DUNAÚJVÁROSI EGYETEM

MÉRNÖKINFORMATIKUS BSC

SZAKDOLGOZAT

GÉPI TANULÁS-VEZÉRELT KÉPFELISMERŐ IOS ALKALMAZÁS FEJLESZTÉSE

Szabó Benedek Áron

mérnökinformatikus jelölt

A-002-INF-2022

Dunaújvárosi Egyetem
Informatikai Intézet

Sorszám: A-002-INF-2022

TÉMAKIÍRÁS

Szabó Benedek Áron (Neptun kód: CU0FGG)
Mérnökinformatikus BSc
hallgatója részére

A szakdolgozat címe:
Gépi tanulás-vezérelt képfelismerő iOS alkalmazás fejlesztése

A szakdolgozat fő fejezetei:

1. Forrásgyűjtés, gépi tanulás szoftveres képfelismerés során betöltött szerepének megértése

2. Fejlesztői környezet összeállítása, rendelkezésre álló szoftveres segédkönyvtárak felkutatása

3. Képfelismerő mintaalkalmazás fejlesztése iOS platformra

4. Az elkészült alkalmazás tesztelése, a teszteredmények kiértékelése

5. Az implementáció során felmerült nehézségek feltárása, továbbfejlesztési lehetőségek részletezése

Belső konzulens (neve): Dr. Katona József

Külső konzulens (neve, aláírása): Szabó Sándor _____

A szakdolgozat beadásának időpontja: 2022. május 9.

.....
Dr. Nagy Bálint, Intézetigazgató

Dunaújvárosi Egyetem
Informatikai Intézet

Sorszám: A-002-INF-2022

SZAKDOLGOZAT - KONZULTÁCIÓS LAP

Szabó Benedek Áron (Neptun kód: CU0FGG)
Mérnökinformatikus BSc
hallgatója részére

A szakdolgozat címe:
Gépi tanulás-vezérelt képfelismerő iOS alkalmazás fejlesztése

1. Konzultációs időpontok:

Belső konzulens	
dátum	aláírás

Külső konzulens	
dátum	aláírás

2. A szakdolgozat beadható:

Belső konzulens

igen nem

_____ év _____ hó ____ nap

.....

belső konzulens aláírása

Külső konzulens

igen nem

_____ év _____ hó ____ nap

.....

külső konzulens aláírása

Dunaújvárosi Egyetem
Informatikai Intézet

Sorszám: A-002-INF-2022

SZAKDOLGOZAT - BÍRÁLATI LAP

Szabó Benedek Áron (Neptun kód: CU0FGG)
Mérnökinformatikus BSc
hallgatója részére

A szakdolgozat címe:
Gépi tanulás-vezérelt képfelismerő iOS alkalmazás fejlesztése

1. A szakdolgozat bírálatra bocsátható:

igen nem

A külső bíráló neve: _____

_____év ____ hónap ____ nap

.....
Intézetigazgató

Szakdolgozatát sikeresen megvédte.

Dunaújváros, _____év _____ hó ____ nap

.....
ZVB elnöke

Dunaújvárosi Egyetem
Informatikai Intézet

Sorszám: A-002-INF-2022

NYILATKOZAT A SZAKDOLGOZATRÓL

Alulírott Szabó Benedek Áron (Neptun kód: CU0FGG) Mérnökinformatikus BSc hallgatója

kijelentem, hogy

Gépi tanulás-vezérelt képfelismerő iOS alkalmazás fejlesztése című szakdolgozatomat (nyomtatott és elektronikus formában) a Dunaújvárosi Egyetem oktatói és hallgatói, szükség esetén más érdeklődők:

O felhasználhatják (pl. hivatkozás alapjául, olvasótermi használatra) későbbi munkájukhoz a szerzői jogok tiszteletben tartása mellett).

O nem használhatják fel (titoktartási nyilatkozat csatolása mellett).

Ugyanakkor kijelentem, hogy a szakdolgozat saját kutató munkám eredménye.

Dunaújváros, _____év _____ hó ____ nap

.....
aláírás

Kivonat

Gépi tanulás-vezérelt képfelismerő iOS alkalmazás fejlesztése

Szabó Benedek Áron

Szakdolgozatom célkitűzése egy olyan gépi tanulás-alapú iOS mobilalkalmazás implementálása, illetve az ennek alapjául szolgáló bináris klasszifikációs modell betanítása, amely alkalmas a bemenetén megjelenő bőrelváltozások képeinek osztályozására aszerint, hogy az azokon látható elváltozás rosszindulatú-e vagy sem.

A modell minőségét, így a klasszifikáció várható pontosságát a mintaadatok begyűjtése és felcímkézése, majd az elkészített mintahalmaz adott problémakör kontextusában meghatározott adatpontjai szerint történő kiválogatása és technikai előkészítése útján biztosítottam.

A modelltanításra kiválasztott Create ML segítségével egy, a célplatformra optimalizált konvolúciós modell képességeit feature extractor szerepkörben hasznosítva, valamint azt egy logisztikai regressziós modellel kaszkádosítva olyan transfer learning pipeline-t alkalmaztam, mellyel a mobilos eszközkörnyezet ismert erőforrás korlátai mellett is elfogadható pontosságú és hatékonyságú megoldást tudtam generálni. A végleges modellverzió eléréséhez számos validációs és verifikációs modellmetrikát vettem figyelembe, melyek mentén a kívánt pontosságot modell- és osztályszinten egyaránt tudtam finomítani.

Az elkészített modellt a Core ML és Vision szoftveres segédkönyvtárak programozói interfészével integráltam mintaalkalmazásomba.

Abstract

Development of a machine learning-driven image recognition application for iOS

Szabó Benedek Áron

The goal of my thesis is to implement a machine learning-based iOS application along with a corresponding binary image classifier model that is capable of classifying images of skin lesions as either benign or malignant.

I achieved the expected model quality and therefore accuracy of predictions by gathering, labeling and curating a set of sample images, while keeping in mind the data points I identified as relevant within the context of the given problem.

By making use of a platform-optimized convolutional model as a feature extractor and cascading it with a logistic regression model within a transfer learning pipeline, I have managed to take advantage of Create ML features to generate a model accurate and efficient enough even within the well-known resource limitations of a mobile device environment. To reach the final, refined model version I considered numerous validation and verification metrics on class- and model levels alike.

I integrated the machine learning model's features into the sample application through the programming interfaces of the Core ML and Vision frameworks.

Tartalomjegyzék

1. Bevezetés 1

1.1. A dolgozat célkitűzései 2

1.2. A dolgozat felépítése 3

2. Gépi tanulás és szoftveres képfelismerés 4

2.1. Elméleti alapfogalmak 5

2.1.1. Mesterséges intelligencia és gépi tanulás kapcsolata 5

2.1.2. Algoritmusok és modellek gépi tanulásban 5

2.1.3. Felügyelt tanulás, klasszifikáció 7

2.2. Képfelismerés induktív tanulással 8

2.2.1. Deep learning 8

2.2.2. A tanulási folyamat ciklusa 9

2.2.3. Bemeneti adatok, megtanult paraméterek 10

2.2.4. Mintaadatok előkészítése 12

2.2.5. Kiugró értékek kezelése 13

2.2.6. Kiegyensúlyozottság és elfogultság 13

3. A probléma elemzése, mintaadatok gyűjtése 15

3.1. Feladatspecifikáció 15

3.2. Problémaelemzés 16

3.2.1. Rákos és nem rákos eredetű bőrelváltozások 16

3.2.2. Rosszindulatú bőrelváltozásokra jellemző vizuális jegyek 17

3.2.3. Diagnosztikus modell betanítása 18

3.3. Data curation, mintaadatok halmazának előkészítése 19

3.3.1. Diagnózisra vonatkozó adatpontok 19

3.3.2. Képalkotás módjára vonatkozó adatpontok 20

3.3.3. Mintagyűjtés 21

3.3.4. Az adathalmaz finomítása 22

4. Modellgenerálás 24
4.1. Fejlesztői környezet összeállítása 24
4.2. Előtanított modellek keresése 25
4.3. Egyedi modell betanítása 26
4.3.1. A céleszköz kiválasztásának szempontjai 26
4.3.2. Optimalizálás Transfer Learning-el 27
4.4. Create ML a gyakorlatban 28
4.4.1. Fájlrendszer szintű címkézés 29
4.4.2. Create ML projekt létrehozása 30
4.5. A tanítás technikai lépései 30
4.5.1. Feature Extraction 31
4.5.2. Logisztikus regresszió 31
4.6. Modelltesztelés, a Create ML verifikációs metrikái 33
4.7. A folyamat szemléltetése 35
4.7.1. Kiindulási konfiguráció 35
4.7.2. Modelljavítás az iterációk számának növelésével 36
4.7.3. Overfitting 37
4.7.4. Dedikált validációs halmaz bevezetése 38
4.7.5. Augmentációk alkalmazása 40
4.7.6. Az eredmény tesztelése 43
4.8. Kiértékelés 45
5. A mintaalkalmazás implementációja 46
5.1. Projektstruktúra és szoftveres architektúra kialakítása 46
5.2. A modell szoftveres integrálása 49
5.2.1. A modell be- és kimeneti interfésze 49
5.2.2. A Core ML segédkönyvtár használata 50
5.3. Vision framework adaptálása 55

5.3.1. VNCoreMLRequest konfigurálása 55
5.3.2. VNImageRequestHandler-ek használata 58
5.3.3. A refaktorálás eredménye 59
6. Az alkalmazás tesztelése 60
7. Továbbfejlesztési lehetőségek 65
7.1. Konvolúciós modellek feature extractor szerepben 65
7.2. Object detection 66
8. Összegzés 68
Irodalomjegyzék 69
Ábrajegyzék 74
Táblázatjegyzék 75

1. Bevezetés

A mesterséges intelligencia, azon belül is a gépi tanulás-alapú algoritmusok és matematikai modellek nem új keletűek. Az emberi agyban található idegsejtek interakciói által inspirált első modellek már az 1940-es évek végén megjelentek, majd az egyre kiterjedtebb kutatási és fejlesztési eredményeknek köszönhetően a gépi tanulás az 1970-es évekre a mesterséges intelligencia külön ágazataként vált le és fejlődött tovább [4].

Mára az ún. neurális hálókon alapuló algoritmusok, és az ezek alapján fejlesztett szoftveres segédkönyvtárak számos üzleti és kutatási probléma megoldásának alapjául szolgálnak [4]. Ilyen, többek között, a szoftveres képfelismerés, mely a rendelkezésre álló hardveres erőforrásoknak köszönhetően immáron mobilos környezetben is elérhetővé vált a szoftverfejlesztők számára.

Az elmúlt években az Apple számos olyan hardveres megoldással állt elő telefonjaikban, melyek kifejezetten neurális hálók számítási kapacitásainak optimális kielégítését biztosítják. Mindezt persze alacsony fogyasztás fenntartása mellett, mely kifejezetten fontos hordozható készülékek akkumulátor idejének megőrzése szempontjából. Ilyen pl. az iPhone-ok CPU chipjébe integrált Neural Engine. [45]

A hardveres fejlesztések képességeinek kihasználásához olyan natív, szoftveres segédkönyvtárak biztosítására van szükség, melyek megkönnyítik és felgyorsítják a gépi tanulás-alapú funkcionalitások integrálását az adott operációs rendszeren futó alkalmazásokba. Ilyen pl. a Core ML, melynek segítségével gépi tanulás-vezérelt modelleket tudunk felhasználni iOS alkalmazásainkban [7]. Az Apple számos olyan eszközt is biztosít számunkra, amelyekkel Core ML-kompatibilis modelleket tudunk készíteni. Ilyen pl. a Create ML vagy a Turi Create [7].

A gépi tanulás elméleti alapjainak megismerése után, megfelelő koncepció összeállítása mellett, már tudunk olyan iOS alkalmazást fejleszteni, mely alkalmas a szoftveres képfelismerés egy adott formájára. Ezen folyamat része a rendelkezésre álló eszközök és segédkönyvtárak felkutatása és használatának elsajátítása, az erre alapuló fejlesztői környezet kialakítása, majd egy egyszerűbb alkalmazás elkészítése. Az alkalmazás tesztelése, valamint a teszteredmények kiértékelése fontos jelentőséggel bír annak elbírálása szempontjából, hogy mennyire effektív megoldást fejlesztettünk adott problémára, és rámutathat további iterációs és továbbfejlesztési irányokra is.

A dolgozat célkitűzése egy olyan gépi tanulás-alapú iOS mobilalkalmazás implementálása, illetve az ennek alapjául szolgáló bináris klasszifikációs modell betanítása, mely alkalmas a bemenetén megjelenő bőrelváltozások képeinek osztályozására aszerint, hogy az adott elváltozás rosszindulatú-e vagy sem.

A gépi tanulás-vezérelt klasszifikációs modellek felügyelt betanításához szükséges alapelvek ismertetése mellett a dolgozat kitér a megoldandó probléma, azaz a bőr malignáns elváltozásainak azonosításához szükséges vizuális adatpontok részletezésére is. Rámutat a gépi tanulás-alapú képosztályozás bőrelváltozások diagnózisában betöltött szerepére, valamint a kettő kapcsolatára az osztályozási modell betanításának kontextusában. Hangsúlyozza a tanításra használt mintaképek összeállításának jelentőségét a betanított gépi modell predikciós pontossága szempontjából, és betekintést nyújt olyan módszerekbe, melyekkel javítani lehet a mintaadatok halmazának minőségét a felhasznált képek begyűjtése és kiválogatása során.

A dolgozatban megismerhetünk olyan, az Apple által biztosított fejlesztőeszközöket, mint pl. a Create ML alkalmazás és framework, melynek segítségével gépi modellünket betaníthatjuk, vagy a Core ML és Vision segédkönyvtárakat, melyek lehetővé teszik az elkészített modell alkalmazásunkba való integrálását Swift nyelv segítségével. Betekintést kapunk olyan metrikákba, melyek mentén az elkészített modellünket és alkalmazásunkat validálhatjuk, tesztelhetjük, és szükség esetén iteratívan tovább finomíthatjuk. Egyúttal a dolgozat demonstrálja a felhasznált módszerek és eszközök limitációit, és szemléltet pár továbbfejlesztési lehetőséget az alkalmazáshoz.

1.1. A dolgozat célkitűzései

A szakdolgozat célkitűzései 5 fő pontba sorolhatók:

- Forrásgyűjtés, gépi tanulás szoftveres képfelismerés során betöltött szerepének megértése
- Fejlesztői környezet összeállítása, rendelkezésre álló szoftveres segédkönyvtárak felkutatása
- Képfelismerő mintaalkalmazás fejlesztése iOS platformra
- Az elkészült alkalmazás tesztelése, a teszteredmények kiértékelése
- Az implementáció során felmerült nehézségek feltárása, továbbfejlesztési lehetőségek részletezése

1.2. A dolgozat felépítése

1. Az első részben, a 2. fejezetben egy klasszifikációs gépi tanulási modell felügyelt betanításához szükséges alapokat ismertetem.
2. A 3. fejezetben részletesen foglalkozok a célként kitűzött mintaalkalmazás feladatspecifikációjával, a probléma elemzésével és a szükséges mintaadatok halmazának előkészítésével.
3. A 4. fejezet a bináris klasszifikációs modell betanítását részletezi: a folyamat során felhasznált céleszköz kiválasztásáról, gyakorlati használatáról, a végeredményhez vezető technikai lépések és logikai megfontolások sorozatáról, végül pedig a produktum verifikációjáról szól.
4. Az 5. fejezet a célkitűzés szerinti mintaalkalmazás implementációját taglalja a kiválasztott szoftveres architektúra, valamint a Core ML és Vision segédkönyvtárak használatának részletezése mellett.
5. A 6. fejezetben az elkészült alkalmazás manuális tesztelésével, illetve a tesztesetek kiértékelésével foglalkozok.
6. Az utolsó részben, a 7. és 8. fejezetben az elkészült applikáció továbbfejlesztési lehetőségeit részletezem, majd összegzem a dolgozat elkészítése során elért eredményeket és megszerzett tapasztalatokat.

2. Gépi tanulás és szoftveres képfelismerés

Programozóként szoftverfejlesztés során számtalan alkalommal használunk feltételes elágazásokat. Vizsgáljuk bizonyos feltétel, avagy feltételek fennállását, majd vezérlőutasítások segítségével elágaztatjuk a programunk egyébként szekvenciális futását a megfelelő utasítások, vagy éppen szubrutinok meghívásával.

Számos repetitív feladat megoldását tudjuk így automatizálni, amennyiben megfelelő információval rendelkezünk a megoldandó problémáról, illetve annak kontextusáról [1-3]. A rendelkezésre álló számítási kapacitásnak köszönhetően programunk effektíven kitermelheti az elvárt produktumot olyan esetekben, ami egyébként az emberek számára időigényes és fárasztó folyamat lenne, ráadásul megfelelő tesztesetek kidolgozásával csökkenthetjük a hibák lehetőségét, ezzel növelve az elvárt eredmények megbízhatóságát.

Minél komplexebb problémakörrel foglalkozunk, annál bonyolultabb megoldási stratégiákat kell kidolgoznunk, ha a hagyományos vezérlési szerkezetekre akarunk támaszkodni. Bizonyos esetekben a logikai érvelésekre alapozott szabályrendszerünk, illetve az ez alapján kidolgozott heurisztikák már csak nagyon nehezen, vagy éppen egyáltalán nem lesznek képesek a kívánt cél hatékony elérésére [1], [3].

Az emberi agy számos ilyen, egyébként összetett feladatot képes megoldani inherens módon. Képes tárgyak felismerésére, azok beazonosítására, vagy éppen megkülönböztetésére, illetve ezek alapján tudatos döntések meghozatalára. A képfelismerés problémakörére nehéz hagyományos döntési heurisztikák mentén megoldásokat találni. Akármilyen nagy szabályrendszer alapján is akarnánk számítógépünket explicit módon programozni, esetenként klasszikus logikai struktúrákba nem szervezhető, vagy a teljesség tekintetében bizonytalan helyzetekkel szembesülünk. Kezelhetetlenül nagy számosságú elágazás és kivétel esetén a logikai elágazás-alapú heurisztikáink egyre kevésbé jelentenek hatékony megközelítést [1], [3].

A gépi tanulás ezekben az esetekben kerül előtérbe. A mögötte rejlő alapgondolatot a következőképpen tudjuk leegyszerűsítve megfogalmazni: ha nem, vagy csak nagyon nehezen tudjuk az egzakt lépéseket megfogalmazni egy probléma megoldásához, pl. egy tárgy felismerésére adott képen, akkor készítsünk helyette egy olyan programot, mely előállítja az elvárt algoritmust számunkra.

A gépi tanulás mögött rejlő tanulási algoritmusok tehát automatikusan előállítják az adott problémakör megoldásához szükséges szabályrendszert [1-3], [5]. A számítógép példák alapján megtanulja és előállítja az eredeti felvetésben szereplő probléma megoldásához szükséges szabályokat, melyeket később így készen tudunk integrálni saját programunkba. Sokszor olyan mintákat is észrevesz a számítógép, melyek a fejlesztők számára nem, vagy csak nehezen megfigyelhetők, ezáltal megbízhatóbb eredményeket produkálva [1-3]. Az alábbi fejezetben a gépi tanulás alapjaival, illetve a képfelismerés során betöltött szerepével foglalkozok.

2.1. Elméleti alapfogalmak

Ahhoz, hogy azonosítani tudjuk a számunkra szükséges gépi tanulás-alapú segédkönyvtárakat, valamint azokat megfelelően tudjuk integrálni alkalmazásunkba, fontos megértenünk a gépi tanulás alapfogalmait, működési paradigmáit és használati jelentőségét.

2.1.1. Mesterséges intelligencia és gépi tanulás kapcsolata

A mesterséges intelligencia mára interdiszciplináris kutatási területté nőtte ki magát. Matematika, pszichológia, biológia és számítógépes tudományok képviselői egyaránt foglalkoznak az emberi agy viselkedését szimuláló modellek és szoftveres megoldások kialakításán. [4]

A mesterséges intelligencia célja, hogy gépek felhasználásával az emberi intelligencia bizonyos aspektusait modellezze és szimulálja [1], [3]. Bár a mesterséges intelligencia és gépi tanulás nagyon szorosan kapcsolódnak egymáshoz, nem minden mesterséges intelligencia épül gépi tanulásra, és nem minden gépi tanulás nevezhető mesterséges intelligenciának. Míg a tanulás folyamatát érthetően asszociálhatjuk bizonyos szintű intelligenciával, a gépi tanulás-alapú rendszerek nem feltétlenül nevezhetőek intelligensnek. A gépi tanulás inkább a mesterséges intelligencia területén használt számos heurisztika és eszköz egyike [1-3].

2.1.2. Algoritmusok és modellek gépi tanulásban

A gépi tanulás centrális alapfogalmai közé tartoznak a gépi tanulási algoritmusok, valamint a gépi tanulási modellek, így fontos megértenünk a kettő közötti különbséget, illetve ezek egymáshoz való viszonyát.

A gépi tanulás során tanulási algoritmusnak, vagy egyszerűen algoritmusnak nevezünk minden olyan mintafelismerő számítógépes eljárást, amely előre definiált bemeneti adathalmazon futtatva, adott gépi tanulási modellt állít elő. Ezekre az algoritmusokra nyugodtan gondolhatunk úgy, mint minden más, a számítástudomány területén előforduló algoritmusra. A döntési fák, lineáris regresszió, logisztikai regresszió, mesterséges neurális hálók mind ebbe a csoportba sorolhatók [1]. Mindegyikben közös, hogy bizonyos bemeneti adatok alapján képesek olyan információk extrapolálására az adathalmazban felismert minták és összefüggések azonosítása révén, melyek alapján ún. prediktív modelleket tudnak előállítani korábban még nem látott adatokkal kapcsolatos előrejelzések készítéséhez [1], [6].

Ezzel szemben a korábban említett gépi tanulási modell, vagy más szóhasználattal prediktív modell, az adott bemeneti adathalmazon futtatott, adott tanulási algoritmus kimenetén jelenik meg. Ez azokat az algoritmus-specifikus adatszerkezeteket, szabályokat és metrikákat foglalja össze, melyeket az algoritmus a futása során állított elő [6]. A modell azt reprezentálja, hogy mit tanult meg a futtatott algoritmus a tanulásra előkészített adathalmazból.

Ha úgy tetszik, gondolhatunk a modellre úgy is, mint egy önálló „program”, mely adatokból, illetve az azokat paraméterként felhasználó predikciós procedúrákból áll. A modell a bemenetén megjelenő adatokat felparametrizálja belső adatszerkezeteivel, majd belső függvényeinek segítségével következtetéseket állít elő a bemeneti adathalmazról.

A modelleket minden esetben tanítani kell. Miután kiválasztottuk a modellünk alapjául szolgáló tanulási algoritmust, annak számos iteráció során példákat kell mutatnunk a megoldani kívánt problémáról. Ezt tanító adathalmaznak is szokták nevezni. [1-3], [6]

A cél az, hogy megfelelő előkészítés eredményeként, a tanítás végére előálló modellünk segítségével következtetéseket vonhassunk le adott adathalmazzal kapcsolatban, úgynevezett predikciókat generálhassunk. A predikció tehát a bemeneti adathalmazról, adott problémakör kontextusában generált megállapítások halmaza [1]. Ilyen például a szoftveres képfelismerés során az adott képen látható tárgyak detektálása, osztályozása stb. Egy modellt megfelelően generizáltnak tekinthetünk, amennyiben olyan adathalmazokra is jó predikciókat generál, amivel korábban nem volt tanítva [1].

Mint azt a mintaalkalmazás implementálása során látni fogjuk, gépi tanulás esetében, a modell betanítási fázisa után a kész modellünk már nem fog új dolgokat tanulni, csupán a meglévő, belső információi alapján dedukciókat levonni. Valójában, ha egy gépi tanulási modellt az alkalmazásunkba integrálunk valamilyen szoftveres megoldás keretében, akkor az már nem fog új dolgokat tanulni.

Természetesen modellünket a háttérben tovább generizálhatjuk a bemeneti adathalmaz finomításával és újabb tanítási ciklusok futtatásával. Ebben az esetben biztosítani kell, hogy az újonnan előállított iteratív modellt referáljuk alkalmazásunk egy újabb verziójában. A gépi tanulásra épülő megoldások egyik nagy kihívása tehát olyan modell előállítása, amely a célnak megfelelően új adatok esetén is jó közelítésekkel él, illetve a szoftveres integrációja biztosítja annak lecserélését a kódbázis jelentősebb módosítása nélkül.

2.1.3. Felügyelt tanulás, klasszifikáció

Amennyiben a gépi modell betanítási folyamata külső aktor által felügyelt környezetben zajlik, úgy azt felügyelt tanulásnak nevezzük. Az aktor, pl. szoftverfejlesztő, utasításokat és információkat biztosít az algoritmusnak azzal kapcsolatban, hogy mit és hogyan tanuljon meg. [1-3], [5]

Felügyelt, vagy más néven induktív tanítást végezni csak az ehhez szükséges, predefiniált tanítási adathalmaz generálásával lehet, melyben az adatok ún. címkékkel vannak ellátva. A labeling, vagy magyarul címkézés folyamata a tanuláshoz használt adatminták további metainformációkkal való összefűzése [2-3]. Az algoritmus számára így együttesen definiálható a feldolgozandó adat, valamint annak adott kontextusban értelmezett szerepe, melyet a hozzárendelt címkék reprezentálnak. Amennyiben a bemeneti adatot α reprezentálja, $f(\alpha)$ pedig a címkézés alapján elvárt kimeneti minta, úgy az induktív tanulás lényege a kettőt összekötő f transzformációs függvény kellően generizált változatának előállítása.

A felügyelt tanulás speciális esete a klasszifikáció, vagy osztályozás. Ennek lényege, hogy előre meghatározott osztályokat készítünk, és a bemeneti adathalmaz elemeit felcímkézzük aszerint, hogy elvárhatóan melyik osztályba tartoznak. N-áris klasszifikáció során a bemeneti adatokat N diszkrét csoportba sorolhatjuk. A szakirodalmak előszeretettel használják a multiklassz- és multinominális klasszifikáció

elnevezéseket is. Ennek egyik alesete a bináris klasszifikáció, ahol két diszkrét halmazzal dolgozunk. [1-3]

A klasszifikációs prediktív modellek tehát diszkrét kategóriákon alapuló válaszokat generálnak. Ennek tipikus példája a betegségek diagnózisa, ahol a már korábban bevezetett jelöléseket használva:

- α a betegek bizonyos tünetegyüttese
- f(α) a diagnosztizált állapotuk,
- f pedig az induktív tanulás eredményeképp előállított diagnosztikus modell.

Mint azt a mintaalkalmazás elkészítése során is látni fogjuk, az induktív tanítás egyik legnagyobb hátránya, hogy sok időbe telik a megfelelő tanító adathalmaz előállítása. Nem csupán a minta adatokat kell kellő gondossággal összegyűjtenünk, hanem az azokhoz tartozó elvárt címkéket is, ráadásul úgy, hogy azok a felhasznált szoftverfejlesztői eszközök számára is értelmezhető adatstruktúraként legyenek reprezentálva.

A teljesség igénye miatt érdemes röviden megemlíteni a felügyeletlen tanulás koncepcióját is. Ilyen az ún. clustering, ahol az algoritmus címkézetlen bemeneti adatok halmazában prognosztizál összefüggéseket és mintákat [1], [3]. A dolgozat további részeiben kizárólag a felügyelt tanuláson alapuló gépi tanulással foglalkozok.

2.2. Képfelismerés induktív tanulással

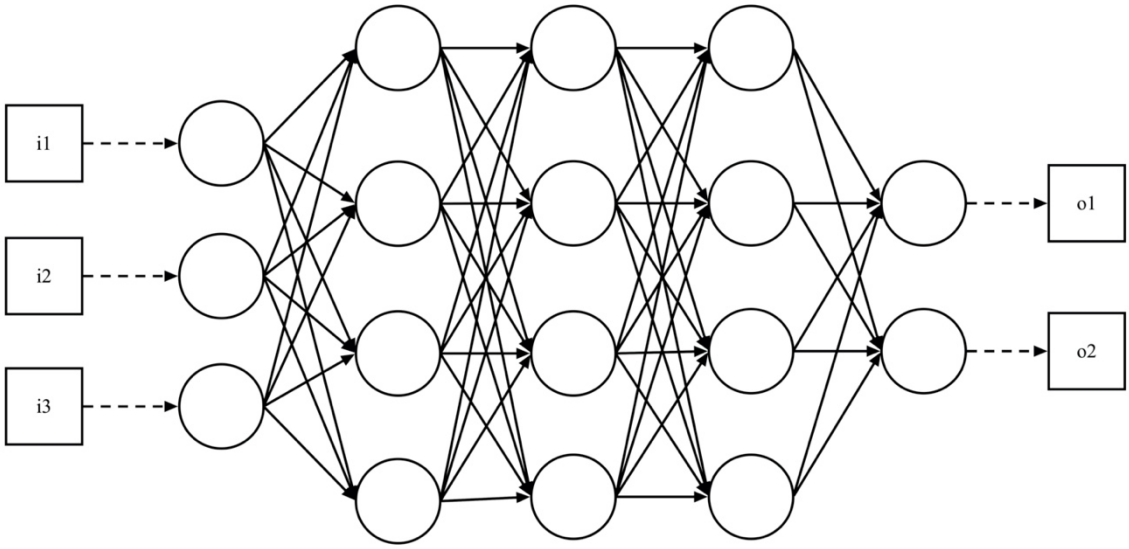
Az alapfogalmak megismerése után rátérhetünk a megoldandó probléma elemzésére, mely a választott mintaprojekt alapján induktív tanulás-alapú, klasszifikációs képfelismerés implementálása mobilos környezetben.

2.2.1. Deep learning

Induktív modellünk alapját egy általunk választott tanulási algoritmus fogja adni, pl. egy mesterséges neurális háló. A felhasználás szempontjából nem kell ismernünk pontosan a neurális háló implementációját, de fontos tisztában lenni a mögötte lévő alapvető koncepciókkal.

Röviden, a neurális hálók az emberi agy működését szimulálják az agyban is található idegsejtek, azaz neuronok közötti interakciók matematikai modellezésével. Ezen csomópontok, vagy mesterséges neuronok, több rétegbe rendezhetők. [1-3]

Az ún. deep learning során olyan neurális hálókat alkalmazunk, melyek több rétegbe szerveződnek, nagyszámú neuronnal rétegenként [1], [3]. Maga a deep learning elnevezés onnan származik, hogy minél több rétegű, azaz minél mélyebb a neurális háló struktúrája, annál komplexebb feladatok megoldását képes megtanulni. Ilyen tipikus probléma pl. a szoftveres képfelismerés, ahol előszeretettel alkalmaznak deep learning-alapú megoldásokat [3]. Az 1. ábrán egy többrétegű mesterséges neurális háló szemantikus vázlatát láthatjuk.



1. ábra: Mesterséges neurális háló szemantikus vázlata (saját szerkesztésű ábra)

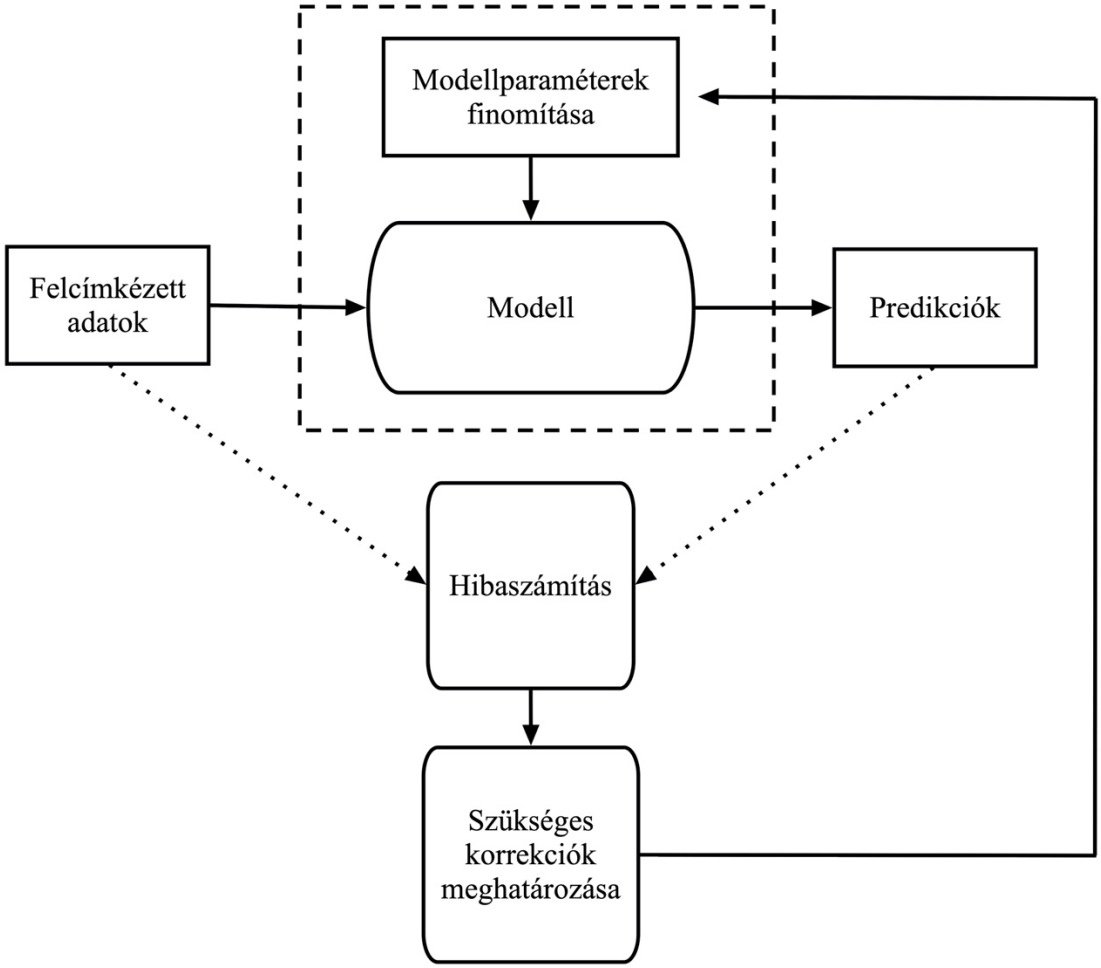
2.2.2. A tanulási folyamat ciklusa

Az induktív, klasszifikációs modellünk betanításához első körben elő kell állítanunk a felismerendő képekből álló bemeneti adathalmazt, és elkészíteni a mintához tartozó címkéket. A tanulás folyamata számos iteráción keresztül zajlik, és a bemeneti adathalmaz méretétől, az iterációk számától, valamint számítógépünk hardveres erőforrásaitól függően huzamosabb ideig is eltarthat.

A modell a bemenetén megkapja a minta adatokat, azaz a képeket, és a hozzájuk tartozó címkéket. Minden egyes adatra generál egy predikciót. Ezt összevetve a címkék alapján elvárt kimenettel felméri és jegyzi a különbségből származó hibát. A hibafaktor segítségével minden iteráció végén javít belső paraméterein, így az újra parametrizált modell a következő iterációban már pontosabb eredményekkel tud szolgálni. [1-3]

Ebből következően nem elég egyszer megmutatni a bemeneti mintákat a modellünknek. Mivel minden egyes iterációval pontosabb és pontosabb eredményekhez

juthatunk, könnyen kirajzolódik előttünk a tanulás repetitív ciklusa: a modell megkapja bemenetén a minta halmazt, predikciókat generál, összevetve azt a referencia értékekkel újra parametrizálja magát, majd újabb tanulási ciklusba kezd mindaddig, amíg el nem éri az általunk beállított maximális iterációk számát. A 2. ábra a tanulási folyamat ciklusát szemlélteti.



2. ábra: A modelltanítási folyamat ciklusa (saját szerkesztésű ábra)

Nehéz megmondani, hogy hány iterációra van szükség ahhoz, hogy viszonylag pontos induktív modellhez juthassunk. A mintaalkalmazáshoz elkészített modell betanítása esetén elsősorban a tanulási algoritmus futtatásához szükséges gép erőforrásai szabtak korlátot, amit figyelembe kell venni az eredmények kiértékelése során.

2.2.3. Bemeneti adatok, megtanult paraméterek

Miután megértettük a tanulási ciklus alapjait, fontos azzal is foglalkoznunk, hogy a bemeneti adatok, valamint a tanulás eredményeként előálló modell paraméterek miként

értelmezendők az általunk kitűzött feladat kontextusában. Mivel a célunk egy olyan modell betanítása, mely képes a bemenetén megjelenő képek klasszifikációjára, fontos specifikálnunk, hogy a tanulás során milyen adatpontok és jellegzetességek mentén kívánjuk ezen képek osztályozását elérni.

A mintaadatok képek formájában jelennek meg a tanulási algoritmus bemenetén, mely a számítógép számára bináris formában tárolt, a képet felépítő pixelek mátrixaként értelmezhető. A pixelek RGB értéke, illetve a mátrixban betöltött pozíciója olyan alacsony szintű információk, amelyekkel nem célunk foglalkozni. Elvárjuk, hogy a kiválasztott tanulási algoritmus ezt helyettünk elvégezze, mi pedig a kitűzött probléma szempontjából fontos adatpontokra koncentrálhassunk.

Első körben tehát definiálnunk kell a probléma szempontjából meghatározó diszkrét halmazokat, illetve azon adatpontokat, mely alapján egy képről el kívánjuk dönteni, hogy pontosan melyik halmazba is tartozzon. A már korábban is említett labeling, azaz címkézés pont erre szolgál. [1-3]

Mint később látni fogjuk, a kiválasztott fejlesztői eszköztől függően ennek számos reprezentációja létezik. Bemeneti adathalmazunkat előkészíthetjük pl. úgy, hogy a mintaképeket egyedi mappákba helyezzük el manuálisan, vagy egy erre megírt script segítségével. Minden mappa adott osztályt reprezentál, illetve minden mappán belüli kép egy, az adott osztályt reprezentáló mintaadat. Ha nem akarjuk fájlrendszer szinten reprezentálni osztályainkat, akkor a közös mappában elhelyezett képeinkhez készíthetünk pl. egy CSV fájlt, melyben minden sor egy képet reprezentál, és összeköti adott kép nevét az ahhoz tartozó osztály azonosítójával.

Ami a tanulás végtermékét illeti, alacsony szinten a modellünk a saját belső algoritmusához igazított paramétereket állít elő [1]. Ezen tanult paraméterek a tanulás során, a predefiniált adatpontok alapján előállított olyan adatszerkezetek, melyek az eredeti adatpontjainkon túlmutató, a modell által a tanulás során felfedezett összefüggéseket reprezentálnak a minta képhalmazunkról.

Érdemes kiemelni, hogy a modell paraméter és modell parametrizálás szakkifejezések magasszintű, a szoftverfejlesztés során előforduló absztrakciók. Az algoritmusok elméleti hátterét taglaló szakirodalom sokszor a modell súlyáról, és az ezt finomító átsúlyozás folyamatáról beszél, melyek a modellalgoritmusok matematikai hátterére utaló, alacsony szintű fogalmak [1], [3]. Mint ahogy az algoritmus belső

működésének pontos részleteivel sem célom foglalkozni, úgy a tanult paramétereket sem fogom közvetlenül felhasználni az implementáció során.

Valójában a számunkra fontos végtermék, azaz a betanított modell, egy egyszerű fájlként jelenik meg. Ennek speciális formátuma lehetővé teszi, hogy az általunk választott segédkönyvtárak importálják és használják modellünket az elkészített alkalmazáson belül.

2.2.4. Mintaadatok előkészítése

A gépi tanulás, illetve deep learning sokat segít nekünk a képek klasszifikációjában, viszont ettől függetlenül, mint minden módszer, ez is produkálhat félrevezető eredményeket. A tanításra használt számítógép erőforrásai korlátozottak, így nem tudunk nagy mintahalmazokkal dolgozni, és túl nagy számú iterációt sem tudunk futtatni a modellünk betanítása során.

Előállított modellünk felfogható olyan „modulként”, mely a bemeneti minták száma, minősége, a tanítási iterációk száma, valamint a probléma komplexitása függvényében úgy lett betanítva, hogy az közel elfogadható mintafelismerő funkciót lát el alkalmazásunkban. Az algoritmus az általunk előkészített és felügyelt adatpontok mentén detektál további mintákat, viszont semmilyen saját kontextussal vagy értelmezéssel nem rendelkezik adott feladattal kapcsolatban.

Az eredmény hatásossága szempontjából rendkívül meghatározó a bemeneti adathalmaz minősége. A tanulási algoritmus könnyen félrevezethető, amennyiben a mintaként kiválasztott képek nem a célnak megfelelően lettek leválogatva. Az adathalmaz alapjául szolgáló képek megfelelő kiválasztását, majd technikai előkészítését angolul data curation-nek nevezik. [2-3], [8]

Előkészítés során számos tényezőt kell figyelembe vennünk. Az algoritmus betanítása során nem csak azok a képek játszanak szerepet, melyeket a modell bemenetén mintaként biztosítunk, hanem azok is, melyeket kihagytunk, vagy éppen kifelejtettünk a mintapopuláció előkészítése során. Fontos tehát előre átgondolnunk, hogy a csoportosított, felcímkézett képeink milyen mértékben segíthetik, vagy éppen vezethetik félre algoritmusunkat.

Mivel klasszifikáció témakörében biztosítunk mintamegoldást, kiválogatáskor segítségünkre lehet a problémakör definíciója, miszerint diszkrét halmazokba való csoportosítást várunk el. Ha feltérképeztük, hogy adott diszkrét halmazok milyen

jellegzetes adatpontok szerint különülnek el, úgy ezen attribútumok mentén, manuálisan, vagy akár automatizálva is tudunk képeket gyűjteni egy populáció kialakításához.

Ha úgy látjuk, hogy a megoldandó probléma felvetésében elvárt csoportok meghatározó attribútumai között van átfedés, úgy a végeredmény tesztelése szempontjából figyelembe kell venni, hogy modellünk nagyobb valószínűséggel osztályozhatja félre a bemenetére adott, korábban még nem látott képeket [1-2]. Ilyen esetekben javíthatunk modellünk minőségén az adathalmaz méretének növelésével, valamint a tanulási iterációk számának növelésével, ezáltal is több esélyt adva modellünknek a nehezebben azonosítható disztinktív, egyedi minták felismerésére.

2.2.5. Kiugró értékek kezelése

Amennyiben szigorúan válogatunk, a csoportokra jellemző mintaképek az adott csoportra jellemző attribútumok alapján közel homogén populációt alkotnak. Bizonyos esetekben ez elvárható. A legtöbb klasszifikációs probléma szempontjából azonban nem elhanyagolhatók az ún. kiugró értékek, melyek olyan adatpontok, amik jelentős mértékben eltérnek a mintavételezés alapjául szolgáló sokaság többi eleménél megfigyelhető, azonos adatpont értékekhez képest [2-3]. A mi esetünkben ez lehet pl. egy, a többi képhez képest dominánsan megjelenő, egyedi szín bizonyos mintaképeken.

Ha a kontextusból ismert, hogy ezen kiugró értékek alapvetően jellemzőek az adott csoportra, és a természetes populációban is előfordulhatnak, úgy nem feltétlenül kell figyelmen kívül hagynunk őket. Sőt, némi kutatással érdemes lehet meghatározni, hogy ezen kiugró értékek mekkora valószínűséggel jelennek meg a természetes populációban, és ennek megfelelő arányban szerepeltetni azokat a mintánkban.

Más megközelítést igényel az, amikor a kiugró érték egy tévesen megválasztott, vagy éppen rossz minőségű minta alapján kerül a halmazba. Mivel modellünk betanításakor ez félrevezető minták azonosításához vezethet, érdemes felkutatni és eltávolítani ezen példa egyedeket a tanításra előkészített képek sokaságából. [1-3]

2.2.6. Kiegyensúlyozottság és elfogultság

Arról már beszéltem, hogy osztályozás esetén hogyan tudunk disztinktív, megkülönböztető attribútumok alapján mintahalmazokat előállítani. Legyen szó bináris, avagy multinominális klasszifikációról, törekednünk kell rá, hogy a diszkrét halmazok mintaegyedei egyenlő, vagy közel egyenlő számban legyenek jelen betanítás során. Az

így előállított adathalmazokat nevezzük kiegyensúlyozottnak, illetve magát a folyamatot kiegyensúlyozásnak [2-3].

Ettől kissé eltérő, de szintén meghatározó fogalom az elfogultság. Amennyiben egy adathalmaz bizonyos elemei nagyobb számban, vagy nagyobb súlyozással reprezentáltak, ún. elfogult adathalmazt kapunk [2-3]. Egy elfogult adathalmaz nem reprezentatív értékű modellünk felhasználási esete szempontjából, így alacsonyabb pontosságú, vagy kifejezetten hibás modellek betanításához vezethet.

Dolgozatomnak nem célja bemutatni olyan módszereket, amelyekkel elfogult adathalmazok hibaforrásait tudjuk kimutatni, ugyanis ezek azonosítása sokszor nagyon bonyolult heurisztikákat követel meg. Mintaalkalmazásunk készítésének szempontjából kiemelendő, hogy a mintát alkotó képek összegyűjtése és előkészítése szintén megtörtént, melyről egy későbbi fejezetben ismertetek részleteket.

3. A probléma elemzése, mintaadatok gyűjtése

A gépi tanulásnak manapság rendkívül sok felhasználási területe van. Mint korábban láthattuk, a modellek betanításához számos algoritmus áll rendelkezésre, melyek matematikai alapjait már hosszú évtizedek óta finomítják [1]. A számítási kapacitás a processzorok, memóriamodulok és akkumulátor technológiák olcsóbbá válásával már mobiltelefonjainkon is rendelkezésre áll ahhoz, hogy gépi-tanulás alapú megoldásokat fejleszthessünk alkalmazásainkhoz.

A megoldandó probléma kiválasztása során figyelembe kell venni az elérhető algoritmusok, fejlesztői eszközök és segédkönyvtárak mellett még egy tényezőt: milyen adatokat tudunk mobilos környezetben feldolgozni, és milyen formában. Amennyiben praktikusak akarunk lenni, figyelembe vesszük a mobiltelefonokon manapság elérhető egyedi adatforrásokat, amelyeket a telefonunkba épített kamera, mikrofon, vagy aktivitásérzékelő szenzorok szolgáltatnak.

3.1. Feladatspecifikáció

A célom egy olyan iOS platformon futó, gépi tanulás-alapú mobilalkalmazás, illetve az ennek alapjául szolgáló betanított modell előállítása, mely alkalmas a bemenetén megjelenő bőrelváltozások képeinek bináris osztályozására aszerint, hogy az adott elváltozás rosszindulatú-e vagy sem.

Az applikáció számára az elemzésre váró képet a telefonkészülék beépített kamerája, vagy a fotókönyvtára szolgáltatja. A felhasználó kiválaszthat egy, már korábban eltárolt fotót, avagy készíthet egy újat az érintett bőrfelület lefényképezésével.

Az alkalmazás az integrált gépi tanulási modell és szoftveres segédkönyvtárainak segítségével analizálja, majd osztályozza a felhasználó számára a kiválasztott képet. A bináris klasszifikációnak két kimenete lehet: az alkalmazás rosszindulatú, avagy jóindulatú bőrelváltozást tud prognosztizálni. Jelen esetben nem foglalkozunk a bőrelváltozás típusának pontos meghatározásával.

A probléma elemzése során csupán azokat az adatpontokat veszem figyelembe, mely alapján egy rosszindulatú vagy jóindulatú bőrelváltozás egymástól elhatárolható, majd ezek alapján válogatom le a tanítási és validálási célokra szánt mintaképek halmazát, és tanítom be modellemet.

3.2. Problémaelemzés

Mint láthatjuk, a megoldandó probléma összetettsége miatt fontos időt szánni arra, hogy megértsük, pontosan milyen formában jelenhetnek meg a bőrön elváltozások, és mi alapján mondhatjuk egy adott bőrelváltozásra, hogy jó eséllyel rosszindulatú. A bizonyos bőrelváltozásokra jellemző, disztinktív vizuális jegyek fogják azon adatpontjainkat alkotni, amelyek alapján a tanításra használt mintaképeket összegyűjthetjük, felcímkézhetjük, majd ezután kiválogathatjuk a kívánt cél eléréséhez szükséges minőségű mintahalmaz előállításához.

3.2.1. Rákos és nem rákos eredetű bőrelváltozások

A bőrrák a bőr olyan megbetegedése, mely során malignáns, azaz rosszindulatú sejtek alakulnak ki a bőr szöveteiben [9]. Számos formája és típusa létezik attól függően, hogy a bőr melyik rétegének milyen sejtjei érintettek a kóros képlet kialakulásában. A Basalioma, Spinalioma, vagy a Melanoma Malignum mind a bőr leggyakrabban előforduló rákos elváltozásai közé tartoznak [9].

Az orvosi diagnosztika során számos eszköz áll rendelkezésre a bőr elváltozásaival kapcsolatos pontos diagnózis felállításához. A beteg kórtörténetének és panaszainak felmérése után az orvos megvizsgálja, és szükség esetén biopszia útján laboratóriumi elemzésre küldi az elemzendő bőrfelületből vett szövetmintát. A dermatológusok historikus, azaz történeti analízist is használhatnak, ami során ugyanazon beteg adott bőrfelületeiről készült képi felvételeket tudnak összehasonlítani, ezzel átlátva a tünetegyüttesek időbeli változását. Az elváltozás méretének, színének, és egyéb jellemzőinek alakulása pontos indikátora lehet annak, hogy az eredetileg jóindulatú elváltozás idővel mekkora eséllyel változik malignáns, azaz rosszindulatú képletté. [9-10]

A mintaalkalmazásunk szempontjából figyelembe vehető adatpontok ezzel szemben rendkívül limitáltak. Mivel a modell bemenetén statikus képek jelennek meg, a betegek panaszairól, illetve a tünetegyüttesek időbeliségéről nem kapunk információkat. Nem tudjuk, hogy az adott képlet együtt jár-e pl. fájdalomérzettel, vagy hogy időben miként változott a megjelenése. A modellünk kizárólag a bemenetén megjelenő kép alapján fog klasszifikációs predikciót előállítani, így fontos megismernünk a jóindulatú és rosszindulatú bőrelváltozásokra általánosságban jellemző vizuális jegyeket, függetlenül attól, hogy azt milyen típusú megbetegedés okozza.

3.2.2. Rosszindulatú bőrelváltozásokra jellemző vizuális jegyek

A bőr leggyakoribb, rákos elváltozásainak főbb vizuális jegyei:

- A Laphámrák a bőr felszínéből kiemelkedő, erősen hámló, jellemzően vöröses színű elváltozás.
- Az Aktinikus Keratózis durva felszínű, vöröses vagy rózsaszínes kis méretű folt.
- A Basalioma fényes rózsaszínes, lassan növekvő tömör duzzanat, mely idővel kifekélyesedésre és varosodásra hajlamos.
- A Melanoma Malignum aszimmetrikus, sokszor szabálytalan körvonalú, pár milliméternél nagyobb átmérőjű alakzat formájában jelenik meg. Tipikusan több színből álló bőrsejtek alkotják, mely kékes vöröses elszíneződés formájában a Melanoma környékére is kiterjedhet. [9-10]

Ezen vizuális attribútumokat sokszor szabad szemmel nem, vagy csak nagyon nehezen lehet vizsgálni. A dermatológusok által használt vizuális képalkotó eljárások közül az egyik leggyakoribb a dermatoszkópia, melynek során az orvos egy speciális képalkotó eszköz, az ún. dermatoszkóp segítségével tárja fel a bőrön megjelenő képlet jellemzőit, és határolja be annak esetleges okát [12]. Egy hagyományos dermatoszkóp 10-szeres, míg egy modern video dermatoszkóp akár 70-100-szoros nagyításra is képes, mindezt megfelelő megvilágítás biztosítása mellett [11]. A dermatoszkópos képek kiértékelése során számos adatpontot kell figyelnünk, ám bizonyos irányelvek útmutatásul szolgálhatnak a kívánt diagnózis elérése szempontjából.

A bőr felszínén észlelt, a bőr színét adó pigmentsejtek túlburjánzásából eredő, eltérő színű elváltozásokat és képleteket pigmentációnak nevezzük. Az elnevezés alapvetően a kialakult képlet jól körülhatárolható, eltérő színére utal. A bőr pigmentált elváltozásait elemi szinten vékonyabb vagy vastagabb, egyenes, görbe vagy körkörös vonalak, foltok, pontok és körök alkotják. Ezen alap geometriai formák ismétlődése mintákat alkothat az érintett bőrterületen. Egy minta strukturálatlannak tekinthető, amennyiben a benne ismétlődő geometriai formák közül egyik sem jellemzően domináns. Strukturált, szabályos mintákról akkor beszélhetünk, ha a mintát alkotó geometriai formák között van hangsúlyos típus. [9-10], [12]

A formák mellett a színeknek szintén meghatározó szerepe van a dermatoszkópiában. A bőr színét adó pigmentek közül a melanin a bőr felsőbb rétegeiben

fekete színként jelenik meg. A bőr alsóbb rétegeibe lehúzódó melanin barna, majd szürke és kékes színekben tűnik fel dermatoszkópos vizsgálatok során. [12]

Attól függően, hogy egy pigmentált minta miként strukturált, a bőr mekkora felületét érinti, valamint milyen színű, számos további következtetés levonható a megbetegedés típusával kapcsolatban. Az egyszínű, szabályos, strukturált minták kevésbé jeleznek rosszindulatú elváltozást [12]. A strukturálatlan minták, valamint a szürkés és kékes színárnyalatok a sejtek malignáns túlburjánzására utalnak a bőr felszínközeli vagy mélyebb rétegeiben [12].

3.2.3. Diagnosztikus modell betanítása

Mint láthatjuk, a rosszindulatú bőrelváltozásoknak számos dokumentált vizuális attribútuma van, melyek alapján azonosítani lehet őket. A mi esetünkben a malignáns elváltozások és bőrképletek oka, azaz a megbetegedés típusa nem számít.

Az egyszerűsített modellünk szerinti diagnózis bizonyos, a formákat és színeket meghatározott módon figyelembe vevő heurisztikák mentén felállítható. Míg az orvosok ehhez hasonlóan, adott lépéseket és döntési elveket követve jutnak el a diagnózisig, addig gépi tanulás esetén nem akarjuk ezen heurisztikákkal a modellünket közvetlenül programozni. A célunk az, hogy a modell a belső algoritmusának megfelelően, a bemenetére adott, általunk előre felcímkézett és kategorizált adatok alapján maga vonhassa le saját következtetéseit a tanulás során.

Fogalmazhatunk úgy is, hogy felügyelt gépi tanulás keretein belül egy megfelelően reprezentatív mintahalmaz kiválasztásával, számos iterációs cikluson keresztül finomítva indirekt módon vezetjük rá a modellünket olyan belső paraméterek előállítására, amik alapján az később diagnosztikus predikciókat tud előállítani egy korábban még nem látott képről. A korábbi fejezetekben bevezetett jelölésrendszer alapján, ha:

- α a bőr elváltozásainak tünetegyüttese
- $f(\alpha)$ a diagnosztizált állapot (rosszindulatú vagy jóindulatú)
- f pedig az induktív tanulás eredményeképp előállított diagnosztikus modell,

akkor a mi felelősségünk az α tünetegyüttes és az ehhez kapcsolható $f(\alpha)$ felcímkézett diagnózispárok előkészítése a modell számára úgy, hogy a tanulás során előálló f modell a célnak megfelelően generizált legyen. Ehhez szükséges ismernünk a malignáns elváltozásokra tipikusan jellemző vizuális adatpontokat, és úgy kell kiválogatnunk az α -

f(α) kép – diagnózis párokat, hogy abból a modellünk az adott kórképre jellemző vizuális mintákat maga is megtanulhassa.

Míg a betanított modell semmilyen kontextussal nem rendelkezik adott problémakörrel kapcsolatban, csupán képeket, és az azokhoz tartozó (malignáns – nem malignáns) diagnózisokat kapja meg, addig a tanulást felügyelő külső aktor, azaz a szoftverfejlesztő felelőssége a probléma megfelelő szintű átlátása, és az implementációhoz szükséges mintaadatok reprezentatív halmazának az összeállítása.

A gépi tanulás előnye, és egyben hátránya ott jelenik meg, hogy a tanulás eredményeképpen olyan vizuális mintákat, összefüggéseket és következtetéseket is le tud vonni modellünk, amelyeket mi magunk nem látnánk át. Mint látni fogjuk, nem megfelelően előkészített, vagy túl kicsi minták esetén ez félrevezető eredményekhez is vezethet.

3.3. Data curation, mintaadatok halmazának előkészítése

Mintaalkalmazásunk modelljének egyik legmeghatározóbb alapja a megfelelően összeállított és felcímkézett képek halmaza, melyeket tanításra, validálásra és tesztelésre egyaránt fogunk használni. A megfelelő minőségű és mennyiségű minta begyűjtése, kiválogatása és rendszerezése, azaz a data curation rendkívül időigényes feladat, és komoly átgondolást igényel [2]. A folyamat bizonyos részei automatizálhatók script-ek segítségével, de esetünkben jórészt manuális munkafolyamatokról van szó.

Első körben be kell szereznünk a mintának szánt képeket, és az azokhoz tartozó metaadatokat, melyek segítségével meg tudjuk határozni a képeken látható bőrelváltozások diagnózisát, illetve a képalkotás módját. Ezen metaadatok halmaza mindenképpen szükséges ahhoz, hogy az osztályozáshoz szükséges címkéket megfelelő módon tudjuk a képeinkhez hozzárendelni, valamint az adathalmaz minőségét garantálni tudjuk.

3.3.1. Diagnózisra vonatkozó adatpontok

A továbbiakban diagnózisnak csak és kizárólag azt az adatpontot tekintjük, mely a bőrelváltozás esetleges malignáns vagy jóindulatú állapotára vonatkozik. A modellünk szempontjából a diagnózis egyéb elemei, pl. a bőrbetegség pontos neve és típusa ignorálhatók. Mivel bináris klasszifikáció során két osztállyal dolgozunk, a címkék értékei a malignáns és jóindulatú diagnózisokat reprezentálják.

Az internetes forrásokban jelenleg publikusan hozzáférhető képek jelentős része nem, hiányosan, vagy nehezen verifikálható módon rendelkezik a diagnózisra vonatkozó metaadatokkal. Mivel a diagnózis számunkra elsődleges adatpont, annak helyességét mindenképpen ellenőriznünk kell, még a begyűjtött kép osztályozása előtt.

A Google keresőmotorjában célzott kulcsszavakkal indított, véletlenszerű keresésekre kapott találati képek esetén a diagnózis helyességének manuális ellenőrzése lassú és nehézkes folyamat. A képek beszerzése és címkézése felgyorsítható, illetve bizonyos szinten automatizálható, ha célzott kereséseket végzünk, és olyan adatforrásokat választunk ki, melyek nagy mennyiségben, metaadatokkal összerendelve nyújtanak hozzáférést a számunkra szükséges bőrelváltozásokat tartalmazó képekhez.

Némi kutatás után rátaláltam az ISIC, azaz az International Skin Imaging Collaboration publikus kollekciójára. Az ISIC digitális archívuma az egyik legnagyobb, publikusan elérhető gyűjtemény, mely nyilvános képforrásként szolgál kutatáshoz, oktatáshoz, valamint diagnosztikai mesterséges intelligencia-algoritmusok fejlesztéséhez és teszteléséhez. A továbbiakban képforrásként az ISIC digitális archívumát fogom használni. [13]

3.3.2. Képalkotás módjára vonatkozó adatpontok

A képalkotás módja nem feltétlenül szükséges adatpont ahhoz, hogy a vonatkozó címkéket hozzárendelhessük a megfelelő képekhez, viszont annál fontosabb a képek kiválogatása során. A megbízhatóbb eredmény szempontjából fontos szem előtt tartanunk, hogy a különböző képalkotási módszerekkel előállított képek eltérő eredményhez vezethetnek még akkor is, ha a diagnózis szempontjából azonos mintát tartalmaznak [1-3].

A képforrásként használt ISIC archívum képeinek jelentős része jelenleg dermatoszkopikus úton került rögzítésre, így megfelelő számosságú mintahalmaz összegyűjtésekor be kell látnunk, hogy az ilyen képalkotási módszerrel készült képeink többségben lesznek. [13]

Amennyiben az adott diagnózisosztályt reprezentáló mintahalmazban a dermatoszkopikus és egyéb képalkotási módszerekből származó képek nem kiegyensúlyozhatók, úgy a kisebbségben lévő, egyéb képalkotási módszerekből származó képek zajként fognak megjelenni a modellben. A zaj olyan zavaró tényező a mintaadatok halmazában, mely nehezen azonosítható, félrevezető eredményeket produkál a modellünk

betanítása során [2-3]. Dolgozatomnak nem célja a zajkeresés bonyolult folyamatának bemutatása, helyette a már előre azonosított, képalkotási módszerből származó zajforrást igyekszem megszüntetni.

A betanítás során a továbbiakban dermatoszkópos képalkotással kinyert mintaképekkel fogok dolgozni. A képalkotási adatpont ilyen típusú lekötése a mintaadatok halmazának homogenitását számtalan további szempontból biztosítja, melyek alapvetően a dermatoszkópos vizsgálatok körülményeiből inherens módon következnek: a mintaképeink megfelelő nagyítással és megvilágítással fognak rendelkezni, fókuszukban pedig elvárhatóan egyetlen, izolált bőrelváltozás lesz.

Nem szabad elfelejteni, hogy minden egyes adatpontra vonatkozó döntés meghatározó lesz a modellünk viselkedése szempontjából. Minden modell olyan típusú adatokon tud legpontosabban operálni, amelyekkel az be lett tanítva [2-3]. A tesztelés során látni fogjuk, hogy a képalkotási adatpont fent említett rögzítése legalább annyi hátránnyal jár, mint amennyi előnyt jelent. Egyik oldalról megszüntettük a mintaadatok összeállításakor az eltérő módon készített képekből származó zajforrást. Másrészről számíthatunk rá, hogy az így betanított, alkalmazásba integrált modellünk nem fogja tudni pontosan klasszifikálni telefonunk kamerájának képeit, hisz egy dermatoszkóp és telefonkamera más minőségben tud képeket szolgáltatni.

3.3.3. Mintagyűjtés

Mint azt korábban említettem, a felhasznált képek az ISIC digitális archívumából származnak. Ahhoz, hogy képesek legyünk mintaadataink effektív begyűjtésére, első körben át kell látnunk az archívum struktúráját, a képek és a hozzájuk tartozó metaadatok elérhetőségét, formátumát, valamint az azok kereséséhez, szűréséhez és letöltéséhez rendelkezésre álló eszközöket.

Az ISIC gyűjteményében jelenleg közel 70.000 kép érhető el publikusan a rájuk jellemző adatpontok és attribútumok halmazával együtt [14]. A metaadatokat képző attribútumok egy része strukturált, mely lehetővé teszi a képek keresését és szűrését a kiválasztott attribútum értékek mentén. Adatgyűjtés során az archívum weboldal-alapú portálját, valamint az azt működtető webes API (Application Programming Interface) végpontjait egyaránt igénybe vettem. Míg a webes API a képek automatizált keresése és szűrése során volt nagy segítséggel, addig a weboldal elsősorban a képek kiválogatása során játszott fontos szerepet.

A képek letöltéséhez szükséges linkekhez, valamint az azokkal asszociált attribútumok listájához a publikusan hozzáférhető webes API `/images/search/` végpontján keresztül jutottam el. A képkeresési végpont URL paraméterként várja a kulcs-érték párokba szervezett keresési paramétereket, és JSON (JavaScript Object Notation) formátumba rendszerezve szolgáltat válaszokat. [14-15]

A következő keresési paraméterek, valamint az azt felhasználó curl script összeállítása nagy segítséggel voltak a metaadatok begyűjtése során malignáns elváltozás-t ábrázoló, dermatoszkopikus képalkotásból származó képek esetén:

```
curl -X GET "https://api.isic-
archive.com/api/v2/images/search/?limit=1000&query=benign_malignant%3Amalignant%20AND
%20image_type%3Adermoscopic" -H "accept: application/json"
```

Ugyanez jóindulatú elváltozásokra vonatkozó adathalmaz esetén:

```
curl -X GET "https://api.isic-
archive.com/api/v2/images/search/?limit=1000&query=benign_malignant%3Abenign%20AND%20
image_type%3Adermoscopic" -H "accept: application/json"
```

A fenti lekérdezések 1000-ben limitálják a lekért minták számát. A válaszként kapott JSON struktúra képekre lebontva tartalmazza azok archívum béli egyedi azonosítóját, letöltési URL-jét, valamint információt a forrásról, a képalkotás módjáról és a számunkra szignifikáns diagnózisról.

Mivel a JSON formátum elsősorban gépi feldolgozásra, és nem emberi olvasásra optimalizált, a tényleges letöltés előtt a képeket manuálisan is visszaellenőriztem az ISIC portál galéria funkciójával. Az egyedi képazonosító birtokában a rendszer könnyedén betölti a keresett kép és vonatkozó attribútumainak emberi olvasásra alkalmas nézetét.

3.3.4. Az adathalmaz finomítása

Az adatgyűjtés eredményeként előállt minta további strukturálásra és válogatásra szorul. A képek halmazát a primer adatpontunk, azaz a diagnózis alapján két osztályra kell bontanunk. A szelekció során ezen diszjunkt osztályok kiegyensúlyozottságát úgy biztosítottam, hogy azokba ekvivalens számosságú mintaadatot válogattam. A képalkotás módszerét, mint másodlagos adatpontot már a képgyűjtés során figyelembe vettem, és a korábban ismertetett limitációk miatt homogén, kizárólagosan dermatoszkopikus módszerekkel készített képeket válogattam ki.

A kiegyensúlyozás módszere mellett igyekeztem más metrikák mentén is javítani a felhalmozott minta minőségén. Csökkentettem az adathalmaz elfogultságát azáltal, hogy lehetőség szerint sötétebb és világosabb bőrfelületekről készült felvételeket egyaránt szerepeltettem a mintában. A bőrszínen kívül más adatpontok, pl. az érintett bőrfelület szőrrel való lefedettségét szintén figyelembe vettem a célként kitűzött klasszifikáció kontextusában.

Bár szőrrel borított, sima, sötétebb és világosabb bőrfelületekről készült felvételek egyaránt feltűnnek az adatok között, a minta ezen adatpontok mentén vett teljes kiegyensúlyozottságát és elfogulatlanságát nem tudtam biztosítani, elsősorban a rendelkezésre álló minta alacsony számosságából adódó limitáció miatt. Az ebből származó esetleges eltéréseket figyelembe kell vennünk a tesztek későbbi kiértékelése során.

Ennél már komplexebb kérdést vet fel, hogy ún. kiugró értékek azonosíthatók-e az összekészített mintában, és ha igen, akkor milyen vizuális jegyek alapján érdemes ezeket kiszűrni. Tekintve a problémakör összetettségét, és a rosszindulatú bőrelváltozások vizuális megjelenésének számtalan variációját, további szakirányú egészségügyi előképzettség hiányában a kiugró értékek lehetséges előfordulásával nem foglalkoztam.

4. Modellgenerálás

Az alkalmazás középpontjában a célnak megfelelően betanított, bináris klasszifikációra alkalmas modell áll. Mint látni fogjuk, a modell szoftveres integrálására és használatára a Core ML segédkönyvtárat választottam ki, így kulcsfontosságú volt olyan eszközök azonosítása, melyek segítségével Core ML-kompatibilis modelleket tudtam előállítani, avagy szükség esetén Core ML által értelmezhető formátumúvá tudtam konvertálni azokat.

4.1. Fejlesztői környezet összeállítása

A fejlesztői környezet kialakítása során a mintaalkalmazás implementálásához szükséges fejlesztői eszközök és szoftveres segédkönyvtárak felkutatása, kiválasztása, valamint szükség esetén bekonfigurálása volt a célom a feladatspecifikációban definiált iOS-es célarchitektúrának megfelelően.

Az implementáció különböző fázisaiban a macOS operációs rendszerre telepíthető Xcode IDE 13.x-es fő verzióját, illetve az abban elérhető fejlesztői segédprogramokat használtam. Az Xcode olyan integrált fejlesztői környezetet biztosít, melynek segítségével natív iOS alkalmazások implementálhatók Swift nyelven. Magában foglalja a fejlesztéshez elengedhetetlen Swift Compiler nyelvi fordító- és linkelő eszközeit, továbbá grafikus felületet nyújt a forrásfájlok szerkesztéséhez, teszteléséhez és verziókövetéséhez.

Míg a forrásfájlok összeállításához és fordításához az Xcode de-facto szabvány iOS-es fejlesztői körökben, addig a gépi tanulási modell betanításához és integrálásához számos különféle további lehetőség közül választhatunk. A rendelkezésre álló opciók felmérése után választásom a Create ML grafikus segédprogramra, illetve az azt működtető Create ML segédkönyvtárra esett.

A Create ML segítségével jól parametrizálható és skálázható platform-optimalizált klasszifikációs modelleket tudtam készíteni, melyeket aztán kevesebb technikai nehézség árán tudtam az alkalmazás projektjében elhelyezni és kezelni az Xcode erre kifejlesztett támogató funkcióit kiaknázva.

A felhasznált iOS-es szoftveres segédkönyvtárak közül a gépi tanulás-specifikus Core ML-t szeretném kiemelni. A Core ML könyvtár nyújtotta API szoftveres absztrakcióinak segítségével a betanított modellem predikcióit a mintaalkalmazás

számára értelmezhető módon tudtam feldolgozni kódból, majd olvasható formátumra konvertálás után megjeleníteni a felhasználó számára.

4.2. Előtanított modellek keresése

Lényeges kiemelni, hogy a Core ML framework nem csupán egy szoftveres segédkönyvtár, hanem gépi tanulási modellek terjesztésére használható nyílt fájlformátum is egyben [23]. Első megközelítésben olyan előre elkészített, Core ML-kompatibilis megoldásokat igyekeztem felkutatni, melyek modelljét az alkalmazásba beágyazva a kívánt diagnosztikus funkciót meg lehet valósítani.

Az Apple fejlesztői portálon számos, előre betanított képklasszifikációs modell érhető el, melyek más és más modellarchitektúrán alapulnak. Ilyenek pl. a MobileNetV2, Resnet50 vagy SqueezeNet. Ezen modellek követik a Core ML fájlformátumát, és kifejezetten képek multinominális klasszifikációjára lettek tervezve.

A modellspecifikációk áttanulmányozása után azonban be kellett látnom, hogy ezen modellek túlságosan generikus adathalmazon lettek betanítva ahhoz, hogy a kitűzött probléma megoldása során használhatók legyenek. A felismert paraméterek magas száma, valamint a dokumentált klasszifikációs pontosság ellenére ezen modellek nem tudnak közvetlenül hatékony és eredményes támogatást nyújtani olyan speciális esetben, mint a feladatkitűzés szerinti bőrelváltozás-klasszifikáció.

A Core ML standardja mellett azonban számos más, gépi modell-specifikus fájlformátum is létezik. Ilyenek pl. a PyTorch, Keras vagy TensorFlow segítségével generált modellek, melyeket szükség esetén Core ML-kompatibilis formátumra lehet konvertálni a coremltools nevű Python csomag felhasználásával [24].

Következő lépésként olyan modellek felkutatásába kezdtem, melyek nyíltan elérhetőek, és szükség esetén Core ML formátumúvá alakíthatók. Sajnos ezen módszer mentén sem jártam sikerrel. A talált megoldások genericitása mellett számos olyan limitációba ütköztem, melyek meggátolták a kipróbálásra szánt modellek konverzióját.

Mint kiderült, bizonyos modellstruktúrák, így neurális háló rétegtípusok és műveletek nem konvertálhatók az elvárt formátumra, mivel azokat a Core ML könyvtár eleve nem támogatja [24]. A megfelelő háttérismeret birtokában ezen modellek természetesen módosíthatók, konvertálhatók majd kipróbálhatók, ez azonban nem célja dolgozatomnak.

4.3. Egyedi modell betanítása

Mint láthattuk, előre betanított modellek használata nehézkes, és legtöbb esetben nem célravezető. A legtöbb, általam fellelt modell túl generikus volt, illetve technikai korlátok következtében nem, vagy csak igen jelentős mértékű fejlesztői munka árán lett volna konvertálható az elvárt formátumra. Ezen tapasztalatok birtokában másfajta megközelítésben haladtam tovább, és saját, feladatspecifikus Core ML modell betanítását tűztem ki célul.

4.3.1. A céleszköz kiválasztásának szempontjai

Teljesen új, Core ML-kompatibilis képklasszifikációs modellek betanítására számtalan eszköz áll rendelkezésre, ám a megfelelő kiválasztásához érdemes átgondolnunk, hogy milyen sémák mentén szeretnénk a modellünket betanítani és használni. A kiszemelt eszköznek képesnek kell lennie a kitűzött feladat ellátására, mindezt gyorsan tanulható és kezelhető interfész, valamint egyszerűen értelmezhető validációs-metrikák biztosítása mellett. A modell betanítása a gépi tanulás súlyponti fázisa, a nem testhezállóan megválasztott modellgenerátor pedig akár akadályozhat is a munkafolyamat során.

A céleszköz kiválasztása során számos feltételt vettem tekintetbe. Első körben a probléma kontextusából indultam ki. Az előkészített mintahalmazon bináris klasszifikációt kell alkalmazni, az ehhez szükséges minta egyedeket pedig a predefiniált malignáns-nem malignáns adatpontok mentén szétválasztani. A felügyelt tanítás paradigmáit követve ugyan más adatpontokat is figyelembe vettem az adathalmaz előkészítése során, ha jobban belegondolunk, azok a modell számára nem láthatóak. A diagnózis, mint primer adatpont az egyetlen attribútum, mely dedikált címke formájában megjelenik a modell bemenetén. Elég tehát olyan modellgenerátort választani, melynek bemeneti halmaza mindenfajta bonyolítás nélkül, a legegyszerűbben parametrizálható a klasszifikációs címkék mentén.

Másodrészt, szem előtt tartottam az elkészült modell tervezett felhasználási környezetét. A mobilalkalmazások méretét több szempontból is érdemes alacsonyan tartani. Minél nagyobb egy alkalmazás, annál nagyobb adatforgalmat generál letöltés és telepítés közben, több helyet foglal a háttértáron, több időbe telik elindítás során a memóriába betölteni, és nagyobb memória lenyomattal fog rendelkezni futás közben. Mivel a megvalósítandó modell valójában egy .coreml kiterjesztésű fájl, melyet az iOS

alkalmazásba csomagolva használunk, így célravezető ezen fájl méretének alacsony szinten tartása. Ezen gondolatmenet mentén figyelemmel voltam arra, hogy a kijelölt eszköz képes legyen a modell méretének minimalizálására, pl. platformspecifikus optimalizációk kiaknázásával.

Végül, de nem utolsó sorban, ügyelnem kellett a modell betanítási kontextusára, azaz arra a számítógépre, melyen a tanítás folyamatát terveztem futtatni. A modellek algoritmusai a mögöttük lévő matematikai optimalizációknak köszönhetően előszeretettel használnak vektoralapú processzorműveleteket. Így olyan, macOS-en is elérhető eszközt kerestem, mely képes a számítógép grafikus kártyáját igénybe venni a tanítási folyamatok felgyorsítása érdekében.

4.3.2. Optimalizálás Transfer Learning-el

A betanítás során használt számítógép hardveres erőforrásai számítási kapacitás és memória szempontjából egyaránt korlátozottak. Bár a modelltanító- és generáló segédeszközök zöme rendelkezik az eljárást gyorsító optimalizációkkal, azok önmagukban sokszor nem elegendőek. Amennyiben a problémakör szemantikája engedi, és lehetőség van a tanulási folyamat finomítására egyéb heurisztikák mentén is, úgy ajánlatos ezeket megkeresni és eljárás szinten alkalmazni.

Ezen logika mentén olyan megközelítés adaptálására törekedtem, mely viszonylag effektíven képes az említett limitációk okozta hátrányok, legfőképpen a betanítási és verifikációs fázisok futtatási idejének lecsökkentésére úgy, hogy közben az ne legyen aránytalan mértékű kihatással a végtermék, azaz a modell klasszifikációs pontosságára. Némi tájékozódó ismeretkutatás után rátaláltam a transfer learning-alapú optimalizációs heurisztikákra.

A transfer learning metodikája szerint egy már meglévő, előtanított modellre alapozva tanítjuk be saját modellünket, ezzel is felhasználva az előtanított modell már korábban elsajátított predikciós képességeit és gyorsítva a modellgenerálás folyamatát [25]. Maga a transfer learning elnevezés arra utal, hogy egy meglévő tudásanyagra építve, annak átadásával végezzük a tanítást.

Az első próbálkozásra talált Core ML modelleknek nem tudtam közvetlen hasznát venni, mert nem az elvárt kimenettel rendelkeztek és túlságosan generikus célt szolgáltak. A bőrfelületek rákos elváltozásainak képi klasszifikációs vizsgálata annyira speciális terület, melyre nem találtam előtanított, Core ML-kompatibilis modellt. Míg

feltételezhető, hogy a transfer learning annál effektívebb, mennél nagyobb az előtanításra használt és saját mintaadat halmazunk közti átfedés, addig azt is látnom kellett, hogy kifejezetten erre előoptimalizált tudásanyaggal bíró egyedi modellt nem találok.

Ennek ellenére semmiképpen sem redundáns az ezen modellek betanítása során felhalmozott tudásanyag, illetve az ennek előállításához felhasznált erőforrások mértéke: számtalan formára, színre és ezek relációjára vonatkozó vizuális adatpontot tartalmaznak a specifikációs sémájuk szerint. A saját modellem generálásához rendelkezésre álló hardveres erőforrások és mintaadatok korlátait figyelembe véve úgy határoztam, hogy transfer learning által megtámogatott betanítást fogok végezni, elvárva, hogy ezáltal csökken a betanításhoz szükséges időtartam, és a verifikációs metrikák szerint nem romlik jelentősen a kimenet minősége sem.

A gyakorlatban ez a tanítás részfolyamatokra bontását, majd ezen részeljárások kaszkádosítását jelentette. Az első szakaszban a célra kiválasztott előtanított modell felhasználásával a mintahalmaz egyedeire vonatkozó sajátos jellemvonások halmazát nyertem ki. Ezen karakterisztikus jellemvonások a modell belső, parametrizált adatszerkezetei szerint reprezentált sajátos információk, az első tanítási fázis végtermékei. Felfoghatjuk úgy is, hogy a bemeneten pixelek összetett mátrixaként megjelenő képeket az azokra jellemző, modell által összeállított jellemvonások leegyszerűsített absztrakciójává alakítottam.

A predefiniált címkék szerinti klasszifikációs modell tényleges előállítása a tanítás második fázisában történt meg. A bemeneten a mintaadatok képek helyett immár az első fázis során kinyert, szimplifikált absztrakciók formájában jelentek meg. A tanítás nem alacsony szintű pixelek, hanem a képek jellemvonásaira jellemző magasabb szintű elvonatkoztatások mentén ment végbe, ezáltal is felgyorsítva a műveletet.

Ennek konkrét, adott segédeszközzel felügyelt folyamatát és technikai részleteit a későbbiekben taglalom. Egyelőre azt lényeges kiemelni, hogy olyan céleszköz kiválasztása volt szempont, mely támogatja a transfer learning sémáját, és a könnyű kezelhetőség végett elrejti a háttérben zajló részfolyamatok összetettségét.

4.4.Create ML a gyakorlatban

A modell betanítására kiszemelt eszköz a Create ML framework és az arra épülő grafikus segédalkalmazás lett. A Create ML a célkitűzés szemszögéből megfelelő

választás, ám ennek átlátásához, valamint hatékony használatba vételéhez meg kellett ismernem annak felépítését és működési paradigmáit.

A Create ML az Apple Swift alapú, feladat-orientált, kifejezetten klasszifikációs problémák megoldására optimalizált fejlesztői eszköze. Szervező eleme az ún. transfer learning pipeline, mely lehetővé teszi előre betanított, általános célú alapmodelljeinek saját adatainkkal való specializálását. Ahogy említettem, a Create ML feladat-orientált segédeszköz. A megoldandó problémához illő feladat kiválasztása a felhasználó feladata, ezután a rendszer a specifikált feladathoz leginkább illő alapmodellt magától választja ki és rendeli hozzá a kontextushoz. A felhasználó fókuszában így a modellspecifikáció helyett a feladat megfelelő kiválasztása és felparametrizálása áll. A Create ML nagy adathalmazokon előtanított alapmodelljei felgyorsítják a betanítás folyamatát, mindezt a számítógép grafikus kártyájának hardveres erőforrásait kihasználva macOS operációs rendszeren. [16-17]

A Create ML grafikus felülete, valamint az azt működtető, Swift nyelven programozható segédkönyvtár voltaképpen ugyanazt a funkcionalitást látják el [18]. Modellem kivitelezésekor a Create ML alkalmazás grafikus felületére, valamint kisebb, célzott Swift scriptek írására egyaránt támaszkodtam. A tanítás menete mindkét esetben ugyanazokat a lépéseket követeli meg.

4.4.1. Fájlrendszer szintű címkézés

Kezdetben a mintahalmaz képeit a Create ML számára interpretálható módon rendszereztem és címkéztem fel. A Create ML a predefiniált osztályokat automatikusan, a bemeneti adatminta fájlrendszer szintjén kialakított struktúrája szerint azonosítja [17]. Az adathalmazom képeit három részre bontottam: egy betanítási-, egy validációs-, és egy teszthalmazra. Ezek mindegyike saját mappát kapott, azon belül két további almappát a malignáns és nem malignáns osztályoknak dedikálva. A mintaképek példányainak megfelelő mappába történő szervezése egyben a mintaadatok felcímkézését is megtestesítette.

A Create ML számára külön-külön biztosítottam ezen mappák fájlrendszerbeli elérési útvonalait, valamint azok felhasználási célját, azaz, hogy tanulásra, validálásra, vagy éppen verifikálásra szántam azokat. A Create ML beolvasás után identifikálta az elvárt osztályok számát és elnevezését az adott útvonalon megtalált almappák száma és nevei alapján.

Szeretném kiemelni, hogy betanításra osztályonként 1000-1000, validálásra és tesztelésre pedig további 200-200 mintaképet használtam .jpg formátumba tömörítve, az arra tervezett mappába rendezve. Az osztályok egyedei között átfedés nincs, tehát diszjunktak, ezáltal garantálva a validálás és tesztelés során előállított metrikák megbízhatóságát.

4.4.2. Create ML projekt létrehozása

A modell betanításához deklarált kontextust egy ezen célra létrehozott, .mlproj kiterjesztésű, Create ML-specifikus projektfájlba szerveztem. Ez nem összekeverendő a folyamat produktumaként generált, .mlmodel kiterjesztésű, Core ML-konform modell fájljával, melyet a projektben definiált kontextus beállításai alapján állítottam elő. A Create ML feladat-orientált eszköz, így a projekt létrehozásakor a feladathoz illő képklasszifikációs sémát állítottam be a projektnév és általános leírás attribútumai mellett.

A projektfájlban a betanítás konfigurációinak különféle variációit úgy mentettem el, hogy azokon keresztül nyomon tudjam követni a modelltanítás evolúcióját, valamint futtatás után az adott beállításhalmaz szerinti eredmény pontosságát mérő metrikák összességét. A Create ML képességeire támaszkodva számos különféle beállítás mentén végeztem betanítást, majd értékeltem ki azok sikerességét.

Annak érdekében, hogy a rendelkezésre álló apparátust hatékonyan tudjam kezelni, továbbá képes legyek a Create ML által futtatás során kijelzett, betanításra vonatkozó valós idejű információkat értelmezni, szükségem volt a Create ML gyakorlati működésének részletes megismerésére és átlátására.

4.5. A tanítás technikai lépései

A Create ML alkalmazása során a modell betanítását iterációkban, a framework elnevezéseivel élve ún. epoch-okban végeztem. Egy epoch során a Create ML minden egyes tanításra kiszemelt mintaegyedet egyszer futtatott át a modellen.

A tanítás műveletét így pl. nem csak a mintahalmaz számosságának változtatásával, hanem az iterációk számának variálásával is tudtam skálázni. Minél több iterációt konfiguráltam be, annál több alkalma volt a modellnek a mintaadatok feldolgozására.

4.5.1. Feature Extraction

A tanítás folyamatának optimalizálása végett a Create ML transfer learning-alapú, kétfázisos pipeline architektúrát követ. Az első fázis, az ún. feature extraction során a bemenetre helyezett, 299x299 pixel felbontású képeket a Create ML erre specializált, VisionFeaturePrint.Scene elnevezésű, előtanított konvolúciós alapmodellje átdolgozta, és lebegőpontos számok 2048 elemből álló vektorává alakította [20], [46]. A tanításra szánt adathalmaz több száz képét a rendszer természetesen nem egyszerre töltötte be a fizikai memóriába. A Create ML a képek metaadatait ún. MLDataTable struktúrákban tárolta el, a vonatkozó képeket viszont ténylegesen csak akkor töltötte be a számítógép memóriájába, amikor azok a feldolgozás közben sorra kerültek [40].

A transfer learning gyakorlati szerepének tanulmányozásakor az alábbi szempontok mentén haladtam: A felhasznált VisionFeaturePrint.Scene konvolúciós modell 299x299 pixel méretű színes képek processzálására alkalmas [20], [46]. Továbbá, a pixelek színének RGB-reprezentációja 3 számot igényel a piros, zöld és kék színkomponensek intenzitásértékének kifejezésére. Ezek alapján minden egyes bemeneten megjelenő kép 299x299x3=268.203, azaz nagyságrendileg kb. 270.000 számmal írható le.

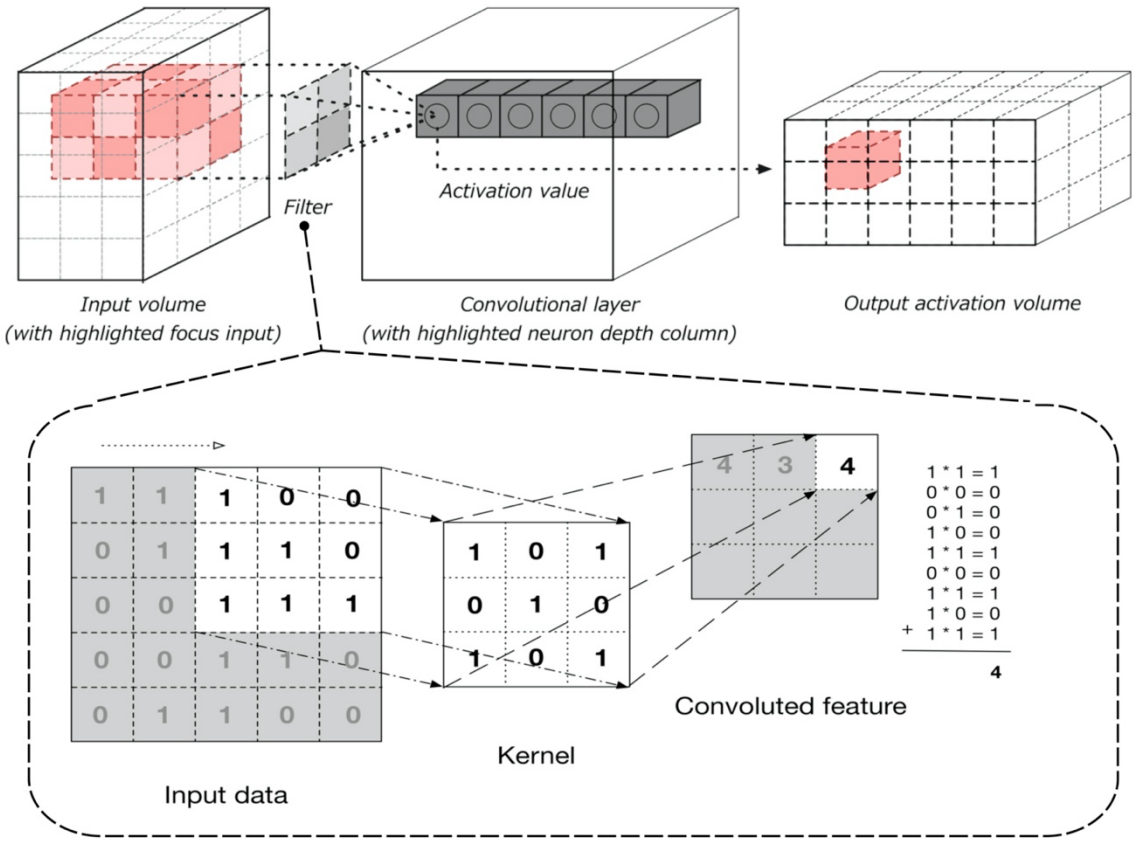
A feature extraction művelete után ezen képek a VisionFeaturePrint.Scene kimenetén már mindösszesen 2048 összetartozó szám együtteseként, a képekből kinyert ún. feature vektorok formájában tűntek fel. Az alapmodell tehát a képek komplex pixelmátrixát az azok vizuális attribútumaiból absztrahált adatpontok vektorára redukálta. Ezen absztrakciós adatpontok pontos jelentésével mélyebben nem foglalkoztam, azokra a kép tartalmának és karakterisztikáinak magas szintű reprezentációjaként tekintettem, melyeket a Create ML a háttérben automatikusan továbbított a pipeline második szakaszába. A 3. ábra a konvolúció alapú vektorizálás belső működését szemlélteti.

Gyakorlati tapasztalat alapján a feature extraction minden esetben a betanítás legidőigényesebb fázisa volt. Ennek ellenére az alapmodell felhasználásával sikerült a bemeneti mintaképeket egy számottevően kisebb méretű, letisztult adatstruktúrára redukálni, mely a későbbiekben jelentősen lerövidítette a klasszifikációs modell parametrizálási idejét.

4.5.2. Logisztikus regresszió

Bár az első fázisban a Create ML VisionFeaturePrint.Scene alapmodellje minden egyes képet feature vektorrá alakított, ezen lebegőpontos számvektorok önmagukban még

nem elegendők az általuk reprezentált képek bináris klasszifikációjához. A Create ML pipeline második fázisában a rendszer logisztikus regresszió alapján operáló klasszifikációs modellje a feature vektorok halmazát újbóli transzformációk során a predefiniált malignáns-nem malignáns osztályokon értelmezett valószínűségi eloszlás konstrukciójává alakította [21].

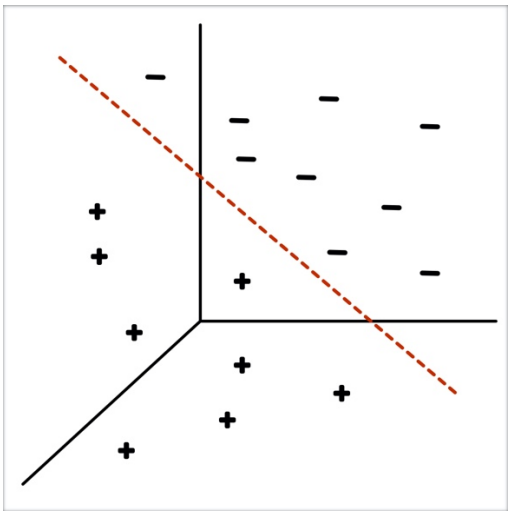


3. ábra: Konvolúció alapú feature extractor modell belső működése [3]

Annak átlátásához, hogy a Create ML klasszifikációs modelljének motorja, vagyis a logisztikus regresszió algoritmusa a bemenetére helyezett 2048 elemű számvektorokat miként dolgozta fel, illetve miként gyorsította ezen eljárást a feature extraction korábbi transzformációs művelete, értelmeznem kellett az algoritmus szemantikáját az adott probléma kontextusában.

A modell a mintákat, a bemeneti vektorok számossága alapján, egy 2048 dimenziójú hipertérben helyezte el, majd azokat a predefiniált osztályokra jellemző, általa detektált disztinktív attribútumok mentén csoportosította. A csoportosítás jelen esetben egy olyan hipersík meghatározása volt, mely a 2048 dimenziójú teret a malignáns-nem malignáns feature vektorok diszjunkt halmazára választotta szét. [1] A 4. ábra a

logisztikus regresszió által generált döntési határ egyszerűsített illusztrációját tartalmazza.



4. ábra: Logisztikus regresszió döntési határa (saját szerkesztésű ábra)

A Create ML modelltanítási paraméterei között szereplő iterációszám változtatásával így a tanítás első, feature extraction fázisára nem voltam kihatással, csupán azt tudtam kontrollálni, hogy a logisztikus regresszió hányszor dolgozza fel a mintaadatok vektorának halmazát, ezáltal pontosítva a regresszió végeredményeként előállított klasszifikációs modellt.

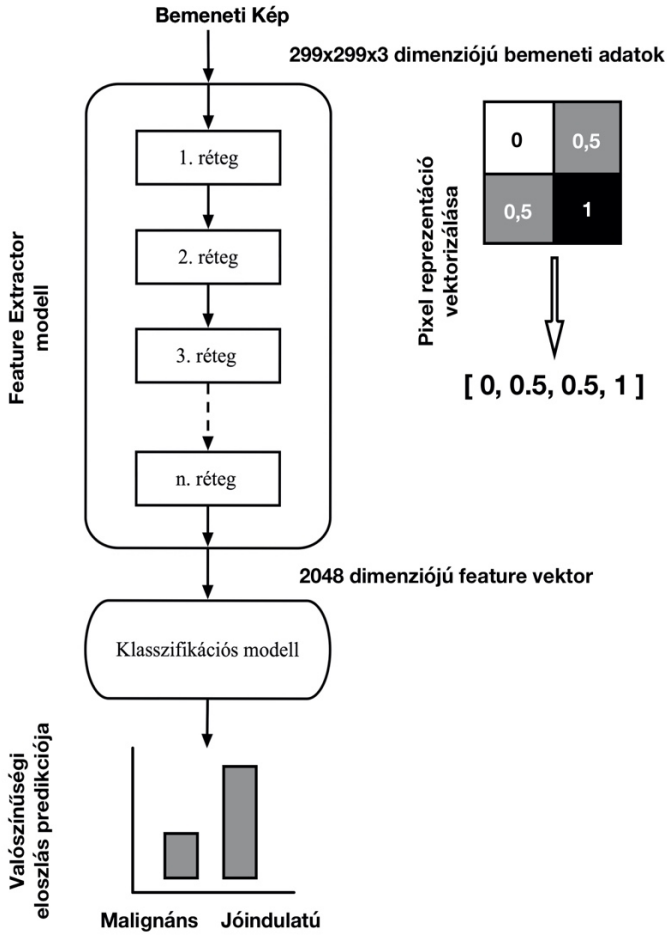
A feature extraction minden egyes tanítási eljárás elindításakor a Create ML pipeline elején, egyszer futott le. Miközben a feature extraction a pipeline futtatási idejének jelentős részét tette ki, az ezután következő logisztikus regresszió futásidejében relatíve elhanyagolható volt, még az iterációszám növelése mellett is. Annak köszönhetően, hogy a regressziós modellnek nem a képek nyers pixelmátrixával, hanem az abból kinyert, 2048 elemű absztrakciós adatstruktúrájával kellett operálnia, az iterációszám módosításakor nem kellett a futásidő aránytalan növekedésére számítanom. Az 5. ábrán a Create ML transer learning pipeline architektúráját láthatjuk.

4.6. Modelltesztelés, a Create ML verifikációs metrikái

A megfelelően strukturált forrásadatok és tanítási paraméterek specifikálása után a pipeline előállította az .mlmodel formátumú, bőrelváltozások klasszifikációjára képes célmodellt. A modellfájl applikációba integrálása előtt azonban ellenőriznem kellett a

végeredmény pontosságát. A Create ML a generált modell tesztelésére és validálására számos lehetőséget és metrikát szolgáltat, melyeknek a verifikáció során hasznát vettem.

A klasszifikáció osztályszintű pontosságának ellenőrzésekor a Create ML erre bevezetett, ún. precision és recall metrikáira támaszkodtam. A precision és recall a predefiniált osztályokra külön-külön számolt, százalékban kifejezett érték.



5. ábra: A Create ML transfer learning pipeline architektúrája (saját szerkesztésű ábra)

A precision azt mutatja meg, hogy az összes, a modell szerint adott osztályba predikált kép hány százaléka tartozik ténylegesen az adott osztályba [26]. Ha az osztályhoz tartozó precision értéke alacsony, akkor a modell gyakran osztályoz félre képeket, és állítja róluk hamis pozitív predikció mentén, hogy azok az adott osztályba tartoznak. Hamis pozitívnak tekinthetünk minden olyan predikciót, mely adott kép adott osztályhoz tartozását úgy állítja, hogy közben az állítás effektíve hamis. A betanítás során

a precision minél magasabb százalékos értékére törekedtem a hamis pozitív predikciók visszaszorítása érdekében.

A recall százalékos értéke ezzel szemben más szemantikát követ. Arra világít rá, hogy az ismerten adott klasszba tartozó egyedek hány százalékát fedezte fel a modell verifikáció során [26]. Az osztályra számolt recall alacsony értéke azt indikálja, hogy számos, adott osztályba tartozó egyedet nem tudott felismerni a modell, ezáltal hamis negatív predikciókat generálva. Egy predikció hamis negatív, amennyiben nem az adott osztályba sorol olyan képet, amely ténylegesen oda tartozik.

Az eredmények elbírálása során a recall vagy precision alacsony értéke felhívta a figyelmem a potenciálisan problémás osztályokra, és segített a modelltanítás célzott finomhangolásában.

Az osztályszintű metrikák mellett a Create ML további három mérőszámát tartottam szem előtt. Ezek rendre az ún. training accuracy, validation accuracy és testing accuracy. Ahogy azt nevük is sugallja, ezen értékek a modell betanítása, validálása és tesztelése során azt tárják fel, hogy az erre dezignált mintaképek halmazának hány százalékáról képes a modell helyes predikciókat generálni [22]. Amint azt a később részletezett példák során látni fogjuk, ezen metrikák más és más kontextusban biztosítottak holisztikus információt a generált modellem elvárható minőségéről.

4.7. A folyamat szemléltetése

A modell végső, felhasználásra szánt változatához a Create ML pipeline technikai működését, verifikációs metrikáit és parametrizálhatóságát fejben tartva, inkrementális lépésekkel, a korábban ismertetett szemantikákat és megoldásokat követve jutottam el. Az összes modellverzió lekövetése azok nagy száma és a tanítási kontextusok bonyolultsága miatt nem célom. A továbbiakban a folyamat szemléltetésére kiemelek pár meghatározó mérföldkövet, azok környezetét, minőségi metrikáit, valamint a hozzájuk vezető, inkrementális fejlesztési döntések mögötti megfontolásokat.

4.7.1. Kiindulási konfiguráció

A legelső összeállításban rögzítettem a tanításra szánt, 1000-1000 darabból álló mintahalmazt, melyek alapján a rendszer automatikusan felismerte a predefiniált címkéket. Az iterációk számát 25-ben maximáltam. Mivel más konfigurációt nem módosítottam, a Create ML tanítás előtt a minta egyedek halmazának közel 5%-át

leválasztotta, és validációs célra automatikusan félretette. A Create ML ezen célú, a folyamat validálási metrikájának a kiszámítására irányuló funkcióját auto validation-nek nevezik. Az auto validation nagyjából 95%-5% arányban automatikusan szétosztja a tanítási adathalmazt, amennyiben külön validációs mintákat nem definiálunk a rendszer számára. A félrerakott mintát sosem használja közvetlenül tanításra, ám validálja vele a modell hatékonyságát, és az ez alapján elkészült visszacsatolással finom hangolja a logisztikus regresszió bizonyos paramétereit.

Auto validation-re támaszkodva az eredmény 100%-os training accuracy és 93,9%-os validation accuracy lett, 3 perc futtatási idő és 25 iteráció után. A jóindulatú osztályt 94% precision és 95% recall, a malignáns osztályt pedig 94% precision és 92% recall értékek jellemzik. Ezek alapján a malignáns csoport elemeit a modell valamivel nehezebben ismeri fel, avagy könnyebben klasszifikálja félre, mint a jóindulatú egyedeket. A 100%-os training accuracy nem véletlen, a rendszer a betanításra korábban felhasznált képeit hiba nélkül tudja osztályozni. Ennél mérvadóbb mérőszám a 93,9%-os validation accuracy, mely a modellt a félrerakott, tanítás során sosem látott képek alapján validálja. A későbbiekben a modell minőségét a training accuracy helyett szigorúan a validation accuracy értékére alapozva korrigáltam.

4.7.2. Modelljavítás az iterációk számának növelésével

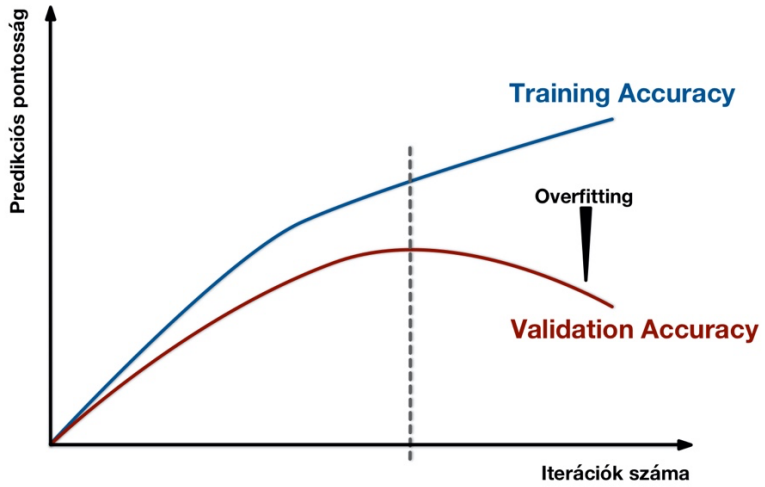
További lépésként az iterációk maximális számát emeltem, változatlan paraméterek mellett, első körben 50-re. A tanítás szintén 3 perc alatt fejeződött be, 50 iteráció után, ám a Create ML immáron 96,5%-os validation accuracy-t mutatott. A precision és recall a jóindulatú osztály esetében 97%-97%-ra, a malignáns osztály esetében pedig 96%-96%-ra emelkedett. A modell minősége mind holisztikus szemszögből, mind osztályokra lebontva javult, az osztályszintű metrikák közti diszkrepancia pedig 1%-ra csökkent.

Ugyanezen gondolatmenetet követve az iterációk maximális számának ismételt duplázásával további javulást feltételeztem. A pipeline 4 perc futásidő és 87 iteráció után konvergált, azaz némileg korábban befejezte futását, minthogy a dezignált iterációszám felső, 100-as határértékét elérje. A validation accuracy értéke meglepő módon visszaesett 94,7%-ra, mindeközben pedig a precision és recall a jóindulatú osztály esetében 95%-95%-ra, a malignáns osztály vonatkozásában pedig 94%-94%-ra romlott le. Az osztályszintű metrikák feltűnően együtt mozogtak, a két halmaz közti értékdifferencia továbbra is 1% maradt.

4.7.3. Overfitting

Mivel az egyetlen paraméter módosítás a maximális iterációszám ismételt megduplázása volt, a validation accuracy értékének ily mértékű visszaesése, valamint az osztályszintű metrikák ilyen szoros együtt mozgása mögött jó eséllyel az ún. overfitting jelensége áll. Overfitting akkor következik be, amikor nagyszámú iteráció során a modell bizonyos mintaadatokkal már túl sokszor találkozott. A minták detektálása helyett a modell így elkezdi memorizálni, majd bizonyos iterációszám után egy az egyben azonosítani a konkrét, rendszeresen ismétlődő minta egyedeket a vonatkozó osztályokkal [2-3]. A konkrét esetre levetítve ez annyit jelent, hogy némely képeket a modell már túl sokszor látott, így 87 iteráció után valószínűsíthetően elkezdte megjegyezni azokat.

Az overfitting jelenségének szemléltetése mellett azért is tartom mérvadónak ezen mérföldkő kiemelését, mert az átláthatóan érzékelteti a validation accuracy relevanciáját és a training accuracy-hez képesti különbözőségeit. A training accuracy praktikus mérőszám, stabil 100%-os értéke jelen esetben a mintahalmaz megfelelő preparáltságára utal. Ennél szignifikánsan alacsonyabb training accuracy komoly problémákat indikálna a mintahalmaz kompozícióját, kiegyensúlyozottságát vagy elfogultságát illetően.



6. ábra: Overfitting jelenségének kimutatása validációs metrikák segítségével (saját szerkesztésű ábra)

Másrészről láthatjuk, hogy a training accuracy önmagában véve félrevezetően optimista mérőszám tud lenni. Csupán ezen metrika változatlan, 100%-os értéke mentén

az overfitting jelenségét és a modell tényleges pontosságát nem lettem volna képes feltárni. A training és validation accuracy értékei közti nagy diszkrepancia kardinális faktor volt az overfitting okozta modelltorzulás kimutatása során, melyet a 6. ábra szemléltet.

4.7.4. Dedikált validációs halmaz bevezetése

A training és validation accuracy közti rés nem csupán overfitting-ből származhat. Ha a validációs halmaz elemei nem reprezentánsak a malignáns-nem malignáns osztályokra nézve, a validáció mérőszámai lecsökkenhetnek. Ugyanez tapasztalható, amennyiben a tanítási és validációs minták valamilyen adatpont mentén szétválnak, így pl. esetemben valamelyik halmazba javarészt sötétebb-világosabb, vagy éppen szőrrel fedett-fedetlen bőrfelületű mintaképek kerülnek.

Eddig a pontig a Create ML auto validation funkciójára támaszkodtam. Bár használata kényelmes, a további minőségnövelés céljából dedikált validációs halmaz bevezetése mellett döntöttem. Indíttatásom mögött a következő okfejtés állt: A training accuracy konstans, 100%-os értéke egyértelmű visszaigazolás volt a mintahalmaz minőségét tekintve. Az alapból kiegyensúlyozott minták halmazából az auto validation során a Create ML véletlenszerűen, a tanítási alaphalmaz kárára csippentett le 5%-nyi képet, és ezt használta validációs metrikáinak számítása során. Ezzel a tanítási halmaz számossága 95%-ra esett vissza, mely ugyan elsőre nem tűnt jelentősnek, de mégis negatívan befolyásolta a logisztikus regressziós modell tanítási eredményességét. Ezzel egyidőben a véletlenszerű kiválasztás mentén nem tudtam kontrollálni az előállított validációs halmaz különböző adatpontok menti homogenitását, kiegyensúlyozottságát és elfogulatlanságát, ami a validation accuracy korábban említett visszaeséséhez vezetett.

A mintahalmazok számossága és minősége feletti közvetlen ellenőrzés végett hosszútávú célként az alaphalmaz számosságának fix 1000-1000 elemen tartását, és osztályonként további 200-200 számosságú, dedikált validációs halmaz bevezetését tűztem ki. A modell így a tanítási alaphalmaz 20%-nak megfelelő, felügyelt összetételű képhalmaz alapján tudott validálni.

A validációs osztályhalmazok rögzítése után, első körben, az iterációk maximális számát felére, azaz 50-re csökkentettem. Elvárásom, miszerint a produktum validation accuracy mérőszáma nagyobb vagy egyenlő lesz a korábban auto validation felhasználásával futtatott, 50-es iterációszámú betanítással, beigazolódott: tizedes értékre

pontosan 96,5% training accuracy-t számolt a rendszer 50 befejezett iteráció után. Érdekes megfigyelni, hogy mindez alatt a futásidő 4 percre emelkedett fel a feldolgozandó képek magasabb számának köszönhetően. A feature extraction VisionFeaturePrint.Scene modellje 2000 helyett immáron 2400 képet alakított feature vektorok halmazává, ami harmadával növelte a pipeline futásidejét. A precision és recall jóindulatú képek esetén rendre 96%-98%, malignáns esetben pedig 97%-96% voltak. A gyengébben teljesítő malignáns osztály precision metrikája 1%-os javulást mutatott a korábbi 96%-hoz viszonyítva, mindezt úgy, hogy közben a modellszintű training accuracy értéke változatlanul 100%-os maradt.

A modell azonban nem konvergált. Ez azt jelenti, hogy az iterációszám felső, 50-es limitjén túl a rendszer még tudta volna optimalizálni a logisztikus regresszió belső paramétereit. Ne feledjük el azt sem, hogy auto validation-re támaszkodva, 100-as iterációkorlát mellett a rendszer ugyan konvergált korábban, de metrikái romlottak az overfitting következtében. A kontrollált validáció, valamint az általam előválogatott validációs halmaz előnyeire apellálva az iterációszámot ismét 100-ra állítottam, és bíztam a számolt metrikák stabilitásában. Ezen lépést azért is véltem esszenciálisnak, mert a beállítások bizonyos kombinációjával el kívántam érni a modell konvergenciáját, mely pontban több iterációt már nem végez a Create ML, még akkor sem, ha azt a beállítások engednék. Konvergált modell pontosságát adott beállítások kontextusában ugyanis már nem lehet javítani.

Az eredmény biztató. Változatlanul 4 perc futásidő, ám ezúttal 96 iteráció utáni konvergencia mellett a validation accuracy értéke összességében 96,8%-ra javult. Az osztályokra levetített precision és recall továbbra sem mutattak romlást, azok a korábban megszokott határok körül mozogtak: jóindulatú képek esetén rendre 96%-98%, malignáns esetben pedig 98%-96% értékeket vettek fel. Míg a jóindulatú osztály stagnált, a rosszindulatú csoport precision értéke 1%-os javulást indikált. A tanítási, illetve validálási mintahalmazok számosságának és kompozíciójának előnyös megválasztásával sikerült megtörni a korábbi iterációszám növelést kísérő metrikaromlást. Az adathalmazok méretnövelése nem mindig elegendő az overfitting kiküszöbölésére, esetemben ez a halmazösszetétel minőségének köszönhető.

Az overfitting jelenségét modell közelibb matematikai szemantikák mentén is ki lehet küszöbölni. Ilyen pl. a belső modellparaméterek túlsúlyozását büntető regularizáció, mely a modell bizonyos megtanult jellemvonások menti elfogultságának detektálásával

és visszaszorításával törekszik az overfitting megelőzésére. A regularizáció összetett heurisztikáival nem célom foglalkozni, helyette a data curation műveletével igyekeztem az elfogadható modellminőséget garantálni.

4.7.5. Augmentációk alkalmazása

Legutóbbi konfigurációm szerint a Create ML pipeline 96 lépésben konvergált, és minden eddiginél megbízhatóbb eredményt szolgáltatott. A 96,8%-os training accuracy a mintaalkalmazás kontextusában már számomra elfogadható modellpontosság, a kiindulási verzióhoz képest sikerült közel 3 százalékpontos minőségjavulást elérnem.

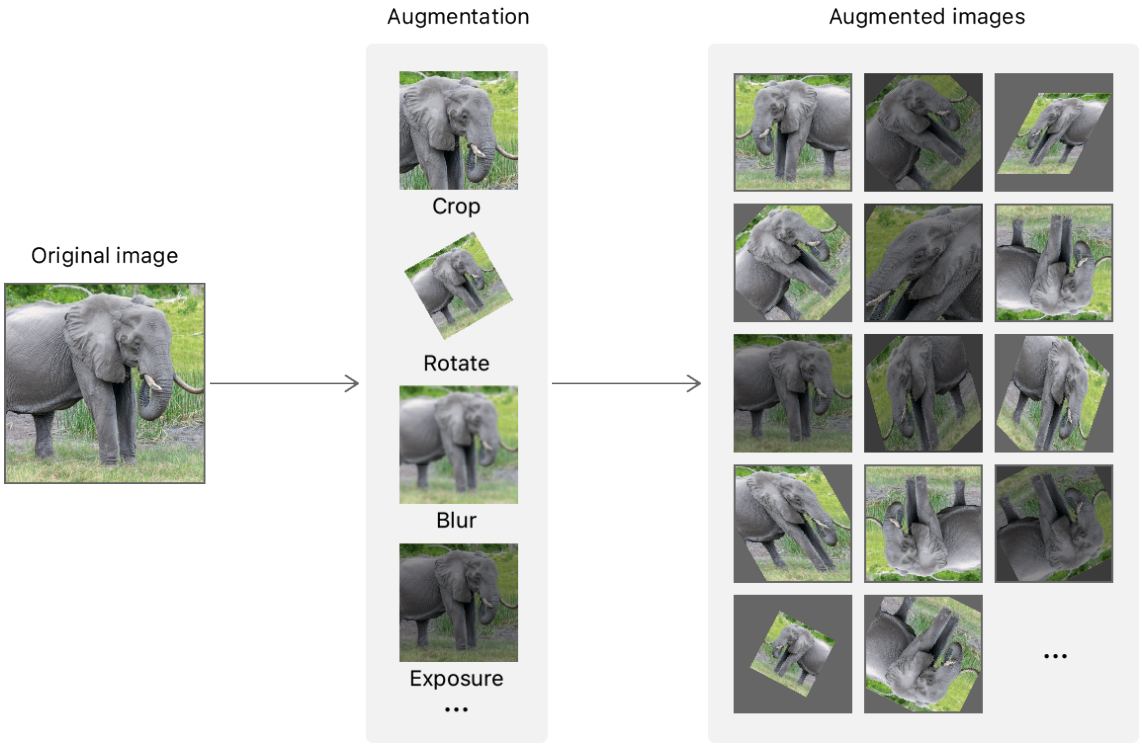
A Create ML klasszifikációs eszköztárában azonban van még egy eddig kihasználatlan opció, az ún. adat augmentációk alkalmazása. Adatok augmentálása során a rendszer a kiindulási adathalmaz elemein bizonyos, a problémakör kontextusában értelmezhető átalakításokat végez, majd ezen módosított egyedeket lementve megsokszorozza a rendelkezésre álló tanítási alaphalmaz elemeinek számát [22]. A háttérben gyakorlatilag a minta eredeti elemeinek lemásolt, és bizonyos elvek mentén transzformált verziói állnak.

Képklasszifikáció feladatkörében a Create ML az alábbi augmentációs lehetőségeket ajánlja fel, melyek vizuális reprezentációját a 7. ábrán láthatjuk:

- Add noise: a képek másolataihoz pixelszintű torzításokat, azaz zajt ad;
- Blur: a képek elmosásával – elhomályosításával transzformál;
- Crop: az eredeti képek bizonyos részeit véletlenszerűen kivágja;
- Expose: a képek sötétebb és világosabb verzióit készíti el;
- Flip: a képeket vízszintesen és függőlegesen is tükrözi;
- Rotate: a képeket véletlenszerűen elforgatja. [19], [22]

A malignáns bőrelváltozások vizuális jegyeit szem előtt tartva úgy találtam, hogy a képek forgatása (Rotate) és tükrözése (Flip) semmiféle negatív hatással nincs az eredeti mintahalmazra, hisz az nem módosít a dermatoszkópos képalkotással kinyert képek lényegi tartalmán. Egy rosszindulatú bőrelváltozásnak nincs orientációja, azt bármelyik irányból figyelve is ugyanazokkal a disztinktív attribútumokkal bír. Ugyanez szintén elmondható a tükrözés műveletéről. Ennek fejében a Rotate és Flip augmentációkat adoptáltam a modelltanítás során.

A megvilágítás (Expose) és elmosás (Blur) várható eredményeit már másképp értékeltem. A dermatoszkópos képalkotás jól megvilágított, éles képeket szolgáltat a vizsgálni kívánt bőrfelületről. Ezzel szemben számítanom kell arra az esetre, amikor a megvalósítandó mobilalkalmazás használata során a felhasználó közvetlenül a mobilkészülék kamerájából érkező kép alapján szeretne osztályozást végezni. Ezen képek megvilágítása és fókusza, a képalkotás forrásaként szolgáló mobilkamera optikai képességeinek korlátait figyelembe véve, vélhetően alacsonyabb minőségű lesz. Ennek ellensúlyozására az Expose és Blur augmentációs lehetőségeket szintén alkalmaztam a későbbiekben.



7. ábra: A Create ML augmentációs eszköztára [22]

Ezzel ellentétben, a fentebb említett lehetőségek közül a Crop és Add noise funkcióit egyáltalán nem applikáltam a mintahalmaz elemeire. Az Add noise, azaz a képek pixelszintű, randomizált eltorzítása a bőrképletek alapvető vizuális adatpontjait tette volna tönkre a méretek, formák, élek és színek megzavarásával. Ahogy zajt sem érdemes meggondolatlanul a mintába belecsempészni, úgy azok véletlenszerű kimetszését, azaz a Crop augmentációs műveletét is indokolatlan transzformációnak találtam jelen esetben. A vizsgált bőrfelület egy előre ismeretlen részének elvesztése legalább olyan dezinformáló lehet, mint zaj hozzáadása a képhez.

Folytatásként a Blur, Expose, Flip és Rotate augmentációkat konfiguráltam be a pipeline újbóli futtatása előtt, minden más beállítás érintetlenül hagyása mellett. A Create ML kiegészítette a mintaképek halmazát a kívánt augmentációk szerint, ennél fogva a feature extraction műveletideje jelentősen megnövekedett. A pipeline 7 perc alatt zárta le a 100 iterációt, ám ezalatt nem konvergált. 96,6%-os validation accuracy mellett a modell épphogy két tizedesjegy pontossággal romlott, a precision és recall értékei pedig teljesen változatlanok maradtak. A nagyfokú futásidő növekedés ellenére érdemi javulást nem tapasztaltam. Az augmentálás drága, erőforrás igényes eszköz, így alkalmazásakor a ráfordított idő megtérülését a modellmetrikák javulásának formájában vártam el. Ezek alapján adott a kérdés, hogy a különféle augmentációk képesek-e a modellem további finom hangolásához érdemben hozzájárulni, amennyiben az általuk generált mintapopuláció növekmény, változatlan eredmények mellett, láthatóan közel duplájára emeli a feature extraction végrehajtási idejét.

A kérdés megválaszolásához a konvergálás sikerességét kellett vizsgáljam. Augmentálás nélkül 100 iteráción belül, egészen pontosan 96 iteráció befejeztével a pipeline végig tudott futni, majd konvergálva lezárult. Az augmentációknak köszönhetően a rendszernek immár jóval több mintaképet kellett feature vektorrá konvertálnia, majd logisztikus regressziós modelljével klasszifikálnia. Egészen pontosan a korábbi 1000 kép helyett osztályonként 2000-2000 képpel tanított, valamint 200 helyett 400-400 képpel validált a pipeline. A képek számával arányosan hosszabbodott a feature extraction fázisa, és ezáltal a pipeline futásideje is. Mivel a logisztikus regresszió időigénye elhanyagolható a feature extraction fázisához képest, csak többszöri átvizsgálás után szembesültem a ténnyel, miszerint a logisztikus regresszió optimalizálásához valójában nem volt elég iterációs lehetősége a rendszernek.

Ezen sarkalatos faktorra ráeszmélve a következőkben 200-ban fixáltam az iterációk maximális számát, majd a pipeline-t újraindítottam. Emlékeztetőül, az overfitting jelenségét ezt megelőzően 100 iteráció alatt, a tanítási- és validálási alaphalmazok manuális konfigurációjával sikerült visszaszorítanom. Az iterációszám további növelésével meg volt az esélye annak, hogy a modell újfent elkezdi memorizálni a számára rekurrensen megmutatott képek egy részét. Az iterációszám további növelése csupán azért volt számba vehető, mert az augmentációk során a rendszer legbelül megsokszorozta mintahalmazainak méretét. Ugyanazon képek repetitív felhasználása helyett a további iterációkban az eredeti minták különféle módokon átalakított másolatait

futtatta át a logisztikus regresszió modelljén, így annak kevesebb lehetősége volt egy-egy specifikus kép tartalmának megjegyzésére.

Az irány helyesnek bizonyult. A pipeline futásideje 7 percben állandósult, viszont 116 iteráció után konvergált, és a klasszifikációs modell optimalizálásával lezárult. Korábban nem látott, 97,5%-os validation accuracy-t sikerült elérnem, méghozzá overfitting nélkül. Az osztályszintű metrikák végre kitörtek korábban megszokott értéksávjukból, és mindkét csoport szemszögéből javuló tendenciát mutattak. A jóindulatú csoport precision értéke 97%, a recall 98% lett. Mind a hamis pozitív, mind a hamis negatív predikciók arányát sikerült 2-3% körüli tartományba leszorítani. A rosszindulatú halmaznál ezen metrikák rendre 98% precision és 97% recall lettek, ami a korábban kissé rosszabbul teljesítő csoportra nézve azt jelenti, hogy eredményesen utolérte komplementerosztályának pontosságát.

4.7.6. Az eredmény tesztelése

Az elkészített modell jelenlegi, optimalizált változatához a training és validation accuracy mérőszámainak lekövetésével, az adott mérföldkövek beállítási környezetére alapuló megfontolások mentén jutottam el. Utolsó lépésként a produktumot a Create ML beépített lehetőségeit felhasználva, kifejezetten ezen megfontolásból kiválogatott, osztályonként 200-200 elemű teszthalmazok segítségével verifikáltam. Hangsúlyoznom kell, hogy a betanítási- és validálási alaphalmazok egyetlen elemét sem alkalmaztam a modell befejező kontrollálása során. A tanítási-, validációs- és teszthalmazok eredeti, augmentálás nélküli formájukban rendre 1000, 200 és 200 számosságú, maradéktalanul diszjunkt halmazok.

A verifikációra szánt teszthalmaz beállítása után hagytam, hogy a Create ML lefuttassa verifikációs folyamatát, és előállítsa az ún. testing accuracy értékét. A 100%-os training accuracy és 97,5%-os validation accuracy mellett a testing accuracy értéke így 90% lett, azaz láthatóan alacsonyabb, mint a validációs metrika indikátora. A korábbi tapasztalatok alapján be kellett látnom, hogy a Create ML betanítási és validálási műveletei az alapmodellek adott számossága és kompozíciója mellett nem finomíthatók tovább a végeredmény torzítása, pl. overfitting nélkül. A testing accuracy a modell végső verifikációs mérőszáma, mely szerint a modell várhatóan 10-ből 9 esetben helyes predikcióval fog élni a bőrelváltozások klasszifikációja során.

A rendszer osztályszinten is verifikálta a modellt. Jóindulatú elváltozásokra 88% precision és 94% recall, malignáns mintákra pedig 93% precision és 87% recall értéket számolt. Recall alapján a modell a jóindulatú elváltozások 94%-át, a malignáns képleteknek pedig 87%-át tudta azonosítani. A precision szerint a malignáns predikciók 93%-ban, a jóindulatúak pedig 88%-ban voltak helyesek.

Azt, hogy ezen eredményt a mintaalkalmazás kontextusában mennyire találtam elfogadhatónak, jelentős mértékben befolyásolták a mintahalmaz, illetve a használt eszköz korlátai. A Create ML számos konfigurációja mentén már nem tudtam olyan kombinációt beállítani, ami a korábban kifejtett okok miatt ne járt volna torzulással. Az összegyűjtött adathalmaz számosságát pedig nem állt módomban úgy növelni az adatforrás limitációinak tükrében, hogy annak összetétele a data curation során figyelembe vett adatpontok mentén ne kezdjen el romlani. Továbbá mérlegelnem kellett azt a tényt, hogy adott teszthalmazra finom hangolt, folyamatosan javítgatott modell egy idő után várhatóan elfogult lesz a felhasznált teszthalmaz irányába, ami a holisztikus képet tekintve szintén nem kívánatos torzulás esetemben.

Továbblépés előtt azonban elengedhetetlen volt felderítenem a validation és testing accuracy értékei közti differencia okát. Mint kiderült, a különbség forrása jó eséllyel a Create ML tanító algoritmusának belső működésében keresendő. A logisztikus regresszió modellje kiindulási állapotában teljesen véletlenszerű súlyozásokkal rendelkezett, mely iterációról iterációra módosult a predikciók hibaértékének, az úgynevezett veszteségnek a csökkentése végett. A logisztikus regresszió modellje a tanítóhalmaz mellett a validációra használt egyedek hibaértékeit is beépítette veszteségminimalizáló függvényébe, ezáltal módosítva saját súlyozását [1], [3]. Ezen súlyokra a szakirodalom sokszor paraméterként, vagy validáció esetén ún. hiperparaméterként referál, a folyamatot pedig hiperparametrizálásnak hívja [3], [27]. Mivel a validációs halmaz képei maguk is befolyásolják a modell végállapotát, a validation accuracy értéke, bár kulcsfontosságú metrika, a tanítás során sosem lehetett teljes mértékben független, elfogulatlan mérőszám. Ebből következett a két metrika közötti százalékos eltérés.

A modell számára ellenőrző, és egyben visszacsatolási pontokat nyújtó validáció tehát olyan iteratív művelet, mely modellfejlesztés közben releváns iránymutatással szolgált. Ezzel szemben a verifikáció során a modell végső állapotának kiértékelésére szánt minőségi mérőszámokat állítottam elő, egyszer, a folyamat legvégén.

4.8. Kiértékelés

A Create ML végeredményben megfelelt a modellgenerálás folyamatával és produktumával szemben támasztott, korábban már részletezett elvárásaimnak. A macOS operációs rendszerre optimalizált pipeline kamatoztatta a számítógép grafikus kártyájának vektorműveleti képességeit: a mintegy 2000 egyedet számláló tanítási-, és további 400 egyedet számláló validációs mintahalmazokat még augmentációs segédtechnikák alkalmazása mellett is 7 perc alatt feldolgozta, és 90%-os pontosságú modellt készített. Természetesen ennek elengedhetetlen előfeltétele volt a mintahalmaz kompozíciójának korábban részletezett megfontolások szerinti biztosítása.

Az eszköz bemeneti parametrizálása, illetve beállítási lehetőségei elégségesek voltak a kitűzött cél eléréséhez, miközben a pipeline kezelhetősége és minőségi metrikáinak értelmezhetősége egyáltalán nem akadályoztak, vagy lassítottak a munkában.

A végső produktum, azaz a Core ML-kompatibilis képklasszifikációs modell mérete mindösszesen 17KB lett, és minden olyan iPhone készülékre elérhető, mely legalább 12-es fő verziójú iOS rendszert futtat. A modell mérete a képességeit tekintve lényegesen kicsi, ami kifejezetten a Create ML kétfázisú, transfer learning-re épülő pipeline architektúrájának köszönhető. A feature extraction műveletéért felelős VisionFeaturePrint.Scene modell ugyanis az iOS operációs rendszer integrált része, így az nem került szállításra az elkészült modellel. A végleges .mlmodell fájl csak a klasszifikálásra betanított logisztikus regresszió modelljét tartalmazza. Ezzel egyidőben be kell látni, hogy az optimalizált modellszerkezet elvesztette portabilitását, és csak iOS operációs rendszer kontextusában futtatható, mely az általam kitűzött feladatspecifikációt nézve teljes mértékben elfogadható veszteség.

5. A mintaalkalmazás implementációja

A feladatspecifikációban célként kitűzött iOS alkalmazás magját adó, Core ML-alapú klasszifikációs modell betanítását követően rátértem az annak keretet adó szoftver implementálására. Az Xcode projektfájl létrehozása, a fordítási beállítások és célplatform konfigurálása után a projekt struktúra kiépítése, azaz a forrásfájlok szervezési elvének kialakítása következett. Az alkalmazás forráskód szintű architektúrájának, azaz a tervezési- és architekturális mintáknak a praktikus kiválasztásával a modell integrálásához szükséges keretrendszer alapjait fektettem le. Végül, de nem utolsó sorban a beágyazott modell predikcióit a Core ML framework által nyújtott programozói interfészen keresztül értem el és jelenítettem meg a felhasználó számára, bizonyos, az információk olvashatóságát növelő formázásokat követően.

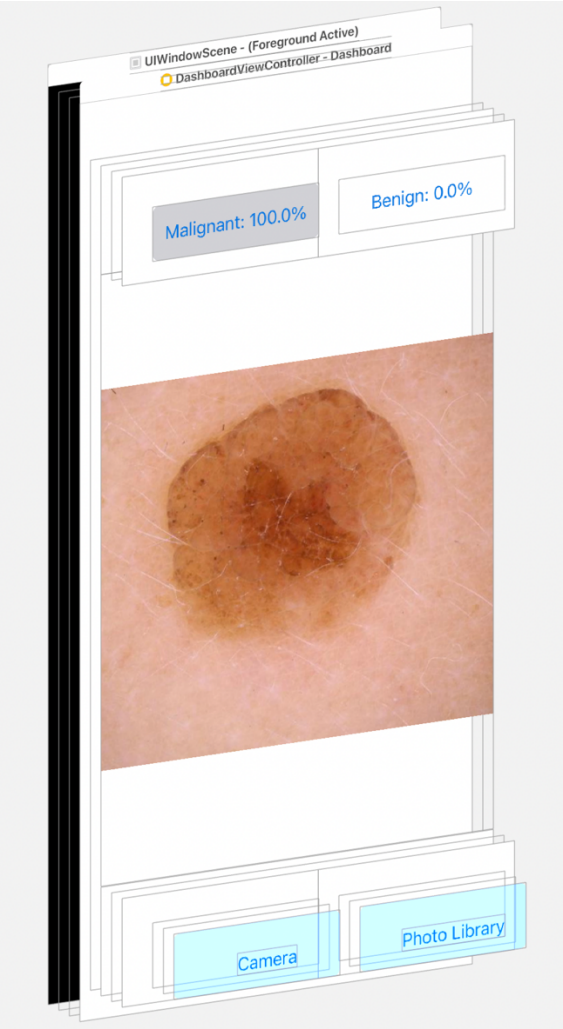
5.1. Projektstruktúra és szoftveres architektúra kialakítása

A fejlesztői környezet megválasztását és kiépítését egy korábbi fejezetben már taglaltam. A projektet Xcode 13 fejlesztői környezetben, Swift nyelv segítségével valósítottam meg iOS 15-öt futtató iPhone készülékek processzorarchitektúráját megcélozva. Bár az implementáció fókusza elsősorban a Core ML segédkönyvtár fejlesztői interfészének bemutatására irányul a képességek és kezelhetőség szempontjából, érdemes rövid kitérőt tenni az általános architektúra szervezőelemeire, illetve azok szerepére a szoftverkomponensek kommunikációja során.

A szoftver felhasználói felületének megtervezése és kivitelezése során elsősorban a használhatóságra, illetve az eredmények könnyű kiértékelhetőségére törekedtem. Az egyképernyős alkalmazás felületét a UIKit grafikus könyvtár segítségével valósítottam meg az Xcode grafikus felülettervezőjével és ún. storyboard fájlok felhasználásával. A storyboard az Xcode speciális, grafikus felületek kialakítására létrehozott fájlformátuma [30]. A képernyőn megjelenő nézetstruktúra kompozícióját, illetve a nézetelemek méreteit és egymáshoz való elhelyezkedését a storyboard fájlok vizuális elemkönyvtára, valamint az azokon értelmezhető ún. autolayout constraint-ek, azaz igazítási kényszerek felhasználásával alakítottam ki.

A végeredményként kapott felületet három részre osztottam. A képernyő közepén a felhasználó által kiválasztott, fotókönyvtárból vagy kamerából importált kép jelenik meg. Alatta a kamera és fotókönyvtár választási lehetőségeket biztosító vezérlőgombok, felette pedig a kiértékelés eredményét osztályonként százalékos formában megjelenítő

információs sáv címkéi láthatók. A megfelelő gomb megnyomása, majd a kép importálása és feldolgozása után a modell klasszifikációs predikciója a képernyő tetején, a magasabb százalék érteket kapott osztály kiemelésével jelenik meg. Az applikáció nézethierarchiájának felépítését a 8. ábra illusztrálja.



8. ábra: Az alkalmazás nézethierarchiája (saját szerkesztésű ábra)

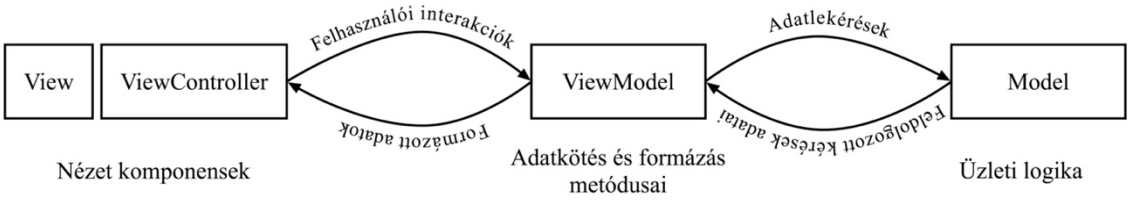
A képernyő kódbeli reprezentációja a `UIViewController` rendszerosztályból származó, `DashboardViewController` elnevezésű komponens lett, melyet a rendszer az egyedileg hozzárendelt azonosító alapján asszociált a kinézetet leíró, szintén Dashboard nevet viselő storyboard fájlhoz. Objektumorientált paradigmák mentén ezen osztály felelőssége referenciát tárolni a képernyőn megjelenő, fa struktúrába szervezett nézetarchitektúra gyökérelemére, az ún. gyökérnézetre, továbbá egységbe szervezni az ezen nézetek életciklusát végig követő delegát metódusok implementációját [31].

A nézetelemek mellett az alkalmazásnak számos más, releváns szereppel bíró komponense is van, melyeket a Model-View-ViewModel, azaz röviden MVVM egész projektet átívelő programarchitektúrája mentén alakítottam ki és kapcsoltam össze. Az MVVM neve valójában a felelősségi körök szerinti komponensszeparációból származik [28]. A korábban taglalt View, azaz nézet a felhasználó által látott összetevőket takarja [28]. A Model, mint azt neve is sugallja, a releváns üzleti logikákat szervezi egységbe [28]. Az alkalmazásban ez az ún. `DashboardModel` nevű protokollt, és az azzal konform `LesionClassifierDashboardModel` nevű osztályt jelenti. Ez kizárólagos, direkt kapcsolatban áll az alkalmazásba integrált Core ML klasszifikációs modellel, így a modellpredikciók egyedüli forrásaként szolgál a többi elem számára. A ViewModel, koncepciója szerint, a nézet és modellkomponenseket köti össze [28]. Egyfajta közvetítőként továbbítja a felhasználói interakciókból származó adatlekéréseket a Model irányába, valamint az elvárások szerint formázza és továbbítja a Model válaszait a View, azaz nézetkomponensek felé [28]. A programban ez a `LesionClassifierDashboardViewModel` osztályt, és az annak elvárt működését leíró `DashboardViewModel` protokollt takarja. A szoftver Xcode projektjét megnyitva azt láthatjuk, hogy a forrásfájlok és az azokat tartalmazó szervezőkönyvtárak a felelősségi körök szerint szétbontva, az arra utaló elnevezésekkel ellátva kerültek szétosztásra, ezáltal is elősegítve a projektstruktúra kereshetőségét és átláthatóságát.

Bár az egynézetes alkalmazás funkcionalitása nem követeli meg, a képernyő navigáció architektúráját az ún. Coordinator minta szerint készítettem elő a jelenlegi, valamint a későbbi, potenciális bővítések során hozzáadandó képernyők kezelésére. A Coordinator felelőssége a különböző képernyőkontextusok közötti váltások koordinálása, felügyelete, illetve az adott kontextushoz tartozó View, ViewModel, és Model típusú komponensek példányosítása és egymáshoz rendelése egy ún. `Coordinator` interfész sémájára [29]. Az ennek megfelelő `AppCoordinator` osztály az alkalmazás teljes kontextusát felügyeli, az applikáció betöltése után a `DashboardCoordinator` már előzőleg ismertetett, `DashboardViewController` elnevezésű, fő képernyőt reprezentáló kontextusát példányosítja és jeleníti meg a felhasználó részére.

A protokollok, vagy más néven interfészek az architektúra elemei közötti kommunikáció sémáját fektetik le. Azért is kulcsfontosságúak, mert segítségükkel átláthatjuk a szoftveres osztályok szolgáltatásait az elérhető műveletek, illetve az azok által szolgáltatott adatok pontos szignatúrájának formájában [32]. A későbbi, Vision

framework adaptálását ismertető fejezetben látni fogjuk, hogy a felelősségi körök ilyen szintű szeparációja lehetővé teszi a háttérben felhasznált technológiák lecserélését minimális, csupán az érintett komponenseket érintő változtatások mellett. Az MVVM architektúra, továbbá a komponens-specifikus protokollok bevezetésével az osztályok refaktorálhatóságát és az alkalmazás potenciális skálázhatóságát egyaránt tudtam biztosítani. A 9. ábrán az alkalmazás MVVM architektúrájának szemantikus felépítését láthatjuk.



9. ábra: MVVM architektúra (saját szerkesztésű ábra)

5.2. A modell szoftveres integrálása

A Create ML segítségével betanított és legenerált modellt a Core ML segédkönyvtár interfészén keresztül integráltam az alkalmazásba. A LesionClassifier.mlmodel nevű, Core ML-kompatibilis modellfájt az Xcode projekthez hozzáadva, majd az applikáció forráskódját lefordítva a modell használatra készen állt. Annak klasszifikációs képességeit a Core ML gépi tanulás-alapú segédkönyvtár programozói interfészén, valamint az Xcode beépített modellkezelő funkcióin keresztül értem el, miközben a fejlesztői környezet a modellfájlba csomagolt alacsony szintű algoritmusokat és azok betanult paramétereit teljes mértékben elrejtette, ezzel is optimalizálva a fejlesztés menetét.

5.2.1. A modell be- és kimeneti interfésze

A képklasszifikációs modell predikciós interfészét Xcode-ban megtekintve azt láthatjuk, hogy az a bemenetén 299x299 pixel felbontású színes képeket vár, melyeket a benign és malignant, azaz magyarul jóindulatú és malignáns osztálycímkék mentén képes szétválogatni. Kimenetén a `classLabelProbs` nevű, predefiniált osztálycímkék mentén indexelt asszociatív tömbben a bemeneti kép adott osztályokhoz tartozásának valószínűségi értékeit adja vissza lebegőpontos számok formájában. A szintén kimeneten elérhető `classLabel` karaktersztring pedig a legnagyobb valószínűségű osztálycímke elnevezését tartalmazza adott bemeneti kép tekintetében.

A modell felhasználása során tehát figyelembe kellett vennem, hogy az nem csupán egy osztálycímkét predikál, hanem technikailag az adott címkéjű osztályokba tartozás asszociatív tömbbe rendezett valószínűségi eloszlását számolja ki és adja vissza. Mivel bináris klasszifikációról van szó, a valószínűségi eloszlás jelen esetben kettő darab, 0 és 1 közötti pozitív számot jelent, melyek összege 1. Ezeket úgy is interpretálhatjuk, mint egy bizonyossági értéket, mellyel a modell jelzi számunkra, hogy adott kép adott osztályhoz tartozása mennyire valószínű a generált predikció alapján. Minél magasabb ez az érték adott osztályra nézve, annál bizonyosabb a modell a számolt predikcióban. Bináris klasszifikáció révén, a nagyobb valószínűséggel rendelkező osztályt kiválasztva megkaptam az elvárt diagnózist. Az eredmények kiértékelése során figyelembe vettem továbbá azt a tényt, hogy minél bizonytalanabb a modell, annál jobban közelítenek a valószínűségi eloszlás mérőszámai egymáshoz és az 50% körüli értékhez.

5.2.2. A Core ML segédkönyvtár használata

A LesionClassifier modell kódbeli eléréséhez szükséges forráskódokat az Xcode az első projektfordítás során legenerálta. Ennek centrális eleme a `LesionClassifier` nevű osztály, mely egy MLModel típusú változón keresztül az alkalmazásba csomagolt modell betöltéséért és eléréséért felelős [33].

Mindezt különböző szignatúrájú predikciós metódusokra támaszkodva valósítja meg. Ezek közül kiemelném a `LesionClassifierInput`-ot váró, és `LesionClassifierOutput`-ot visszaadó szignatúrájú predikciós függvényt, melynek be- és kimeneti paraméterei szintén a Core ML által absztrahált magas szintű segédosztályok:

```
func prediction(input: LesionClassifierInput,
                options: MLPredictionOptions) throws -> LesionClassifierOutput {
   let outFeatures = try model.prediction(from: input, options: options)
   return LesionClassifierOutput(features: outFeatures)
}
```

A Core ML rendszer belső implementációja szerint ún. CVPixelBuffer formátumú képeket kezel, melyek egy speciális, kCVPixelFormatType_32BGRA típusú képpuffer által alacsony szinten reprezentált, 299x299 felbontású színes pixelmátrixok [34]. A `LesionClassifier` osztály egy másik, általam közvetlenül alkalmazott predikciós kisegítőfüggvénye ilyen CVPixelBuffer típusú pufferelt képet vár bemenetén, melyet

aztán egy `LesionClassifierInput` típusú segédosztályba csomagolva továbbít feldolgozásra a korábban már említett predikciós függvény felé:

```
func prediction(image: CVPixelBuffer) throws -> LesionClassifierOutput {
    let input_ = LesionClassifierInput(image: image)
    return try self.prediction(input: input_)
}
```

Mind a `LesionClassifierInput`, mind a `LesionClassifierOutput` a Core ML MLFeatureProvider protokollja szerint operáló segédosztályok. Az input osztály elsődleges felelőssége a CVPixelBuffer típusú bemeneti kép segédmetódusokon keresztüli eltárolása és visszaadása, míg az output segédosztály a már korábban említett `classLabelProbs` és `classLabel` nevű kimeneti változókat fogja közre az objektumorientáltság szemantikáját követve [35].

A segédkönyvtár használatba vételéhez elengedhetetlen volt a modellspecifikus segédosztályok, valamint azok metódusainak tanulmányozása és pontos megértése. Az alkalmazásban a Core ML belső modelljével kizárólagosan a `LesionClassifierDashboardModel` osztály kommunikál. A modell típusú osztály `subject` nevű belső változójában eltárolja a felhasználó által kiválasztott, viewmodel által továbbított képet, majd szükség esetén lekéri és visszaadja az elvárt predikciókat a belső, `lesionClassifier` nevű modellreferenciáját igénybe véve. A képek kiválasztásához felhasznált, UIImagePickerController típusú komponens a képeket ún. UIImage formátumban adja vissza a kamerából és fotókönyvtárból egyaránt [36].

Míg ezen formátum testhezálló a képek kijelzőn való megjelenítéséhez, addig a `LesionClassifierDashboardModel` osztály ezt minden predikciós hívás előtt átalakítja a klasszifikációs modell számára is értelmezhető CVPixelBuffer pixelpufferré az elvárt bemeneti követelmények és kényszerek alapján. A `getPixelBufferRepresentation` nevű függvény figyelembe veszi a tanulási modell bemeneti előírásait, majd visszatér az MLFeatureValue objektumba csomagolt kép puffer-alapú reprezentációjával:

```
func getPixelBufferRepresentation(of image: UIImage) -> CVPixelBuffer? {
    guard let cgImage = image.cgImage,
          let inputDescription = lesionClassifier.model
                                                 .modelDescription
                                                 .inputDescriptionsByName["image"],
          let modelImageConstraint = inputDescription.imageConstraint else {
```

```
        return nil
    }

    let options: [MLFeatureValue.ImageOption: Any] = [
        .cropAndScale: VNImageCropAndScaleOption.centerCrop.rawValue
    ]

    let feature = try? MLFeatureValue(cgImage: cgImage,
                                      constraint: modelImageConstraint,
                                      options: options)

    return feature?.imageBufferValue
}
```

Mivel a klasszifikáció művelete, így a valószínűségi eloszlás előállítása hosszabb időt is igénybe vehet, a klasszifikációs függvényhívást aszinkron módon, magas prioritású háttérszálon indítom, majd a kiszámolt eredményeket egy Swift closure, azaz Swift nyelvben értelmezett magas szintű függvény pointer segítségével továbbítom feldolgozásra a `generateClassification` metódusban:

```
func generateClassification(
    for image: UIImage?,
    completionHandler: @escaping (ClassificationInfo) -> Void
) {
    DispatchQueue.global(qos: .userInitiated).async { [weak self] in
      if let self = self,
         let image = image,
         let pixelBuffer = self.getPixelBufferRepresentation(of: image),
         let prediction = try? self.lesionClassifier.prediction(image: pixelBuffer) {
            completionHandler(prediction.classLabelProbs)
         } else {
            completionHandler([:])
         }
    }
}
```

A closure használata biztosítja, hogy az esetlegesen elhúzódó modelloperáció időközben ne blokkolja az alkalmazás fő szálának futását. Az aszinkron függvények visszatérési értékeinek closure segítségével történő processzálását completion handler mintának nevezzük [37]. Nevét a visszatérésre felhasznált, függvényparaméterként átadott, és az aszinkron függvény törzsében kiszámolt értékekkel felparametrizálva meghívott closure-ről, azaz a completion handler-ről kapta [37].

Az applikáció működtetése közben az adatok áramlási irányát, illetve azok komponens-specifikus formátumait az MVVM architektúra keretrendszere előre lefekteti. A felhasználó a `DashboardViewController` vezérlőgombjainak segítségével tudja kiválasztani az általa preferált képforrásból az analizálásra szánt, UIImage formátumú képet, melyet a `LesionClassifierDashboardViewModel` kisegítő metódusai juttatnak el a `LesionClassifierDashboardModel`-nek tárolásra.

A képernyő frissítése során a viewcontroller osztály lekéri, majd beállítja háttérének a modell aktuálisan tárolt képét, aztán triggereli a viewmodel-en keresztül a klasszifikációs operációt. Mivel a művelet aszinkron módon, háttérszálon fut, így a visszatérési értékeket mind a viewcontroller, mind a köztes kommunikációért felelős viewmodel closure formájában várja. A `LesionClassifierDashboardModel` osztály a generált `LesionClassifierOutput` típusú predikciót, egészen pontosan annak `classLabelProbs` nevű asszociatív tömbjét completion handler segítségével adja át a `LesionClassifierDashboardViewModel`-nek a korábban részletezett `generateClassification` metódus törzsében.

A viewmodel a megkapott valószínűségi eloszlást a felhasználó számára értelmezhető adatreprezentációvá alakítja, majd továbbítja a viewcontroller-nek a `fetchClassificationInfo` elnevezésű eljárásában definiált closure szerint:

```
func fetchClassificationInfo(
    completionHandler: @escaping (FormattedClassificationInfo) -> Void
) {
    model.classifyCurrentSubject { classifications in
        var formattedResults = FormattedClassificationInfo()

        var benignLabelValue: String = "Benign: n/a"
        var shouldHighlightBenign: Bool = false
        var malignantLabelValue: String = "Malignant: n/a"
        var shouldHighlightMalignant: Bool = false

        if let benignProbability = classifications["benign"],
           let malignantProbability = classifications["malignant"] {
            benignLabelValue =
              Self.formatResultsLabelValue(for: LesionClassificationType.benign,
                                           using: benignProbability)
            malignantLabelValue =
              Self.formatResultsLabelValue(for: LesionClassificationType.malignant,
                                           using: malignantProbability)

            if benignProbability > malignantProbability {
                shouldHighlightBenign = true
```

```
                } else {
                    shouldHighlightMalignant = true
                }
        }

        formattedResults["benign"] = (formattedInfo: benignLabelValue,
                                      shouldHighlight: shouldHighlightBenign)
        formattedResults["malignant"] = (formattedInfo: malignantLabelValue,
                                         shouldHighlight: shouldHighlightMalignant)

        completionHandler(formattedResults)
    }
}
```

A closure belsejében a címkék szövegének formázott verzióját a klasszifikáció típusa, valamint a hozzátartozó megbízhatósági valószínűség alapján a `formatResultsLabelValue` nevű függvény állítja elő:

```
static func formatResultsLabelValue(for type: LesionClassificationType,
                                    using probability: Double) -> String {
    String(format: "%@ %.1f%%",
           type == .benign ? "Benign:" : "Malignant:",
           probability * 100)
}
```

A formatált szövegek, illetve a magasabb predikciós valószínűséggel rendelkező osztály kiemelésére vonatkozó információk a `FormattedClassificationInfo` elnevezésű, tuple típusú adatszerkezeten keresztül kerülnek továbbításra a nézetkomponens felé. Végül, a `DashboardViewController` a megkapott szöveges adatokat a megjelenítésre szánt címkékhez hozzárendeli, és az alkalmazás fő szálán frissíti a képernyőt a `classificationHandler` változóban tárolt completion handler alapján:

```
lazy var classificationHandler = { [weak self] (results: FormattedClassificationInfo)
in
    DispatchQueue.main.async {
       guard let self = self else { return }

       if let benignInformation = results["benign"] {
          self.benignPredictionLabel.text = benignInformation.formattedInfo
          self.benignPredictionLabel.backgroundColor =
                benignInformation.shouldHighlight ? .systemGray4 : .white
       }

       if let malignantInformation = results["malignant"] {
```

```
            self.malignantPredictionLabel.text = malignantInformation.formattedInfo
            self.malignantPredictionLabel.backgroundColor =
                  malignantInformation.shouldHighlight ? .systemGray4 : .white
        }
    }
}
```

Végeredményben a View, Model és ViewModel komponenseknek megfelelő osztályok a saját felelősségi körük alapján, a működésüket leíró protokollok sémájára operálnak, méghozzá a felhasználási szint szerinti adatreprezentációk szerint. A viewcontroller nézetosztály képi (UIImage) és karaktersztring (String) adatokat kap és jelenít meg, a modell az alacsony szintű CVPixelBuffer képreprezentációk mentén asszociatív tömbben (Dictionary) tárolt valószínűségi eloszlásokat nyer ki belső klasszifikációs modelljéből, míg az adattranszformációs rétegnek megfelelő viewmodel ezen asszociatív tömb elemeit kapja meg és transzformálja karaktersztringekké a nézetek számára.

5.3. Vision framework adaptálása

A Core ML alacsony szintű szoftveres absztrakciói mentén, az MLModel és kisegítő osztályainak interfészéből kiindulva sikeresen integráltam a klasszifikációs modellt applikációmba. A Core ML programozói interfésze azonban túlságosan modellspecifikus ahhoz, hogy effektíven skálázható legyen. Ezért következő lépésként az Apple által biztosított Vision framework magasabb absztrakciós szintű képfeldolgozó interfészét adaptáltam, mely a kód jelenlegi állapotának némi refaktorálását követelte meg.

A Vision segédkönyvtár számos képkezelési optimalizációt tartalmaz eszköztárában, mely a Core ML modell működtetése során elrejti annak alacsony szintű működését és adatstruktúráit. A kamera és fotókönyvtár UIImage formátumú képeit szükség szerint átméretezi és konvertálja, mindezt a modell által használt CVPixelBuffer teljes elfedése mellett. Míg a Core ML központi eleme az MLModel osztály, addig a Vision feladatleíró absztrakciók, ún. VNCoreMLRequest objektumok, valamint az azokat elindító VNImageRequestHandler osztályok köré szerveződik. [38]

5.3.1. VNCoreMLRequest konfigurálása

Az MVVM architektúrából eredendően az egyetlen alkalmazáskomponens, mely a Core ML modellel eddig közvetlenül kommunikált, a

`LesionClassifierDashboardModel` osztály volt. Ennek megfelelően a következő módosításokat ezen osztály belső implementációjában hajtottam végre a Vision framework funkcionalitásának integrálásához. A korábbi segédfüggvények kitörlése után `imageAnalysisRequest` néven létrehoztam a modell saját VNCoreMLRequest referenciáját:

```
lazy var imageAnalysisRequest: VNCoreMLRequest = {
    do {
        let lesionClassifier = try LesionClassifier(configuration: .init())
        let visionCoreMLModel = try VNCoreMLModel(for: lesionClassifier.model)

        let analysisRequest = VNCoreMLRequest(model: visionCoreMLModel,
                                completionHandler: { [weak self] request, error in
          guard let self = self,
                let requestHandler = self.imageAnalysisRequestHandler else {
              return
          }
              requestHandler(request, error)
          })

          analysisRequest.imageCropAndScaleOption = .centerCrop

          return analysisRequest
    } catch {
          fatalError("Error message: \(error)")
    }
}()
```

A feladatleíró komponens genericitása, valamint annak példányosításához szükséges erőforrásigények figyelembevételével az objektum újra hasznosítása mellett döntöttem, így a klasszifikációs kérések során minden képet ezen VNRequest alosztályon keresztül dolgoztam fel. A VNCoreMLRequest működéséhez elengedhetetlen modellinformációkat, annak inicializálása során, egy ún. VNCoreMLModel kapcsolóobjektum segítségével biztosítottam [38].

Releváns eltérés, hogy a Core ML modelljének programozói interfésze szinkron függvényhívásokat használ, így az alkalmazás reszponzivitásának megőrzése érdekében a közvetlen modellpredikciós hívásokat egy dedikált háttérszálra átvezetve, aszinkron módon kellett meghívnom. A VNCoreMLRequest ezzel szemben eleve completion handler minta alapján dolgozik, és az inicializáláskor átadott closure alapján továbbítja a kérések eredményeit [38].

Szintén a VNCoreMLRequest belső paramétere az ún. `imageCropAndScaleOption` változó [39]. Ahhoz ugyanis, hogy a kérés elfogadhatóan tudja végrehajtani a bemeneti képek automatizált konverzióját CVPixelBuffer formátumra, specifikálni szükséges, hogy a képek transzformálása során milyen aspektusmegőrzési heurisztikát használjon a rendszer. Erre alapvetően három lehetőség áll rendelkezésre, melyeket a 10. ábra szemléltet:

- scaleFill
- scaleFit
- centerCrop. [39]

A scaleFill 299x299 méretű négyzetté alakítja a bemeneti képet, ám nem őrzi meg annak eredeti méretarányait, így téglalap alakú képek esetén azok a hosszabb dimenzió mentén összenyomva, torzítva jelennek meg [39]. A scaleFit ugyan megőrzi a méretarányokat, viszont torzítás helyett átlátszó pixelekkel tölti ki a hiányzó képrészleteket a lekicsinyített kép pixelmátrixában [39]. A bőrelváltozások színe és formája olyan kritikus adatpontok, melyek túlságosan torzulhatnak ezen képtranszformációs heurisztikák mellékhatásaként, így azokat ki kellett zárnom.

10. ábra: Vision framework 'crop and scale' transzformációi [39]

A centerCrop ezzel szemben először lekicsinyíti a bemeneti képet a rövidebb oldala mentén 299 pixel méretűre, majd kivágja az így keletkezett kép középső, 299x299 pixel méretű négyzetes részét [39]. Szélesebb képek esetén azok bal- és jobb széle, míg magasabb képek esetében azok felső- és alsó széle fog elveszni a transzformáció végén. Figyelembe véve, hogy a modell betanítása során felhasznált dermatoszkopikus képek fókuszában eleve a vizsgált bőrelváltozás állt, arra a következtetésre jutottam, hogy a centerCrop transzformációs heurisztika alkalmazása, bár némi információ veszteséggel jár, nem fogja jelentős mértékben befolyásolni a klasszifikációs modell működését.

Cserébe az applikáció használata és tesztelése során esszenciális elvárás a vizsgált bőrfelület fókuszpontban tartása, különösen kameraképek készítése közben.

5.3.2. VNImageRequestHandler-ek használata

Míg az `imageAnalysisRequest` változó a `LesionClassifierDashboardModel` egyszer példányosított, a képek feldolgozási kéréseit automatizáló objektuma, addig ezen klasszifikációs kéréseket a képenként külön-külön inicializált VNImageRequestHandler típusú objektumokon keresztül indítottam el.

Számítva arra, hogy a modell leterhelt memória- vagy processzorkapacitás esetén megcsúszhat a predikciós válaszok generálásával, a képek osztályozását végző `generateClassification` függvény belsejében helyileg példányosított VNImageRequestHandler klasszifikációs kérését explicit háttérszálon futtattam:

```
func generateClassification(for image: UIImage?) {
   guard let image = image, let transformedImage = image.cgImage else { return }

   let imageOrientation = CGImagePropertyOrientation(
                              rawValue: map(image.imageOrientation).rawValue
                           ) ?? .up

   DispatchQueue.global(qos: .userInitiated).async {
      let imageRequestHandler = VNImageRequestHandler(cgImage: transformedImage,
                                                      orientation: imageOrientation)
      do {
         try imageRequestHandler.perform([self.imageAnalysisRequest])
      } catch {
         let error = GeneralError(message: "Failed to perform classificationRequest",
                                  domain: .operationFailed)
         AppLogger.log(
            message: "Classification failed: \(error.localizedDescription)",
            event: .error
         )
         return
      }
   }
}
```

Erre az MLModel predikciós függvényének végrehajtása esetében felmerült, korábban már részletezett megfontolások miatt jelen esetben is igény volt. A VNImageRequestHandler létrehozásához, majd a VNCoreMLRequest ezen keresztüli elindításához szükség volt a kép UIImage reprezentációjának Core Graphics framework

szerinti, egy szinttel alacsonyabb CGImage formátumra konvertálására, továbbá a kép orientációjának specifikálására [38].

A képorientáció olyan bemeneti paraméter, mely alapján a Vision framework döntéseket hoz az esetleges rotációs transzformációk szükségességével kapcsolatban a VNCoreMLRequest automatizált, pixelmátrixra vonatkozó átalakítási eljárásai során [38]. A VNImageRequestHandler tehát az adott kép CGImage reprezentációja, valamint annak orientációja szerint példányosított objektum. Minden kép esetén újabb handler objektum példányosítása szükséges, még az arra vonatkozó klasszifikációs kérés elindítása előtt. A mintaalkalmazás szempontjából kiemelendő, hogy a bemeneti képek orientációja nincs hatással a kimenet klasszifikációs pontosságára, sőt, modellgenerálás során explicit rotációs augmentációkat is igénybe vettem a modell validációs pontosságának növelése érdekében.

5.3.3. A refaktorálás eredménye

A Vision framework bevezetése elsősorban a Model réteget, valamint a Model és ViewModel közötti kommunikációs protokollokat érintette. Az eredményeket immár VNClassificationObservation típusú objektumok tömbjén keresztül értem el és alakítottam olvasható szöveggé a `LesionClassifierDashboardViewModel` adatátalakító closure-jének magjában. A VNCoreMLRequest és VNImageRequestHandler absztrakcióinak köszönhetően sikerült a kódot eltávolítani a gépi modell programozási interfészének megkötéseitől, ezáltal pedig hosszú távon skálázhatóvá tenni.

A helyesen megválasztott architektúra létjogosultságát az is erősíti, hogy az újabb segédkönyvtár bevezetése csak és kizárólag az érintett programkomponenseket, valamint azok kommunikációját érintette az új adatreprezentációk mentén.

6. Az alkalmazás tesztelése

Modellgenerálás során számos validációs és verifikációs metrikát vettem figyelembe a különböző modellverziók tesztelése és továbbfejlesztése céljából. A modellverifikációs eljárások mellett az elkészült mobilalkalmazást is teszteltem, méghozzá manuális funkcionális tesztesetek kidolgozásával és végrehajtásával.

A gyakorlatban ez olyan tesztképhalmaz összeállítását és klasszifikálását jelentette, melyet a modell betanítása során nem használtam. A malignáns-jóindulatú képosztályok a Lesions_ManualTesting mappán belüli benign (jóindulatú) és malignant (malignáns) almappákba kiválogatott 10-10 kép alapján állnak össze, melyeket a már korábban ismertetett ISIC képarchívumból válogattam ki e célra. A 11. ábrán a verifikáció során felhasznált jóindulatú bőrelváltozások mintaképeit láthatjuk.

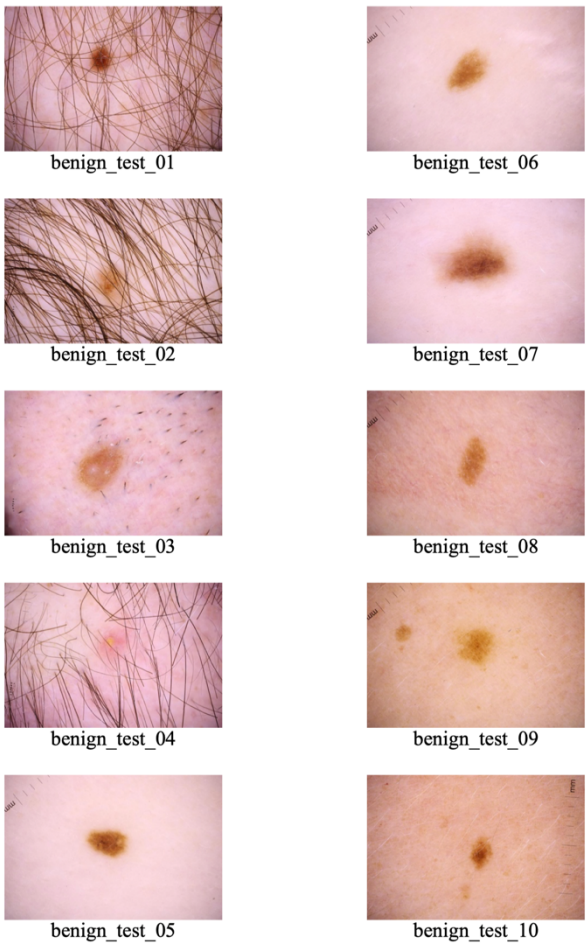
Mivel az alkalmazás a fotókönyvtárból importált képek, valamint kamerával készített fotók alapján is képes diagnózis felállítására, így a képek és fotók analízisét külön tesztesetnek vettem. Az 1. táblázat a jóindulatú képeken végrehajtott tesztesetek eredményeit mutatja.

1. táblázat: Jóindulatú mintaképekre vonatkozó manuális tesztesetek eredményei

	Fotókönyvtárból importálva		***Kamerával készítve***	
Képazonosító	***Predikció***	***Konfidencia***	***Predikció***	***Konfidencia***
benign_test_01	jóindulatú	100%	jóindulatú	100%
benign_test_02	jóindulatú	100%	jóindulatú	100%
benign_test_03	**malignáns**	**96,5%**	**malignáns**	**50,7%**
benign_test_04	jóindulatú	100%	jóindulatú	100%
benign_test_05	jóindulatú	100%	jóindulatú	99,4%
benign_test_06	jóindulatú	100%	jóindulatú	100%
benign_test_07	jóindulatú	100%	jóindulatú	91,1%
benign_test_08	jóindulatú	59,8%	jóindulatú	98,7%
benign_test_09	jóindulatú	100%	jóindulatú	74,2%
benign_test_10	jóindulatú	99,6%	jóindulatú	100%

A tesztek elvégzéséhez egy iPhone 8 Plus típusú, iOS 15-ös rendszerverziójú telefonkészüléket használtam. Az egyik teszteset mentén a kijelölt képeket elmentettem az eszköz fotókönyvtárába, majd onnan azokat a futó alkalmazásba importáltam. A másik

teszteset során a kiválasztott képeket a telefonkészülék kamerájával fotóztam be kiértékelésre.



11. ábra: Jóindulatú bőrelváltozások egyedi tesztazonosítóval ellátott verifikációs képhalmaza (saját szerkesztésű ábra)

A táblázatot áttekintve azt látjuk, hogy a képek forrása nem befolyásolja a klasszifikáció pontosságát a mintahalmaz kontextusában: 10-ből 9 predikció mindkét teszteset során megfelelő volt, ami 90%-os verifikációs pontosságot jelent. Ez megegyezik a felhasznált modellverzió automatizált verifikációs tesztje során kapott 90%-os testing accuracy mérőszámával, és egyben arra is rámutat, hogy a Vision framework integrálása során alkalmazott automatikus képkonverziós eljárások nem rontottak az alkalmazás várható predikciós képességein.

A helyes predikciók halmazát tekintve azonban észrevehető, hogy a kamerával befotózott képek konfidenciaértéke valamivel alacsonyabb tendenciát mutat: a 3-as és 9-es tesztfotók esetén rendre 50,7% és 74,2% konfidenciát kapunk, míg az 5, 7 és 8-as

fotókat az applikáció már 90% feletti megbízhatósággal klasszifikálja. A változó fényviszonyok és kameraállások mellett ez betudható annak is, hogy a telefonkészülék kamerája nem rendelkezik olyan optikai képalkotó képességekkel, mint a modell betanításához használt képek dermatoszkópos képalkotó eljárása.

A 3-as sorszámú félrediagnosztizált kép esetén érdekes megfigyelni, hogy a kamerával készített elemzés során a konfidencia jelentősen csökkent, így immáron 50% közeli bizonyossággal állította a modell, hogy malignáns képet lát. Bár az applikáció gyakorlatilag nem tudta meghatározni az elvárt képosztályt, és továbbra is hamis negatív predikcióval szolgált, annak konfidencia értéke azt sugallja, hogy ugyanezen kép fotókönyvtárból importált verziójához képest a klasszifikáció igazságértéke az osztálycímke szerint elvárt irányba javult a kamera igénybevételével.

A következő lépés a malignáns képek teszthalmazának ugyanezen esetek mentén történő kielemzése volt. A 12. ábra a rosszindulatú elváltozások verifikációs célra kiválogatott mintaképeit szemlélteti, a tesztesetek eredményeit pedig a 2. táblázat mutatja.

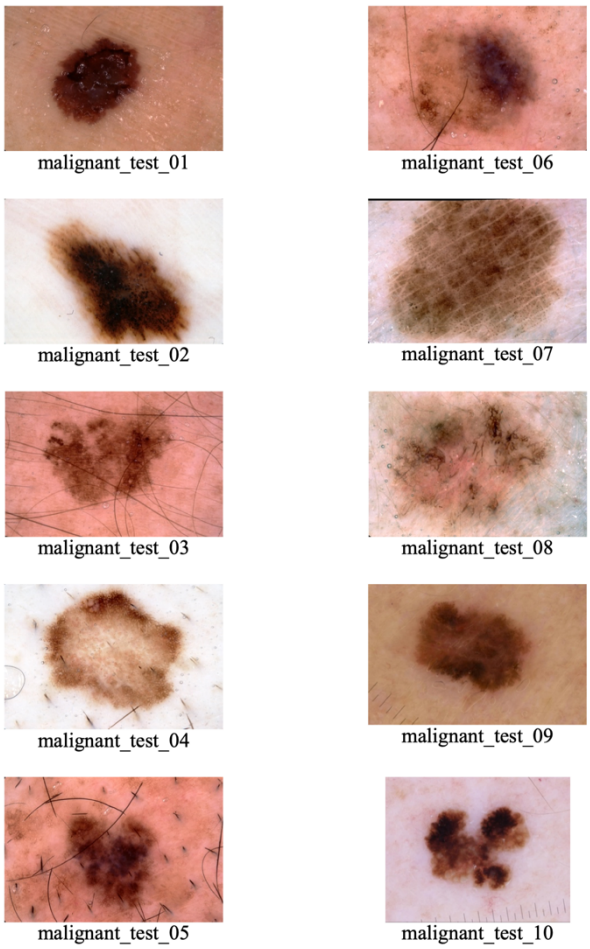
2. táblázat: Malignáns mintaképekre vonatkozó manuális tesztesetek eredményei

	Fotókönyvtárból importálva		***Kamerával készítve***	
Képazonosító	***Predikció***	***Konfidencia***	***Predikció***	***Konfidencia***
malignant_test_01	malignáns	100%	malignáns	100%
malignant_test_02	malignáns	100%	malignáns	100%
malignant_test_03	malignáns	100%	malignáns	100%
malignant_test_04	malignáns	100%	malignáns	100%
malignant_test_05	**jóindulatú**	**90%**	malignáns	100%
malignant_test_06	malignáns	100%	malignáns	100%
malignant_test_07	**jóindulatú**	**57,4%**	malignáns	100%
malignant_test_08	malignáns	100%	malignáns	100%
malignant_test_09	malignáns	100%	malignáns	100%
malignant_test_10	malignáns	100%	malignáns	100%

A teszteset mérőszámain végig tekintve láthatjuk, hogy a kameraképek alapján végrehajtott klasszifikációk mindegyike az elvártnak megfelelő, azaz 100%-os pontosságot kapunk. Ezzel szemben a fotókönyvtár felhasználása során a predikciós pontosság 80%-os értéket mutat. Az applikáció 10-ből 2 esetben hamis negatív predikciót generált, ami az automatizált modelltesztekhez képest 10%-os romlást indikál. Eközben

a konfidencia értéke majdnem minden esetben 100% körüli, tehát a pozitív predikciókat az alkalmazás mindenféle bizonytalanság nélkül állapítja meg.

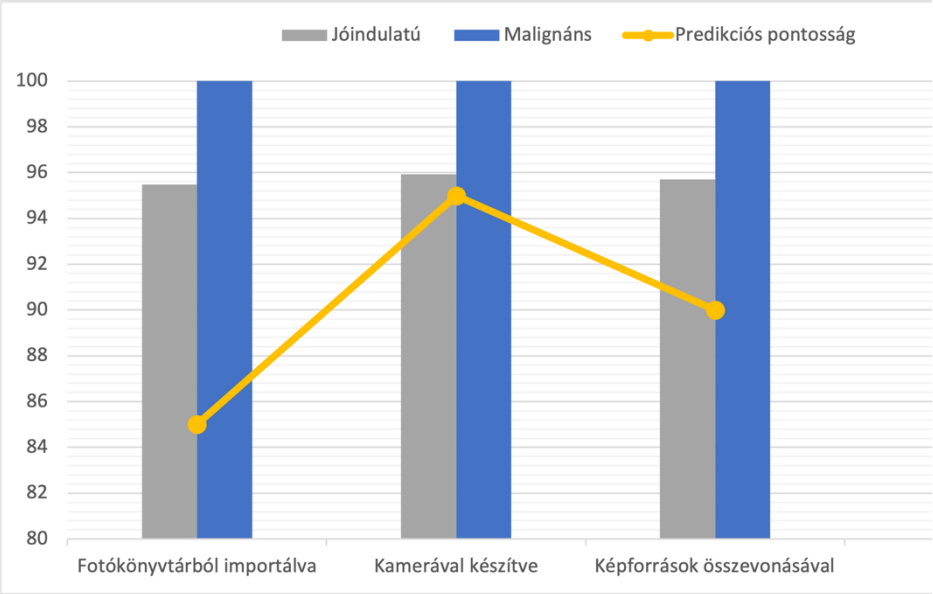
A mintaképek kontextusában ezt interpretálhatjuk úgy, hogy a modell a malignáns elváltozások beazonosításában közel hasonló pontosságot mutat, mint jóindulatú esetekben, ám ezen predikciókat magasabb konfidenciával állítja elő. A két teszteset közötti differencia elsősorban a kameraképeknél mutatkozik meg. A modell a jóindulatú elváltozásokat, bár megtalálja, magasabb bizonytalanságot mutat. Ez indikátora lehet a korábban már megemlített, a kameraképek minőségéből eredő pixelzajoknak, melyek a modellt befolyásolva az eredményt bizonyos mértékig a malignáns irányba torzítják.



12. ábra: Malignáns bőrelváltozások egyedi tesztazonosítóval ellátott verifikációs képhalmaza (saját szerkesztésű ábra)

A fotókönyvtárból importált képek átlagos predikciós pontossága 85%, a kameraképeké 95%. A helyesen klasszifikált malignáns képeket mindkét tesztesetben 100%-os átlagos konfidenciával azonosította a rendszer, a fotókönyvtárból és kamerából

származó jóindulatú képek vonatkozásában ezen érték rendre 95,48% és 95,93% lett. A kamera és fotókönyvtár képforrások osztályonként összevont tesztesetei alapján a helyes predikciók átlagos konfidenciája a jóindulatú osztály vonatkozásában 95,71%, ugyanezen érték malignáns esetben pedig 100%. Az átlagos predikciós pontosság 90%-ot mutat, amennyiben a képforrások között nem teszünk különbséget. A számolt statisztikákat a 13. ábra illusztrálja.



13. ábra: Helyes predikciók átlagos konfidenciaértékei a precíziós pontosság viszonylatában (saját szerkesztésű ábra)

Összességében tehát kimondhatjuk, hogy a manuális tesztek eredményei a modelltanítás során használt automatizált verifikáció minőségi metrikáit tükrözik. Míg a Vision framework pixeltranszformációra vonatkozó automatizálási eljárásai nem befolyásolták az eredmény minőségét, addig a kameraképek esetleges fényviszonyai, illetve a képalkotásra használt kamera korábban említett pixelzaja már megjelennek az eredmények konfidencia értékeiben. Első ránézésre a malignáns predikciókra vonatkozó konfidencia konzisztens 100%-os értéke azt jelzi, hogy a malignáns képeket könnyebben beazonosítja az applikáció. Mindeközben arra is felhívja a figyelmet, hogy jóindulatú elváltozások bizonyos jegyei, illetve a kamerából származó pixelzajok hatására az applikáció hajlamos lehet malignáns irányban elfogult predikciók generálására.

7. Továbbfejlesztési lehetőségek

A bináris klasszifikációs modell korábbi fejezetekben ismertetett módszerek és megfontolások mentén történő elkészítése, majd annak Core ML és Vision szoftverkönyvtárakkal való integrálása komplex munkafolyamat. A célplatform és a Core ML által támasztott technikai követelmények pre-determinálják a kitűzött célfeladat magját adó modell betanítási kontextusát, a felhasználható segédeszközöket és módszereket.

A modell minőségét garantáló mintahalmaz egyedeinek felkutatása és előkészítése, a rendelkezésre álló források limitációi miatt, időigényes és behatárolt folyamat volt. A bőrelváltozásokról verifikált diagnózissal rendelkező, publikusan elérhető képek száma korlátos, így az adatgyűjtés és data curation koncepciói mellett a következőkben olyan módszerekkel és eszközökkel is foglalkozok, melyek az elérni kívánt modell működését technikai megközelítésből finomítják.

7.1. Konvolúciós modellek feature extractor szerepben

A transfer learning heurisztikája nagy mértékben elősegítette a modellgenerálás felgyorsítását és optimalizálását. Ennek keretében a jól megválasztott, kis méretű, célorientált és gyors számítási kapacitású konvolúciós modellek hozzájárulhatnak a modell potenciális finomításához [41]. A VisionFeaturePrint.Scene mellett a MobileNet vagy SqueezeNet konvolúciós modelljeit szintén lehet feature extractor szerepkörben kamatoztatni, ám figyelembe kell venni, hogy a végeredményként kapott modell mérete, gyorsasága, pontossága és alapvető technikai paraméterei eltérőek lesznek [42].

A VisionFeaturePrint.Scene egy viszonylag nagy méretű konvolúciós modell, mely 299x299-es felbontású képekből 2048 dimenziójú feature vektorokat nyer ki [20], [46]. Mivel az iOS operációs rendszer része, így nem kerül szállításra a transfer learning pipeline részeként, és nem kompatibilis más platformokkal. A felhasználásával generált, kb. 17KB méretű végmodell mindeközben 90%-os klasszifikációs precizitást mutatott verifikáció során.

A SqueezeNet egy kisebb méretű, memória-optimalizált konvolúciós modellarchitektúra, mely relatíve gyors számítási kapacitással rendelkezik. Kézenfekvő megoldást kínál, amennyiben sebességoptimalizálás a cél. Ezen gyorsaságnak azonban

ára van. A 227x227 pixelen dolgozó modell sokkal kevesebb megtanult paramétert szállít, ennélfogva jelentősen kisebb pontosságot mutat működés közben. [42]

Amennyiben a sebesség optimalizálása mellett az elkészített modell pontossága és portabilitása is elvárás, akkor a MobileNet értelemszerű választás. A 224x224 méretű képbemeneteket váró modellarchitektúrának több tanult paramétere van, illetve belső optimalizációknak köszönhetően gyorsabb is, mint a SqueezeNet. Ellenben sokkal nagyobb méretű és több memóriát foglal, mely mobilos környezetben nem mindig elfogadható erőforrás veszteség. [42-43]

Természetesen számos más, a feladatra alkalmas modellarchitektúra létezik még. A legújabbak közül említésre méltóak az EfficientNet modellverziói, melyek a bemeneti felbontás, illetve a neurális háló szélessége és mélysége mentén összetett módon skálázhatók. [42]

Mint láthatjuk, az igénybe vett modell precizitása, mérete, memóriaszükséglete és gyorsasága olyan tényezők, melyeket együttesen mérlegelve tudjuk a testhezálló feature extractor modellt kiválasztani. A meglévő transfer learning pipeline koncepcióját számos konvolúciós modell mentén lehet finomítani, ám ehhez szükséges ezek technikai jellemzőinek előzetes felkutatása és megismerése, továbbá olyan modellgeneráló eszközök használata, melyek képesek a kiválasztott modellarchitektúrák kezelésére.

7.2. Object detection

Az eddigiekben a bináris klasszifikációra alkalmas transfer learning pipeline feature extractor modellarchitektúrájának potenciális finom hangolását részleteztem. A bőrelváltozások analízise során azonban nem csupán klasszifikációs módszereket lehet alkalmazni. A diagnózis felállításában az is segíthet, ha a bemeneti képek bőrfelületein az elvátozásokat mutató régiókat beazonosítjuk, majd azokat egyenként klasszifikáljuk.

Az ún. object detection modellek pont így operálnak. Kimenetükön a kép egy adott régióját befogó téglalap adatai, valamint az adott régióban detektált tárgy klasszifikációja jelenik meg egy százalékos konfidencia érték kíséretében. Ezen konfidencia indikálja, hogy a modell mennyire biztos a detektált régióban, illetve az ahhoz rendelt klasszifikációs kategóriában. A transfer learning pipeline architektúrája ez esetben annyiban változik, hogy a feature extractor konvolúciós modellje mellé egy, a régiók detektálására alkalmas modellréteg is beépítésre kerül. [44]

Az object detection sokat finomíthat a modell működésén azáltal, hogy kiemeli a bőrfelületek vizsgálandó részleteit, ezáltal kiszűrve a többi képrégióból származó esetleges pixelzajt. Hátránya viszont az, hogy betanítás során a detektor modellnek immáron nem csak osztálycímkéket, hanem a mintaképeken látható bőrelváltozások régióit körül határoló téglalapok adatait is meg kell határozni, méghozzá olyan formátumban, amit a betanításra használt segédeszköz is értelmezni tud [44]. Amennyiben ezen adatok nem állnak rendelkezésre a mintához csatolt metaadatként, úgy azok manuális elkészítése szignifikánsan növeli a data curation folyamatának komplexitását és idejét.

8. Összegzés

A dolgozat célkitűzésében szereplő, bőrelváltozások bináris klasszifikációjára alkalmas gépi tanulási modell iOS platformra optimalizált változatának elkészítése, majd annak Core ML framework segítségével történő integrálása olyan megközelítéseket és módszereket követelt, melyek alapján a mobilos környezetben ismeretes erőforrás korlátok mellett is megbízható pontosságú és hatékonyságú applikációt tudtam készíteni.

A mintaadatok begyűjtése és felcímkézése, majd az elkészített mintahalmazok adott problémakör kontextusában meghatározott adatpontok mentén történő előkészítése és kiválogatása időigényes és komplex munkafolyamatnak bizonyult. A mintahalmaz, így egyben a klasszifikáció várható pontosságát a probléma elemzése során kiválasztott adatpontok szerint biztosítottam, figyelembe véve a rendelkezésre álló adatok mennyiségi és minőségi limitációit.

A modell betanítására felhasznált céleszköz és heurisztikák megválasztása során a célarchitektúra, valamint az arra elérhető szoftveres segédkönyvtárak által támasztott technikai követelményeket vettem figyelembe. A VisionFeaturePrint.Scene konvolúciós modell képességeit feature extractor szerepkörben hasznosítva, majd azt egy logisztikai regressziós modellel kaszkádosítva olyan transfer learning pipeline-t alkalmaztam az e célra megválasztott Create ML eszköz segítségével, mely a megoldás képességeihez képest elenyésző méretű, és relatíve nagy pontosságú modellt generált. A végleges modellverzió eléréséhez számos validációs és verifikációs modellmetrikát vettem figyelembe, melyek mentén a kívánt pontosságot modell- és osztályszinten egyaránt tudtam finomítani. Az overfitting elkerülése érdekében augmentációs segédtechnikákat is alkalmaztam, ügyelve arra, hogy ezzel ne torzuljanak a predikciók alapját adó sarkalatos adatpontok.

Az elkészített modellt a Core ML és Vision framework programozói interfészével integráltam, mindezt az alkalmazáskomponensek feladatkörének szeparációját, így a skálázhatóságot is biztosító MVVM architektúra keretein belül. Az elkészített alkalmazás működését manuális tesztesetek alapján ellenőriztem és értékeltem ki.

Végül, a megszerzett tapasztalatok birtokában az applikáció továbbfejlesztési lehetőségeire is kitértem további alkalmazható modellarchitektúrák, illetve az object detection előnyeinek és hátrányainak tömör, célravezető kiemelésével.

Irodalomjegyzék

[1] Shai Ben-David, Shai Shalev-Shwartz: “Understanding Machine Learning: From Theory to Algorithms”, Cambridge University Press, New York, 2014.

[2] John Paul Mueller, Luca Massaron: “Machine Learning For Dummies”, John Wiley & Sons Inc., Hoboken, 2016.

[3] Adam Gibson, Josh Patterson: “Deep Learning: A Practitioner’s Approach”, O’Reilly Media Inc., Sebastopol, 2017.

[4] K. D. Foote, “A brief history of machine learning”, *DATAVERSITY*, 20-Jan-2022. [Online]. Elérhető: https://www.dataversity.net/a-brief-history-of-machine-learning/. [Letöltve: 2022.03.10.]

[5] J. Brownlee, “Basic concepts in machine learning”, 14-Aug-2020. [Online]. Elérhető: https://machinelearningmastery.com/basic-concepts-in-machine-learning/. [Letöltve: 2022.03.18.]

[6] J. Brownlee, “Difference between algorithm and model in machine learning”, Machine Learning Mastery, 19-Aug-2020. [Online]. Elérhető: https://machinelearningmastery.com/difference-between-algorithm-and-model-in-machine-learning/. [Letöltve: 2022.03.18.]

[7] M. Hollemans, “Apple machine learning in 2020: What's new?”, machinethink. [Online]. Elérhető: https://machinethink.net/blog/new-in-apple-machine-learning-2020/. [Letöltve: 2022.03.18.]

[8] P. D. Steve Leven, “Machine Learning's Secret Sauce: Curation”, Medium, 26-Aug-2020. [Online]. Elérhető: https://towardsdatascience.com/machine-learnings-secret-source-curation-e8c3107dcc13. [Letöltve: 2022.03.18.]

[9] “Skin cancer: Types, symptoms, risk factors & treatment”, Cleveland Clinic. [Online]. Elérhető: https://my.clevelandclinic.org/health/diseases/15818-skin-cancer. [Letöltve: 2022.03.19.]

[10] “Skin cancer”, Dana. [Online]. Elérhető: https://www.dana-farber.org/skin-cancer/. [Letöltve: 2022.03.19.]

[11] “Dermatoscope: What is it used for and what does it see?”, Medical News Today. [Online]. Elérhető: https://www.medicalnewstoday.com/articles/dermatoscope. [Letöltve: 2022.03.19.]

[12] C. Rosendahl, A. Cameron, I. McColl, and D. Wilkinson, “Dermatoscopy in routine practice”, Australian Family Physician. [Online]. Elérhető:

https://www.racgp.org.au/afp/2012/july/dermatoscopy-in-routine-practice. [Letöltve: 2022.03.19.]
[13] “Overview of the ISIC Collaboration”, ISIC Archive. [Online]. Elérhető: https://www.isic-archive.com/#!/topWithHeader/tightContentTop/about/aboutIsicOverview. [Letöltve: 2022.03.21.]
[14] “Content and Layout of the Archive”, ISIC Archive. [Online]. Elérhető: https://www.isic-archive.com/#!/topWithHeader/tightContentTop/about/isicArchiveContent. [Letöltve: 2022.03.21.]
[15] “Infrastructure of the Archive”, ISIC Archive. [Online]. Elérhető: https://www.isic-archive.com/#!/topWithHeader/tightContentTop/about/isicArchiveInfrastructure. [Letöltve: 2022.03.21.]
[16] “Create ML: Create machine learning models for use in your app.”, Apple Developer Documentation. [Online]. Elérhető: https://developer.apple.com/documentation/createml. [Letöltve: 2022.03.28.]
[17] “Create ML: Creating an Image Classifier Model”, Apple Developer Documentation. [Online]. Elérhető: https://developer.apple.com/documentation/createml/creating_an_image_classifier_model. [Letöltve: 2022.03.28.]
[18] “MLImageClassifier: A model you train to classify images.”, Apple Developer Documentation. [Online]. Elérhető: https://developer.apple.com/documentation/createml/mlimageclassifier. [Letöltve: 2022.03.28.]
[19] “MLImageClassifier ImageAugmentationOptions”, Apple Developer Documentation. [Online]. Elérhető: https://developer.apple.com/documentation/createml/mlimageclassifier/imageaugmentationoptions. [Letöltve: 2022.03.28.]
[20] “VisionFeaturePrint - Core ML Format Reference documentation.”, Apple Coremltools Github Documentation [Online]. Elérhető: https://apple.github.io/coremltools/mlmodel/Format/VisionFeaturePrint.html#. [Letöltve: 2022.03.28.]
[21] “MLLogisticRegressionClassifier”, Apple Developer Documentation. [Online]. Elérhető:

https://developer.apple.com/documentation/createml/mllogisticregressionclassifier. [Letöltve: 2022.03.28.]
[22] “Create ML: Improving Your Model’s Accuracy”, Apple Developer Documentation. [Online]. Elérhető: https://developer.apple.com/documentation/createml/improving_your_model_s_accuracy/. [Letöltve: 2022.03.28.]
[23] “Core ML Model Format Specification - Core ML Format Reference documentation”, Apple Coremltools Github Documentation. [Online]. Elérhető: https://apple.github.io/coremltools/mlmodel/index.html. [Letöltve: 2022.03.28.]
[24] “Use coremltools to convert models from third-party libraries to Core ML.”, Coremltools API Reference. [Online]. Elérhető: https://coremltools.readme.io/docs. [Letöltve: 2022.03.28.]
[25] J. Brownlee, “A gentle introduction to transfer learning for Deep learning”, Machine Learning Mastery, 16-Sep-2019. [Online]. Elérhető: https://machinelearningmastery.com/transfer-learning-for-deep-learning/. [Letöltve: 2022.03.28.]
[26] “Create ML: Precision and Recall”, Apple Developer Documentation. [Online]. Elérhető: https://developer.apple.com/documentation/createml/mlclassifiermetrics/3005453-precisionrecall. [Letöltve: 2022.03.29.]
[27] J. Brownlee, “What is the difference between a parameter and a hyperparameter?”, Machine Learning Mastery, 16-Jun-2019. [Online]. Elérhető: https://machinelearningmastery.com/difference-between-a-parameter-and-a-hyperparameter/. [Letöltve: 2022.03.31.]
[28] “Introduction to MVVM”, objc.io, 01-Jun-2014. [Online]. Elérhető: https://www.objc.io/issues/13-architecture/mvvm/. [Letöltve: 2022.04.03.]
[29] S. Khanlou, “The Coordinator”, Khanlou, 20-Jan-2015. [Online]. Elérhető: https://khanlou.com/2015/01/the-coordinator/. [Letöltve: 2022.04.03.]
[30] M. Manferdini, “IOS storyboards in Xcode: The ultimate guide”, matteomanferdini.com, 05-Nov-2019. [Online]. Elérhető: https://matteomanferdini.com/ios-storyboards-xcode/. [Letöltve: 2022.04.03.]
[31] “UIViewController: An object that manages a view hierarchy for your UIKit app”, Apple Developer Documentation. [Online]. Elérhető:

https://developer.apple.com/documentation/uikit/uiviewcontroller. [Letöltve: 2022.04.03.]
[32] A. Gaidukov, “An introduction to protocol-oriented programming in swift”, Toptal Engineering Blog, 05-Dec-2016. [Online]. Elérhető: https://www.toptal.com/swift/introduction-protocol-oriented-programming-swift. [Letöltve: 2022.04.03.]
[33] “MLModel: An encapsulation of all the details of your machine learning model”, Apple Developer Documentation. [Online]. Elérhető: https://developer.apple.com/documentation/coreml/mlmodel/. [Letöltve: 2022.04.03.]
[34] “CVPixelBuffer: An image buffer that holds pixels in main memory”, Apple Developer Documentation. [Online]. Elérhető: https://developer.apple.com/documentation/corevideo/cvpixelbuffer-q2e. [Letöltve: 2022.04.03.]
[35] “MLFeatureProvider: An interface that represents a collection of values for either a model's input or its output”, Apple Developer Documentation. [Online]. Elérhető: https://developer.apple.com/documentation/coreml/mlfeatureprovider/. [Letöltve: 2022.04.03.]
[36] “UIImagePickerController: A view controller that manages the system interfaces for taking pictures”, Apple Developer Documentation. [Online]. Elérhető: https://developer.apple.com/documentation/uikit/uiimagepickercontroller/. [Letöltve: 2022.04.03.]
[37] Darren, “Understanding completion handlers in swift”, Programming With Swift, 18-Jul-2019. [Online]. Elérhető: https://programmingwithswift.com/understanding-completion-handlers-in-swift/. [Letöltve: 2022.04.03.]
[38] “Classifying Images with Vision and Core ML”, Apple Developer Documentation. [Online]. Elérhető: https://developer.apple.com/documentation/vision/classifying_images_with_vision_and_core_ml. [Letöltve: 2022.04.03.]
[39] “imageCropAndScaleOption: An optional setting informing the Vision algorithm how to scale an input image”, Apple Developer Documentation. [Online]. Elérhető: https://developer.apple.com/documentation/vision/vncoremlrequest/2890144-imagecropandscaleoption. [Letöltve: 2022.04.03.]
[40] “MLDataTable: A table of data for training or evaluating a machine learning model,” Apple Developer Documentation. [Online]. Elérhető:

https://developer.apple.com/documentation/createml/mldatatable/. [Letöltve: 2022.04.03.]
[41] M. Hollemans, “How fast is my model?”, machinethink. [Online]. Elérhető: https://machinethink.net/blog/how-fast-is-my-model/. [Letöltve: 2022.04.06.]
[42] M. Hollemans, “New Mobile Neural Network Architectures”, machinethink. [Online]. Elérhető: https://machinethink.net/blog/mobile-architectures/. [Letöltve: 2022.04.06.]
[43] M. Hollemans, “Mobilenet version 2”, machinethink. [Online]. Elérhető: https://machinethink.net/blog/mobilenet-v2/. [Letöltve: 2022.04.06.]
[44] M. Hollemans, “One-stage object detection”, machinethink. [Online]. Elérhető: https://machinethink.net/blog/object-detection/. [Letöltve: 2022.04.06.]
[45] S. Hollister, “Apple says its new A13 Bionic chip brings hours of extra battery life to new iPhones”, The Verge, 10-Sep-2019. [Online]. Elérhető: https://www.theverge.com/circuitbreaker/2019/9/10/20857177/apple-iphone-11-processor-a13-cpu-speed-graphics-specs. [Letöltve: 2022.04.06.]
[46] M. Hollemans, “An in-depth look at core ML 3”, machinethink. [Online]. Elérhető: https://machinethink.net/blog/new-in-coreml3/. [Letöltve: 2022.04.06.]

Ábrajegyzék

1. ábra: Mesterséges neurális háló szemantikus vázlata (saját szerkesztésű ábra) 9
2. ábra: A modelltanítási folyamat ciklusa (saját szerkesztésű ábra) 10
3. ábra: Konvolúció alapú feature extractor modell belső működése [3] 32
4. ábra: Logisztikus regresszió döntési határa (saját szerkesztésű ábra) 33
5. ábra: A Create ML transfer learning pipeline architektúrája (saját szerkesztésű ábra) 34
6. ábra: Overfitting jelenségének kimutatása validációs metrikák segítségével (saját szerkesztésű ábra) 37
7. ábra: A Create ML augmentációs eszköztára [22] 41
8. ábra: Az alkalmazás nézethierarchiája (saját szerkesztésű ábra) 47
9. ábra: MVVM architektúra (saját szerkesztésű ábra) 49
10. ábra: Vision framework 'crop and scale' transzformációi [39] 57
11. ábra: Jóindulatú bőrelváltozások egyedi tesztazonosítóval ellátott verifikációs képhalmaza (saját szerkesztésű ábra) 61
12. ábra: Malignáns bőrelváltozások egyedi tesztazonosítóval ellátott verifikációs képhalmaza (saját szerkesztésű ábra) 63
13. ábra: Helyes predikciók átlagos konfidenciaértékei a precíziós pontosság viszonylatában (saját szerkesztésű ábra) 64

Táblázatjegyzék

1. táblázat: Jóindulatú mintaképekre vonatkozó manuális tesztesetek eredményei 60
2. táblázat: Malignáns mintaképekre vonatkozó manuális tesztesetek eredményei 62