ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK	Hallgató neve:	**Kovács Dániel**
2024	Hallgató törzskönyvi száma:	**T/008491/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Szoftvertervezés és -fejlesztés Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Kovács Dániel**
Törzskönyvi száma: T/008491/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

Fafajták meghatározása neurális háló alkalmazásával
Determining tree species with neural network

Intézményi konzulens: Dr. Tusor Balázs
Külső konzulens:

Beadási határidő: 2024. december 15.

A záróvizsga tárgyai: Számítógép architektúrák
Szoftvertervezés és -fejlesztés specializáció

A feladat

A hallgató feladata egy olyan webes szoftver elkészítése, amely képes falevelekről készült képek alapján meghatározni konvolúciós neurális háló segítségével, hogy a falevél melyik fafajtához tartozik az előre definiált 28 fafajta közül.

A dolgozatnak tartalmaznia kell:

- bevezetést,
- szakirodalom összefoglalását és egyéb megoldások bemutatását,
- rövid összefoglalót a neurális hálókról,
- alkalmazni kívánt neurális háló felépítését és tesztelésének leírását,
- az elkészített rendszer felépítésének bemutatását,
- az elkészített rendszer működésének bemutatását,
- az elkészített rendszer tesztelésének folyamatát és eredményeit.

ÓBUDAI EGYETEM * Neumann János Informatikai Kar * 1. Budapest

Ph.

.....
Dr. habil. Vámossy Zoltán
intézetigazgató

A szakdolgozat elévülésének határideje: **2026. december 15.**
(ÓE HKR 54.§ (10) bekezdés szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat / diplomamunka saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban / diplomamunkámban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 2024. 12. 01.

Horváth Dániel

hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve: KOVÁCS DÁNIEL

Neptun Kód: [redacted]

Tagozat: NAPPALI

Telefon: [redacted]

Levelezési cím (pl.: lakcím): [redacted]

Szakdolgozat / Diplomamunka[1] címe magyarul:

FA FAJTÁK MEGHATÁROZÁSA NEURÁLIS HÁLÓ ALKALMAZÁSÁVAL

Szakdolgozat / Diplomamunka[2] címe angolul:

DETERMINING TREE SPECIES WITH NEURAL NETWORK

Intézményi konzulens: Dr. TUSOR BALÁZS

Külső konzulens: .....

Kérjük, hogy az adatokat nyomtatott nagy betűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2024.03.07.	SZAKDOLGOZAT KÖVETELMÉNYEK ÁTBESZÉLÉSE	Tusor Balázs
2.	2024.03.21.	EDDIGI HALADÁS BEMUTATÁSA	Tusor Balázs
3.	2024.04.11.	ÁLTALÁNOS KONZULTÁCIÓ	Tusor Balázs
4.	2024.04.18.	ÁLTALÁNOS KONZULTÁCIÓ	Tusor Balázs

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat / Szakdolgozat I. / Szakdolgozat II. / Projektlabor 1 / Projektlabor 2 / Projektlabor 3 / Záródolgozati projekt / Diplomamunka I / Diplomamunka II / Diplomamunka III / Diplomamunka IV [3] tantárgy követelményét teljesítette, beszámolóra / védésre [4]bocsátható.

A konzulens által javasolt érdemjegy: .....

Budapest, 2024.05.13.....

Tusor Balázs

.....

Intézményi konzulens

[1] Megfelelő aláhúzandó!
[2] Megfelelő aláhúzandó!
[3] Megfelelő aláhúzandó!
[4] Megfelelő aláhúzandó!

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve:	Neptun Kód:	Tagozat:
KOVÁCS DÁNIEL	DAKLXU	NAPPALI

Telefon: [redacted] Levelezési cím (pl.: lakcím):

Szakdolgozat / Diplomamunka[1] címe magyarul:

FA FAJTÁK MEGHATÁROZÁSA NEURÁLIS HÁLÓ ALKALMAZÁSÁVAL

Szakdolgozat / Diplomamunka[2] címe angolul:

DETERMINING TREE SPECIES WITH NEURAL NETWORK

Intézményi konzulens:

Dr. TUSOR BALÁZS

Külső konzulens:

.....

Kérjük, hogy az adatokat nyomtatott nagy betűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2024.09.24.	SZAKDOLGOZAT I. EREDMÉNYEINEK ÁTTEKINTÉSE	Tusor Balázs
2.	2024.10.08.	ÁLTALÁNOS KONZULTÁCIÓ	Tusor Balázs
3.	2024.11.11.	EDDIGI HALADÁS BEMUTATÁSA	Tusor Balázs
4.	2024.11.19.	ÁLTALÁNOS KONZULTÁCIÓ	Tusor Balázs

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat / Szakdolgozat I. / Szakdolgozat II. / Projektlabor 1 / Projektlabor 2 / Projektlabor 3 / Záródolgozati projekt / Diplomamunka I / Diplomamunka II / Diplomamunka III / Diplomamunka IV [3] tantárgy követelményét teljesítette, beszámolóra / védésre [4]bocsátható.

A konzulens által javasolt érdemjegy: 5

Budapest, 2024.11.26.....

Tusor Balázs

.....

Intézményi konzulens

---

[1] Megfelelő aláhúzandó!
[2] Megfelelő aláhúzandó!
[3] Megfelelő aláhúzandó!
[4] Megfelelő aláhúzandó!

ABSZTRAKT

A szakdolgozat célja egy olyan webalkalmazás kifejlesztése, amelynek fő feladata, hogy egy neurális háló segítségével azonosítsa a felhasználó által feltöltött falevél alapján a fa fajtáját, illetve egy másik neurális hálóval becslést adjon arra, hogy a kép melyik évszakban készülhetett. A fákat osztályozó neurális háló egy mély tanulásos konvolúciós, ImageNet adathalmazon előtanított VGG16-os kibővített modell, az évszakokat meghatározó neurális háló pedig egy szekvenciális neurális háló. Az alkalmazás ezenkívül még számos felhasználókezelési funkciót is kínál.

A dolgozatban az irodalomkutatás részeként bemutatom, hogy egyre inkább teret hódít a gépi tanulás a képek osztályozásában, kiemelve a mély tanulás általános alkalmazhatóságát, különös tekintettel a konvolúciós neurális hálókra. Áttekintem az évszakok meghatározására alkalmazható módszereket, és az információk alapján meghatározom, hogy milyen technikákat kívánok felhasználni.

A rendszer megtervezésének részeként kifejtem, hogy milyen követelményeknek kell megfelelnie az alkalmazásnak, majd áttekintem, hogy az irodalomkutatás eredményeként kapott eljárásokat és egyéb megoldásokat hogyan kívánom felhasználni. Megtervezem a neurális hálókat és adathalmazaikat, valamint meghatározom a webalkalmazás felépítését.

A megvalósításban áttekintem, hogy a tervezés fázisában kialakított rendszerterv hogyan került megvalósításra, kezdve a neurális hálókkal, majd pedig az API és webalkalmazással.

Végül pedig összefoglalom a dolgozat tartalmát.

ABSTRACT

The aim of this thesis is to develop a web application whose main task is to use a neural network to identify the type of tree based on the leaf uploaded by the user and another neural network to estimate the season in which the image was taken. The tree classification neural network is an extended VGG16 model pre-trained on the ImageNet dataset, while the season estimation neural network is a sequential neural network. Additionally, the application offers various user management functions.

In the thesis, as part of the literature review, I demonstrate the increasing role of machine learning in image classification, highlighting the general applicability of deep learning, particularly convolutional neural networks. I also review methods for determining seasons and identify the techniques I plan to use based on this information.

As part of the system design, I outline the requirements the application must meet and explain how I intend to utilize the procedures and solutions identified in the literature review. I design the neural networks and their datasets, as well as the structure of the web application.

In the implementation section, I review how the system design developed during the planning phase was realized, starting with the neural networks and then the API and web application.

Finally, I summarize the content of the thesis.

TARTALOMJEGYZÉK

1 BEVEZETÉS

1.1 Feladat leírása

A szakdolgozatomban egy olyan webalkalmazás elkészítése a cél, amelynek fő funkciója, hogy egy falevélről készült kép alapján neurális hálóval meghatározza a fa fajtáját. Emellett pedig rendelkezik egy másodlagos funkcióval, ami szintén neurális háló segítségével próbálja megbecsülni, hogy a levélről készült kép milyen évszakban készülhetett. Ezeken kívül az alkalmazás rendelkezik még a használatához szükséges funkciókkal.

Az alkalmazás használatához a felhasználónak regisztrálnia kell, ezért szükség van felhasználókat kezelő funkciókra:

- regisztrálás
- bejelentkezés / kijelentkezés
- felhasználói adatok módosítása
- profil törlése
- barátok kezelése (hozzáadás, törlés, megtekintés)

Amennyiben a felhasználónak már létezik regisztrált fiókja, az alkalmazásba belépve láthatja a barátait, a saját fa gyűjteményét a megfelelő navigációs gombokra kattintva.

A felhasználó bejelentkezés után tud új fát hozzáadni a gyűjteményéhez. A megfelelő funkciót kiválasztva fel tudja tölteni a falevélről készült fényképet és eltudja küldeni feldolgozásra. A feldolgozás során végbe megy a fénykép normalizálása és ezek után megkapják a már betanított neurális hálók. A kép nem kerül elmentésre. A feldolgozás után a felhasználó válaszban visszakapja az adott fa fajtáját és a becsült évszakot. Ekkor a felhasználó eldöntheti, hogy el akarja-e menteni a gyűjteményébe a fát, ha igen akkor a fafajta hozzáadódik a felhasználó fa gyűjteményéhez, az alkalmazás jelzi a felhasználó számára, ha már létezik az adott fa fajta a gyűjteményben. A neurális háló 28 fajta, Magyarországon őshonos lombhullató fát képes meghatározni.

A gyűjtemény funkciót kiválasztva a felhasználó kilistázva láthatja az általa már megtalált fa fajtákat. Itt egy adott fát kiválasztva elolvashatja a fa fajta jellemzőit: a nevét, a latin nevét, egy rövid leírását, az érdekességet amennyiben van ilyen és egyéb fajta specifikus jellemzőket.

Egy felhasználó rákereshet más fiókokra és barátnak jelölheti őket. Amennyiben a másik fél elfogadja a jelölést, mindkét felhasználó láthatja a másik felhasználót a saját a barát listájában, és megtekinthetik egymás gyűjteményét.

1.2 Dolgozat szerkezete

A dolgozatban először az irodalomkutatás tekintetében (2. szakasz) megvizsgáljuk, hogy milyen ehhez hasonló megoldások lelhetőek fel a világon (2.1 fejezet) és áttekintjük, hogy milyen módszereket lehet alkalmazni a falevelek felismerésére, valamint az évszakok meghatározásához (2.2 fejezet).

Ezt követően áttérek a rendszer megtervezésére (3. szakasz), ahol leírom milyen követelményeket kell teljesítenie a megoldásnak, milyen módszereket és technikákat alkalmazok a probléma megoldására, kezdve a fa fajtákat osztályozó neurális hálóval (3.3 fejezet), majd az évszakok meghatározására használt neurális háló tervezetére (3.4 fejezet). Majd pedig áttekintem magának az alkalmazásnak a felépítését (3.5 fejezet).

Mindezek után következik a megvalósítás (4. szakasz), ahol először leírom a neurális hálók megvalósításának részleteit (4.1 - 4.2 fejezetek), majd áttérek a Flask API alkalmazás működésére (4.3 fejezet), utána pedig a felhasználóval interakciókat végző webalkalmazás szerkezetét és működését taglalom (4.4 fejezet).

Végül pedig a dolgozat az összefoglalásban zárul (5. - 6. szakasz).

2 IRODALOMKUTATÁS

A természetben több száz fajta fa lelhető fel, és a megkülönböztetésük meglehetősen nehéznek is bizonyulhat. A botanikusok és a növénytanban jártas emberek azonban ennek ellenére is képesek egy pillantással azonosítani egy adott fa fajtáját a hozzátartozó falevél jellemzői alapján. Felmerül tehát a kérdés, hogy vajon lehetséges gépi tanulást alkalmazni a levéltípusok automatikus osztályozására.

A falevél klasszifikáció egy gyorsan növekvő része a mély tanulásnak. Az egyik talán legelterjedtebb módszer a levelek osztályozására a CNN-modell[1] alkalmazása, amelyet gyakran használnak a mély tanulás képfeldolgozási alkalmazásakor[1].

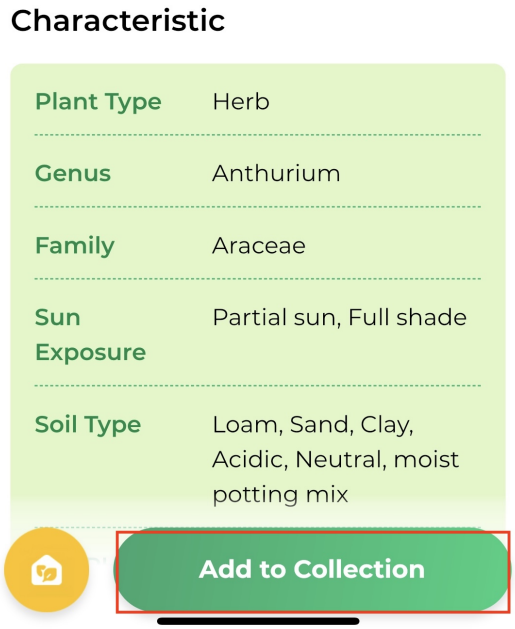
2.1 Hasonló fejlesztések

Ebben a szakaszban bemutatásra kerül két szoftver, amelyeket kiindulóalapként szeretnék felhasználni a saját megoldásomhoz.

2.1.1 LeafSnap

LeafSnap egy Androidra és iOS-re is ingyenesen letölthető okostelefonos applikáció, amely falevelekről készült képek alapján képes meghatározni fa fajokat. Az alkalmazás felülete a 2.1. ábrán látható. Az alkalmazás jelenleg az ismert növényfajok 90%-át képes felismerni [2]. Az alkalmazás használata rendkívül egyszerű. Csak készíteni kell egy fotót a falevélről, majd az alkalmazás automatikusan felismeri a fajtát. Ehhez internet kapcsolat szükséges, mert az alkalmazás egy hatalmas növényeket tartalmazó adatbázist használ, de, képes galériából feltöltött képeket is feldolgozni. A falevél feldolgozása után a fafajtát, hozzá lehet adni egy gyűjteményhez [3].

[1] Convolutional Neural Network (CNN)



2.1. ábra: Fa fajta hozzáadása gyűjteményhez

2.1.2 Plant.id

A Plant.id egy weboldal, aminek célja, hogy demonstrálja a készítők növény felismerési képességüket a vevőiknek. Elsősorban növények felismerésére szakosodtak, de már készítettek szoftvert gombák, és bogarak felismerésére is. A rendszer ingyenesen használható, de korlátozva van. Hetente összesen 5 darab növényt lehet azonosítani vele. Van egy mobil alkalmazásuk is, de ez a webalkalmazással ellentétben, nem alkalmaz gépi tanulást, hanem szakértő botanikusok végzik az azonosítást manuálisan a kép alapján [4].

A készítők biztosítanak még egy gépi tanulást alkalmazó API-t[2] , ami beküldött fénykép alapján határozza meg a növény fajtáját. A rendszer megközelítőleg 12 050 fajtát ismer, legyen az vadon élő növény vagy kertben termesztett. Megközelítőleg 87.2%-os pontossággal képes meghatározni a növény típusát. Mindezt €0.05 / lekérdezésért biztosítják [5].

2.2 Módszerek és technikák

Ez a szakasz ismerteti, hogy milyen tudományos leírások születtek már a növények leveleinek felismerésére, azonosítására.

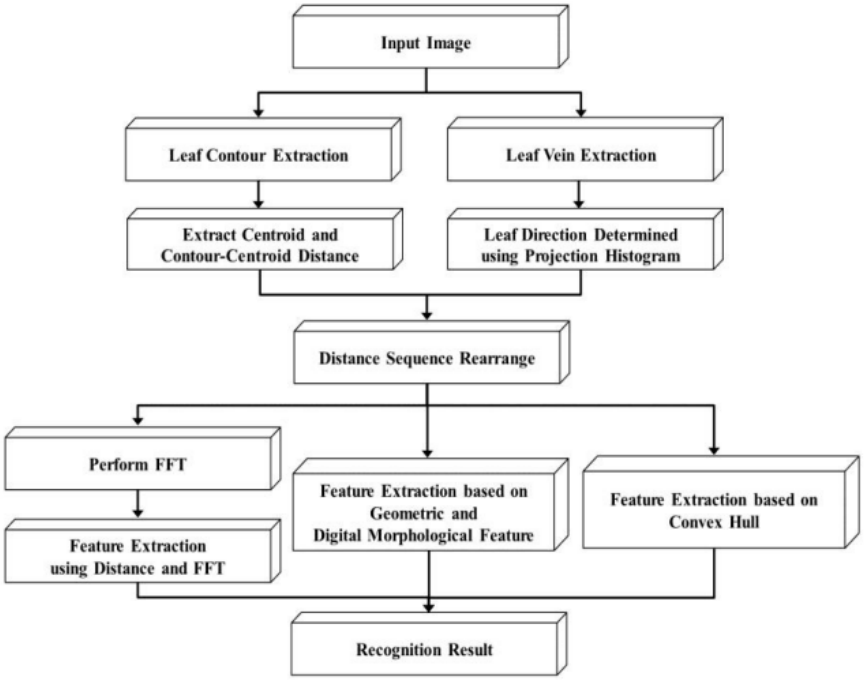
[2]Application Programming Interface (API)

2.2.1 Levél felismerés

A két legnépszerűbb jellemző, amit levélről készült kép alapján történő növény felismeréshez használnak a szín és a forma. A szín vizsgálatánál a szín hasonlóságot két kép között a szín hisztogramok összehasonlításával mérhetjük. A forma alapú vizsgálatban régió és körvonal jellemzőket kezelnek időtartományos adatonként. Ezt a módszert erősen korlátolják az évszakok és, hogy hogyan különböztetjük meg a levél elejét és végét.

2.2.2 Felismerés levél véna és forma alapján

A levél vénái segíthetnek a levél irányának meghatározásában és utána Gyors Fourier Transzformáció (FFT)[3] segítségével frekvenciatartományos adatokként kezeljük az adott levél körvonalának és súlypontjának távolságát[6]. A számításokhoz tartozó folyamatábra a 2.2. ábrán látható.



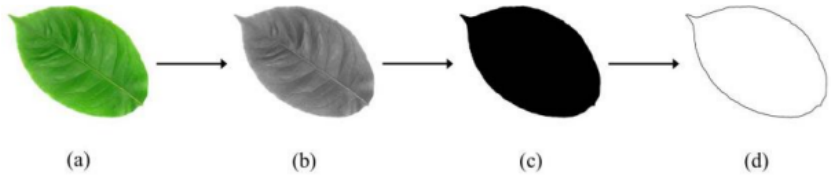
2.2. ábra: Véna és forma alapján történő levél azonosítás folyamatábrája

A levél színe függ az évszakoktól és befolyásolják egyéb környezeti tényezők, ezért ennél a megoldásnál a színeket nem vették figyelembe, hanem az alábbi képlettel (2.1) a képet szürke skálára konvertálták:

[3]Fast Fourier Transform (FFT)

$$Gray = 0.299 * R + 0.587 * G + 0.114 * B \tag{2.1}$$

Ezt követően, hogy megkapják a levél körvonalát, a szürkére skálázott képet bináris képpé alakították. Ha az adott pixel szürke intenzitás értéke kevesebb, mint a meghatározott T küszöbérték, akkor a pixel intenzitás értéke 0 lesz, ellenkező esetben pedig 255.



2.3. ábra: Példa a kontúr meghatározásához: (a) a bemeneti kép, (b) szürkére skálázott kép, (c) bináris kép, (d) levél kontúrja

A véna kivonáshoz szintén szürke skálára konvertálják a képet majd nyitó műveleteket[4] hajtanak végre. Más szóval eróziós műveleteket hajtanak végre a tágító eljárások után. Ezután vesszük a szürkeárnyalatos képet és a nyitó műveletek után kapott képet, majd vesszük a különbözetüket. Ennek a különbözet képnek a binárissá alakított verziója lesz a levél vénák kivonata.

A levelek felismeréséhez 21 jellemzőt vonnak ki egy levélből: 10 darab jellemző a súlypont-körvonal távolságból, az FFT magnitúdókból és fázisból, 10 másik jellemzőt morfológiai folyamatokkal határoztak meg, felhasználva 4 alap geometriai jellemzőt, 5 véna jellemzőt, az utolsó kivont jellemzőt a konvex testből szerzik. Majd az FFT folyamat által kapott magnitúdók közül a 10 legnagyobbat vetik össze.

A 2.3. ábrán láthatjuk a falevélről képen végzett transzformációk lépéseit. Az (a) lépésben még az eredeti képet láthatjuk, majd a 2.1. képlet segítségével a bemeneti képet szürkére skálázzák, ez lesz a (b) jelű kép. Ezután a (c) lépésben láthatjuk, hogy a szürkére skálázott képen végzet küszöbölés milyen eredményt adott. Végül pedig utolsó lépésben, hogy megkapjuk a (d) jelű eredményt, különböző matematikai műveletek után kivonjuk a (c) jelű lépés eredményét a (b) lépés eredményéből. A végeredmény a levél körvonala lesz.

[4]matematikai morfológiai művelet

2.2.3 Felismerés Konvolúciós Neurális Hálóval

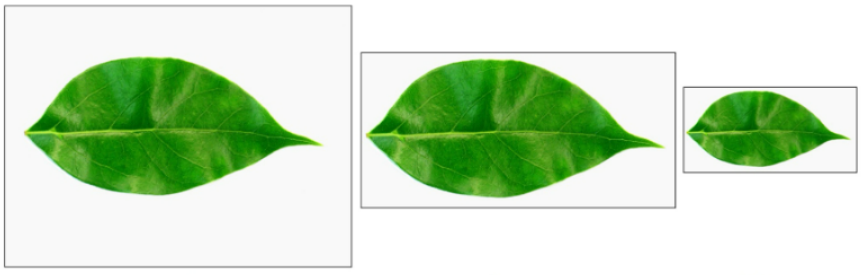
Napjaink egyik legnépszerűbb mély neurális hálója a Konvolúciós Neurális Háló[5] (továbbiakban CNN). A neve a konvolúcióból, lineáris matematikai művelet két mátrix között, ered. A CNN remek teljesítményt nyújt gépi tanulásos problémák megoldásában, különösen olyan alkalmazásokban, amik képi adatokkal dolgoznak. Nagy előnye más minta felismerési eljárásokkal szemben, hogy nem kell tudnia, hogy egy adott elem hol helyezkedik el a képen [7].

Számtalan különböző CNN architektúra típust találhatunk. Ilyen például a 2012-ben bemutatott AlexNet, ami talán az egyik legismertebb architektúra. Az AlexNet 8 rétegből áll és az ImageNet adatbázison végzett osztályozási feladatokra használták [8].

Másik architektúra a VGGNet, amelyet a Visual Geometry Group fejlesztett ki. Ez már 16-19 rétegből áll és szintén az ImageNet adatbázison végzett osztályozási feladatokra használták [9].

Van még a GoogleNet architektúra, ezt a CNN modellt a Google fejlesztette ki, Inception architektúrán alapszik. Segítségével a kutatók mélyebb hálózati struktúrákat tudnak kiépíteni, miközben a számítási komplexitást nem növelik. A modellbe épített Inception modulok több konvolúciót is képesek párhuzamosan futtatni, hogy több különböző jellemző pontot vonjanak ki.

A probléma GoogleNet CNN modellel történő, egyik lehetséges megoldása: előkészítési folyamat gyanánt képkivágással csökkenthető a számítások sokasága, mivel csökken a vizsgálni kívánt felület (2.4. ábra). Ez után pedig a kép átadásra kerül egy alap GoogleNet modell megvalósításnak. A 2.5. ábrán látható a GoogleNet model architektúrája.



2.4. ábra: Képkivágás a felület csökkentéséshez

[5]Convolution Neural Network (CNN)

Type	Filter size / stride
Conv	3 × 3 / 2
Conv	3 × 3 / 1
Conv padded	3 × 3 / 1
Ppool	3 × 3 / 2
Conv	3 × 3 / 1
Conv	3 × 3 / 2
Conv	3 × 3 / 1
3×Inception	Figure 10(a)
5×Inception	Figure 10(b)
2×Inception	Figure 10(c)
Pool	8 × 8
Linear	Logits
Softmax	Classifier

2.5. ábra: Alap GoogleNet CNN modell struktúrája Inception modulokkal

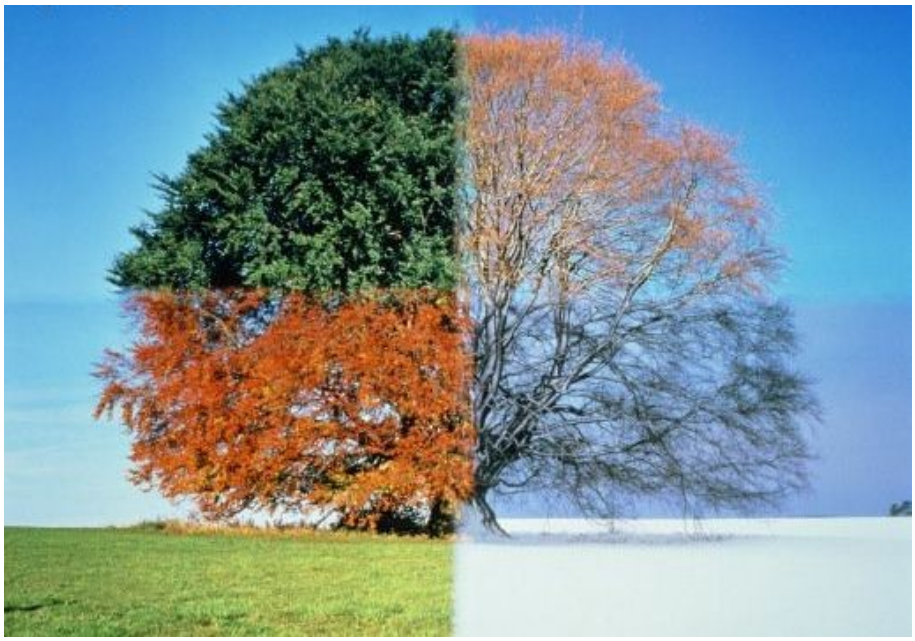
A „padded” konvolúciós művelet célja, hogy csökkentse az adatveszteséget a „pooling” művelet előtt. Ezek után folytatódnak a konvolúciós műveletek. Miután keresztül jut a 10 inception modulon, bekerül egy 8x8-as pool-ba. A folyamat végén végbe megy a „softmax” klasszifikáció, amit akkor használnak, ha több osztály a lehetséges kimenet.

2.2.4 Évszak meghatározása

Az évszakok meghatározásának egyik legegyszerűbb megközelítése az, ha megvizsgáljuk azokat a vizuális különbségeket, amelyek leginkább megkülönböztetik az évszakokat egymástól. Ezek közé tartoznak a természet adott évszakban való domináns színei.

A négy évszak - tavasz, nyár, ősz, tél - megkülönböztetése viszonylag egyszerű, ha a természet színeire koncentrálunk. Tavasszal a világosabb zöld árnyalatok dominálnak,

egy kevés barnával keverve. Nyáron a legtöbb fa már sűrű lombozattal rendelkezik, amely mélyebb zöld színben pompázik. Az ősz a négy évszak közül a legszínesebb, a lombozatban leginkább a sárga és vörös különböző árnyalatai jelennek meg. Végül a tél a legkevésbé színes évszak, amelyet leginkább a szürke árnyalatai, valamint az elhalt barna levelek jellemeznek. Az évszakok domináns színeit a 2.6. ábrán szemléltetem.



2.6. ábra: A négy évszak színekkel reprezentálva

Ezekből adódik, hogy az évszak meghatározása relatív egyszerűen vonatkoztatható a kép színeiből.

2.2.5 Domináns színek kigyűjtésének módszerei

Az évszakok meghatározása a falevélképek alapján a domináns színek elemzésével fog történni. A színek kinyerése érdekében különféle algoritmusok alkalmazhatók. Az alábbiakban három elterjedt módszert vizsgálok meg, melyek mindegyike sajátos előnyöket kínál a képi adatok elemzésében. Ezek a módszerek különböző szempontok alapján közelítik meg a domináns színek meghatározását, és mindegyik alkalmas lehet az évszak azonosítására.

Az első algoritmus a K-Means klaszterezés. Ez egy széles körben használt algoritmus, amely a képi színek klaszterekbe rendezésével segít a domináns színek kinyerésében. Az algoritmus működési elve az, hogy a képet előre meghatározott n számú klaszterre osztja fel, majd minden egyes klaszterhez egy középpontot (centroidot) rendel. Ezek a középpontok képviselik a klaszterek legjellemzőbb színeit. A folyamat iteratív, és addig

folytatódik amíg a középpontok el nem érik a stabil állapotukat, az az már nem változnak jelentősen az iterációk során. Ez a módszer gyors, és könnyen implementálható, valamint jól alkalmazható olyan képeken, ahol a domináns színek száma előre becsülhető [10].

A második algoritmus a Gauss-keverék modellek[6] (GMM). Ez szintén egy elterjedt megközelítés a domináns színek kivonására, amely a K-Means algoritmushoz képest nagyobb rugalmasságot biztosít. A GMM több Gauss-eloszlás segítségével modellezi a képen található színek eloszlását, ami különösen előnyös lehet, ha a színek eloszlása nem homogén vagy nem szabályos klaszterekből áll. A Gauss-keverékek segítségével nemlineáris klaszterhatárok is meghatározhatók, így az algoritmus rugalmasabb a K-Meanshez képest, hiszen képes különböző eloszlású színcsoportokat is felismerni. Ez különösen hasznos lehet olyan képek esetében, ahol a színek átmenetesen, egymásba olvadva jelennek meg [11].

Az utolsó algoritmus, amit most megvizsgálunk az a Mean Shift algoritmus. Ez egy nem-paraméteres klaszterezési módszer, amely akkor igazán hatékony, amikor a képi színek sűrűség szerinti csoportosítása kívánatos. Az algoritmus a K-Means eljárással ellentétben, nem igényel előre meghatározott klaszterszámot, hanem a színek sűrűsége alapján hoz létre csoportokat, azaz azokat a régiókat fogja klaszterekké alakítani, ahol a színek gyakorisága meghalad egy bizonyos küszöböt. Ez a módszer különösen előnyös olyan képeknél, ahol a színek eloszlása nem egyenletes. Ennek az eljárásnak a hátránya, hogy nagyobb számítási teljesítményt igényel, különösen nagy felbontású képek esetében [12].

2.3 Irodalomkutatás eredménye

A kutatás eredményeképpen arra jutottam, hogy a legmegfelelőbb megközelítés a fa fajták meghatározásához a VGG16 modell felhasználása a neurális hálózatban, valamint a falevelek erezetének és körvonalának kiemelése. A VGG16 modellt az ImageNet adathalmazon előtanították, és bizonyítottan nagy pontossággal képes osztályozni különböző objektumokat. A falevelek erezetének és körvonalának kiemelése azért fontos, mert ezek a jellemzők alapvetőek a fafajok megkülönböztetésében, és segíthetnek csökkenteni a neurális hálózat tanításának komplexitását, javítva annak hatékonyságát és pontosságát.

Az évszakok meghatározásához a képen található domináns színek kiválasztását tervezem, és a K-Means klaszterező algoritmust fogom használni erre a célra. A K-Means eljárás különösen jól alkalmazható ebben az esetben, mivel lehetőséget ad arra, hogy előre meghatározott számú klasztert hozzak létre, amely lehetővé teszi a tíz legdominánsabb szín azonosítását a képen. Ezzel a megközelítéssel az algoritmus az RGB-színtérben csoportosítja a képpontokat, és minden egyes klasztert egy reprezentatív átlagos színnel jelöl

[6] Gaussian Mixture Models (GMM)

meg. A kiválasztott színek a neurális hálózat bemeneteként szolgálnak, hogy az évszakot pontosan be lehessen sorolni.

Ez a módszer nemcsak hatékony, hanem a klaszterek előre definiálása miatt konzisztens színkivonást is biztosít, így a képek évszak szerinti osztályozása megbízhatóbbá válik.

3 FELHASZNÁLT MÓDSZEREK ÉS TECHNIKÁK

Ebben a fejezetcsoportban egy előzetes rendszertervet állítok fel, amely megfelelő alapként szolgál a megvalósítási munkák során.

Tervezés szempontjából a szoftvert két fő szakaszra bontom szét. Az egyik szakasz foglalkozik a neurális hálóval, ami a szoftver központi eleme, hiszen ez végzi majd el a falevelek felismerését, illetve az évszakok meghatározását. A másik főszakasz pedig maga a webalkalmazás lesz, ami a felhasználói interakciókért felel.

3.1 Funkció lista

Az alábbi felsorolásban láthatóak azok a funkciók, amiket a webalkalmazásnak teljesítenie kell:

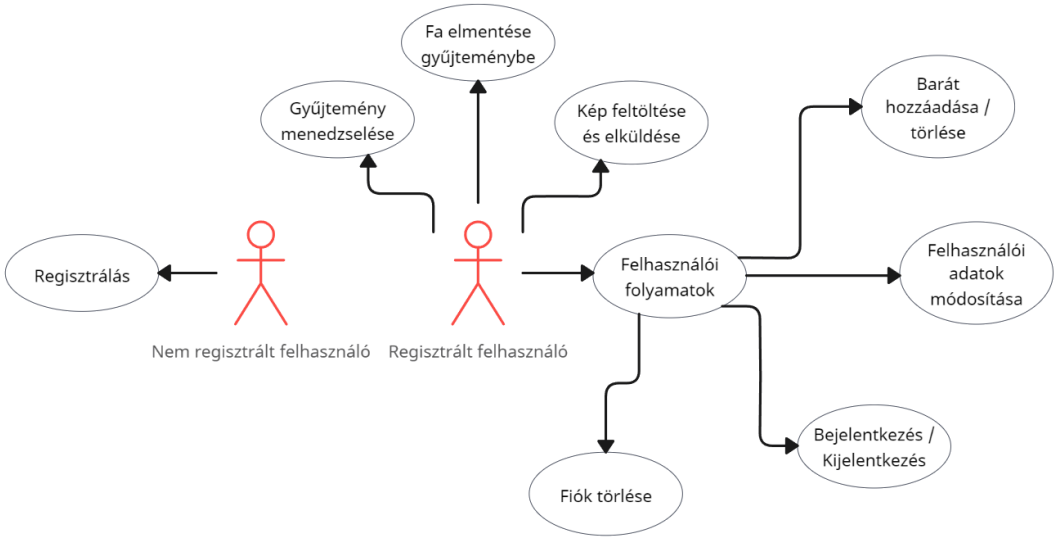
- Felhasználó regisztrálása (a jelszó legalább 8 karakter, kis- és nagybetűket, valamint számokat egyaránt tartalmaz)
- Felhasználó bejelentkezés / kijelentkezés
- Felhasználói adatok módosítása (felhasználónév, jelszó, profilkép)
- Felhasználói fiók törlése
- Másik felhasználó keresése név, felhasználónév vagy e-mail cím alapján
- Felhasználó hozzáadása a barátlistához
- Felhasználó törlése a barátlistáról
- Barát felhasználó fagyűjteményének megtekintése
- Falevélről készült kép feltöltése és elküldése feldolgozásra
- Feldolgozásról kapott eredmény kiíratása / megjelenítése
- Új fa fajta elhelyezése a felhasználó fagyűjteményébe
- Saját fagyűjtemény megtekintése
- Fagyűjteményből kiválasztott fafajta adatainak megjelenítése
- Fa törlése a felhasználói gyűjteményből
- Teljes gyűjtemény visszaállítása alapállapotba

3.2 Felhasználói szintek

A rendszer szempontjából a felhasználók két főcsoportra bonthatóak: regisztrált felhasználókra és nem regisztrált felhasználókra.

Értelemszerűen a még nem regisztrált felhasználó nem férhet hozzá a program funkcióihoz, egyedül a regisztrálás lehetséges a számára. Ezzel szemben a már regisztrált felhasználó minden funkcióhoz hozzáfér. Ez a szétbontás látható a 3.1. ábra use-case

diagramján.



3.1. ábra: Felhasználói use-case diagram

3.3 Fa fajtákat meghatározó neurális háló

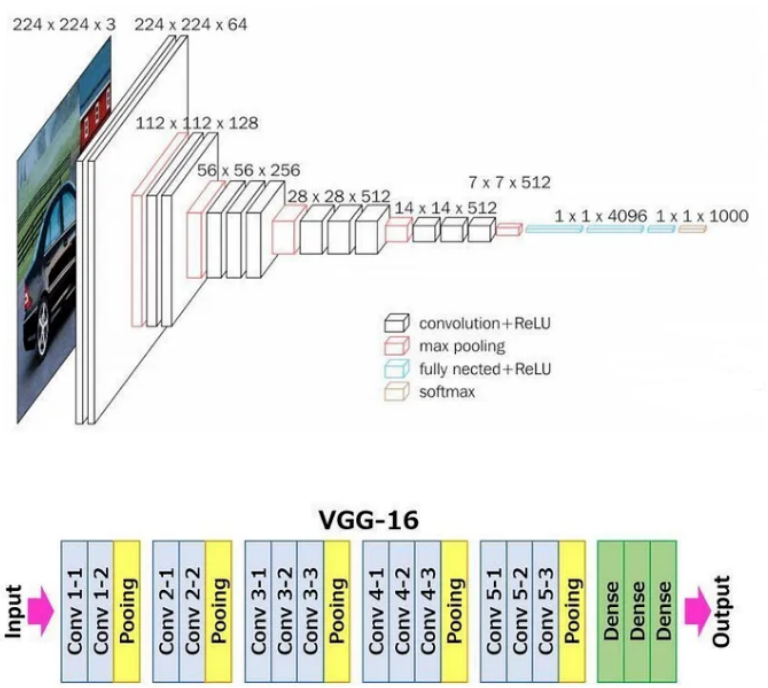
Ebben a fejezetben a falevelek felismeréséhez felhasználni kívánt neurális háló megtervezését taglalom.

Először megvizsgálom a VGG-16-os modell tulajdonságait, majd rátérek a tanításhoz használni kívánt adathalmazra, végül pedig megtervezem, hogy milyen módosításokat hajtok végre az alap modellen.

3.3.1 VGG-16

A VGG-16, mint ahogy már korábban említettem, egy konvolúciós neurális háló, amelyet Karen Simonyan és Andrew Zisserman dolgozott ki az Oxfordi Egyetemen. A 16 rétegű modell nagy előrelépést jelentett, mert képes volt 92,7%-os pontosságot elérni az ImageNet által meghirdetett 2014-es ILSVRC versenyen, amivel az egyik legjobb lett[13].

A modellt gyakran használják képfelismerési feladatokban, különösen akkor, ha nagy mennyiségű képadat áll rendelkezésre. A modell bemeneti követelménye viszont csak 224x224 pixel méretű, 3 csatornás képeket fogad el, amire figyelni kell a tanításnál.



3.2. ábra: VGG16 architektúra

Az 3.2. ábrán láthatjuk, hogy bemenetnek egy 224 x 224-es képet vár, amit még megszorzunk 3-mal, a három színcsatorna miatt.

A modell nagy előnye, hogy egyszerű és könnyen érthető architektúrával rendelkezik, azonban ez egy nagy méretű modell, mindösszesen 138 millió paraméterrel rendelkezik, ami miatt a betanítása sok időt vehet igénybe. De szerencsére van megoldás erre a problémára.

3.3.2 ImageNet

Az ImageNet egy nagy képeket tartalmazó adatbázis, amelyet vizuális objektumfelismerő szoftverek kutatásához hoztak létre. Először 2009-ben került bemutatásra. Manapság ez az adathalmaz már több, mint 1,2 millió képet tartalmaz, amit 1000 különböző osztályba csoportosítanak [14].

Az ImageNet csapata szinte minden évben megrendez egy versenyt, az ILSVRC-t[7] és a verseny győztesei, mint amilyen a VGG-16, előfordul, hogy publikálják a tanítás után elért modelljüket. Ezek a modellek már rendelkeznek a betanított paraméterekkel, a felhasználónak már csak minimális módosításokat kell rajta végezni, hogy a saját projektjében is alkalmazni tudja [15].

[7]Large Scale Visual Recognition Challenge

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 fc1 (Dense)                 (None, 4096)              102764544

 fc2 (Dense)                 (None, 4096)              16781312

 predictions (Dense)         (None, 1000)              4097000

=================================================================
Total params: 138357544 (527.79 MB)
Trainable params: 138357544 (527.79 MB)
Non-trainable params: 0 (0.00 Byte)
```

3.3. ábra: ImageNet-en előtanított VGG16 modell

A 3.3. ábrán láthatjuk a VGG16 modell felépítését. Az ábra legalján látható, hogy összesen 138357544 paraméterrel rendelkezik. Az ábra nagyrészét viszont a modellben egymást követő rétegek adják. Látható, hogy egy réteg milyen elnevezéssel rendelkezik, milyen típusú, milyen kimeneti mérete van és hogy hány tanítható paraméter található benne. Az ábrán fentről lefelé haladnak a rétegek a bemenettől a kimenet felé. A rétegeket öt blokkra bontották, ezek a blokkok vagy kettő, vagy három darab kétdimenziós

konvolúciós réteget tartalmaznak, majd pedig lezárásként egy kétdimenziós pooling réteget csatolnak rá. Az öt blokk után a kapott kimenetet kisimítják, majd ezt kötik rá két teljesen összekötött rétegre, ez az fc1 és fc2, majd pedig lezárásként az 1000 neuronos predictions teljesen összekötött réteg jön, ami már az 1000 osztályra adott osztályozás eredményét adja.

3.3.3 Adathalmaz a fafajtákhoz

A neurális háló célja, hogy 28 fafaj leveleit tudjam megkülönböztetni képeken. Ehhez a Missouri Egyetem által összegyűjtött adathalmazt használom fel, ami tartalmazza a falevelek képeit és a hozzájuk tartozó címkéket [16]. A bemeneti képeket úgy kell átalakítani, hogy megfeleljenek a VGG16-os modell elvárásainak, azaz 224x244 pixel méretűnek és 3 csatornásnak (RGB) kell lennie. A felismerni kívánt 28 fafajta az alábbi listában olvasható:

- madárberkenye,
- lisztesberkenye,
- háziberkenye,
- barkócaberkenye,
- szelídgesztenye,
- közönséges bükk,
- hamvas éger,
- mézgás éger,
- kecskefűz,
- ezüst hárs,
- nagylevelű hárs,
- kislevelű hárs,
- tatár juhar,
- korai juhar,
- virágos kőris,
- magas kőris,
- rezgőnyár,
- fehér nyár,
- szürke nyár,
- molyhos nyír,
- közönséges nyír,
- hegyi szil,
- vénic-szil,
- mezei szil,
- kocsánytalan tölgy,

– sajmeggy
– zselnicemeggy

3.3.4 Fa fajtákat meghatározó neurális háló felépítése

Az eddigi ismeretek alapján, ebben a projektben egy az ImageNet adathalmazon már előre betanított konvolúciós neurális hálót a VGG-16-ot használom fel, a falevelek képeinek osztályozására. A VGG-16 egy 16 réteg mély neurális háló, amely az ImageNet adatbázison tanult és képes több ezer objektumkategóriát felismerni a képeken [17]. A VGG-16 előnye, hogy egyszerű, egységes szerkezetű, és képes nagyon jó pontosságot elérni a képfeldolgozás területén.

A neurális háló megvalósításához Tensorflow és Keras keretrendszereket alkalmazok, mindezt Python nyelven.

Mint már említettem az előre betanított VGG16-os modellt gyakran igazítani kell a saját igényeinknek megfelelően, mivel alapból az ImageNet adatbázis miatt, 1000 osztályra végezne klasszifikációt. A VGG16-os modellt használom alapnak, de ezen már nem végzek tanítást, hanem eltávolítom az utolsó rétegeket, amik az osztályozásért felelnének, és kiegészítem egy 120 majd egy 60 neuronos teljesen összekötött[8] réteggel, végül pedig, hogy megkapjuk a 28 kimeneti osztályt, az utolsó réteg egy szintén Dense réteg, ezúttal 28 neuronnal és softmax aktivációs függvénnyel, ami lehetővé teszi a 2-nél több osztály közötti azonosítást.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 120)               3010680

 dense_1 (Dense)             (None, 60)                7260

 dense_2 (Dense)             (None, 28)                1708

=================================================================
Total params: 17734336 (67.65 MB)
Trainable params: 3019648 (11.52 MB)
Non-trainable params: 14714688 (56.13 MB)
```

3.4. ábra: Saját kiegészített modell

[8]Dense layer

Habár a vgg16 alapmodellen már nem végzünk tanítást, de a kiegészítés miatt még így is marad 3019648 tanítandó paraméter.

A 3.4. ábrán már a saját modellem szerkezete látható. Látszik, hogy a teljes paraméter szám 17734336, de ebből az alap modellen való tanítás letiltása miatt, kiesik 14714688 paraméter. A bemenettől a kimenetig való irány itt is fentről lefelé halad, az első jelzett réteg a vgg16 maga az alap modell, ennek a szerkezete a 3.3. ábrán megtekinthető, bár valójában az a bemeneti réteget tartalmazza és ez öt darab blokkot, mert a kimeneti rétegeket kicseréltük. Az alapmodell után megint lapítunk a flatten réteggel, majd következik két teljesen összekötött réteg, az egyik 3010680 tanítható paraméterrel, a másik pedig 7260 paraméterrel. Végezetül pedig egy utolsó 28 neuronos teljesen összekötött réteg jön, ami a kimenetet biztosítja a 28 osztály közötti klasszifikációhoz.

A tanítás befejeztével a modellt HDF[9] fájl formátumban mentjük el, és ezt a fájlt fogjuk használni a webalkalmazásban.

3.4 Évszakokat meghatározó neurális háló

Ebben a fejezetben az évszakok becsléséhez használt neurális háló megtervezését taglalom.

Először a tanítási adathalmaz megalkotását írom le, majd utána áttérek a használni kívánt neurális háló tervére.

3.4.1 Adathalmaz az évszakokhoz

Ennek a neurális hálónak a célja, hogy a kép domináns színeiből meghatározza az évszakot. Ehhez szükség van olyan képekre, amelyek hűen tükrözik az adott évszak termeszétbeni állapotát. Az adathalmaz létrehozásához a Flickr API-t használtam, amely lehetővé teszi a releváns képek automatikus letöltését és gyűjtését.

A Flickr egy népszerű fotómegosztó platform, amelyet a SmugMug üzemeltet. A platform lehetőséget biztosít felhasználók számára, hogy fényképeket és videókat töltsenek fel, oszthassanak meg, és kapcsolatba léphessenek egymással. A Flickr API programozott hozzáférést biztosít a Flickr adatbázishoz, lehetővé téve a képek keresését és letöltését az API-kulcs és titkos kulcs felhasználásával [18].

A rendszer használatához először regisztrálni kell egy alkalmazást a Flickr API oldalán, hogy megkapjuk az API és privát kulcsot. Ha ez megvan akkor ezeknek a kulcsoknak a felhasználásával létre lehet hozni a kapcsolatot az API-val. Végül pedig bizonyos ke-

[9]Hierarchical Data Formats, .h5 kiterjesztés

resési kulcsszavak mentén elkezdődhet a keresés az adatbázisban, majd a kapott képeket letöltöm és elmentem.

Ezzel a módszerrel és az utólagos manuális szűréssel minden évszakhoz egyenként több mint 400 releváns képet kaptam.

Végezetül, hogy megkapjam a tanításhoz használni kívánt adatokat, a kapott képekből ki kell vonnom a domináns színeket, majd elkezdődhet a tanítás.

3.4.2 Évszakokat meghatározó neurális háló terve

A következőkben egy olyan neurális háló architektúra-tervet ismertetek, amely alkalmas az évszakok osztályozására a képek 10 legdominánsabb színe alapján. Az architektúra létrehozásához, a fafajtákat meghatározó hálóhoz hasonlóan, szintén a Tensorflow és Keras keretrendszereket használok Python programozási nyelven.

A modell bemeneteként a fényképen fellelhető tíz domináns szín RGB-értékei szolgálnak, ez összesen 30 bemeneti paramétert jelent. A hálózat felépítése és paraméterezése az osztályozási feladat hatékonyságának növelésére szolgál, figyelembe véve az egyszerűséget és a modellek megbízhatóságát.

A modell egy szekvenciális teljesen összekötött hálózat, amelynek főkomponensei a sűrű rétegek (dense layers), melyek a bemeneteket feldolgozva egyre mélyebb absztrakciós szintekre juttatják el az adatokat.

A bemenet egy harminc elemű vektor, amely a kép 10 legdominánsabb színének RGB-értékeit tartalmazza. A dense rétegeken Leaky ReLU[10] aktivációs függvényt alkalmazok. A hálózat túltanulásának érdekében pedig Dropout rétegeket használok, amely véletlenszerűen inaktiválja a neuronok egy részét minden tanítási iterációban. A tervezett dropout arány 0.2, vagyis a neuronok 20%-a lesz véletlenszerűen kikapcsolva minden egyes lépésben. Ez a módszer elősegíti, hogy a modell jobban általánosítható legyen új adatokon.

A kimeneti réteg négy neuront tartalmaz majd a négy évszak számára. Mivel többosztályos osztályozásra van szükség ezért az utolsó rétegen softmax aktivációs függvényt alkalmazok. A softmax függvény a kimenetet valószínűségi értékké alakítja, ahol az értékek összege 1, így minden osztályhoz hozzáférhető egy valószínűségi pontszám.

A modell optimalizálására Adam optimalizációt használok, amely adaptív tanulási rátát kínál, ezáltal gyorsabb konvergenciát biztosítva a képzési folyamat során. A kezdeti tanulási rátát 0,001-re állítom be.

A tanítás során early stopping technikát is kívánok alkalmazni, amely figyeli majd a

[10]Leaky Rectified Linear Unit

validációs veszteséget, és megállítja a tanítást, ha a modell teljesítménye már nem javul.

Ez az architektúra lehetővé teszi a szezonális különbségek felismerését a domináns színek alapján, és a rétegek, valamint az optimalizálási paraméterek megválasztása elősegíti a hálózat hatékony és pontos működését.

3.5 Webalkalmazás

Habár a neurális háló felel a projekt fő funkciójáért, de önmagában nem sok hasznát vesszük, ha nem lehet vele interakcióba lépni. Emiatt van szükség egy webalkalmazásra, ami megvalósítja, leegyszerűsíti a kommunikációt a neurális háló és a felhasználó között.

3.5.1 Fő webalkalmazás

A webalkalmazást ASP.Net Core keretrendszer segítségével hozom létre. Az ASP.Net Core egy nyílt forráskódú cross-platform változata az ASP.NET keretrendszernek [19].

Előnyei közé tartozik a cross-platformsága, mert így operációs rendszertől függetlenül tud futni, akár Windows, MacOS vagy Linux rendszereken, a Blazor oldalak alkalmazhatósága, ami lehetővé teszi, hogy C# kódot használjak a böngészőben a JavaScript mellett. Magas teljesítményt nyújt, gyorsabb, mint más keretrendszerek, például: Node.js, PHP vagy a legtöbb Java keretrendszer. Számos keretrendszert támogat, ami leegyszerűsíti, gyorsítja a fejlesztést, mint az ASP.Net Core Identity keretrendszer, ami egy sablont ad a felhasználók kezelésre, így azt nem kell külön megírni. Ezeket figyelembe véve az ASP.NET Core megfelelőnek tűnik az alkalmazás létrehozásához.

A webalkalmazás célja, hogy segítse a kommunikációt a neurális háló és a felhasználó között, de kiegészítésre kerül egyéb felhasználói funkciókkal. Az ASP.Net Core Identity keretrendszer segítségével létrehozok regisztrálási és bejelentkezési felületet, mert a funkciókat csak regisztrált felhasználók számára szeretném elérhetővé tenni. Minden felhasználó a szokásos tulajdonságokon kívül, mint a felhasználónév, e-mail cím és jelszó, rendelkezik még egy gyűjteménnyel, amiben gyűjtheti az általa talált fafajtákat. Ahhoz, hogy a felhasználó fákat rendelhessen a profiljához használnia kell egy erre kirendelt külön oldalt, ahol feltöltheti a fényképet .jpg vagy .png formátumban.

Viszont az ASP.NET Core nem tudja önmagában kezelni a neurális hálót, ezért szükség van egy másik webalkalmazásra, ami kezelni tudja a betanított modellünket.

3.5.2 FlaskAPI alkalmazás

A FlaskAPI egy könnyű és rugalmas Python keretrendszer, amely elsősorban web API-k és webalkalmazások fejlesztésére szolgál. Egyszerű de erős eszközöket ad a fejlesztőknek a HTTP kérések kezelésére, és válasz generálásra. A FlaskAPI a mikrokeretrendszerek elvét követi, azaz a modularitást és minimalizmust helyezi előtérbe.

A keretrendszer rugalmasságának és a Python programozási nyelv használatának köszönhetően egyszerűen felhasználhatom a Tensorflow és Keras keretrendszereket, hogy becslést végezzek a betanított neurális hálókkal.

A program egy API hívás után elvégzi a kívánt módosításokat a kapott képen, végrehajtja a predikciót, mind a két neurális háló esetében, majd kiértékeli az eredményt és azt küldi vissza válaszként.

4 MEGVALÓSÍTÁS

Ebben a fejezetben a tervezett rendszer összetevőit részletezem. A rendszert négy fő komponensre osztottam fel.

Konvolúciós Neurális Háló (CNN): Ez a komponens felelős a fafajták felismeréséért. A konvolúciós neurális háló az inputként kapott képek elemzésére szolgál, és azokat a megfelelő fafajtákhoz rendeli.

Szekvenciális neurális háló: Ez a komponens végzi el az évszakok becslését a képekből kinyert domináns színek alapján.

Python Flask API alkalmazás: Ennek a komponensek a feladata a feltöltött képen adott transzformációk elvégzése, majd a kinyert és átalakított adatok átadása a neurális hálóknak a becslések elvégzésére, mind a fafajták meghatározása, mind az évszakok meghatározása esetében, majd a kapott eredmények visszaküldése.

ASP.NET Core webalkalmazás: Ez a komponens felelős a felhasználói interakcióért. Lehetővé teszi a felhasználó számára, hogy az alkalmazás különböző funkcióit, beleértve a képfelismerést és az eredmények megjelenítését, egyszerűen és hatékonyan használják.

Ezen komponensek együttesen teszik lehetővé a rendszer összehangolt működését, a képfelismeréstől kezdve egészen a felhasználó felületig, ami lehetővé teszi a hatékony kommunikációt és felhasználói élményt.

4.1 Konvolúciós neurális háló

A neurális hálót Python [20] nyelven készítettem el, Tensorflow [21] és Keras [22] keretrendszerek segítségével. A modellem alapját a Visual Geometry Group által kifejlesztett VGG-16-os modell adja, mely előre tanított rétegeket tartalmaz az ImageNet adatbázison. Hogy a modellt használni lehessen a projekten, módosítani kell, mert alapból az ImageNet adathalmaz 1000 osztályára végezne klasszifikációt.

4.1.1 Alap modell módosítása

Először is, az alapmodell után egy Flatten réteget helyeztem el, hogy a képek térbeli szerkezetét lapos vektor formájába alakítsam át.

Azután, a modell pontosságának növelése érdekében két további réteget kapcsoltam a modellhez. Az első új réteg egy 120 neuronból álló, és a ReLU[11] aktivációs függvényt alkalmazó teljesen összekötött réteg. Ezt követően egy másik dense réteget adtam hozzá,

[11] Rectified Linear unit

amely 60 neuronból áll, szintén ReLU aktivációs függvénnyel. De ez még nem lenne elég ahhoz, hogy 28 különböző osztály között végezzünk klasszifikációt. Szükség van még egy utolsó teljesen összekötött rétegre, ami 28 neuronból áll és softmax aktivációs függvényt alkalmazok rajta, így kimenetként a 28 osztályra végez majd osztályozást.

Optimalizáló algoritmusként a modell betanításához az Adam algoritmust választottam, amelynek tanulási rátája 0,001-re állítottam.

Ezek a hozzáadott rétegek segítenek a modellnek abban, hogy a képek reprezentációit megfelelően értelmezze és osztályozza a 28 kategóriába.

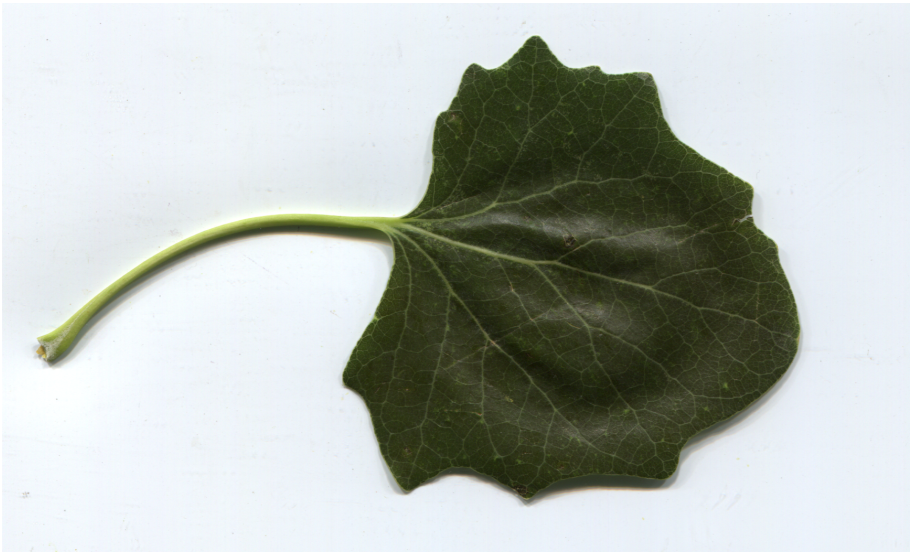
Magának az alap VGG16 modellnek a rétegein már nem végeztem tanítást, csak a hozzácsatolt rétegeken. A tanítás során early stoppingot is alkalmaztam, ami 5 epoch után, ha nem állt be javulás a validációs adathalmazon értelmezett pontosságban leállítja a tanítást.

4.1.2 Adathalmaz és előkészítése

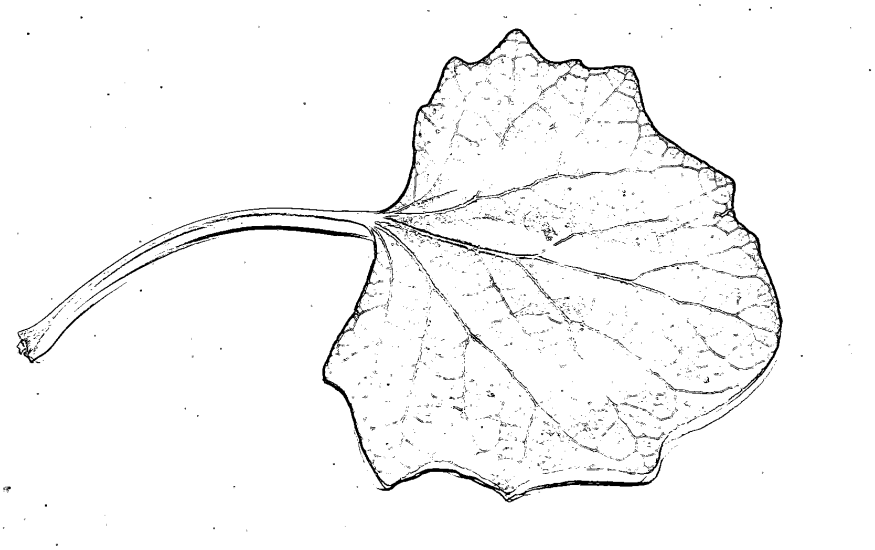
A Missouri University által összegyűjtött adathalmaz [16] mérete miatt a tanítás során adat augmentációt alkalmaztam, illetve, hogy a bementi adat illeszkedjen a VGG-16-os alapmodell elvárásaihoz 224x224 pixeles-re konvertáltam a képeket.

Ez az adathalmaz színes képeket tartalmaz, ami csökkentheti a betanított modell hatékonyságát, hiszen az évszakok változásával megváltozik a levelek színe. Éppen ezért a képeken olyan transzformációkat hajtottam végre, hogy végeredményül csak a levelek erezete és körvonala látszódjon. Majd ezekután a módosított adathalmazon végeztem el a tanítást.

A módosítás előtti állapot a 4.1. ábrán látható, míg a kapott eredmény a 4.2. ábrán.



4.1. ábra: Kép a módosítások előtt, populus canescens



4.2. ábra: Kép a módosítások után, populus canescens

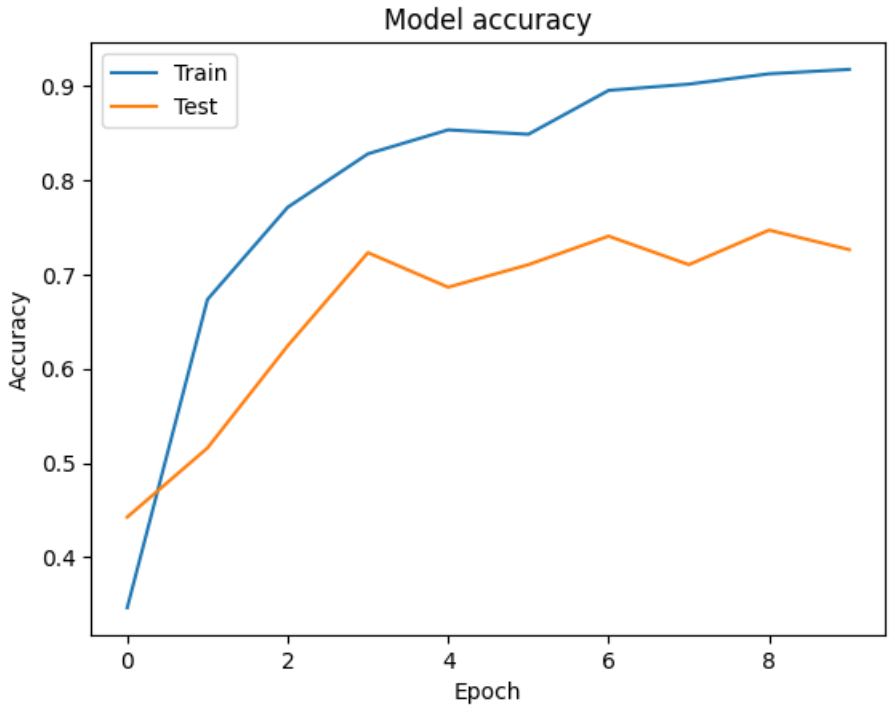
Ilyen módon a modellel megközelítőleg 70%-os pontosságot tudtam elérni. A tanítás epochonkénti eredménye a pontosságra a 4.3. ábrán látható, míg a veszteségre a 4.4. ábrán. Egy epoch az egy tanítási ciklusnak felel meg. A pontosság értékének kiszámolására az alap accuracy számítást alkalmaztam, mivel az adathalmaz minden osztálynak megközelítőleg ugyan annyi eleme van, tehát a helyes besorolásokat osztottam az össze minta számával. A háló teljesítményének alaposabb megértése érdekében kiértékeltem a teszt adathalmazon, amely alapján a modell megközelítőleg 0,86-os F1-score értéket ért el. Az F1-score a precision (pontosság) és a recall (érzékenység) harmonikus átlaga, amely egyetlen értékben összegzi a modell osztályozási képességét.

A precision azt méri, hogy az összes pozitívként besorolt minta közül mennyi volt valóban helyes, míg a recall azt mutatja meg, hogy az összes ténylegesen pozitív minta közül mennyit sikerült helyesen azonosítani. Az F1-score értéke az alábbi 4.1 képlettel számolható ki:

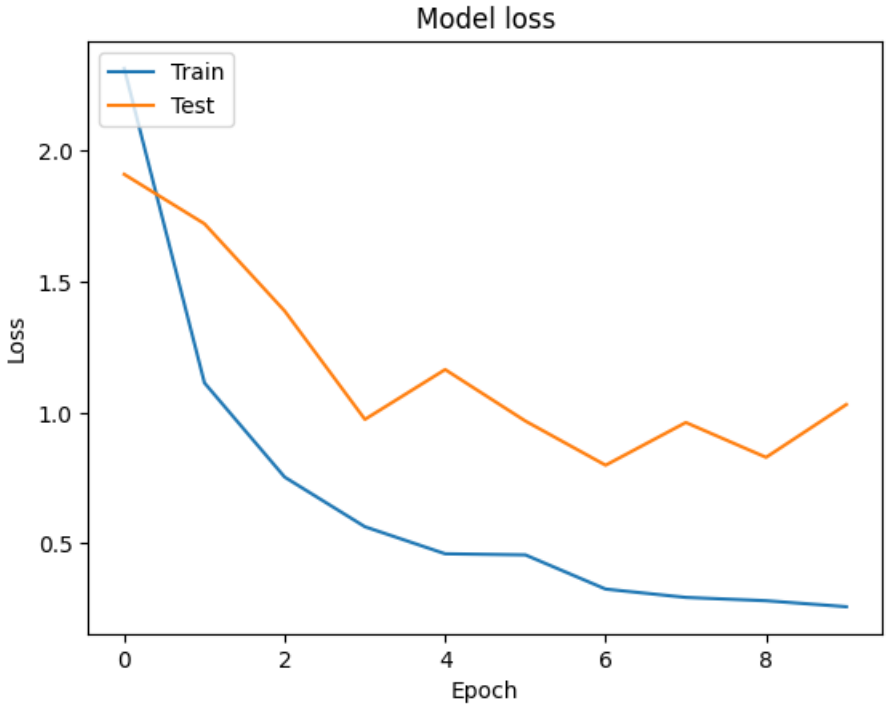
$$F1 = 2 * \frac{precision * recall}{precision + recall} \tag{4.1}$$

Az F1-score értéke 0 és 1 között mozog, ahol az 1 a tökéletes osztályozást jelenti, míg a 0 azt mutatja, hogy a modell teljesen hatástalan. Az én modellem esetében az elért 0,86-os F1-score azt mutatja, hogy a háló viszonylag jól kiegyensúlyozta a helyes pozitív osztályozásokat (precision) és a ténylegesen pozitív osztályokat (recall).

Ez az érték fontosabb lehet, mint az általános pontosság (accuracy), mivel az F1-score kiegyensúlyozatlan adathalmazok esetén is megbízható mérőszám, figyelembe véve mind a helytelen pozitív besorolásokat (False Positives), mind a kihagyott pozitív mintákat (False Negatives). Ennek köszönhetően pontosabb képet ad a modell valódi teljesítményéről.



4.3. ábra: Tanítás során elért pontosság értékek epochonként



4.4. ábra: Tanítás során elért veszteség értékek epochonként

4.2 Szekvenciális neurális háló

Ebben a szakaszban először áttekintjük a tanítási adathalmaz létrehozását végül pedig magának a neurális hálónak a felépítését és tanítását.

Mind az adathalmaz létrehozásához mind a neurális háló megalkotásához a Python programozási nyelvet választottam.

4.2.1 Tanítási adathalmaz létrehozása

Mint ahogy azt már korábban a tervezési fázisban említettem a neurális hálóhoz való adathalmazt a Flickr API használatával letöltött, majd utólag szelektált képekből alakítottam ki, méghozzá K-Means algoritmussal.

Az adatok kinyerésében három fő könyvtár játszik szerepet: cv2, numpy és scikit-learn, Ezek mindegyike széles körben használt eszköz a gép látás és gépi tanulás területén, amelyek segítségével hatékonyan és gyorsan feldolgozhatók a képek, valamint a kinyert jellemzők alapján taníthatók a modellek.

A cv2 az OpenCV nyílt forráskódú könyvtár, amelyet a gépi látási feladatokra fejlesztettek ki. A kódban a kép beolvasására és színkonverzióra használtam [23].

A NumPy egy Python könyvtár, amit elsősorban nagy tömbök (mátrixok) és numerikus adatok kezelésére használnak [24].

Végül pedig a skicit-learn egy olyan Python-alapú gépi tanulási könyvtár, amely számos algoritmust biztosít osztályozási, regressziós és klaszterezési problémák megoldásához. A kódban csak a K-Means klaszterező algoritmus implementációját használtam fel [25].

A képek feldolgozása egy négy évszakot reprezentáló mappastruktúrából indul ki: tél, tavasz, nyár és ősz. A mappák nevei és tartalmuk alapján a képeket az évszakokhoz rendeltem, amelyekhez a kód egy numerikus címkét (0 = tél, 1 = tavasz, 2 = nyár és 3 = ősz) kapcsol.

A domináns színek meghatározása a már többször említett K-Means klaszterezési algoritmus segítségével történt. Az eljárás lényege, hogy egy adott kép képpontjait az RGB színtérben csoportosítja, és tíz klaszterközéppontot (centroidot) azonosít, amelyek a kép legjellemzőbb színeit reprezentálják.

Ehhez első sorban be kell olvasni a képet, amihez a cv2 könyvtár `imread()` függvényét használtam, majd az így kapott eredményt RGB színrendszerbe konvertáltam a `COLOR_BGR2RGB` konverzióval.

Ezek után a kép mátrixát képpontok listájává alakítottam `reshape()` metódus segítségével, így mindegy egyes képpont egy (R, G, B) értéket hordozó vektorként jelenik meg.

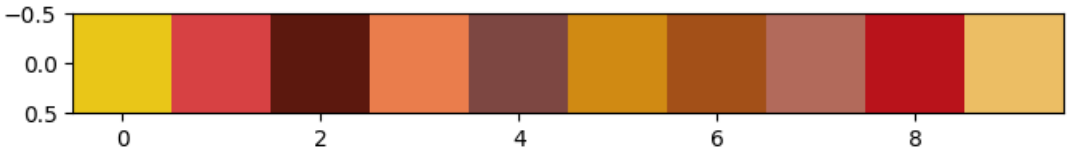
Az át alakítást követően meghívtam a scicit-learn könyvtár KMeans algoritmusát, méghozzá 10 klaszterre. Az eredményeket a `cluster_centers_` attribútum segítségével nyerjük ki.

Utolsó lépésként a klaszterközéppontokat 8 bites egész számokká alakítom, hogy azok kompatibilisek legyenek az RGB színértékekkel.

Ezt a folyamatot az összes képen végrehajtom, és mivel ez meglehetősen időigényes az eredményeket két tömbbe gyűjtöttem, X a feldolgozott jellemzővektorok tömbje, Y pedig az évszakokat reprezentáló címkék tömbje. Majd a két tömböt .npy kiterjesztésű fájlokban eltároltam.

Az alábbi képek a folyamat kiinduló állapotát (4.5. ábra) és a kapott végeredményt (4.6. ábra) jelenítik meg.

4.5. ábra: Kiindulási kép



4.6. ábra: K-Means klaszterezési eljárás után kapott 10 domináns szín a képen

4.2.2 Neurális háló felépítése az évszakok meghatározásához

Az évszakok meghatározására fejlesztett neurális háló egy szekvenciális modell, amely a Tensorflow és Keras keretrendszerek segítségével kerül megvalósításra. A modell bemeneti adatai a képekből kinyert 10 legdominánsabb szín RGB-értékei (30 bemeneti paraméter), míg a kimenet négy osztályra (tél, tavasz, nyár és ősz) végzett valószínűségi becslést ad. Az alábbiakban részletesen bemutatom a modell architektúráját és tanításának folyamatát.

A neurális háló felépítése a következő rétegekből áll. A bemeneti réteg, ahol a bemeneti adatok mérete 30, amely az RGB színértékekből adódik (10 szín * 3 csatorna). Ez a réteg fogadja a normalizált adatokat, és továbbítja azokat a rejtett rétegekhez.

Az első rejtett réteg egy teljesen összekötött (dense) 64 neuronos réteg. Ezen a rétegen a Leaky ReLU aktivációs függvényt alkalmazom, amely a hagyományos ReLU aktivációval ellentétben lehetőséget ad arra, hogy a negatív értékeknél is kis mértékben aktív maradjon a neuron, csökkentve a "dead neuron" probléma előfordulását. Ez is paraméte-

rezhető, én az alpha paramétert 0,01 értékre állítottam.

Az első réteg után következik egy Dropout réteg, amely 20%-os valószínűséggel deaktivál véletlenszerűen neuronokat a tanítási folyamat során. Ez csökkenti a túlilleszkedés[12] kockázatát, különösen igaz ez kis méretű adathalmazok esetén.

Ezek után egy újabb Dense réteg következik, ezúttal 32 neuronnal és szintén Leaky ReLU aktivációval. Ez a réteg tovább mélyíti a modell kapacitását, lehetővé téve bonyolultabb mintázatok felismerését.

Az utolsó réteg a kimeneti réteg, ami szintén teljesen összekötött, de csak 4 neuronnal, amelyek a négy évszak becslését adják. A kimeneti réteg aktivációs függvénye a softmax lett, amely biztosítja, hogy a kimeneti értékek valószínűségi eloszlást képezzenek, így a négy évszakra vonatkozó osztályok valószínűségei összeadódnak és 1-et adnak. Az architektúra szerkezete a 4.7. ábrán látható. A modell 4196 tanítható pereméterrel rendelkezik.

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 64)                1984

 leaky_re_lu_4 (LeakyReLU)   (None, 64)                0

 dropout_2 (Dropout)         (None, 64)                0

 dense_7 (Dense)             (None, 32)                2080

 leaky_re_lu_5 (LeakyReLU)   (None, 32)                0

 dense_8 (Dense)             (None, 4)                 132

=================================================================
Total params: 4196 (16.39 KB)
Trainable params: 4196 (16.39 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

4.7. ábra: Évszakokat osztályozó neurális háló szerkezete

A modell tanításához a keresztentrópia veszteségfüggvényt[13] alkalmazom, mivel ez ideális választás több osztályos kategorizálási problémák esetén, ahol az osztálycímkék numerikusak. Az optimalizálásra az Adam algoritmus szolgál, 0,001-es tanulási rátával, amely gyors és hatékony konvergenciát biztosít.

A tanítás során a korai leállás[14] technikát is alkalmazom, amely a modell túlilleszke-

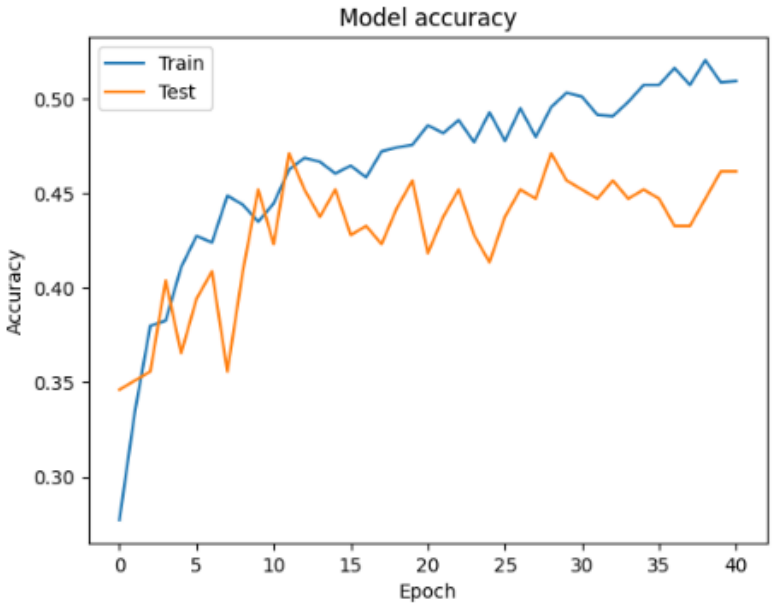
[12] overfitting
[13] sparse_categorical_crossentropy
[14] early stopping

désének megelőzésére szolgál. Az algoritmus a validációs veszteség értékét figyeli, ha 10 egymást követő epoch során nem tapasztalható javulás, a tanítás leáll. A legjobb validációs eredmény elérésekor mentett súlyokat visszaállítja, ezzel biztosítva az optimális modellváltozatot.

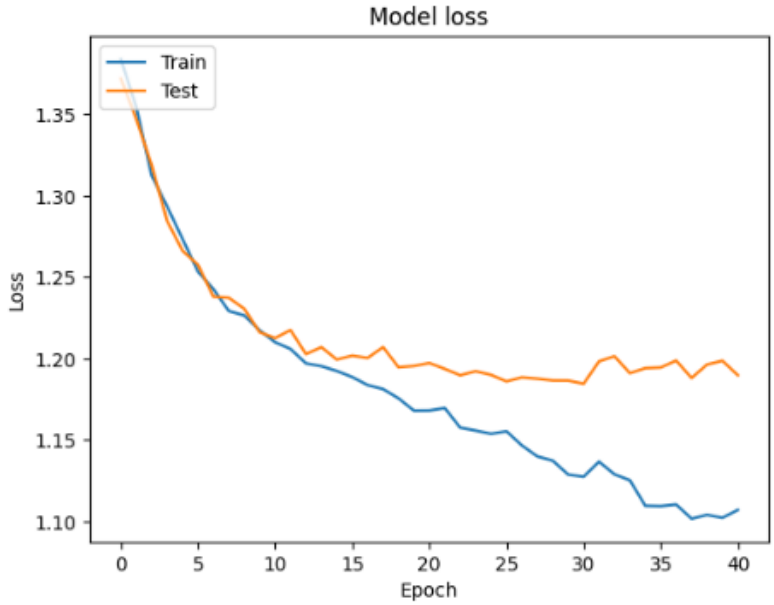
A tanítás végeztével a modell hatékonysága nem kiemelkedő, de az általános információ szolgáltatáshoz, amire felhasználom elegendő. A tanítás epochonkénti pontossági eredménye a 4.9. ábrán látható, míg a veszteség értékek a 4.10. ábrán. A pontosság számításához a fa fajtákat osztályozó neurális hálónál alkalmazott eljárást alkalmaztam, mivel az adathalmaz itt is relatív kiegyenlített, valamint kiszámítottam a 4.1. képlettel a betanított modell F1 értékét is, ami a 4.8. ábrán megtekinthető. A modellt, a fákat azonosító neurális hálóhoz hasonlóan HDF formátumban mentem el.

```
F1 Score: 0.5002504942075038
Precision: 0.5191208974331094
Recall: 0.5168269230769231
```

4.8. ábra: Évszakokat osztályozó neurális háló eredményei



4.9. ábra: Évszakokat osztályozó neurális háló pontossági értékei epochonként



4.10. ábra: Évszakokat osztályozó neurális háló veszteség értékei epochonként

4.3 Flask API

A Flask API [26] alkalmazás célja, hogy a betanított neurális hálók segítségével képeket dolgozzon fel, predikciót végezzen és az eredményeket a felhasználó számára visszaküldje. Az alkalmazás több különálló modult tartalmaz, amely a képek előfeldolgozását, a neurális hálók betöltését, valamint a predikciók és az eredmények kezelését végzik.

4.3.1 API alkalmazásban felhasznált csomagok

Az API működéséhez az alábbi csomagokra volt szükségem:

- Flask: A REST API működését biztosító könnyűsúlyú keretrendszer,
- Tensorflow és Keras: A betanított neurális hálók betöltésére és predikciók futtatására.
- Pillow és skimage: A képek előfeldolgozásához szükséges könyvtárak
- Matplotlib: A transzformált képek mentéséhez és megjelenítéséhez.
- Scikit-learn: A kiemelt színek meghatározása K-Means klaszterezési algoritmus segítségével

4.3.2 API alkalmazás felépítése és működése

A neurális hálók tanítása után a modell paramétereit HDF5 formátumban mentettük el, amelyeket a Flask API indításakor betöltünk, hogy biztosítsuk a predikciók gyors és ha-

tékony végrehajtását.

Az API egyetlen fővégponttal rendelkezik, ez a `/predict`, amely egy POST [27] kérést fogad el egy kép fájl formájában, és a lentebb leírt lépések végrehajtásával tér vissza a becslések eredményével.

Először a feltöltött képet ideiglenesen eltároljuk a fájlrendszerben, majd ellenőrizzük, hogy létezik-e az adott fájl, illetve, hogy megfelelő-e a formátuma.

A fájl mentését és ellenőrzését követően a kép feldolgozása két fő lépésben történik: transzformáció és jellemzők kinyerése, amelyeket a `transform_image()` és az `extract_dominant_colors()` metódusok végeznek el. Először a transzformációt írom le, utána pedig a képek kiemelését.

A transzformáció célja, a kép egyszerűsítése és az él- és kontúrinformáció kiemelése, hogy a fafajtákat felismerő neurális háló számára releváns adatokat biztosítsunk. A `transform_image()` metódus a következő lépéseket hajtja végre.

Először a képet szürkeárnyalatosra konvertálja, amely lehetővé teszi az élek hangsúlyozását majd meghatározzuk a szürkeárnyalatos kép optimális Otsu-küszöbértékét, amelyet a skimage könyvtár `filters.threshold_otsu()` metódusa generál, ez után pedig a küszöbérték alapján a kép pixelei bináris osztályokba kerülnek.

Ezt követően a `restoration.denoise_tv_chambolle()` metódussal kisimítom a képet, csökkentve a zajt, és megkönnyíti a későbbi feldolgozást. A képen ez után kontrasztfokozás érdekében az intenzitás értékek átskálázásával kiemelem a releváns részleteket.

Végül már csak inverzálom a képet az `np.invert()` metódus segítségével és végső lépésben a `filters.sobel()` metódussal kiemelem a kép éleit. Az élesített kontúrok informatív bemenetként szolgálnak a további feldolgozáshoz

A transzformáció végeztével a kapott eredményt szintén elmentjük a fájlrendszerben, majd pedig a `preprocess_image()` metódussal megnyitjuk, ami úgy alakítja át a transzformált képet, hogy a fafajtákat meghatározó neurális háló feltudja azt dolgozni.

A második főlépésben az eredeti színes képből olvassuk ki a domináns színeket az `extract_dominant_colors()` metódussal. Ez a folyamat nagyon hasonló ahhoz, amit az évszakokat osztályozó neurális háló adathalmazának létrehozásához használtam. A kép pixel értékeit vektorrá alakítva átadom a K-Means algoritmusnak, hogy képezze le a tíz szükséges klasztert, amelyek a kép domináns színeit reprezentálják. Ezeket az értéket normalizáljuk (0 és 1 közé), majd egy bemeneti vektorként előkészítjük az évszakok osztályozásáért felelős neurális háló számára.

A neurális hálók betöltése a Flask API indításakor történik meg. Az előfeldolgozott

kép és a kinyert domináns színek bemenetként szolgálnak a két különálló modell számára. Mindkét modellen elvégezzük a megfelelő bemenettel a predikciót, majd a becslések eredménye JSON formátumban kerül visszaküldésre. A predikciók eredményei a legmagasabb valószínűséggel rendelkező osztály alapján kerülnek meghatározásra az `np.argmax()` függvény segítségével.

A folyamat részeként az ideiglenesen mentett képek automatikusan törlődnek a fájlrendszerből, ezzel biztosítva a biztonságos működést és az erőforrások hatékony kezelését. Az API robosztus hibakezelési mechanizmust alkalmaz, amely részletes hibajelentéseket nyújt a felhasználó számára, miközben fenntartja a rendszer zavartalan működését.

Ez az integrált felépítés lehetővé teszi az API számára, hogy hatékonyan és pontosan végezze el a képfeldolgozást és a predikciókat, miközben biztosítja a moduláris fejlesztési lehetőségeket a jövőbeni bővítésekhez.

4.4 Webalkalmazás

Az ASP.NET Core webalkalmazás célja, hogy intuitív felhasználói felületet biztosítson a rendszer számára, lehetővé téve a felhasználók kezelését, valamint a fénykép alapján történő predikciók kezelését. Az alkalmazás több rétegre bontott architektúrát követ, amely biztosítja a modularitást, a könnyű karbantarthatóságot, valamint a különböző funkciók logikai elkülönítését. Az alkalmazás fő rétegei:

- Model: Az üzleti logikához szükséges adatszerkezetek és entitások.
- Data: Az adatbáziskapcsolat és adattárolás kezeléséért felelős réteg.
- Repository: Az adatbázis műveletek absztrakciója, amely közvetlenül kezelő a tárolt adatokat.
- Logic: Az üzleti logika rétege, amely a Repository által biztosított funkciókat kiegészíti és irányítja.
- Client: Az ASP.NET Core MVC alapú felület, amely a felhasználói interakciók kezeléséért és a végpontok biztosításáért felelős.

Ebben a fejezetben sorban haladva végig veszem az alkalmazás rétegeit.

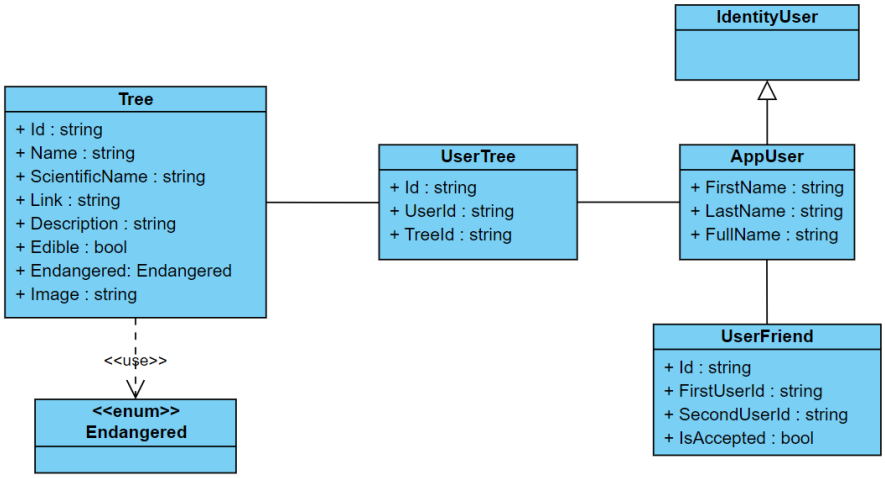
4.4.1 Model réteg

A Model réteg az alkalmazás adatmodelljeit tartalmazza, amelyek meghatározzák az adatstruktúrákat és azok közötti kapcsolatokat. Ilyen az `AppUser` osztály, ami `IdentityUser` osztályból származik, amihez szükség van az ASP.NET Core Identity keretrendszerre, és ez az osztály feleltethető meg az alkalmazás felhasználójának. [28] De itt található meg a `Tree` osztály is, ami pedig a fa fajták modellje.

A több-a-többhöz kapcsolatok miatt összekötő osztályok is vannak ebben a rétegben. Ilyen a `UserFriend` osztály, amivel a baráti kapcsolatokat lehet leírni, vagy a `UserTree` osztály, ami azt reprezentálja, hogy egy adott felhasználó hozzáadott egy predikcióval kiválasztott fát a fiókjához.

Ezen kívül még van egy enum osztály az `Endangered` is a fa fajták veszélyeztetettségi szintjének számára, ez lehet: Nem veszélyeztetett, Veszélyeztetett, Kihalt vagy pedig Ismeretlen állapotú.

Az osztályok közötti kapcsolatok UML diagramja (lásd 4.11. ábra) jól szemlélteti a réteg összetettségét és az entitások közötti viszonyokat.



4.11. ábra: Model réteg fő osztályainak diagramja

4.4.2 Data réteg

A Data réteg az adatbáziskapcsolat és az adatmodell tárolásának kezelését biztosítja az Entity Framework Core segítségével.

Az `ApplicationDbContext` osztály, amely a `DbContext` osztályból származik, ezt az adatbáziskapcsolatot valósítja meg. Ez az osztály tartalmazza az adatbázisban tárolt entitásokkal való kommunikációt biztosító `DbSet` tulajdonságokat. A `Users` a felhasználói adatok tárolására, a `Trees` a fa fajták tárolására, a `UserTrees` a felhasználókhoz kapcsolt fák tárolására és végül a `UserFriends` a felhasználók közötti kapcsolatok tárolására.

Az adatbázis létrejöttekor inicializálok az `OnModelCreating` metódus segítségével egy alapértelmezett adminisztrátor felhasználót és a konvolúciós neurális háló által ismert 28 fa fajtát.

Az adatbázis konfigurációja .NET 6-on alapul, ami az alábbi NuGet csomagokat tartalmazza:

- Microsoft.AspNetCore.Identity.EntityFrameworkCore: Ez a csomag tartalmazza az azonosítás kezeléséhez szükséges osztályokat és funkciókat.
- Microsoft.EntityFrameworkCore: Az Entity Framework Core keretrendszer, amely lehetővé teszi az adatbázis műveletek végrehajtását.
- Microsoft.EntityFrameworkCore.Proxies: Ez az opció lehetővé teszi a lazy loadingot az alkalmazásban.
- Microsoft.EntityFrameworkCore.SqlServer: Az SQL Server adatbáziskezelési támogatás.
- Microsoft.EntityFrameworkCore.Tools: Ez az eszközök csomagja, amely segíti az Entity Framework Core parancssori műveleteit.

Az Entity Framework Core segítségével megvalósított adatbáziskapcsolat nemcsak robusztus, hanem lehetővé teszi a fejlesztők számára, hogy minimális erőfeszítéssel módosítsák vagy bővítsék az adatstruktúrát.

4.4.3 Repository és Logic réteg

Az alkalmazások fejlesztése során a szoftverarchitektúrák részeként a Repository és Logic rétegek kritikus szerepet töltenek be az adatok kezelésében és az üzleti logika megvalósításában. Ezek a rétegek a szoftverek strukturális és funkcionális elkülönülését szolgálják, lehetővé téve az alkalmazások számára az adatok hatékony kezelését és az üzleti folyamatok egyértelmű és szétválasztott megvalósítását.

A Repository réteg az adatelérés és adatmanipuláció központi helyszíne. Feladata az adatbázis kapcsolódás kezelése és az adatok lekérdezése, hozzáadása, frissítése és törlése az adatbázisból. Ez a réteg általában interfészeket és osztályokat tartalmaz, amelyek az adatbázishoz való hozzáférést biztosítják, miközben elrejtik a konkrét adatbázis implementáció részleteit.

A Logic réteg az üzleti logika székhelye. Ennek a rétegnek az a feladata, hogy az üzleti szabályokat és logikát valósítsa meg az alkalmazásban. Ide tartozik az adatok feldolgozása, validálása, azokon végzett műveletek és az üzleti folyamatok megvalósítása. Ez a réteg általában független az adatbázistól, és az adatokon végzett műveleteket az üzleti szabályokhoz és követelményekhez igazítja.

Az elválasztás ezek között a rétegek között lehetővé teszi, hogy az alkalmazás könnyebben karbantartható legyen. Például, ha az adatbázis struktúrája változik, a Repository réteget kell frissíteni, míg az üzleti logikát változatlanul lehet hagyni, mivel az független az adatbázis konkrét implementációjától.

Ezen rétegek szétválasztása hozzájárul az alkalmazások modularitásához és skálázhatóságához is. A Repository réteg könnyen cserélhető más adatbázisokkal való működés esetén, mivel az csak az adatelérés interfészét biztosítja. Ugyanakkor a Logic réteg a vállalati logikát kezeli, ami kevésbé függ az adateléréstől, így nagyobb fokú rugalmasságot és kezelhetőséget biztosít az alkalmazás számára.

Ezen két réteg együttesen az alkalmazások architektúráját hatékonyabbá és skálázhatóbbá teszi, lehetővé téve az adatok kezelését és az üzleti logika megvalósítását úgy, hogy az könnyen karbantartható, könnyen bővíthető és szilárd alapot nyújt a fejlesztéshez.

A Repository réteg a struktúrájában egy kizárólagos osztályt és a vele párosított interfészt tartalmazza, az `ApplicationRepository` osztályt. Ez az osztály foglalkozik az alkalmazás főmodelljeinek CRUD[15] műveleteinek megvalósításával. Az ApplicationRepositoryban találhatók meg ezek a funkciók, melyek az adatok létrehozásától a frissítésen át a törlésig terjednek. Az osztály működéséhez elengedhetetlen az `ApplicationDbContext` osztály, amit a repository osztály konstruktorában dependency injection segítségével kap meg.

A Logic réteg hasonló felépítésű, és itt az `ApplicationLogic` osztály nyújtja a különböző funkciókat, szintén egy vele párosított `IApplicationLogic` interfésszel. A logic osztály feladata az `ApplicationRepository` osztály metódusainak meghívása, amelyekkel a Logic réteg valósítja meg az alkalmazás üzleti logikáját. Ehhez a funkcióhoz a Logic réteg is a dependency injection elvét használja, így fér hozzá az `ApplicationRepository` metódusaihoz.

A Repository és Logic rétegek közötti együttműködés jól szemléltethető egy konkrét példával, amelyben egy adott fafajta adatait kérdezzük le a fafajta nevének ismeretében. A folyamat során a Logic réteg felelős a bemeneti adatok előkészítéséért, míg a Repository réteg végzi az adatbázis-műveletet.

A Logic rétegben található metódus feladata, hogy egy adott fafajta nevét megkapva előkészítse azt az adatbázisban történő kereséshez. Például, a bemenetként kapott név nagybetűs formátumra alakítása biztosítja, hogy az összehasonlítás kis- és nagybetű érzékenységtől függetlenül helyes eredményt adjon. A Logic réteg ezután meghívja a Repository réteg megfelelő metódusát, amely a név alapján keres a fák adatbázisában.

A Repository rétegben az adatbázis-lekérdezés az alkalmazás adatbázisát kezelő DbContext használatával történik. A rendszer megkeresi azt a sort a fák táblájában, amelynek neve megegyezik a megadott értékkel. Ha talál egyezést, visszaadja a megfelelő modellt, amely tartalmazza a fafajta adatait, például annak nevét, veszélyeztetettségi státuszát, vagy egyéb tulajdonságait.

[15]Create Read Update Delete

Ez a példafolyamat jól bemutatja a rétegek közötti szerepek tiszta szétválasztását. A Logic réteg kizárólag az üzleti logikáért felel, az adatelérés részletei teljes mértékben a Repository rétegben valósulnak meg. Ez az architektúra biztosítja az alkalmazás karbantarthatóságát, mivel a két réteg egymástól függetlenül módosítható vagy bővíthető anélkül, hogy az egyik változtatása szükségszerűen érintené a másikat.

4.4.4 Client réteg

A Client réteg a webalkalmazás ügyféloldali részét valósítja meg, és a felhasználók számára biztosítja a felhasználói felületet. Ez a réteg ASP.NET Core 6 alapú, amely egy nyílt forráskódú, platformfüggetlen és nagy teljesítményű keretrendszer a webes alkalmazások fejlesztéséhez. Az ASP.NET Core 6 modern kódstruktúrája, beépített biztonsági funkciói, gazdag könyvtárkészletei és eszközei révén gyors fejlesztési ciklust és könnyű skálázhatóságot tesz lehetővé.

Ezen réteg egyik kulcsfontosságú eleme az ASP.NET Core Identity keretrendszer, amely a felhasználók hitelesítését, azonosítását és jogosultságkezelését biztosítja. Az Identity lehetővé teszi a felhasználók regisztrációját, bejelentkezését, kijelentkezését, valamint olyan funkciókat, mint a jelszó módosítása vagy fiók törlése. Ezen felül szerepek és jogosultságok kezelését is biztosítja, így az alkalmazás rugalmasan szabályozhatja az egyes funkciókhoz való hozzáférést.

A Client réteg MVC (Model-View-Controller) architektúrát követ, amely egy elterjedt minta a webes alkalmazások tervezéséhez. Az MVC architektúra három fő komponensre osztja az alkalmazást:

- Model: az alkalmazás adatmodelljét és üzleti logikáját tartalmazza.
- View: a felhasználói felület megjelenítésit logikáját valósítja meg.
- Controller: a vezérlési logikát és a felhasználói interakciókat kezeli.

Ez a szerkezet biztosítja a kód modularitását, újrafelhasználhatóságát és könnyű karbantarthatóságát.

A Client rétegben ezeket a NuGet csomagokat használom:

- Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore: Ez a csomag lehetővé teszi, hogy a webalkalmazás diagnosztizálja és kezelje az Entity Framework Core által okozott hibákat, mint például a migrációs hibák vagy az adatbázis kapcsolódási hibák.
- Microsoft.AspNetCore.Identity.EntityFrameworkCore: Ez a csomag lehetővé teszi, hogy a webalkalmazás az Entity Framework Core-t használja az ASP.NET Core Identity keretrendszer adatmodelljének tárolására és kezelésére, továbbá a csomag

biztosítja a szükséges osztályokat, metódusokat, konfigurációkat, a felhasználók, szerepek, igények, tokenek adatbázisban való kezeléséhez.
- Microsoft.AspNetCore.Identity.UI: Ez a csomag lehetővé teszi, hogy a webalkalmazás kész UI komponenseket használjon az ASP.NET Core Identity keretrendszer funkcióinak megjelenítéséhez, mint például a regisztráció, a bejelentkezés, a jelszó változtatás. Valamint ez a csomag biztosítja a szükséges Razor Pages-eket, nézeteket, vezérlőket, stílusokat, a felhasználói felület létrehozásához és testre szabásához.
- Microsoft.EntityFrameworkCore.SqlServer: Ez a csomag lehetővé teszi, hogy a webalkalmazás az Entity Framework Core-t használja a SQL Server adatbázis motorral való kommunikációhoz. Biztosítja a szükséges drivert, konfigurációt, adatbázis specifikus funkciókat az adatbázis kapcsolat létrehozásához és kezeléséhez.
- Microsoft.EntityFrameworkCore.Tools: Ez a csomag lehetővé teszi, hogy a webalkalmazás használja az Entity Framework Core eszközeit, mint például a migrációk, a modell generálás, a kódbázis frissítés, valamint biztosítja a szükséges parancssori és grafikus felületű eszközöket, amelyek segítségével a fejlesztők könnyen kezelhetik az adatbázis változásait és szinkronizálhatják a modellt a kóddal.
- Microsoft.VisualStudio.Web.CodeGeneration.Design: Ez a csomag lehetővé teszi, hogy a webalkalmazás használja a Visual Studio kódgeneráló eszközeit, amelyek segítségével a fejlesztők gyorsan létrehozhatnak és módosíthatnak MVC komponenseket, mint például a modell, a nézet, a vezérlő, a ViewComponent. Ez a csomag biztosítja a szükséges sablonokat, konvenciókat, szabályokat a kódgenerálás elvégzéséhez.

A Client réteg három fő vezérlőt tartalmaz:

- ErrorController: Hibák kezelésére szolgál, különböző felhasználói felületeket biztosítva az egyes hibák esetén.
- HomeController: Az alapfunkciók, például a kezdőlap (Home) és az adatvédelmi oldal (Privacy) kezelését végzi. Ezen felül a barátok funkciót is kezeli, amely magába foglalja a barátok listázását, a baráti kérelmek küldését, valamint azok elfogadását vagy elutasítását.
- TreeController: A fő funkcionalitásokért felel, mint például a fagyűjtemény megtekintése, egy új fa hozzáadása (HTTP POST kérésen keresztül a Flask API alkalmazásnak), valamint a gyűjtemény menedzselése.

A webalkalmazás központi funkciója a fa felismerése és évszak meghatározása neurális hálóval, méghozzá egy falevélről készült kép elküldésével. Ez a funkció a következőképpen zajlik:

- A felhasználó feltölt egy falevélről készült képet a webalkalmazásba, amit a Tree-

Controller osztály `UploadImageForJS` metódusa fogad. A metódus ellenőrzi, hogy a kép érvényes-e, és ha igen, akkor átalakítja byte tömbbé, és elküldi a Flask API alkalmazásnak egy HTTP POST kéréssel.
- A Flask API alkalmazás fogadja a képet, és feldolgozza úgy, hogy a betanított neurális hálók értelmezni tudják, majd a predikciók elvégzése után visszatér az eredménnyel egy JSON üzenetben, amiben a `PredictedClass` mező tartalmazza a fa fajtáját és a `PredictedSeason` tartalmazza a becsült évszakot.
- A webalkalmazás megkapja a JSON üzenetet, és a `PredictionResult` osztály mentén megtörténik a deszerializálás. A webalkalmazás megkeresi a `Tree` osztályban a megfelelő fafajtát a `PredictedClass` alapján, és megjeleníti a kapott eredményt egy felugró ablakban. Ez látható a 4.14. ábrán.
- Ezen az ablakon a felhasználó láthatja a fa tulajdonságait, illetve amennyiben, az adott fa még nem része a gyűjteményének dönthet annak gyűjteménybe csatolásáról.

A Client réteg minden szükséges vezérlő akcióhoz tartozik egy megfelelő nézet, mint TreeCollection, Details, Index. Ezek a nézetek Razor Pages formátumban vannak megírva, ami egy HTML és C# kódot összekötő szintaxis. A Razor Pages segítségével a nézetek dinamikusan generálhatók és frissíthetők az adatok és a logika alapján.

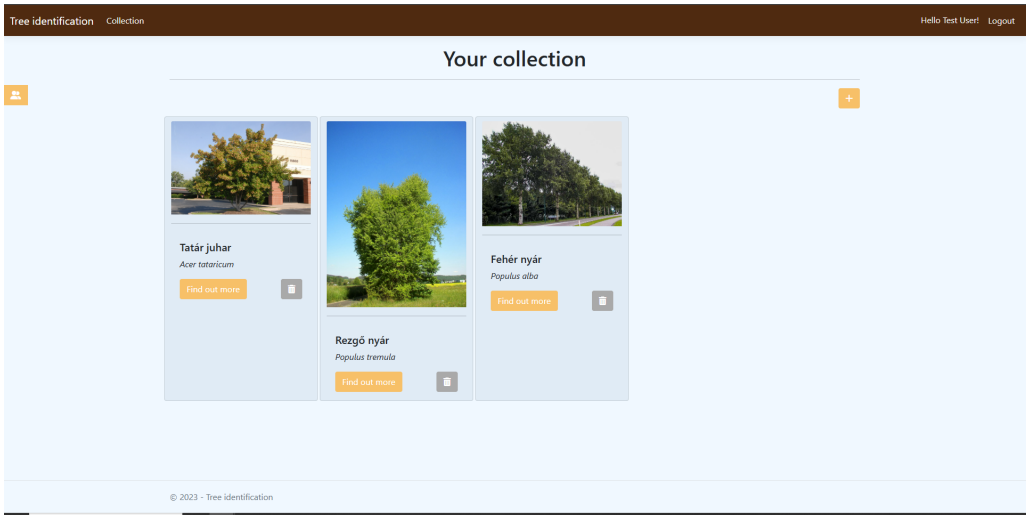
A webalkalmazás felhasználói felületének megtervezéséhez a Bootstrap 5 keretrendszert használok, ami egy népszerű és modern CSS keretrendszer, és amely segítségével reszponzív, mobilbarát és esztétikus weboldalakat lehet létrehozni. A Bootstrap 5 számos előre definiált stílust, komponenst, elrendezést, ikont kínál, amelyeket a webalkalmazás használhat és testre szabhat a saját igényei szerint. A webalkalmazás továbbá használ ViewComponent-eket is, amelyek kisebb, önálló UI elemeket valósítanak meg, amelyeket újra lehet használni a különböző nézeteken, ilyen elemeket használok a barátok és a fák megjelenítéséhez.

A webalkalmazás kliensoldali funkcionalitásának megvalósításához JavaScriptet is használok, amely lehetővé teszi a dinamikus és azonnali felhasználói interakciókat a Razor Pages alapú nézetekkel. Számos modern JavaScript eszközt és technológiát alkalmazok, mint a jQuery AJAX hívások a dinamikus adatlekérdezéshez és -küldéshez. Felhasználói értesítéshez Toastr-t alkalmazok, hogy vizuálisan vonzó és figyelem felkeltő visszajelzéseket adhassak. Végül pedig Bootstrap 5 alapú modális ablakok segítségével biztosítom a részletek megjelenítését.

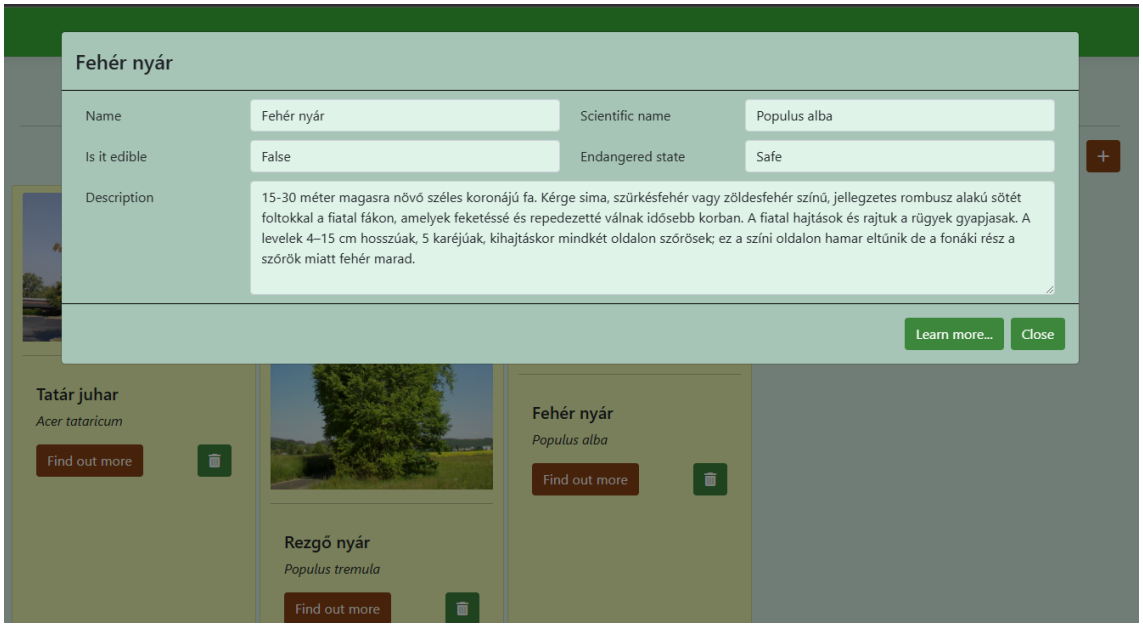
4.4.5 Webalkalmazás felülete

Az alap Bootstrap CSS megjelenést kiegészítettem még további saját CSS stílusokkal, amelyeket az évszakok színei inspiráltak. Az alkalmazás automatikusan az éppen aktuális

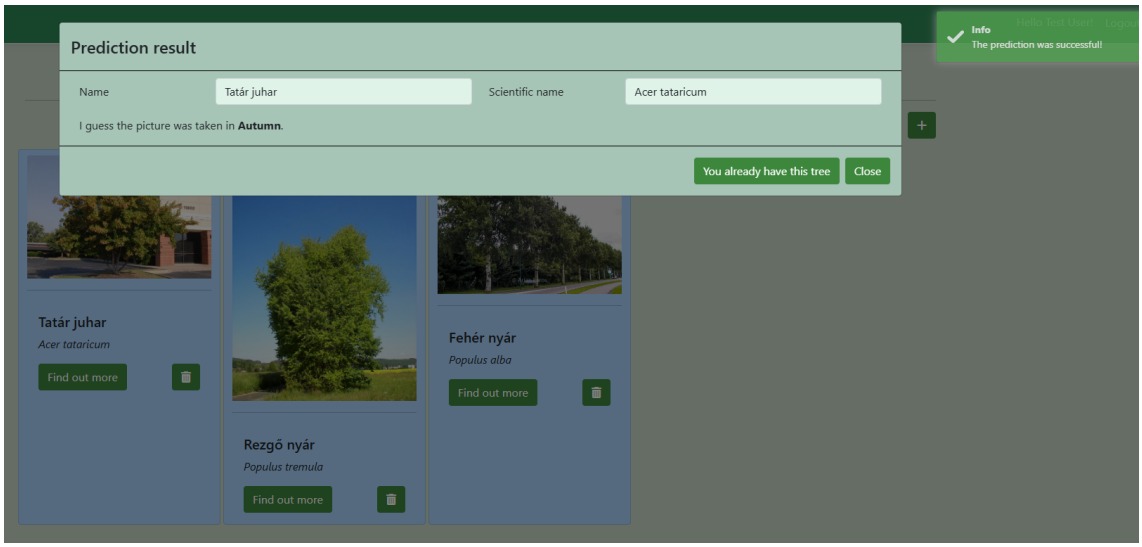
évszakhoz kapcsolódó stílust fogja alkalmazni. Ezek az évszakonkénti stílusok láthatóak a 4.12. ábrán, ahol a gyűjtemény oldal jelenik meg a tél színeiben, vagy a 4.13. ábra, ami tavaszi stílussal jeleníti meg egy fának a részleteit. Látható még, a 4.14. ábrán a predikció eredménye nyári stílusban, és végül pedig a 4.15. ábrán őszies színekben a barátokat kezelő beúszó mező.



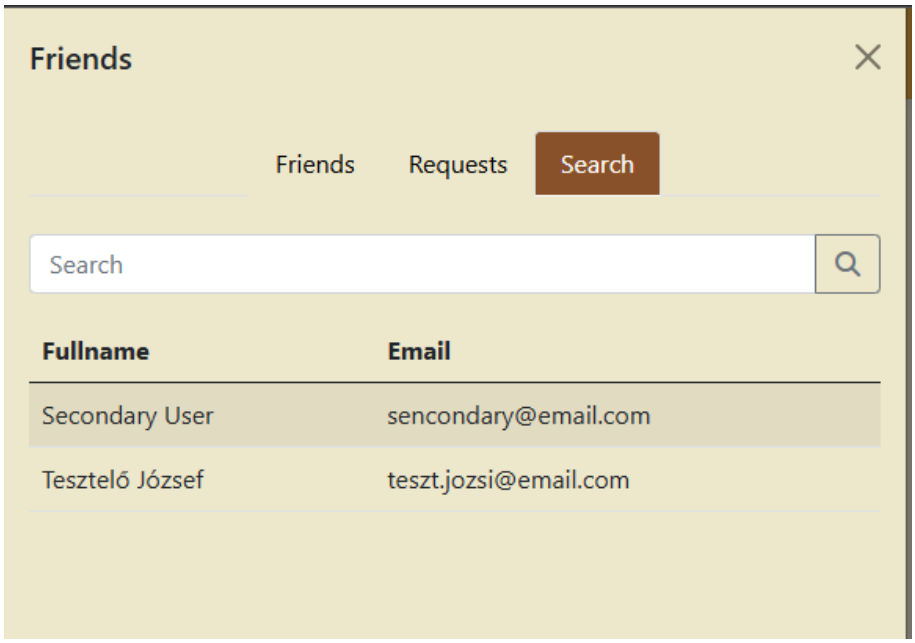
4.12. ábra: Fákat tartalmazó gyűjtemény, téli stílus



4.13. ábra: Adott fa részletei, tavaszi stílus



4.14. ábra: Új fa hozzáadására kapott eredmény, nyári stílus



4.15. ábra: Barátnak jelölési oldal, őszi stílus

5 ÖSSZEFOGLALÁS

A szakdolgozat célja egy olyan webalkalmazás kifejlesztése volt, amely lehetővé teszi a felhasználók számára, hogy falevelek alapján meghatározzák a fafajtákat és az évszakot neurális hálók használatával. Az alkalmazás két fő komponensből áll: egy felhasználóbarát webes interfészből, amely az interakciókat biztosítja, és a neurális háló szolgáltatásból, amely az elemzéseket végzi.

A fejlesztett alkalmazás fő funkciói között szerepel a regisztráció, bejelentkezés, falevelek feltöltése, gyűjtemények kezelése, valamint az eredmények interaktív megjelenítése. A rendszer támogatja az évszakok meghatározását a feltöltött képek domináns színei alapján, valamint a fafajok osztályozását két külön előre betanított neurális háló segítségével.

Az irodalomkutatás részben áttekintettem a hasonló alkalmazásokat, mint például a LeafSnap és a Plant.id, valamint megvizsgáltam a falevelek felismerésének módszereit és technikáit, ideértve a konvolúciós neurális hálókat is. Az évszakok meghatározása terén külön figyelmet fordítottam a domináns színek kinyerésére, például a K-Means klaszterezés használatával. Majd ezek közül kiválasztottam azokat a technikákat, amiket felhasználni kívántam, ilyen a falevél éleinek és erezetének kiemelése, a kép domináns színeinek kiemelése, valamint az alapként használni kívánt neurális hálót.

A neurális háló részlegben bemutattam a projektben alkalmazott technológiákat, például a VGG-16 modellt, amelyet az ImageNet adathalmazon előtanítottak, és továbbfejlesztettem a fafajok felismerésére. Az adathalmaz, amely a Missouri Egyetem nyilvános gyűjteményéből származik, 28 különböző fafaj képeit tartalmazza, és a modell finomhangolása ezen képeken történt. Az évszakok meghatározásához saját, több mint ezer képből álló adathalmazt állítottam össze, amelyeket négy évszak szerinti csoportokba rendeztem, majd előkészítettem őket a neurális háló betanítására.

A rendszer megtervezésére a funkciók listájának és a felhasználói szinteknek a meghatározásával került sor. A felhasználói funkciók közé tartozik a regisztráció, bejelentkezés, falevél-feltöltés, gyűjteménykezelés és barátok kezelése. A felhasználói szintek a regisztrált és nem regisztrált felhasználók közötti különbséget határozzák meg.

Ezek után az alkalmazást szét bontottam két külön álló részre, a funkciójuk alapján és hogy megkönnyítsem a neurális hálók beimportálását a projektbe. Az egyik alkalmazás egy FlaskAPI alkalmazás, ami a betanított neurális hálókat fogja használni, míg a másik ASP.NET Core webalkalmazás pedig a felhasználók és a FlaskAPI alkalmazás közti kommunikációt biztosítja.

Ezeket követően megkezdődött a korábban meghatározott tervek alapján a rendszer

implementációja, tesztelése és finomhangolása törekedve arra, hogy az alkalmazás a tervezett specifikációknak megfelelően működjön. Ez magába foglalja az alkalmazások kódjának megírását, az adathalmaz átalakítását és előkészítését, valamint a neurális háló felépítését, tanítását és kiértékelését.

Mindkét neurális háló esetén elvégeztem az F1-score számítást, amivel egy stabilabb képet kapok a hálók teljesítményéről. A fa fajtákat osztályozó neurális hálónál 0,86-os értéket értem el, ami jónak számít, tekintve, hogy a 28 fafajta közül soknak alig tér el a levele egymástól, az évszakokat becslő neurális háló esetén pedig 0,5-ös értéket kaptam, ami elegendő arra, hogy egy megközelítő becslést adjak az évszakokra. Az alkalmazás teljesítményét számos felhasználói teszt során is validáltam, amelynek eredményeként a rendszer általánosan pontos és felhasználóbarát működést mutatott.

A rendszer jövőbeli fejlesztései során új funkciók bevezetésére helyezném a hangsúlyt, amelyek tovább növelhetik a felhasználói élményt. Ezek közé tartozik például egy kertgondozós minijáték beépítése, amelyben a felhasználók virtuális kerteket alakíthatnak ki, és az alkalmazásban összegyűjtött fafajokhoz kapcsolódó interaktív feladatokat végezhetnek. Ez a funkció nemcsak szórakoztató, de oktató jellegű is lehetne, mivel közelebb hozhatná a felhasználókhoz a természetismeretet. Emellett további közösségi funkciók bővítését is tervezem, például ranglisták és események szervezésének lehetőségét, amelyek erősíthetnék a felhasználói közösséget.

6 SUMMARY

The aim of this thesis was to develop a web application that allows users to identify tree species and seasons based on leaf images using neural networks. The application consists of two main components: a user-friendly web interface that facilitates interaction, and a neural network service that performs the analyses.

The main functions of the developed application include registration, login, leaf image upload, collection management, and interactive result display. The system supports the determination of seasons based on the dominant colors of the uploaded images, as well as the classification of tree species using two separate pre-trained neural networks.

In the literature review section, I examined similar applications such as LeafSnap and Plant.id, and explored methods and techniques for leaf recognition, including convolutional neural networks. In the determination of seasons, particular attention was given to extracting dominant colors, for instance, using K-Means clustering. From these, I selected the techniques I intended to use, such as highlighting leaf edges and veins, emphasizing the dominant colors of the image, and the neural network to be used as a base.

In the neural network section, I presented the technologies used in the project, such as the VGG-16 model pre-trained on the ImageNet dataset, which I further refined for tree species recognition. The dataset, sourced from the public collection of the University of Missouri, contains images of 28 different tree species, and the model was fine-tuned on these images. For the determination of seasons, I compiled my own dataset of more than a thousand images, organized into groups according to the four seasons, and prepared them for neural network training.

The system design involved defining the list of features and user levels. User features include registration, login, leaf image upload, collection management, and friend management. User levels distinguish between registered and non-registered users.

Following this, the application was divided into two separate parts based on their functionality and to facilitate the import of neural networks into the project. One application is a FlaskAPI application that will use the trained neural networks, while the other is an ASP.NETCore web application that ensures communication between users and the FlaskAPI application.

Subsequently, the system was implemented, tested, and fine-tuned according to the previously defined plans, striving to ensure the application operates according to the designed specifications. This includes writing the application code, transforming and preparing the dataset, and building, training, and evaluating the neural network.

For both neural networks, I calculated the F1-score to obtain a more stable picture

of the networks' performance. The neural network for classifying tree species achieved a value of 0.86, which is considered good given that many of the 28 tree species have very similar leaves. For the neural network estimating the seasons, I achieved a value of 0.5, which is sufficient to provide an approximate estimation of the seasons. The application's performance was also validated through numerous user tests, resulting in a generally accurate and user-friendly operation.

Future developments of the system will focus on introducing new features that can further enhance the user experience. These include incorporating a garden care mini-game, where users can create virtual gardens and perform interactive tasks related to the tree species collected in the application. This feature could be both entertaining and educational, bringing users closer to nature knowledge. Additionally, I plan to expand further community features, such as the possibility of organizing leaderboards and events, which could strengthen the user community.

IRODALOMJEGYZÉK

[1] W.-S. Jeon és S.-Y. Rhee, „Plant leaf recognition using a convolution neural network”, *International Journal of Fuzzy Logic and Intelligent Systems*, 17. évf., 1. sz., 26–34. old., 2024. ápr. 13.

[2] Appixi. „LeafSnap Plant Identification”. (2024. ápr. 12.), cím: `https://play.google.com/store/apps/details?id=plant.identification.snap` (elérés dátuma 2024. 04. 12.).

[3] Appixi. „LeafSnap FAQ”. (2024-04-12), cím: `https://leafsnap.app/q&a.html` (elérés dátuma 2024. 04. 12.).

[4] Plant.id. „FAQ for Plant.id API”. (2024-04-12), cím: `https://web.plant.id/faq/` (elérés dátuma 2024. 04. 12.).

[5] Plant.id. „Plant.id API”. (2024-04-12), cím: `https://web.plant.id/plant-identification-api/` (elérés dátuma 2024. 04. 12.).

[6] K.-B. Lee és K.-S. Hong, „An implementation of leaf recognition system using leaf vein and shape”, *International Journal of Bio-Science and Bio-Technology*, 5. évf., 2. sz., 57–66. old., 2024. ápr. 13.

[7] S. Albawi, T. A. Mohammed és S. Al-Zawi, „Understanding of a convolutional neural network”, *2017 international conference on engineering and technology (ICET)*, Ieee, 2017, 1–6. old.

[8] N. Adaloglou. „Best deep CNN architectures and their principles: from AlexNet to EfficientNet”. (2024-04-12), cím: `https://theaisummer.com/cnn-architectures/` (elérés dátuma 2024. 04. 12.).

[9] S. Das. „CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more...” (2024-04-12), cím: `https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5` (elérés dátuma 2024. 04. 12.).

[10] J. A. Hartigan, M. A. Wong és tsai., „A k-means clustering algorithm”, *Applied statistics*, 28. évf., 1. sz., 100–108. old., 2024. nov. 13.

[11] D. A. Reynolds és tsai., „Gaussian mixture models.”, *Encyclopedia of biometrics*, 741. évf., 659-663. sz., 2024. nov. 13.

[12] K. G. Derpanis, „Mean shift clustering”, *Lecture Notes*, 32. évf., 1-4. sz., 16. old., 2024. nov. 13.

[13] K. Simonyan és A. Zisserman, „Very deep convolutional networks for large-scale image recognition”, *arXiv preprint arXiv:1409.1556*, 2024. ápr. 17.

[14] S. V. Lab. „About ImageNet”. (2024. ápr. 17.), cím: https://www.image-net.org/about.php (elérés dátuma 2024. 04. 17.).

[15] LearnOpenCV. „Image Classification using Pre-Trained ImageNet Models in TensorFlow & Keras”. (2024. ápr. 17.), cím: https://learnopencv.com/image-classification-pretrained-imagenet-models-tensorflow-keras/ (elérés dátuma 2024. 04. 17.).

[16] M. U. of Science és S. M. Technology. „Plant Identification on a combined-imbalanced leaf dataset”. (2024-04-14), cím: https://scholarsmine.mst.edu/picild_data/index.html (elérés dátuma 2024. 04. 14.).

[17] Matlab. „VGG-16”. (2024-04-14), cím: https://www.mathworks.com/help/deeplearning/ref/vgg16.html (elérés dátuma 2024. 04. 14.).

[18] Flickr. „The Flickr Developer Guide”. (2024. nov. 14.), cím: https://www.flickr.com/services/developer/ (elérés dátuma 2024. 11. 14.).

[19] Microsoft. „What is ASP.Net Core?” (2024-04-14), cím: https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet-core (elérés dátuma 2024. 04. 14.).

[20] P. S. Foundation. „Welcome to Python”. (2024. ápr. 17.), cím: https://www.python.org/ (elérés dátuma 2024. 04. 17.).

[21] Tensorflow. „Tensorflow”. (2024. ápr. 17.), cím: https://www.tensorflow.org/ (elérés dátuma 2024. 04. 17.).

[22] keras. „Keras”. (2024. ápr. 17.), cím: https://keras.io/ (elérés dátuma 2024. 04. 17.).

[23] OpenCV. „OpenCV - Introduction”. (2024. nov. 16.), cím: https://docs.opencv.org/4.x/d1/dfb/intro.html (elérés dátuma 2024. 11. 16.).

[24] N. Developers. „NumPy documentation”. (2024. nov. 16.), cím: https://numpy.org/doc/stable/ (elérés dátuma 2024. 11. 16.).

[25] scikit-learn developers. „User Guide - Clustering”. (2024. nov. 16.), cím: https://scikit-learn.org/stable/modules/clustering.html (elérés dátuma 2024. 11. 16.).

[26] Pallets. „Welcome to Flask”. (2024. ápr. 17.), cím: https://flask.palletsprojects.com/en/3.0.x/ (elérés dátuma 2024. 04. 17.).

[27] M. Corporation. „POST - HTTP | MDN”. (2024. ápr. 17.), cím: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST (elérés dátuma 2024. 04. 17.).

[28] Microsoft. „IdentityUser Class”. (2024. ápr. 17.), cím: https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.identity.entityframeworkcore.identityuser?view=aspnetcore-1.1 (elérés dátuma 2024. 04. 17.).

ÁBRAJEGYZÉK