img.shape[1] else img.shape[1]

POINT_SIMILARITY_COMPARE_AREA_RADIUS = round(biggerSide*0.00375)
LINE_MIN_LENGTH = round(biggerSide*0.0275)
LINE_CHECK_SIMILARITY_THRESHOLD = round(biggerSide*0.00375)
COMPONENT_OTHER_ENDPOINT_SEARCH_WIDTH = round(biggerSide*0.0125)
COMPONENT_OTHER_ENDPOINT_SEARCH_MAX_LENGTH = round(biggerSide*0.0875)
COMPONENT_MIN_BOX_SIZE = round(biggerSide*0.05)
COMPONENT_BOX_SIZE_OFFSET = round(biggerSide*0.015)
OUTPUT_POINT_SIMILARITY_COMPARE_AREA_RADIUS = round(biggerSide*0.0075)
```

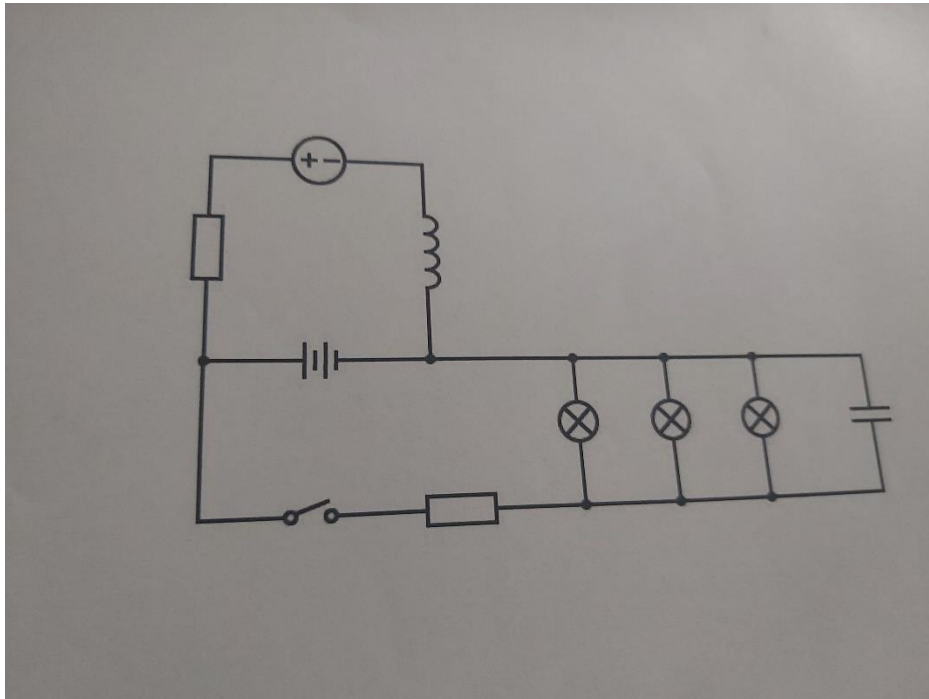
## 3.3 *Bemenet és előfeldolgozás*

A program elején, a bemenet beolvasásához, valamint előfeldolgozásához az opencv eszközei vannak használva, amelyek segítségével a képet felkészíti a szoftver a kapcsolási rajz elemeinek felismerésére. Az előfeldolgozás során a beolvasott kép egyszerűbb formára lesz alakítva, vagyis a cél az adatvesztés nélküli egyszerűsítése a képnek. Ezen kívül különféle transzformációk is végre lesznek hajtva, amelyek a felismerést segítik.

### 3.3.1 Kép beolvasása, egyszerűsítése

A program először ellenőrzi, hogy milyen paramétereket kapott az indításakor. Egy paraméter kötelező, ennek hiányában a program nem fut tovább. Ez a paraméter a felismerésre szánt kép neve. Amennyiben a kép nem a program mappájában van, az elérési utat is meg kell adni. A program először ellenőrzi, hogy lett-e megadva paraméter, amelyet utána meg is vizsgál. Amennyiben nincs megadva, vagy nem létező file-ra mutató elérési útvonal lett megadva, a program leáll.

Amennyiben megfelelő a paraméter, a program futása folytatódik, és beolvassa, és eltárolja a képet (12. ábra). A kép beolvasása az opencv imread függvényének segítségével történik. Az imread által beolvasott kép egy numpy tömb formájában kerül tárolásra, „img” néven.



12. ábra: Bemeneti kép minta

A beolvasott képen utána megkezdődik az előfeldolgozás, amelynek a célja a kép minél egyszerűbbre való alakítása a könnyebb felismerés érdekében úgy, hogy minél kevesebb adat vesszen el a képről.

Miután a kép beolvasása sikeresen megtörtént, utána az első lépés a kép szürkeárnyalatossá alakítása. Ezáltal a kép nem 3 különböző szín adat alapján lesz tárolva, hanem egy adat alapján, amely az adott pixel intenzitását adja meg. A folyamat által a kép fel lesz készítve a további egyszerűsítésre. Szürkeárnyalatossá alakítás után az új kép külön lesz eltárolva, egy „gray” nevű numpy tömbben.

Az átalakítás után a szürkeárnyaltos képen egy medianBlur szűrő van alkalmazva. A szűrő egy  $3 \times 3$ -as, négyzet alakú szűrőt használ. A medián szűrő által minden pixel értéke ki van cserélve a környező pixelek medián (középső) értékére, ezáltal enyhén elmosódás hatást keltve a képen. Segítségével az esetleges zaj eltávolítható a képekről. [3]

A szűrés után történik a legfontosabb előfeldolgozási folyamat, a thresholding. A cél, hogy elkülönítsük a hasznos pixeleket (amelyek a kapcsolási rajzot alkotják) a háttértől. A program az opencv threshold függvényét használja. Ez úgy van paraméterezve, hogy minden pixelt, amelynek a szürkeárnyaltos képen 110-nél nagyobb az intenzitása, azt állítsa 255-re, azaz fehérre, minden mást pedig 0-ra, vagyis feketére. Ezzel elértem, hogy a háttér fehér,

a kapcsolási rajz elemei pedig fekete pixelekből áll. A későbbi feldolgozás miatt ezt azonban meg kell fordítani. Minden pixel új értékét úgy kapjuk meg, hogy 255-ből ki van vonva a pixel értéke, így 255-ből 0 lesz, és fordítva. Ezt a folyamatot a numpy-nak köszönhetően egy sorban meg lehet csinálni, ahogy a kódban látható.

Az újonnan létrejött képen (thresh) a hasznos pixelek fehér színnel, vagyis 255 értékkel léteznek. Ezen a képen végrehajtódik egy morfológiai zárás, amely célja a kapcsolási rajz elemein, de leginkább az összekötő vonalakon esetlegesen keletkezett kihagyások pótlása. Ilyen kihagyások keletkezhetnek a kép minősége, vagy az átalakítások következtében is. A morfológiai zárás szintén egy négyzet alakú,  $3 \times 3$  méretű kernelt használ, amely által a kis hibákat kijavítja, azonban jelentősen nem alakítja át az adatokat, nem rontja el a kapcsolási rajzot.

#### 4. kódrészlet: Bemenet és előfeldolgozás

```
if len(sys.argv) < 2:
    exit("Not enough parameter given!")

if not os.path.isfile(str(sys.argv[1])):
    exit("The file does not exist!")

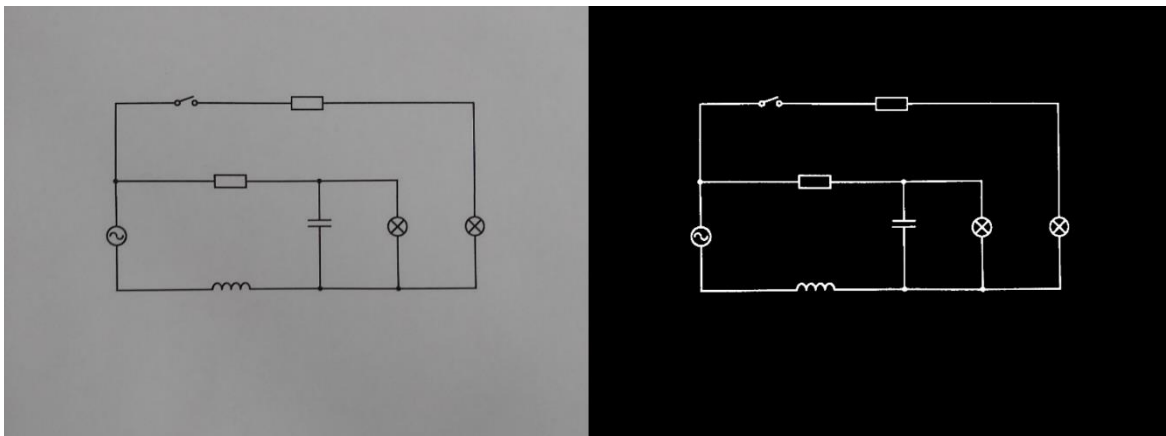
img = cv2.imread(sys.argv[1])
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

gray = cv2.medianBlur(gray, 7)

thresh = cv2.threshold(gray, 110, 255, cv2.THRESH_BINARY)[1]

thresh = 255 - thresh

gray = cv2.morphologyEx(gray, cv2.MORPH_CLOSE,
cv2.getStructuringElement(cv2.MORPH_RECT, (3,3)))
```



13. ábra: Kép előfeldolgozás. Eredeti (bal) és feldolgozott (jobb)

### 3.3.2 Kép transzformációi

Az előfeldolgozás során a képek nem csak leegyszerűsítve vannak, hanem különféle transzformációk is végre vannak hajtva, amelyek nem egyszerűsítik a képeket, nem célzottan csökkentik a feldolgozandó adatokat, viszont úgy alakítják át a képeket, hogy a felismerés a későbbiekben minél megbízhatóbban működhessen.

#### 3.3.2.1 *Madárnézet kialakítása, perspektíva transzformáció*

Mivel egy kapcsolási rajzon a komponensek, és összeköttetések vízszintes és függőleges orientációban vannak, ezért logikus egy ilyen képet is megfelelő forgatási szögből vizsgálni. Egy kép készítése során nem lehet biztosra menni, hogy a képen a kapcsolási rajz tökéletesen lesz. Nem lehet garantálni, hogy a kép tökéletesen felülnézetből készül, hogy a kép nem lesz elforgatva. Az ilyen anomáliák kiküszöbölése miatt fontos a kapcsolási rajz esetében a madárnézet (bird's eye view) kialakítása.

A folyamat a perspektíva transzformáció segítségével valósul meg. A transzformáció az opencv által biztosított függvények segítségével van megoldva. Ezekhez a függvényekhez először meg kell állapítani, hogy a képen hol van a kapcsolási rajz, és az milyen orientációban van. Ezt a programban a vonalkeresés által lett megoldva.

Mivel egy kapcsolási rajzon minden elemnek kapcsolódnia kell más elemekhez, és ezek a kapcsolatok vonalakkal vannak jelölve, ezért lehet feltételezni, hogy a kapcsolási rajz minden oldalát, azaz tetejét, alját, bal és jobb oldalát is legalább részben vonalak alkotják. Ezt kihasználva, a program megkeresi a kapcsolási rajz minden oldalához tartozó, legszélső vonalakat, azaz amelyek nagyrészt határolják a rajzot.

A vonalkeresés a HoughLinesP függvény segítségével van megvalósítva, amely a canny által detektált éleket azonosítja be (5. kódrészlet). Ahogy a kódrészleten is látható, a Canny függvény várja a vizsgálandó képet, és 2 paramétert. A kép jelen esetben a threshold által létrejött kép. A két szám a canny által használt alsó és felső küszöbhatár. Mivel a threshold képen minden pixel vagy fekete, vagy fehér, ezért a gradiens vagy nagyon magas, vagy nagyon alacsony (0), mivel minden pixel között csak két lehetőség van: vagy van, vagy nincs változás. Emiatt jelen esetben a canny paraméterezése nagyrészt tetszőleges.

Annak érdekében, hogy a vonalfelismerés megfelelően működhessen, jó eredményeket adjon, a vonalakat, a felismerés előtt dilatáció van végrehajtva, amely megvastagít minden vonalat. A HoughLinesP úgy van paraméterezve, hogy mindenféle szögben felismerhessen

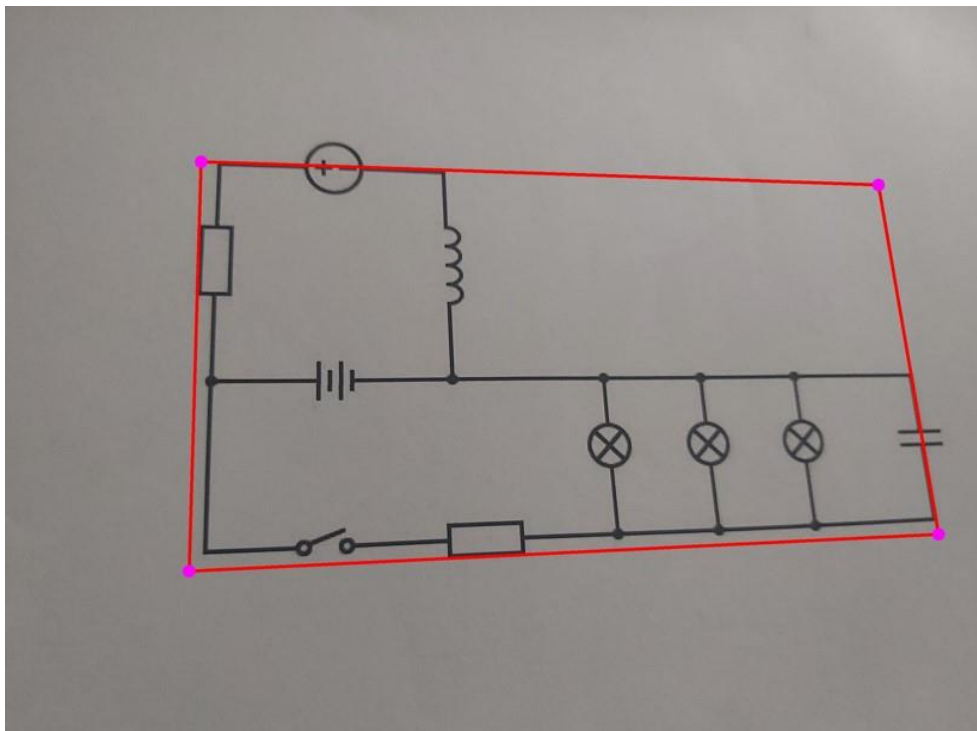
vonalakat, azonban meg van adva egy minimum vonalhossz, amit el kell érnie egy vonalnak, hogy azt beszámíthassa a program a transzformációhoz. Ezt a minimum hosszt egyik, a program elején értéket kapó változó befolyásolja.

#### 5. kódrészlet: Madárnézet vonalkeresés

```
canny = cv2.Canny(thresh, 100, 150)
canny = cv2.dilate(canny, cv2.getStructuringElement(cv2.MORPH_RECT, (3,3)))

linesP = list(cv2.HoughLinesP(canny, 1, np.pi/180,
LINE_PERSPECTIVE_MIN_LENGTH, None, LINE_PERSPECTIVE_MIN_LENGTH, 0))
```

Miután megkaptuk a vonalakat, kiszűrjük belőlük a legfelső és legalsó vízszintes, valamint a legszélső, bal és jobb oldali függőleges vonalakat. Ez a négy vonal fogja körbe a kapcsolási rajzot. Miután megkapjuk a pontokat, ahol ezek a vonalak keresztezik egymást, megkapjuk annak a négyszögnek a négy sarkának koordinátáit, amely négyszög magába foglalja a kapcsolási rajzot, annak a dőlésszögében (14. ábra).



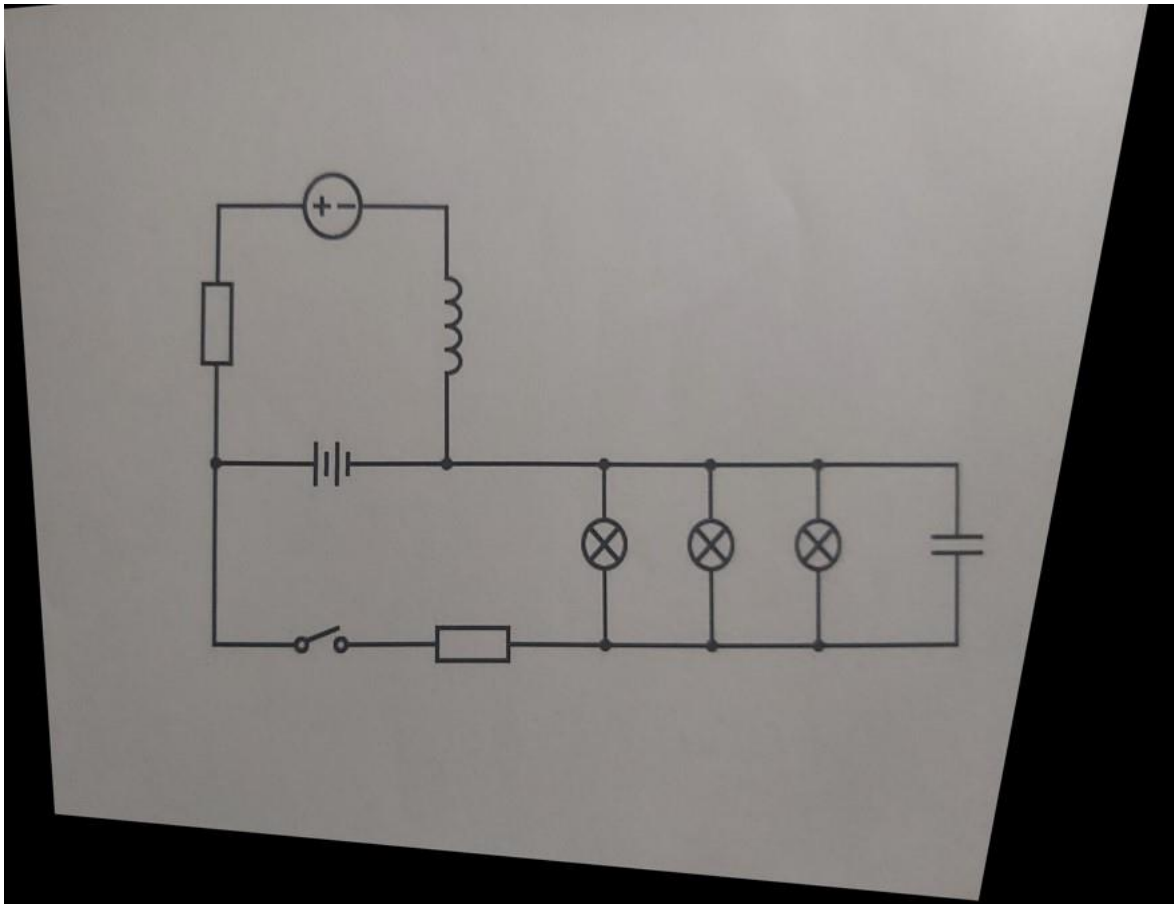
14. ábra: Kapcsolási rajz helyzete, dőlési szöggel

A következő, és utolsó lépés maga a transzformáció végrehajtása (6. kódrészlet). A kapcsolási rajz helyzetének ismeretében, a köré rajzolt négyszög négy sarkának koordinátáiból (pts1), valamint egy hasonló szélességű, és hosszúságú téglalap koordinátái alapján (pts2) az opencv getPerspectiveTransform függvényének segítségével kiszámítható

a perspektíva mátrix. Ezt a mátrixot felhasználva, a warpPerspective függvény segítségével a kép, valamint a threshold kép is átformálható madárnézetű képpé (15. ábra).

#### 6. kódrészlet: Perspektíva transzformáció

```
M = cv2.getPerspectiveTransform(pts1,pts2)
img = cv2.warpPerspective(img,M,(img.shape[1], img.shape[0]))
thresh = cv2.warpPerspective(thresh,M,(img.shape[1], img.shape[0]))
```



15. ábra: Madárnézetű kapcsolási rajz

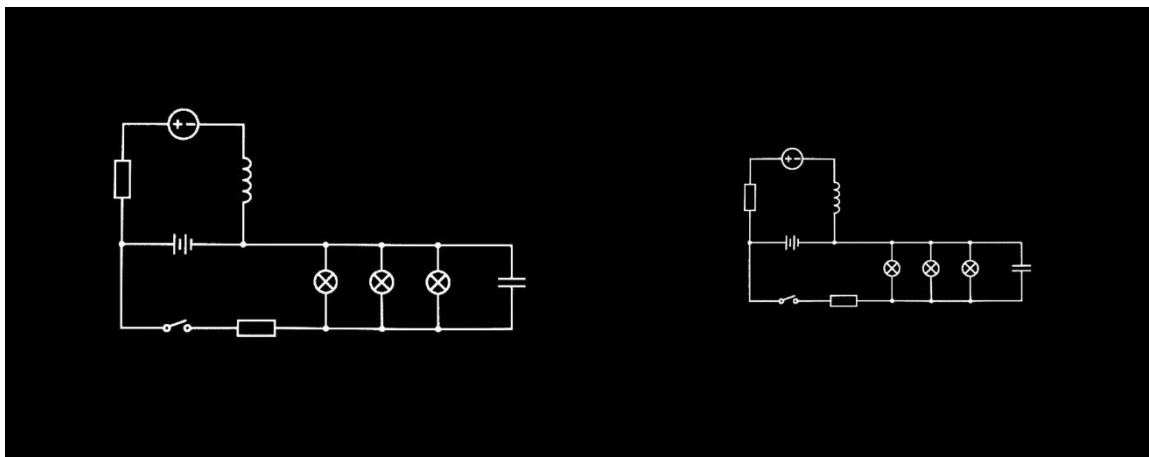
#### 3.3.2.2 Átméretezés, vászon

A program lehetőséget ad a beviteli kép kisebb méreten való feldolgozására. Mivel a kapcsolási rajzról készített képek eltérő távolságból lehetnek fotózva, ezért probléma lehet a program felismerési algoritmusok számára egyes elemek keresése. A túl közelről fotózott képek esetén egy vonal a képen sokkal több pixelt tartalmaz, mint egy távolabbról készített képen. Annak érdekében, hogy a szoftver jól működhessen ilyen körülmények esetén, a program kaphat egy második bemeneti paramétert. Ez a paraméter egy -5 és 5 közé eső egész szám lehet. 0 fölött, minél nagyobb a szám, annál kisebbre lesz méretezve a bemeneti kép.



Amennyiben 0-nál kisebb a szám, akkor a kép nagyobbra lesz méretezve.

A képnek a folyamat során nem változik a mérete. Ez amiatt van, mivel az átméretezett kép rákerül egy akkora vászonra, amekkora alpból volt a bemeneti kép. A vászon fekete színű, azaz 0 értékeket tartalmaz. A folyamat után a threshold kép (a forgatáshoz hasonlóan) látszólag csak egy nagyobb keretet kap, ezáltal módosítás, átalakítás nélkül lehet rajta tovább dolgozni.



16. ábra: Átméretezés előtt (bal) és után (jobb)

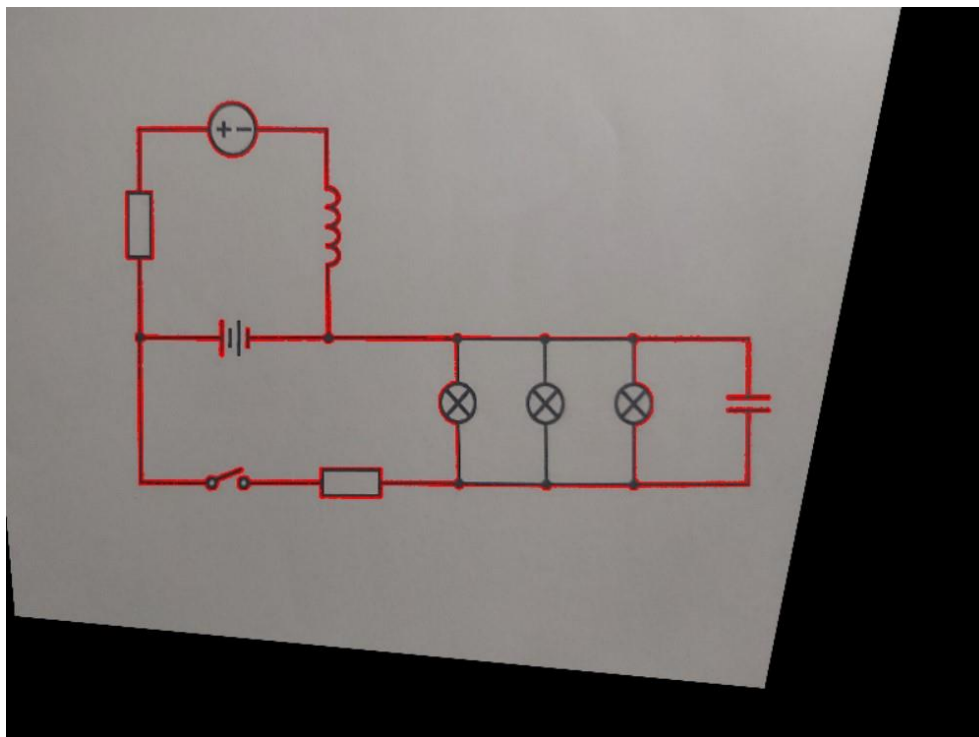
Azáltal, hogy a méretezés során, az átméretezett kép az eredeti kép méretével megegyező vászonra kerül, nem csak átméretezni lehet a képet, hanem a kép széleitől messzebb vinni. Ezáltal, ha több kép ugyanolyan minőségben készült, és a program arra a minőségre jó eredményeket produkál, akkor ugyanazzal a paraméterezéssel végig lehet futtatni a programot a képeken úgy, hogy ha néhány kép túl közelről lett készítve, akkor azok esetén a program általi átméretezés után is ugyanazokkal a jól beállított paraméterekkel fel lehet ismertetni a képet.

### 3.3.2.3 Kimenet eltolás

A program a futása végén generál egy kimeneti file-t, amely egy kapcsolási rajz digitális modelljét tartalmazza. Annak érdekében, hogy ez a kimenet minél jobban legyen megjelenítve, az előfeldolgozás végén meg lesz határozva egy eltolási értékek (offset). Ezek az értékek megadják, hogy a képen mekkora távolságoktól (x és y távolság) kezdődik a kapcsolási rajz. Ennek a haszna az, hogy a kimeneten a kapcsolási rajz a 0-hoz közeli x és y koordinátákhoz legyen közel, ne a kép koordinátáihoz legyen viszonyítva.

A rajz helyének meghatározásához a kontúrkeresés lett alkalmazva. A kontúrkeresés

által megtalálható a képen minden kontúr, azaz minden egybefüggő alakzat. Ezen alakzatok közül a program kikeresi azt a legnagyobb kontúrt, amelyik nem egyezik meg a kép méretével. Ez a kontúr lesz maga a kapcsolási rajz, és ezen kontúr helyzetének koordinátái szolgálnak alapul az eltolási értékek számára.



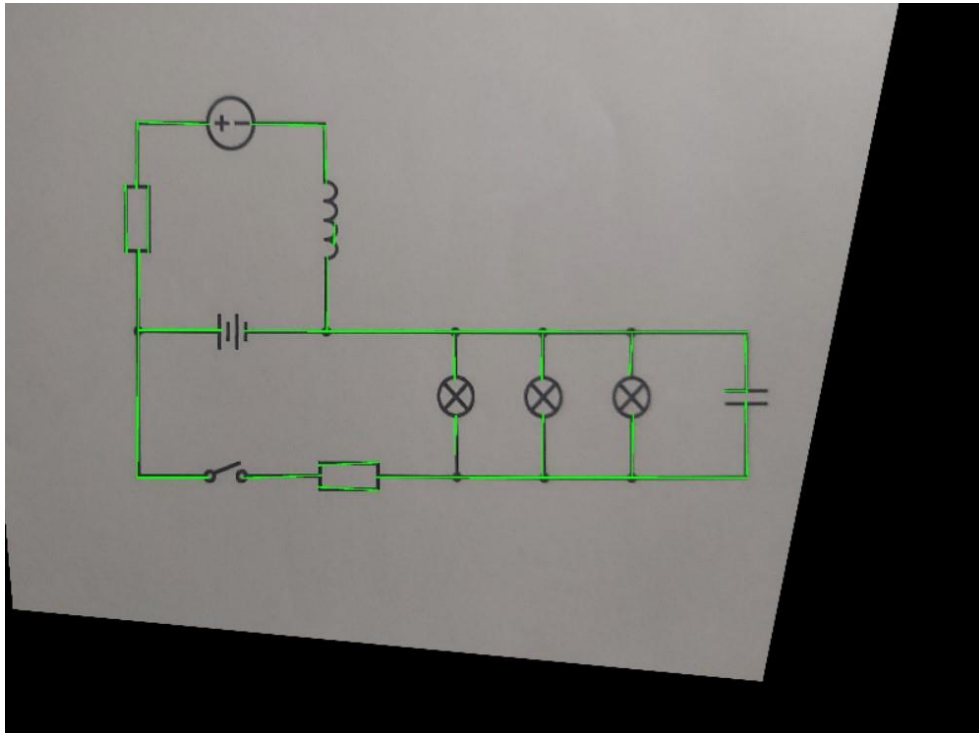
17. ábra: Kontúrkeresés eredménye

### ***3.4 Komponensek és összeköttetések helyének beazonosítása***

#### **3.4.1 Vonalak felismerése, feldolgozása**

A komponensek helyének beazonosítása, valamint az összeköttetések keresése is a vonalkeresésen alapszik. Az előfeldolgozás, valamint a kép transzformálása után egy újabb HoughLinesP metódus segítségével keres vonalakat a program. Az előfeldolgozás során a transzformációk fő célja az volt, hogy ez a vonalkeresés minél jobban működhessen, mivel ez egy kritikus lépés a program szempontjából. Mivel ez egy fontos lépés, ha nem talál vonalat a szoftver, akkor kilép.

Amennyiben sikeres volt a vonalkeresés, akkor a függvény által visszaadott vonalaknak ismert lesz a kezdő-, és végpont koordinátái. Ezen koordinátákból dolgozva a vonalak szűrésen esnek át.



18. ábra: Vonalkeresés eredménye

A szűrés közben egyrészt megkülönböztetjük a vízszintes és függőleges vonalakat egymástól. Ez megkönnyíti a következő folyamatot, amely a hasonló vonalak egybevonása. A folyamat által megpróbálja a program kiszűrni az egymást átfedő vonalakat. Ilyen átfedés bekövetkezhet azért, mert a vonalkeresésnél egy vonalszakasz több vonalként lett felismerve, vagy egy szakaszt több vonal ír le. Ez tapasztalat szerint akkor következik be, amikor egy vonal túl vastag, és több vonalat is le lehet benne írni.

A vízszintes és függőleges vonalak megkülönböztetése egy egyszerű módszerrel történik. Amennyiben egy vonal egyik végpontját  $x_1$  és  $y_1$ , a másik végpontját  $x_2$  és  $y_2$  koordináták írják le, akkor a következő egyenlőtlenség (5) kerül vizsgálatra:

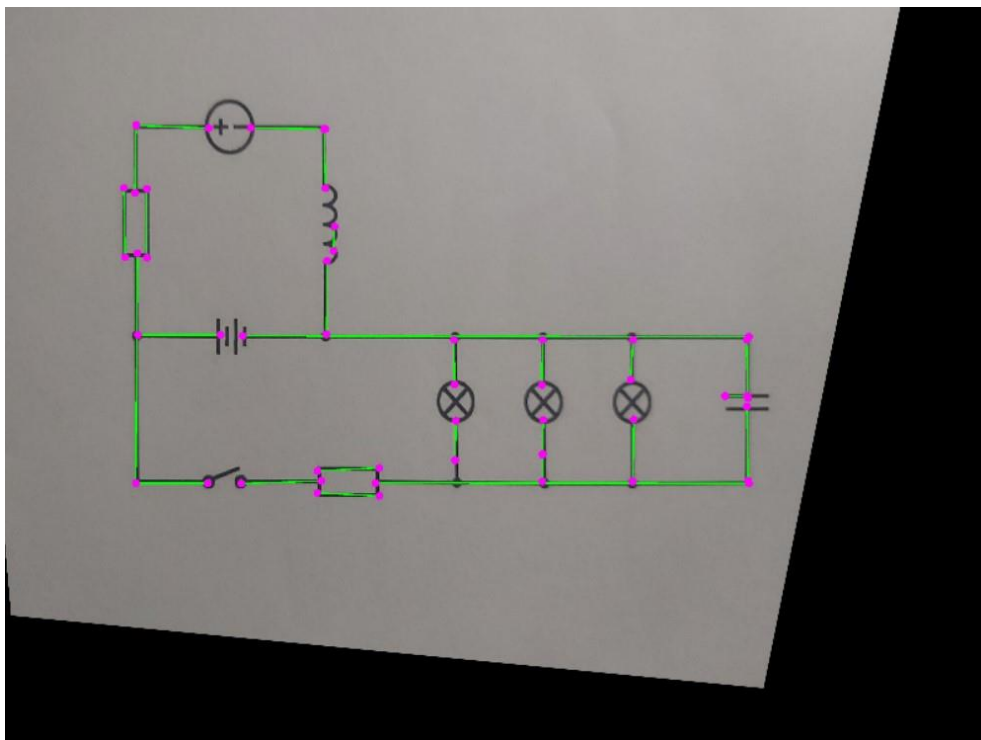
$$|x_2 - x_1| > |y_2 - y_1| \quad (5)$$

Amennyiben a vízszintes koordináták különbsége nagyobb, mint a függőleges koordináták különbsége, akkor a vonal vízszintes, ellenben függőleges.

Hasonló vonalak szűrésekor azt vizsgáljuk minden új vonalnál, hogy az eddig feldolgozott, és ugyanolyan orientációjú vonalak közül van-e olyan, amely átfedi, vagy közel átfedi a vizsgált vonalat. Amennyiben átfedi, akkor a két vonal helyett egy vonal kerül tárolásra, amely a két vonal összekapcsolásából keletkezik.

Az átfedés vizsgálata úgy történik, hogy két vonal esetén, ha a két vonal hosszanti irányú koordinátái átfedésben vannak, akkor meg van vizsgálva, hogy a vonalak ugyancsak hosszanti irányban mért közepén milyen koordinátában van a másik irányú koordinátájuk, azaz ezen a ponton egy vízszintes vonalnál a függőleges koordináta van vizsgálva, és fordítva. Amennyiben a vonalközepe koordináták a két vonal esetén egy határértéken belül vannak, vagyis a két vonal nem csak hosszanti irányban fedi el egymást, hanem nagyon közel is vannak egymáshoz, akkor a két vonal egybe lesz dolgozva.

Az utolsó lépés a vonalak feldolgozásánál, a végpontok szétválogatása. Annak érdekében, hogy a későbbi műveleteket könnyebben el lehessen végezni, a vonalak végpontjai szét vannak válogatva, kategorizálva vannak külön listákba. A válogatás aszerint történik, hogy milyen irányba néznek. Így 4 lista van, 4 különböző irányú végpontoknak. A vízszintes vonalak szét vannak osztva bal és jobb oldali, a függőleges vonalak felső és alsó irányú végpontokká.

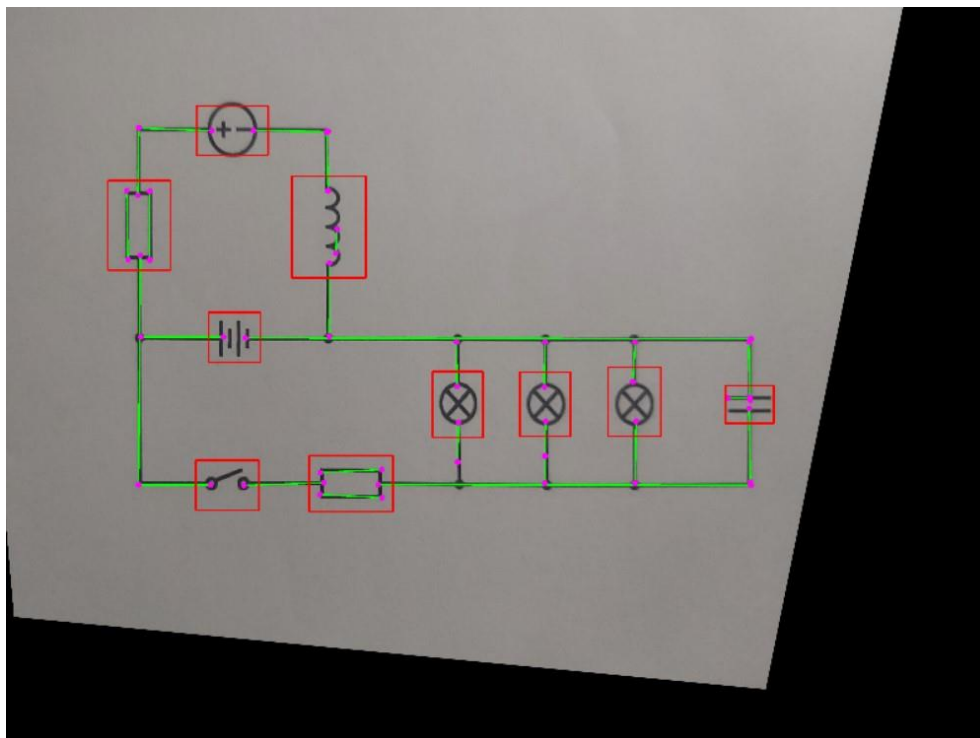


19. ábra: Végpontok azonosítása

### 3.4.2 Komponensek keresése

A komponensek keresése a vonal végpontok alapján történik. A szétválogatás után a jobbra és lefelé néző végpontok vannak vizsgálva. Mindkét esetben a cél egy másik, ellenkező irányba néző végpont keresése. Ezt a folyamatot nagyban segíti a végpontok

szétválogatása, mivel célirányosan lehet keresni az ellenkező irányba néző pontokat. A keresés során a végpont előtt egy téglalap formában van keresve a szembenéző végpont. A téglalap szélessége, hossza, valamint minimum távolsága a kezdőponttól meg van határozva paraméterként a program elején. Amennyiben találat volt egy végpont esetén, még egy vizsgálat végre van hajtva: amennyiben a talált területben a hasznos, azaz nem 0 pixelek száma nem halad meg egy adott határértéket (a komponens méretéhez arányosan), akkor az nem lesz komponensként eltárolva. Ennek célja a hamis találatok szűrése. Amennyiben átesett a szűrőn, akkor a két végpont közötti hely komponensként lesz eltárolva, és a két végpont utána ki lesz véve a vizsgálandó pontok közül.



20. ábra: Talált komponensek

Egy komponens esetén le van tárolva a helyének koordinátái, az orientációja (vízszintes/függőleges), valamint a komponenshez csatlakozó végpontok koordinátái. Az utóbbi külön listában is el van tárolva, mivel a későbbiekben hasznos lesz tudni, hogy egy végpont komponenshez tartozik-e vagy nem.

A komponensek letárolása közben, az újabb komponensek esetén mindig megvizsgálja a program, hogy az új elem hogyan viszonyul a többi komponenshez. Amennyiben igazodik, vagyis közel ugyanolyan koordinátán fekszik, mint egy hasonló orientációjú komponens, akkor ennek a koordinátáit a már letárolt komponens koordinátáihoz igazítja. Vízszintes

fekvésű komponensek esetén az y, függőleges esetén pedig az x koordináta van igazítva. Ennek segítségével elkerülhető olyan hiba, ha a kép nem tökéletesen egyenesre lett transzformálva, és a felismeréskor egy kicsit elcsúsztatott koordinátákkal lesz mentve a komponens, akkor se legyen olyan, hogy a kimeneten az egyik komponens (valamint összeköttetései) elcsúszva jelenjenek meg.

## **3.5 *Komponensek felismerése, CNN***

### **3.5.1 *Komponensek feldolgozása***

Ezen a ponton a komponensek helye már ki van gyűjtve listába, így egyszerűen a listán lévő elemeket kell egyesével felismertetni. Minden listaelem, vagyis komponens esetén a kigyűjtött pozíciójuk alapján egy kép ki van vágva a beolvasott képből. Ez a kis kép tartalmazza a felismerésre szánt komponenst.



21. ábra: Kivágott komponens kép példa

A kivágás után a képen még transzformációk vannak végrehajtva, amelyek felkészítik a felismeréshez. Az egyik ilyen transzformáció a kép átméretezése  $150 \times 150$  méretűre, mivel ezzel a mérettel működik a CNN. A másik transzformáció csak akkor történik meg, ha a komponens függőleges orientációban van. Ilyen esetben  $90^\circ$ -al el van forgatva, az óramutató járásával ellentétes irányba. Ez azért kell, mert a konvolúciós neurális hálózat úgy lett betanítva, hogy csak vízszintes komponenseket ismerjen fel. Ennek oka a következő lépések után érthető lesz.

Miután a transzformációk el lettek végezve, a kapott kép fel van ismertetve a neurális hálózattal. Ennek eredménye egy lista, amely tartalmazza az összes felismerhető elemhez tartozó becslési értéket. Amelyik érték ezek közül a legnagyobb, azt minősítjük a felismert alakzatnak. A becslés után le van mentve a becslés szerinti komponens, valamint a becslési érték.

A következő lépés a már becslésre küldött kép tükrözése a függőleges tengely mentén. Ez a tükrözött kép ismét fel van ismertetve a CNN (3.5.2 fejezet) által. Ennek a célja, hogy el lehessen dönteni, hogy a komponens melyik irányba néz. Alapvetően a végpontok alapján megállapítható, hogy vízszintes vagy függőleges irányban van-e a komponens, azonban azon belül nem tudni, hogy bal/jobbról, fel/le irányba néz. Emiatt a CNN úgy lett betanítva, hogy olyan komponenseket ismerjen fel, amelyek horizontálisak, és egy irányba (jobbra) néznek. Ezáltal, ha a tükrözés után a becslési érték nagyobb, mint az első érték, vagyis tükrözés után biztosabban meg lett állapítva a komponens típusa, akkor el van tárolva, hogy ezt a komponens-t a kimeneten majd tükrözve kell megjeleníteni.

### 3.5.2 CNN

A komponensek felismerésére egy konvolúciós neurális hálózat (CNN) van használva. A hálózat saját képeket felhasználva tanítja magát.

A tanításra szánt képek egy mappában vannak, amelyen belül minden felismerendő komponens képei külön-külön mappákban vannak elrendezve. Minden kép  $150 \times 150$  pixel nagyságú, és mindegyik előzetesen átesett a küszöbölésen, így fekete-fehér képek lettek.

A képekből először adatkészletet (dataset) kell generálni (7. kódrészlet). A keras segítségével, a képek az előre megadott mappából (images\_path) ki vannak olvasva, feldolgozva, és két adatkészlet lesz generálva, hasonló módon: egy tanító (training), és egy érvényesítő (validation) készlet.

#### 7. kódrészlet: Tanító adatkészlet generálás

```
ds_train = tf.keras.preprocessing.image_dataset_from_directory(  
    images_path,  
    labels="inferred",  
    label_mode="int",  
    color_mode='grayscale',  
    batch_size=batch_size,  
    image_size=(img_height,img_width),  
    shuffle=True,  
    seed=123,  
    validation_split=0.2,  
    subset="training",  
)
```

Az adatkészlet létrehozásakor többféle paraméter meg van adva:

- képeket tartalmazó mappa
- címkézés típusa (infered: a mappaszerkezet alapján)
- címkézés módja (int: minden komponens egy szám jellemez)
- színmód (grayscale: szürkeárnyaltos)
- batch mérete (batch\_size: változó, amely egyenlő a felismerhető komponensek számával)
- képek mérete (150 × 150)
- érvényesítésre szánt képek aránya (0.2, vagyis a képek 20%-a érvényesítésre szolgál)
- keverés (adatok keverése, abc rend helyett)
- seed (értéke befolyásolja a keverést, valamint a transzformációkat)

Az érvényesítő adatkészlet is ezekkel a paraméterekkel van generálva.

A végső modell felépítése előtt egy másik modell készül el. Ennek célja az adatkészletben lévő képek átformálása, véletlen szerű transzformálása (8. kódrészlet). A folyamat neve: data augmentation A képeken a következő transzformációk történhetnek, véletlenszerűen:

- függőleges tükrözés
- forgatás (10%)
- nagyítás (20%)
- eltolás (10%)

A százalékos értékek azt mutatják, hogy mekkora az a maximum transzformálás, amelyet a kép kaphat.

#### 8. kódrészlet: Data augmentation

```
data_augmentation = keras.Sequential([
    layers.experimental.preprocessing.RandomFlip("vertical",
input_shape=(img_height, img_width,1)),
    layers.experimental.preprocessing.RandomRotation(0.1),
    layers.experimental.preprocessing.RandomZoom(0.2),
    layers.experimental.preprocessing.RandomTranslation(0.1, 0.1),
])
```

A data\_augmentation model után elkészülhet a végleges CNN modell (9. kódrészlet). A



modell felépítése tesztelések során alakult ki, vagyis tesztelések által a bemutatott modell produkálta a legjobb eredményt a tanítás, valamint a tesztelés során.

A modell elején megtörténik a data augmentation, azaz a bemenet véletlenszerűen át lesz alakítva. Ezután van egy input réteg, amely esetén meg van adva a bemeneti képek méretei. Utána két konvolúciós réteg van, egy 32 és egy 64 szűrővel rendelkező. Mindkét réteg  $3 \times 3$  méretű kernelt, és relu aktivációs függvényt használ, és mindkettő után van egy maxpooling réteg, amely alap értékeket használ, vagyis  $2 \times 2$ -es kernelt, és teljes ( $2 \times 2$ -es kernelnél 2-es) lépést (stride).

Ezek után megtörténik a lapítás (flatten), majd egy 3 rétegű, teljesen csatolt réteg dolgozza fel, és dönti el a végső kimenetet.

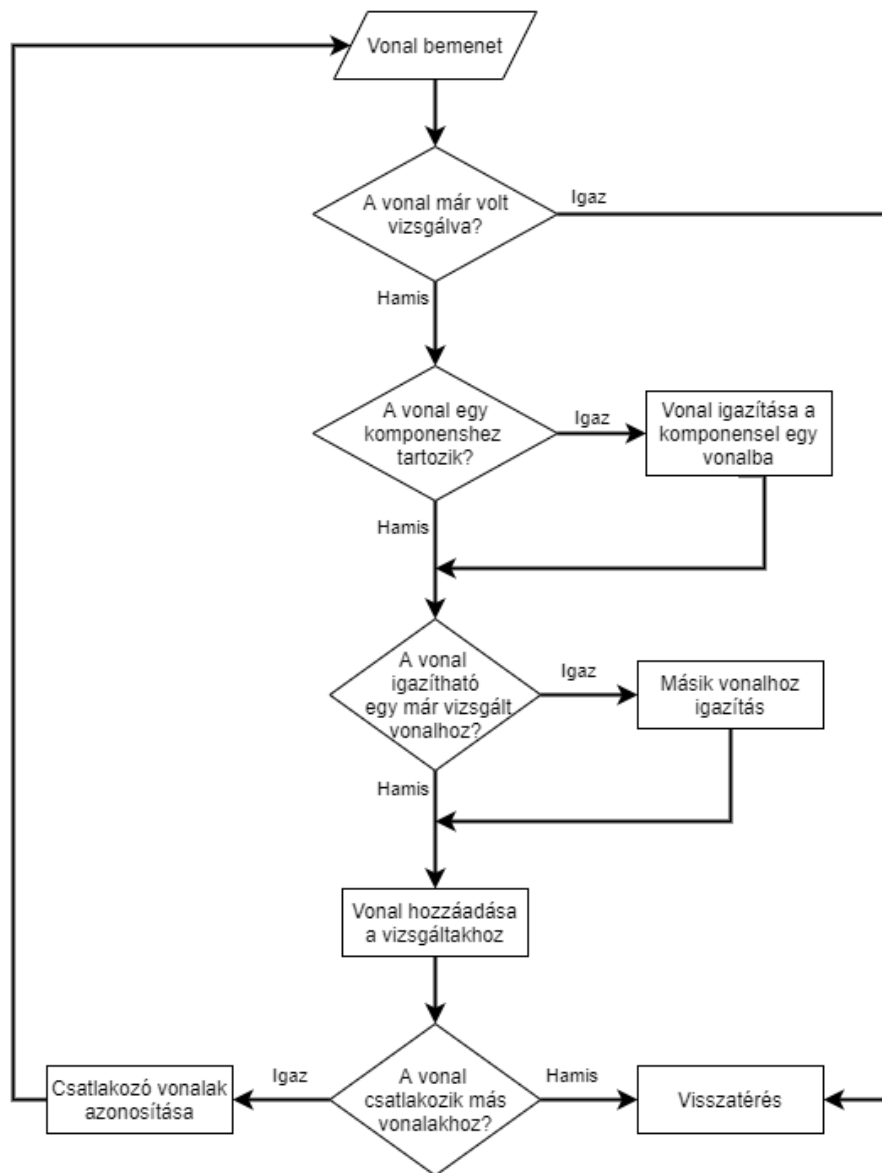
#### 9. kódrészlet: CNN modell

```
model = keras.Sequential([
    data_augmentation,
    layers.Input((img_height,img_width, 1)),
    layers.Conv2D(32, 3, activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, activation='relu'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(comp_count),
])
```

### 3.6 Összeköttetések azonosítása

A képen lévő komponenseket vonalak kötik össze. Ezen összeköttetések már fel lettek ismertetve a vonalkeresés által, azonban ezeket még át kell dolgozni a kimenetre. A bemeneti képen talált vonalak általában nem tökéletesen vízszintes/függőleges. A készített kép szöge, mérete által a vonalak nem egyenesek, nem egységes méretűek, és ezt a transzformáció során se lehet tökéletesen javítani. Emiatt a talált vonalak, ha egy nagyon kicsivel is, de eltérnek egymástól, valamint nem tökéletesen egyenesek, viszont a kimenetre egyenes vonalak kirajzolása a cél. Ebből az okból szükséges az összeköttetések további feldolgozása, átformálása a kimenetre. A program elején, egy állítható paraméter által vezérelhető, hogy a kimenet mekkora legyen. Ez egy méretarány, hogy a kimenet nagysága hányad része legyen a bemenet nagyságának. A későbbiekben a komponensek és összeköttetések ez alapján lesznek átméretezve a kimeneten.

Az átformálás itt úgy történik, hogy a vonalak egyesével vannak feldolgozva. A feldolgozás a komponenshez csatlakozó vonalaknál kezdődik. A komponens vonalak kiválogathatóak, a korábban lementett komponens végpontok által: ha egy vonal egyik végpontja benne van a komponens végpont listában, akkor az a vonal egy komponenshez tartozik. Minden komponens vonal esetén egy rekurzív függvény van meghívva (22. ábra).



22. ábra: Összeköttetéseket feldolgozó rekurzív függvény folyamatára

Az függvény bemenetként megkapja a vizsgálandó vonalat, egy listát az összes vonalról, a komponens végpontokat, egy listát arról, hogy mely vonalak voltak már feldolgozva, valamint egyéb segítő információkat.

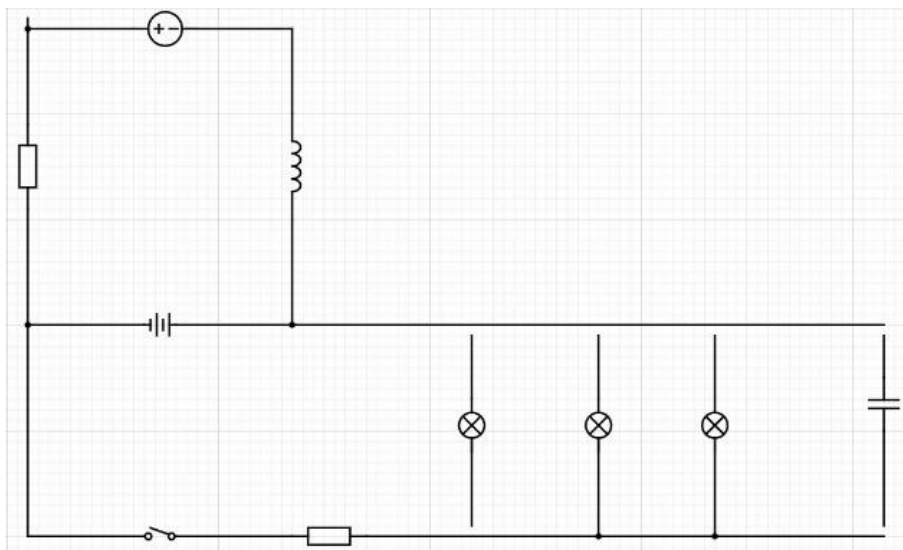
Az algoritmus úgy működik, hogy minden meghíváskor a kapott vonalat ellenőrzi, hogy már fel van-e dolgozva, hogy egyszer már megkapta-e bemenetként. Amennyiben már vizsgálva volt, a függvény visszatér, nem dolgozza fel újra. Amennyiben még fel nem dolgozott vonalat kap, azt először megvizsgálja, hogy komponenshez tartozó vonal-e. Ha komponenshez tartozik, akkor a vonal azon oldalát, amely nem a komponenshez csatlakozik, átalakítja úgy, hogy a komponenssel egy vonalba legyen, ezáltal kiegyenesítve a vonalat.

Ezután a korábban említett vonalak közötti eltérést próbálja minél jobban kiküszöbölni a program. Minden feldolgozott vonal koordinátái a későbbiekben mintapontként szolgálnak, vagyis a későbbiekben kapott vonalakat ezekhez a pontokhoz próbálja az algoritmus igazítani. Ez pontosabban úgy történik, hogy az éppen vizsgált vonal orientációjával ellentétes fekvésű, már feldolgozott, azaz mintavonalakhoz próbálja igazítani a jelenleg vizsgált vonal végpontját. Amennyiben a vizsgált vonal végpontja elég közel van egy másik, ellentétes orientációjú vonalhoz, akkor a végpont hozzá lesz igazítva a másik vonalhoz. Ez a közelség vizsgálat is egy bizonyos határon belül van vizsgálva, amely nagyságát egy, a program elején állítható paraméter vezérel. Amint megtörtént az igazítás, az átalakított vonal hozzáadódik a feldolgozott vonalak közé.

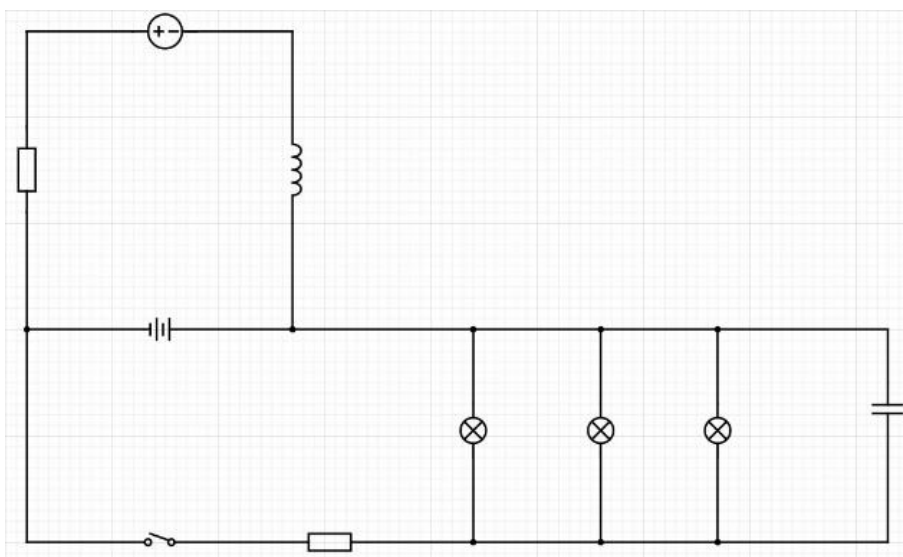
A vonal feldolgozása után jön a rekurzív része az algoritmusnak. Minden feldolgozott vonal esetén megvizsgálja, hogy a vonalnak a vizsgált végpontja csatlakozik-e más vonal végpontjához. Amennyiben csatlakozik, akkor minden csatlakozott vonal esetén újra meghívja önmagát a függvény, ugyanazon paramétereket átadva, csak a vizsgált vonalat változtatva.

A függvény lefutása után a bemenetre adott, már feldolgozott vonalak listája, amely a kezdetekben egy üres lista volt, most tartalmazza az összes feldolgozott vonalat.

A rekurzív algoritmus futásakor a vonalak igazítása jó eredményeket mutat, azonban nem teljes, mivel egy-egy vonal vizsgálatakor még nem áll rendelkezésre minden másik vonal, amelyekhez igazítani lehetne. Ebből az okból, miután minden komponenstől kiindulva fel lettek dolgozva a vonalak, még egy igazító algoritmus lefut, amely hasonlóképpen igazítja a vonalakat egymáshoz, hogy ne legyenek kilógó vonalak (23. ábra, 24. ábra).



23. ábra: Vonal igazítás előtt



24. ábra: Vonal igazítás után

### 3.7 Kimenet generálása

A program végénél, amikor már fel lett ismerve az összes komponens, és összeköttetések, valamint ezek az adatok fel lettek dolgozva a kimenet számára, már csak maga a kimenet megalkotása a feladat. A végső kimenet egy .cddx kiterjesztésű fájl, amely magában foglalja a teljes kapcsolási rajz információit, vagyis komponenseit, összeköttetéseiket, ezek pozícióját, orientációját.

A cddx (Circuit Diagram Document Format) egy olyan tömörített állomány, amely képes eltárolni egy kapcsolási rajz elemeinek elrendezését. A tömörítés folyamata hasonló

más tömörítő folyamatokhoz. Ebből az okból egy ilyen fájl tartalmát el lehet érni, ha egyszerűen át van írva a kiterjesztése .cddx-ről .zip-re. Ilyenkor megnyitható, kicsomagolható, és újra tömöríthető ez a file. Ezt használja ki a program is a fájl generálására. [22]

A kimenet generálása a tömörítés előtti mappaszerkezet kialakításával kezdődik. Amennyiben még nem létezik a kellő mappaszerkezet, akkor a program legenerálja azt (25. ábra). A mappaszerkezetben található fájlok különféle információkat tárolnak: metadata (dátum, cím), a fájlok típusai, a fájlok relációi (melyik file mit tartalmaz), valamint a program szempontjából leg fontosabb, a kapcsolási rajz elemeinek adatai. [22]

```
Circuit.cddx/  
├─ circuitdiagram/  
│   └─ Document.xml  
├─ docProps/  
│   └─ core.xml  
├─ _rels/  
│   └─ .rels  
└─ [Content_Types].xml
```

25. ábra: cddx fájl felépítése

*Forrás: [22]*

Maga a kapcsolási rajz a Document.xml fájlban van tárolva. Ez a file xml formátumban tárolja a kapcsolási rajz felépítését [22]:

- kapcsolási rajz területének mérete
- a rajzon szereplő komponensek típusai
- a komponensek adatai (típus, pozíció, méret, orientáció, forgatás/tükrözési adatok, valamint egyéb, komponens specifikus tulajdonságok)
- összeköttetések adatai (pozíció, orientáció, méret)

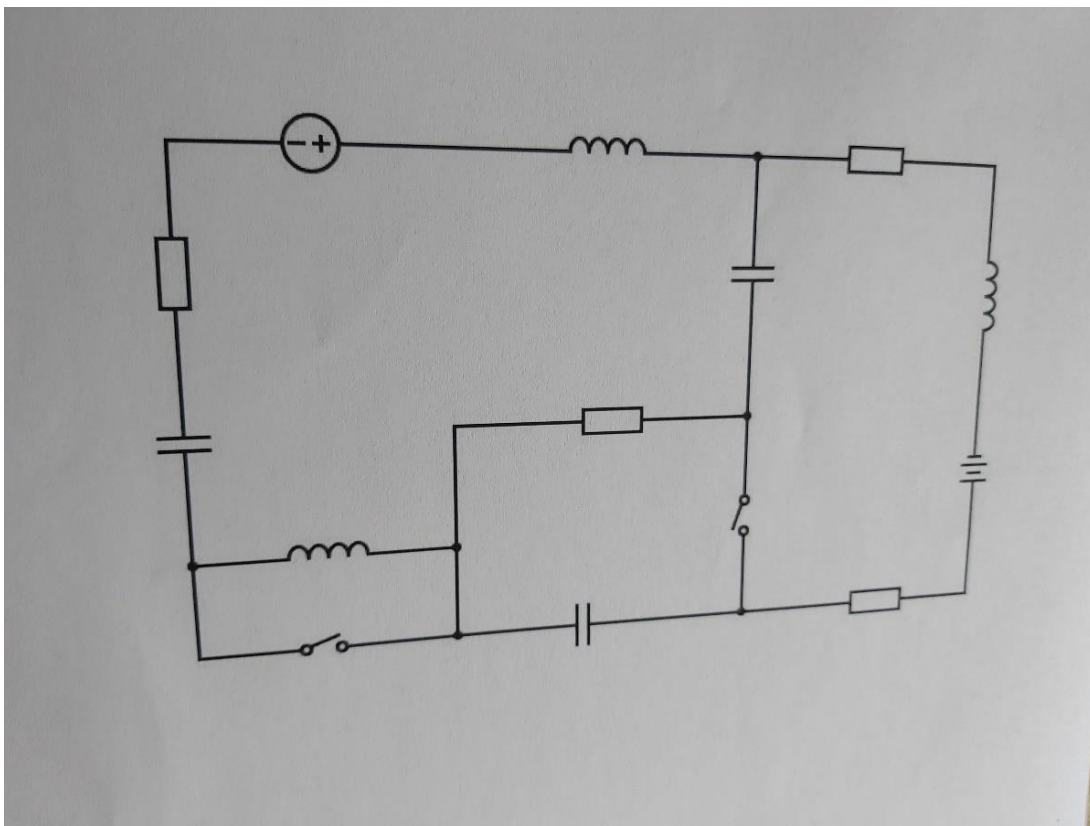
A program ezt a fájlt feltölti a már ismert adatokkal, valamint létrehozza/szerkeszti a többi fájlt, sablonok alapján (a többi fájlban nincs, vagy csak kis eltérés van a különböző kapcsolási rajzok esetén). Miután ez megtörtént, a program a kimeneti fájlokat, vagyis a mappaszerkezetet tartalmazó mappát (output) letömöríti, így keletkezik a kimeneti fájl: output.cddx.

## 4 Tesztelés

### 4.1 Módszer

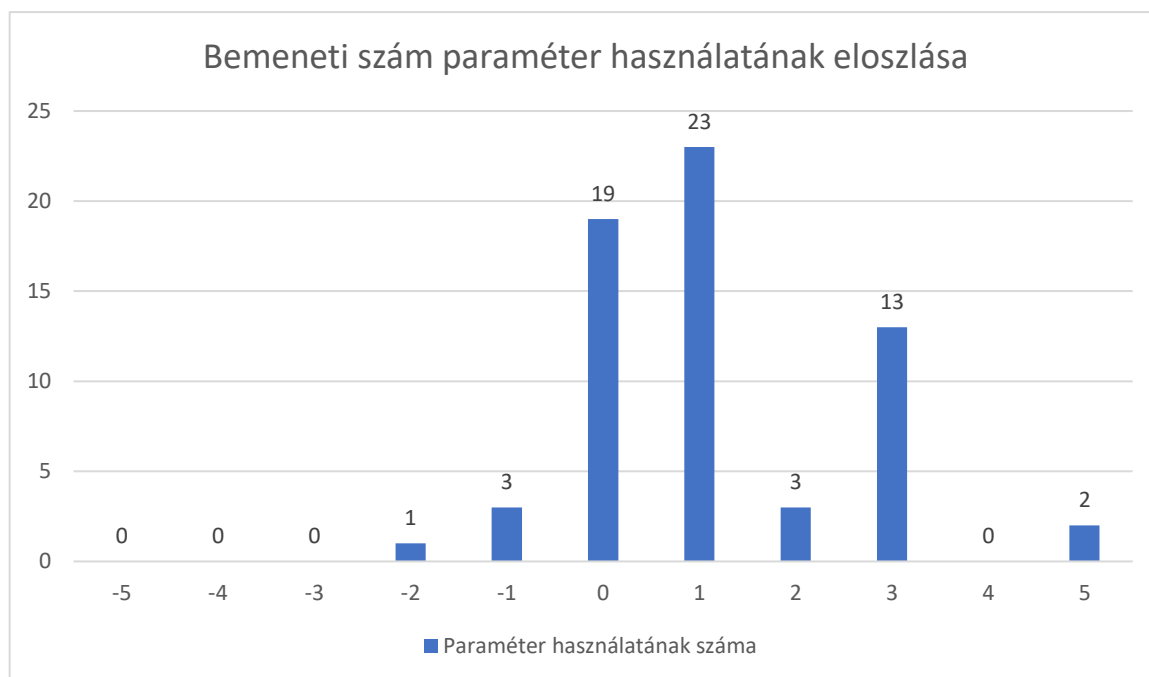
A program tesztelése előre kinyomtatott kapcsolási rajzok segítségével történt. Minden nyomtatott rajz le lett fotózva, és a készített képek szolgáltak a program számára bemenetként. A képek egy telefon kamerájának segítségével lettek készítve, és nagy részük  $4032 \times 3024$  és  $4032 \times 2268$  felbontású.

A tesztképek között vannak, amelyek egy rendes, működő rendszert írnak le, azonban nagy részük inkább csak tesztelés céljából összerakott, nem feltétlenül működő rendszert jelképező kapcsolási rajzok.



26. ábra: Egy tesztkép példa (20-as számú tesztkép)

A tesztelés során végig ugyanaz a program volt használva, ugyanazokkal a program eleji paraméterekkel (2. kódrészlet). Egy eltérés lehetett végig a tesztelés során két kép vizsgálatakor, az a program futtatásakor megadható szám paraméter, amely a bemeneti kép átméretezéséért felel. Ennek segítségével a különféle távolságból készített képek esetén is hasonló formára lehetett hozni a képeket a felismeréshez.



27. ábra: Teszteléskor használt bemeneti paraméterek eloszlása

A tesztelés során, a képek, valamint a felhasznált paraméterek (27. ábra) alapján elmondható, hogy a program jobban kezeli a közelebbről fotózott képeket, mint a távolról készítetteteket. Ez nagyrészt elvárható viselkedés, hiszen a közelebbről fotózott képek több információt tartalmaznak a kapcsolási rajzról, mivel minden komponens, és minden vonal több pixelből tevődik össze.

A teszteléskor 22 különböző kapcsolási rajz volt, amelyekről 64 különböző kép készült. Egy kapcsolási rajzról azért készült több kép, hogy ugyanannál a mintánál, különböző távolságok, dőlésszögek esetén is lehessen tesztelni a programot. Minden mintáról készült egy nagyrészt optimális kép, amely felülnézetből, minimális forgatással készült. Ezen kívül az összesről készült legalább egy olyan kép, amelyen el van döntve, valamint forgatva a kép (különböző mértékben). Végül készültek véletlenszerűen is képek, amelyek készítésénél nem volt figyelembe véve a kép állapota, csak véletlenszerűen lett készítve.

Ahogy később látni lehet (1. táblázat), volt optimális eset (minimális forgatás, felülnézet), forgatás, döntés, döntés és forgatás, extrém döntésses kép, valamint több rajzot is tartalmazó kép.

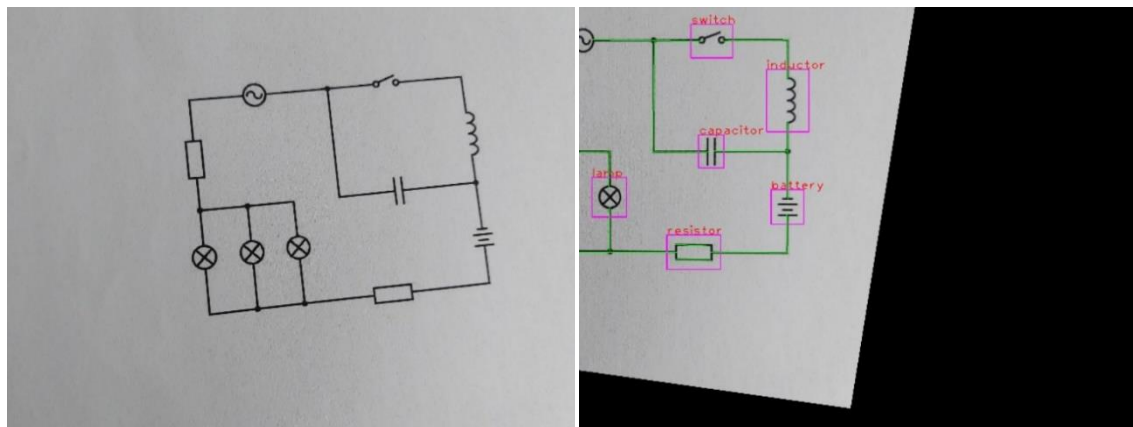
## 4.2 Eredmények, levont következtetések

### 4.2.1 Rajz feldolgozása, felismerése

A tesztek során, egy-egy képen a felismerés 1.5 másodpercet vett igénybe. A bonyolultabb, több elemet tartalmazó képek 2 másodpercet vettek igénybe. Ebben az időben csak a feldolgozás, felismerés és a kimenet tartozik bele. Ezekon kívül a tensorflow importálása, valamint a model betöltése további 3 másodpercet ad a program futására.

#### 4.2.1.1 Előfeldolgozás

Az előfeldolgozás során a binarizálás, és a kép méretezése a vártan megfelelően működött, nem volt velük probléma. A madárnézetbe transzformálás során, 1 teszt képnél, viszont annak majdnem minden méretezésénél előfordult egy hiba. A transzformálás során a program megkeresi a legszélső vonalakat a kapcsolási rajzon, és azok alapján átalakítja a képet. Ennél a képnél azonban nem találta meg a legszélső vonalakat, hanem azoknál beljebb lévő vonalszakaszok alapján transzformálta a képet. Ezen vonalszakaszok ugyanúgy jó forgatási és döntési transzformációt eredményeztek, a kép megfelelően átformálódott, azonban mivel nem a legszélső vonalakat találta meg, a transzformáció során a kép széle levágódik (28. ábra).



28. ábra: Hibás transzformáció előtt (bal) és után (jobb)

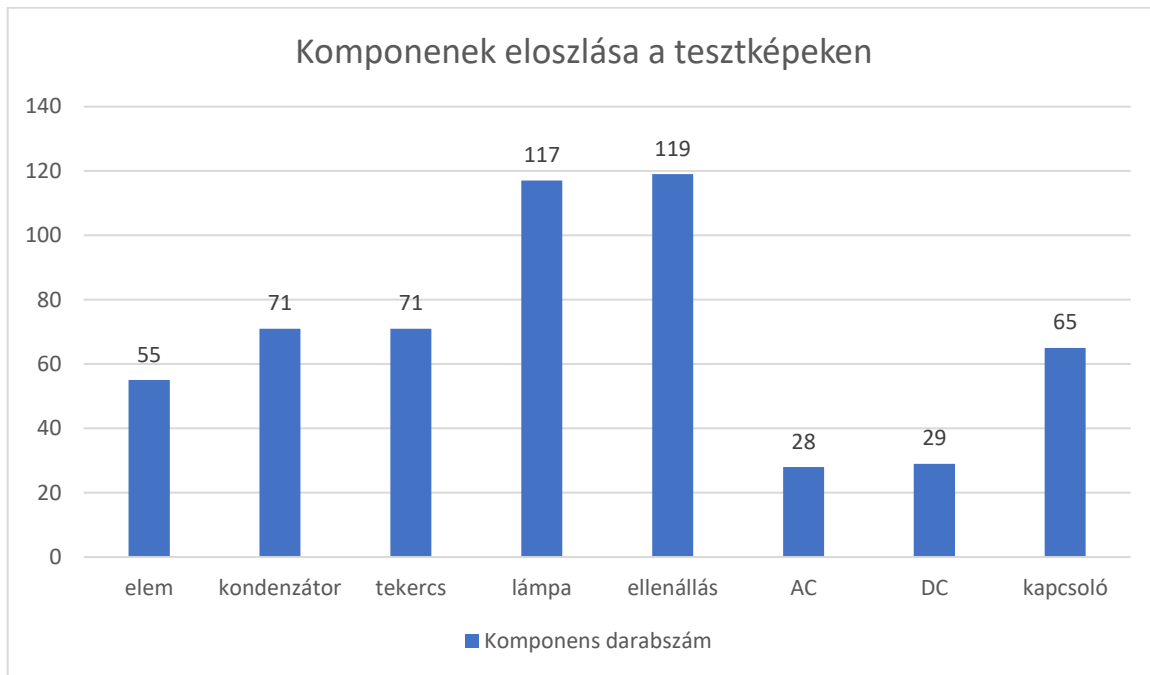
A hiba arra vezethető vissza, hogy ez a kép kisebb felbontású, mint a többi, és az első körben futtatott HoughLinesP függvény nem találja meg az összes vonalat a képen. A kisebb felbontás, valamint a távolról való fotózás miatt kevesebb pixelből áll össze minden része a kapcsolási rajznak. Vonalkeresésnél így nehezebb teljes, egész hosszúságú vonalakat megtalálni. Ezen kívül, mivel a konvolúciós neurális hálózatnak  $150 \times 150$  nagyságú kép kell, ezért a kisebb kép esetén esélyes, hogy a komponenseket fel kell nagyítani erre a nagyságra,



amely a komponens képének torzulásával járhat, ami csökkenti a felismerhetőség esélyét.

#### 4.2.1.2 *Komponensek, összeköttetések felismerése*

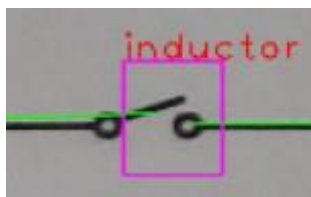
A tesztelés után a tesztképeken összesen 555 komponens volt jelen (29. ábra). Ezekből a program 552 komponensnek találta meg a helyét, és 546 komponenst sikeresen azonosított.



29. ábra: Komponensek eloszlása a tesztképeken (diagram)

A sikertelen azonosítások egyik oka a komponens helyének sikertelen azonosítása, vagy a CNN által másnak lett minősítve egy komponens.

Az általam használt konvolúciós neurális hálózat jó eredményeket adott, 4 esetben tévesztett el egy felismerést, ezáltal a tesztelés során a pontossága 98.91% volt. Legtöbb esetben egy egyenáramú (DC) forrás komponenst tévesen váltakozó áramú (AC) komponensnek ismert fel, valamint volt egy eset, amelynél egy kapcsolót tévesztett össze egy tekercsrel. Utóbbi azért fordult elő, mivel a komponens helye nem lett elég pontosan meghatározva, és a kapcsoló része lelógott a felismerésre szánt képről (30. ábra).

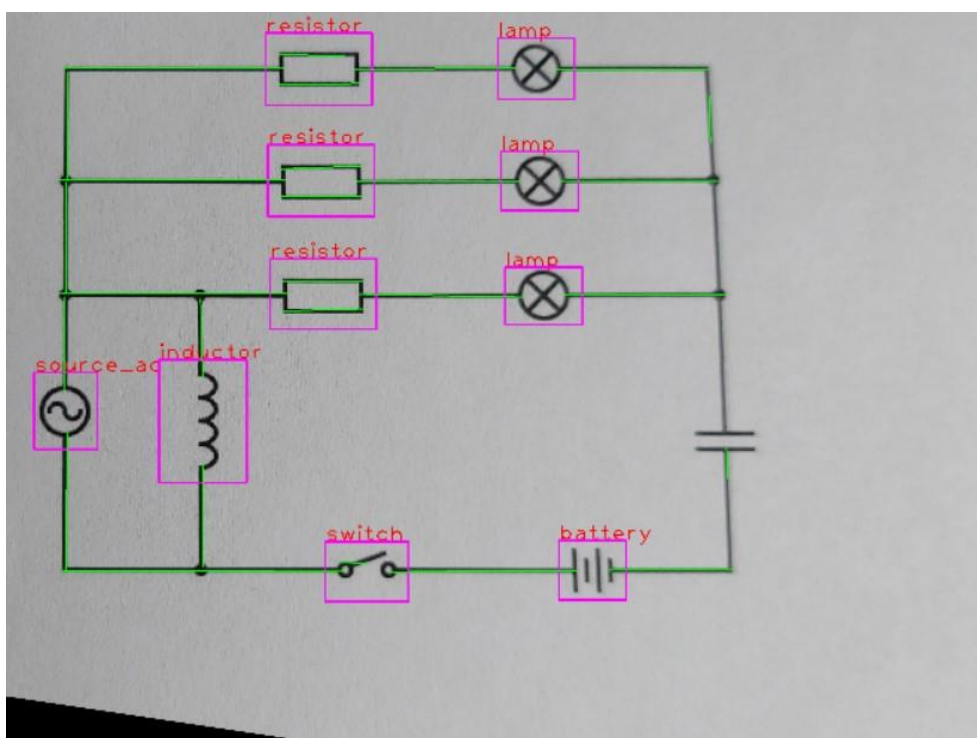


30. ábra: Komponens felismerés hiba

A komponens helyének sikertelen azonosítása 2 ok miatt történt a tesztek során:

1. A vonalkeresés nem járt teljes sikerrel, és nem ismert fel olyan vonalat, amely a komponenshez csatlakozik
2. A vonalkeresés során egy komponens két oldalán lévő vonalak nagymértékben eltérően lettek azonosítva, így nem tudta a program szembenező végpontokként azonosítani őket

A vonalfelismerés hibája általában azért következett be, mivel a képen a vonal nem volt elég vastag, hogy tökéletesen azonosítani lehessen. Ez előfordulhatott azért, mert a kép túl kicsi volt, vagy a dőlésszöge túl nagy volt a képnek, és a transzformálás után a kép egyik oldala nem tökéletesen lett átalakítva. Az utóbbi esetre példa (31. ábra), amikor a kép madárnézetbe való transzformálása nem volt elég jó, és a kép jobb oldalán a vonalak nem voltak elég függőleges helyzetben, hogy a vonalvizsgálatnál a dőlésszög határértékén belül maradjon.



31. ábra: Vonal detektálás hiba (zöld vonal jelképez egy talált vonalat)

Mivel az optimális képeknél is ugyanaz a madár perspektíva transzformáció végbemegy, mint az elforgatott/döntött képeknél, ezért ilyen téren a kétféle tesztkészletek esetén nem volt sok változás, több alkalommal is tapasztaltam, hogy a döntött képek esetén jobb volt a

felismerés hatékonysága.

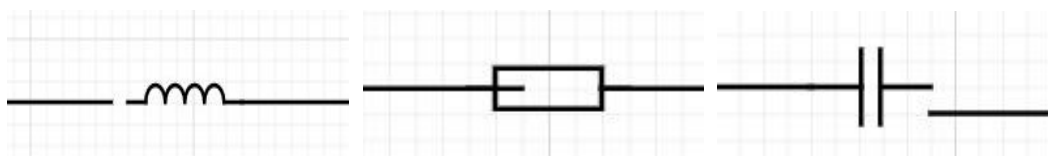
A nagyon extrém módon készített képek esetén azonban már felléptek hibák. Néhány képen a transzformáció után nem lett felismerhető egy komponens helye, valamint ezeknél a képeknél gyakoribbak a kimeneten tapasztalható eltérések.

#### 4.2.2 Generált kimenet

A program végső kimenete egy legenerált output fájl, amelyet beolvasva, szerkeszthető lesz a felismert rajz. A tesztek során a fájl felépítésével, generálásával, tömörítésével nem volt probléma. A kimeneten a megtalált komponenseket megfelelően elhelyezte a térben, megfelelő mérettel és jó helyre.

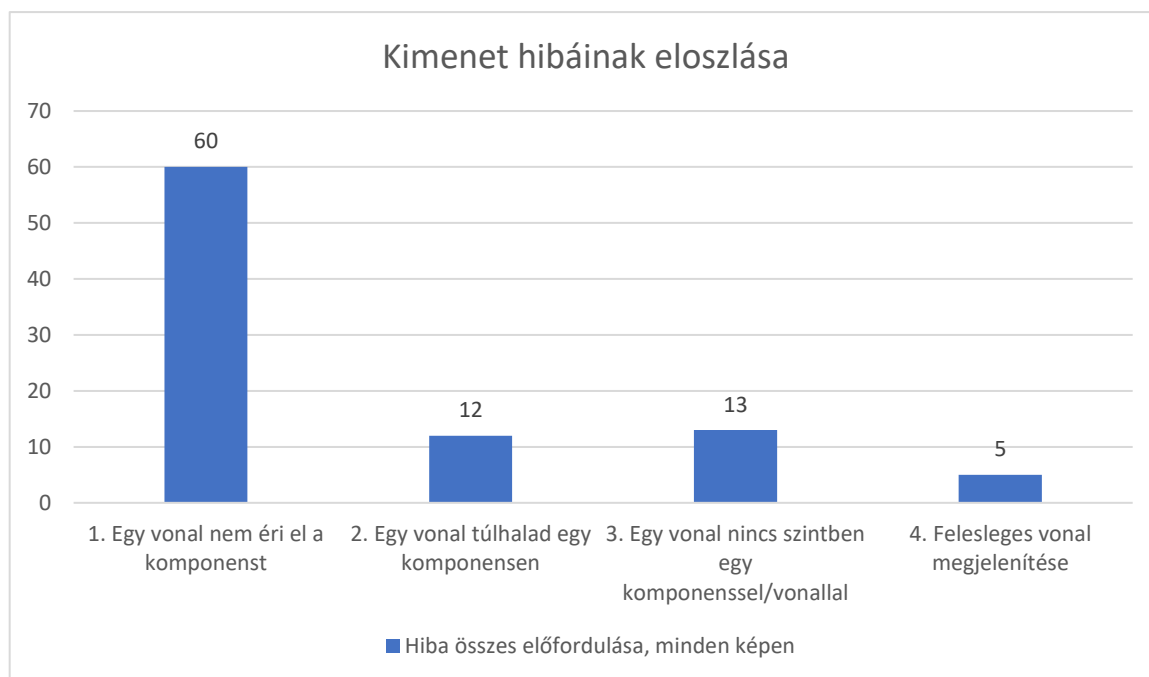
A kimeneten probléma az összeköttetések esetén volt gyakoribb. 64 tesztetből 17 esetén tökéletes volt a kimenet a bemenethez mérten. A többi esetről jellemzően 4 különböző típusú hiba fordulhatott elő (a kimeneten egy egység 10 pixelnek felel meg, mivel minden komponens/vonal pozíciója, mérete 10-zel osztható, 32. ábra):

1. Egy vonal nem éri el a komponenst (maximum 1 egységnyi eltérés)
2. Egy vonal túlhalad egy komponensen (maximum 2 egységnyi eltérés)
3. Egy vonal nincs szintben egy komponenssel/másik vonallal (maximum 1 egységnyi eltérés)
4. Felesleges vonal megjelenítése



32. ábra: Vonalhibák (balról: 1., 2. és 3. típus)

A hibát tartalmazó kimenetek esetén egy képen átlagosan 2 hiba fordult elő (33. ábra). A tesztek közül az is leszűrhető, hogy amennyiben egy képen előfordult egy nagyobb hiba (3. típus), akkor azt követik a kisebb mértékű hibák is. Ez következik abból is, mivel ezen hibák sokkal jellemzőbbek azokon a képeken, amelyeken nagyobb mértékben kellett transzformálni. Ilyen esetekben a vonalfelismerés esetén a vonalak koordinátái jobban eltérhetnek, ezáltal nehezebb azokat egymáshoz viszonyítani, valamint kerekítési eltérések is könnyebben adódnak.

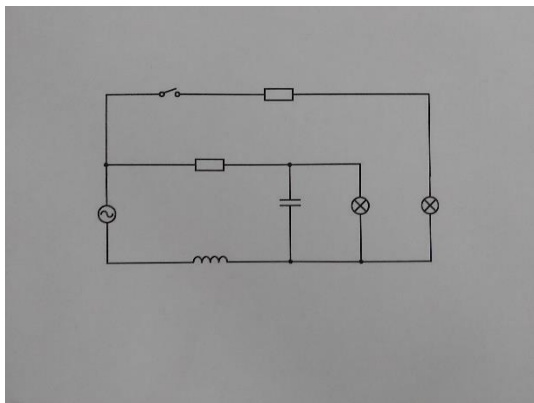
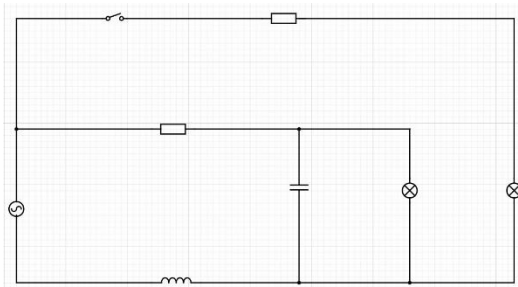
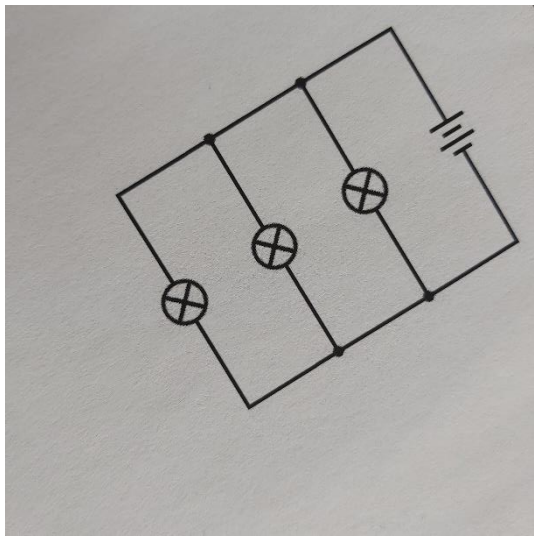
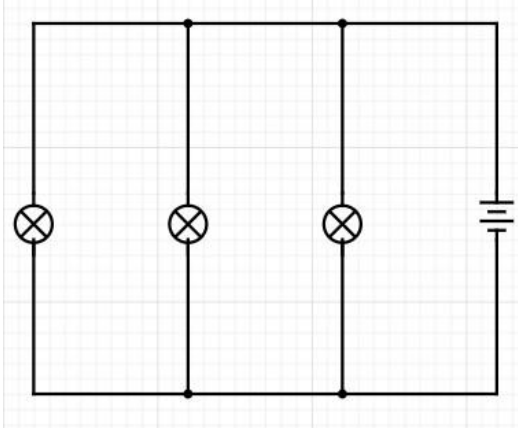
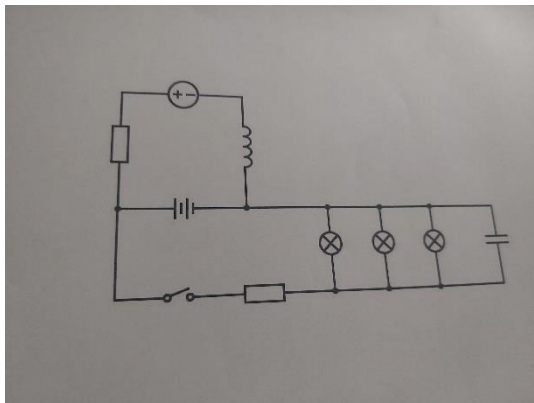
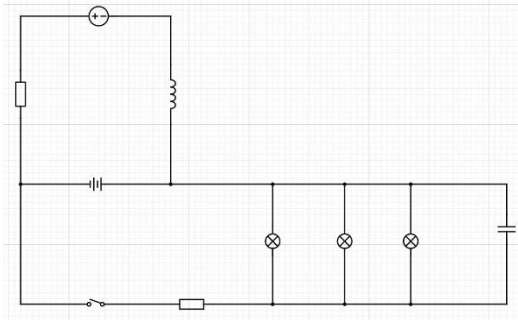


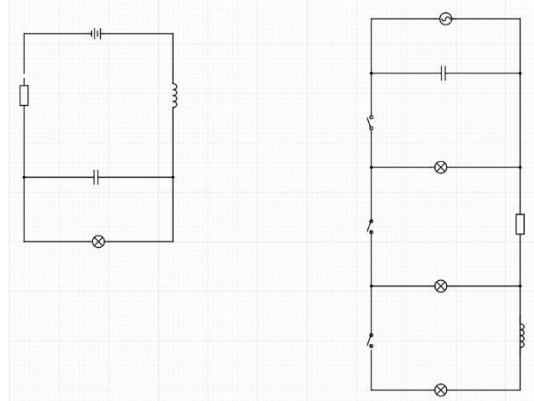
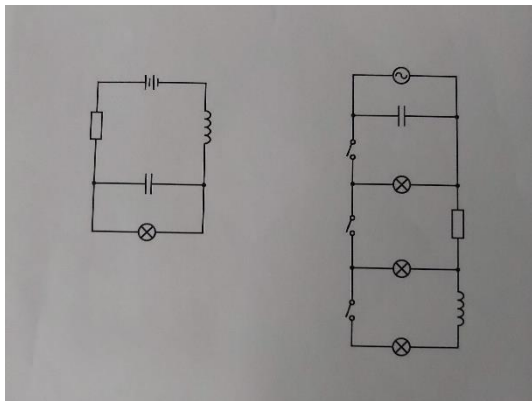
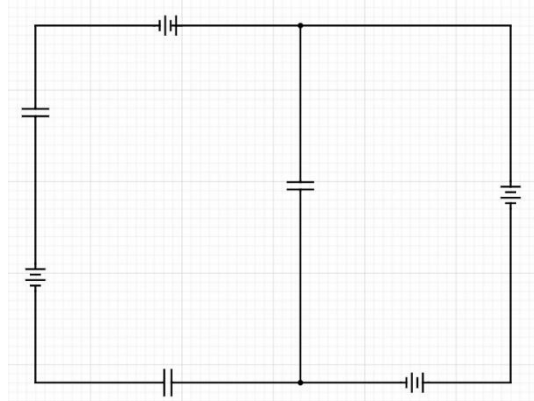
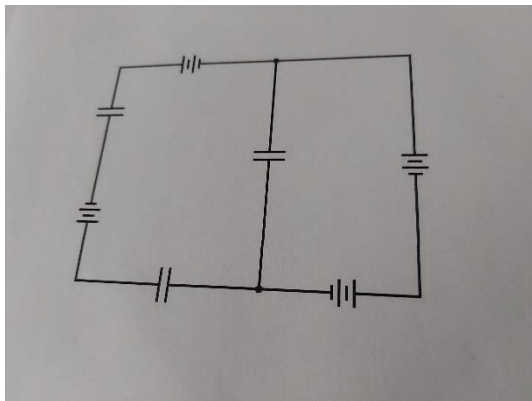
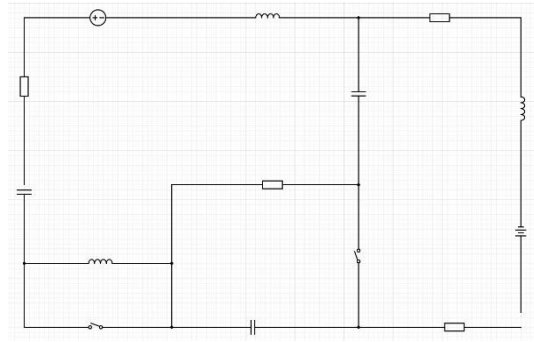
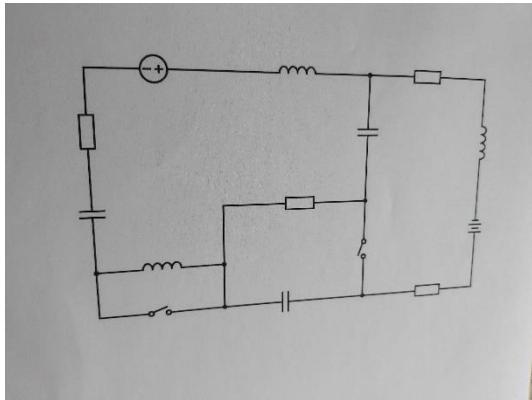
33. ábra: Kimenet hibáinak eloszlása (diagram)

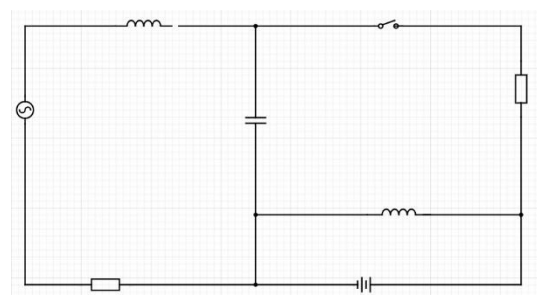
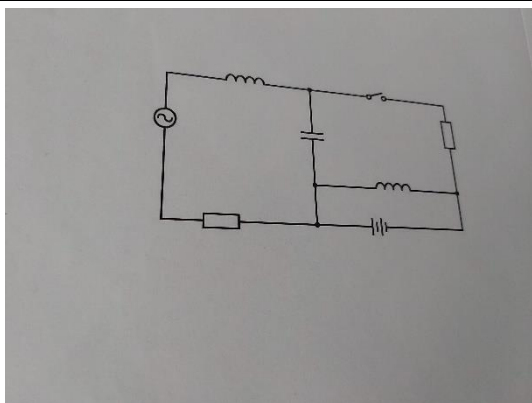
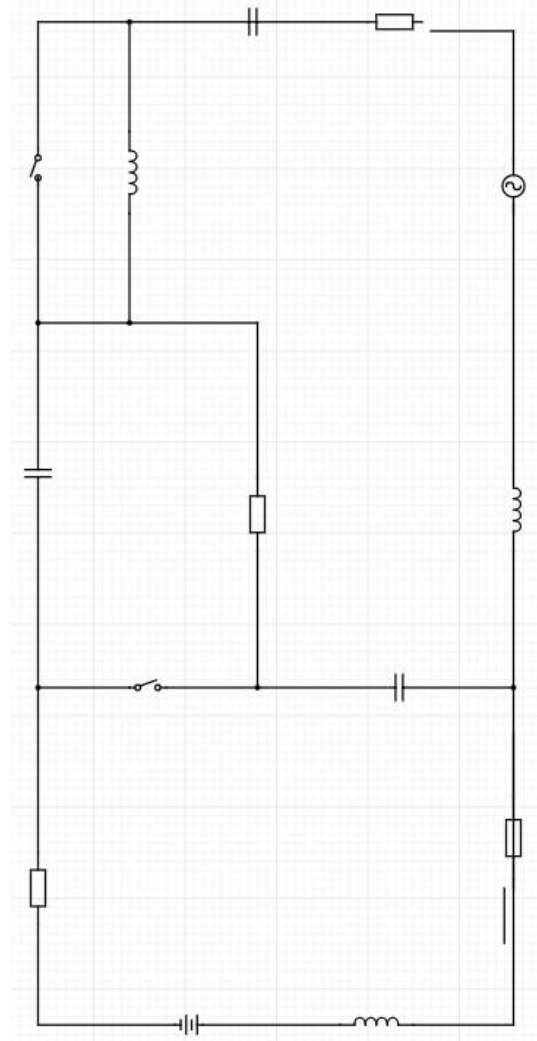
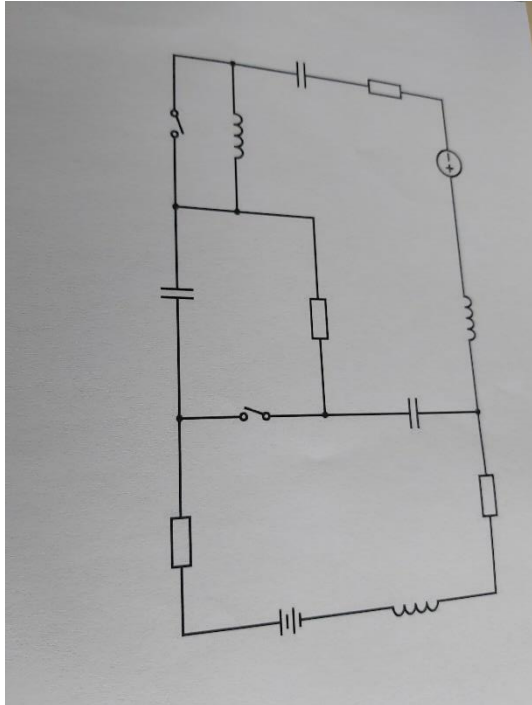
Ahogy a diagrammon is látható (33. ábra), az összesen 555 komponens, és azok összekapcsolása esetén a leggyakrabban probléma az első típusú volt, amihez viszonyítva a többi típus ritkán fordult elő.

A felsorolt hibák közül az első kettőt, valamint a negyediket könnyen lehet kézzel javítani. A 3. típust, az esettől függően eltérő nehézséggel lehet javítani. Ilyenkor lehet, hogy csak egy vonal pozícióját kell megváltoztatni, azonban lehet, hogy egy középen lévő vonalon van a hiba, amely több vonallal/komponenssel is kapcsolatban van. Utóbbi esetben több változtatást is el kell végezni, hogy az eredeti képet visszakapjuk. A tesztek során ez a hiba általában a kép szélén fordult csak elő (a transzformáció miatt), ezért javítása általában egyszerű lehetett.

1. táblázat: Mintaképek (bal), és a tesztelésük eredménye (jobb)





## 5 Összegzés

Az elkészült program bemutat egy megoldást egy digitalizációs problémára. A program által a digitalizált kapcsolási rajz tovább finomítható, szerkeszthető lesz. Ennek eléréséhez különféle képfeldolgozó eszközök, nagyrészt koordinátákkal dolgozó algoritmusok, valamint egy neurális hálózat is együttműködik egy szoftver által.

A bemutatott komponenseket és azok kapcsolatait kereső logika bízató eredményeket produkált. Az elkészült program működése még tovább fejleszthető, jobb eredmények is elérhetőek, valamint csökkenthetőek a bemeneti előkövetelmények: a komponensek keresése kiterjeszthető, hogy több mint 2 vezeték bemenetet tartalmazó komponensek esetén is megtalálja az összeköttetéseket, a neurális hálózatnak további komponensek is betaníthatóak, valamint arányosítás helyett előre elkészített paraméter sablonok segítségével mindenféle méretű képek esetén a legjobb paraméterekkel dolgozhat a program, ezáltal tovább növelve a hatékonyságot. Ebből is látható, hogy a jelenlegi program még nem érte el teljes potenciálját, viszont logikái bővíthetőek, a neurális hálózat tovább tanítható, ezáltal egy erős alapot adva további fejlesztésekhez.



## 6 Irodalomjegyzék

- [1] Nemzetközi standardok:  
<https://www.iec.ch/understanding-standards>, letöltés ideje: 2021.03.18.
- [2] Ellenállás minta:  
<https://circuitspedia.com/resistor-working-types-of-resistor-color-code/>, letöltés ideje: 2021.11.18.
- [3] Bradski, G., Kaehler, A.: *Learning OpenCV*.  
O'Reilly Media, Inc, United States of America, p. 571, 2008.
- [4] Marengoni, M., Stringhini, D.: High Level Computer Vision Using OpenCV  
2011 24th SIBGRAPI Conference on Graphics, Patterns, and Images Tutorials, pp. 11-24, 2011.
- [5] Thresholding technikák:  
[https://docs.opencv.org/3.4/d7/d4d/tutorial\\_py\\_thresholding.html](https://docs.opencv.org/3.4/d7/d4d/tutorial_py_thresholding.html), letöltés ideje: 2021.11.18.
- [6] Rong, W., Li, Z., Zhang, W., Sun, L.: An Improved Canny Edge Detection Algorithm  
Proceedings of 2014 IEEE International Conference on Mechatronics and Automation, pp. 577-582, 2014.
- [7] El Naqa, I., Murphy, M.: Machine Learning in Radiation Oncology: Theory and Applications  
Springer International Publishing, pp. 3-11, 2015
- [8] Abraham, A.: Artificial Neural Networks, in: Handbook of Measuring System Design  
Oklahoma State University, Stillwater, OK, USA, pp. 901-908, 2005.
- [9] Gupta, N.: Artificial Neural Network  
International Conference on Recent Trends in Applied Sciences with Engineering Applications, vol.3, no.1, pp. 1-6, 2013.
- [10] Feng, J., Lu, S.: Performance Analysis of Various Activation Functions in Artificial Neural Networks  
Journal of Physics: Conference Series, pp. 1-6, 2019.
- [11] 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.
- [12] Konvolúciós neurális hálózat felépítése:  
<https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>, letöltés ideje: 2021.11.18.
- [13] 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.
- [14] 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
- [15] 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.
  - [16] Rabbani, M., Khoshkangini, R., Nagendraswamy, H.S., Conti, M.: Hand Drawn Optical Circuit Recognition  
Procedia Computer Science, vol. 84, pp. 41-48, 2016.
  - [17] Del Sole, A.: Visual Studio Code Distilled: Evolved Code Editing for Windows, MacOS, and Linux.  
Apress, p. 215, 2018.
  - [18] Van Rossum, G., Drake, F. L.: Python 3 Reference Manual.  
CreateSpace, 2009.
  - [19] 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.
  - [20] Bradski, G.: The OpenCV Library  
Dr. Dobb's Journal of Software Tools, 2000.
  - [21] 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.
  - [22] CDDX fájl típus  
<https://www.circuit-diagram.org/docs/cddx>, letöltés ideje: 2021.11.11.