Közlekedési táblák felismerése Konvolúciós Neuronháló (CNN) segítségével

Schifler Patricia-Vivien
Informatika III.
Sapientia EMTE

Bevezető

A konvolúciós neuronháló vagy CNN egy olyan mélytanulási technika, amivel általában olyan feladatokat oldanak meg, amihez objektumfelismerés szükséges. Egy ilyen feladat az önvezető autóknál a közlekedési táblák felismerése, ami valljuk be, nem a legegyszerűbb feladat, tekintve, hogy nem csak kontinensenként, de akár országonként is igen eltérő táblákkal találkozhatunk, vagy épp olyanokkal, amik csak adott térségekben vannak jelen. Ennek okán, akár a tapasztalt vezetőknek is újdonságnak számíthatnak egyes táblák.

Ezen dokumentáció egy ilyen projektet hivatott bemutatni, amiben egy CNN-t hozunk létre, Python környezetben, a Keras mélytanulási API segítségével. A modell 11 osztályt hivatott felismerni, többek között a gyalogos átkelőhely táblát, a 80-as sebességkorlát táblát és az elsőbbség szabályozó táblát.

A projekt elkészítése

A projekt felülnézetből két lépésből tevődött össze: adatgyűjtés és a modell elkészítése.

Az adatgyűjtés során több különböző oldalt is megtekintettem, míg végül a Roboflow oldal Universe részlegén, első látásra megfelelő minőségű és mennyiségű adatokat találtam. Ezt követően a Dataset Ninja oldaláról is gyűjtöttem még adatokat. A különböző adathalmazok egyesítéséből összeállt a végső adathalmazom, amit a modell felépítésére is használtam. Az adatok ezt követő kiválogatását két lépésben végeztem el. A *categorize_images.py* fájlban található kód a kapott adathalmazokat a hozzájuk csatolt táblázat alapján a megjelölt osztályokba csoportosította. A *count_imgs_in_dir.py* fájlban levő kód pedig megszámolta, hogy az adott mappákban hány elem található. Ezt követően kiválasztottam azokat a mappákat, amik ígéretesnek bizonyultak. Megbizonyosodtam róla, hogy csak olyan képek szerepelnek a könyvtárakban, amik pontosak, majd ezt követően feltöltöttem őket Google Drive-ra, ahonnan a Google Colab engedélyezést követően könnyedén elérte az adatokat. Az adatok a következő linken érhetők el.

Megjegyzendő, hogy a Roboflow-ról összegyűjtött adatok már alapjáraton preprocesszálva, normalizálva voltak. A képek mérete 640x640 volt, emellett automatikus tájolást végeztek rajtuk, aminek köszönhetően a képek különböző módon vannak elforgatva, valamint automatikus kontrasztolást végeztek kontraszt nyújtással. Emellett, a világosságot és az expozíciót (megvilágítási idő) egy randomizált értékkel változtatták -25 és 25 százalék között, valamint magas frekvenciájú zajt adtak a pixelek 7 százalékához. Emellett minden képből három darab, különböző módon feldolgozott példány van. Valamint, ezek a képek inkább stock fotók, míg a Dataset Ninja-s képek inkább a hétköznapi környezetben megtalálható kontextusok formájában ábrázolják a táblákat.

A Dataset Ninja-s képeket manuális módon választottam ki, mivel nem a Roboflow csoportosítási módszerét követte, illetve nem rendelkezett túl nagy mennyiségű adatmennyiséggel, így inkább kiegészítés céljából volt megfelelő, valamint "*élet ízt*" kölcsönzött az adathalmaznak.

A modell felépítéséhez a Lab7_CNN_Flowers_Recognition példa kódot használtam fel, aminek az alapját Kaggle-ön találjuk meg. A kódot természetesen át kellett alakítanom, hogy a saját adathalmazomnak és az adott feladatnak megfelelően működjön. Ennek okán mind a modellt, mind az adathalmaz betöltéséért felelős részt korrigálnom kellett. Továbbá a modell mentéséért felelős részt implementálnom kellett. A modell felépítésére használt kódot csatolt *CNN_utjelzo_tablak_vegso.ipynb*, a túltanított modell felépítésére használt kódot pedig a *CNN_utjelzo_tablak_tultanitott_vegso.ipynb* fájlokban találjuk meg. Ehhez társítva a kimentett modelleket a *cnn_model.keras* és *cnn_model_tultanitott_vegso.keras* fájlok képezik.

A statisztikák kimentéséért felelős részt is külön implementálnom kellett. A kapott statisztikákat a csatolt *metrics_traffic_signs.csv* és *metrics_traffic_signs_overlearned_final.csv* fájlban találjuk.

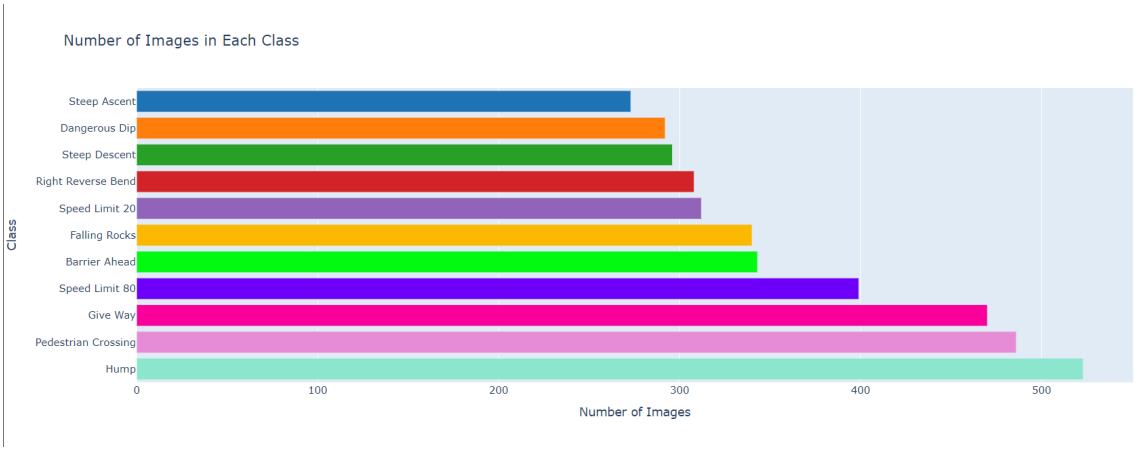
Adathalmaz feltárása

A Roboflow oldal Universe részlegén, első látásra megfelelő minőségű és mennyiségű adatokat találtam. A "*traffic signs*" kulcsszavak alatt több különböző adathalmazt is találtam, amiknek az adatairól előkép is elérhető, így a nagyon rossz minőségű vagy nagyon kis méretű képeket ki tudtam küszöbölni, így ezekkel nem kellett bajlódnom. Viszont, a letöltött adathalmazok ettől függetlenül nem voltak tökéletesek. Bár az innen letöltött adathalmazok mindegyike fel volt osztva tanítási, tesztelési és validálási mappákra, amik mindegyikéhez tartozott egy CSV fájl az elérhető képekről és azok táblák szerinti osztályozásáról és méretéről, nekem olyan könyvtárakra volt szükségem, amik a táblák szerint osztják szét a képeket.
Ennek okán, felhasználva a mappákban talált CSV fájlt írtam egy Python kódot, amivel a mappák tartalmát fel tudtam osztani a megadott osztályok szerint. Bár ez sikeresen elvégezte a dolgát, az osztályok nevei nem voltak minden esetben megfelelőek. Példaképpen olyan nevek voltak, amik számokat jelöltek, amiből következtetni lehet arra, hogy aki osztályozta a képeket, rendelkezett egy listával, hogy melyik szám mit jelöl, de akár emberi hanyagságra is következtethetünk ebből. Emellett, arra is volt eset, hogy különböző adathalmazoknál ugyanazok a táblák más osztálynévvel voltak ellátva. Továbbá, olyan eset is volt, hogy egy tábla rossz osztály névvel volt ellátva, így például a 40-es sebességkorlát tábla bekerült egy 80-as sebességkorlát csoportba.
Így a csoportosítást követően át kellett néznem mindegyik mappát, hogy biztosan megfelelő képek szerepelnek-e benne. Ezt megelőzően viszont készítettem egy

másik Python kódot, aminek segítségével végigiteráltam minden mappán és megszámoltam, hogy hány képet tartalmaz. Ezt követően kiválasztottam azokat, amik 140 és 200 között voltak, ami utólag belátva kevésnek bizonyultak, hiszen akadt olyan mappa, amiben 200 feletti darabszám is volt és jobban megfelelt volna a feladatnak. Miután kiválasztottam ezeket a könyvtárakat, számszerint 11 darabot, végignéztem, hogy biztosan megfelelő képek vannak-e bennük. Amennyiben nem voltak megfelelő képek, de beletartoztak a többi kiválasztott mappa valamelyikébe, áttettem a képet, máskülönben töröltem az adott mappából. Szerencsére ez csak minimálisan csökkentette a képek darabszámát.
Az első adathalmaz preprocesszálása után kipróbáltam a modellt, de mivel nem működött az adott adatmennyiséggel, először arra következtettem, hogy a kiugró értékek miatt működhet rosszul. Ezért a nagyobb adatmennyiséggel rendelkező mappák méretét körülbelül 160 darab képre szűkítettem, de hamar be kellett látnom, hogy nem ez volt a probléma. Rájöttem, hogy a fő problémát az okozza, hogy kevés képpel rendelkezem, ezért kerestem még adathalmazokat, amik megfeleltek az elvárásaimnak. Hasonlóképpen jártam el ezekkel is, mint a legelső adathalmazzal, majd a már korábban kiválasztott könyvtárakhoz tartozó képeket választottam ki.
Az új adatok hozzáadása után azonban, bár jobban működött a modell, még mindig nem volt megfelelő, ezért a Dataset Ninja oldaláról is csatoltam egy adatmennyiséget. Az innen szerzett képek diverzitásának köszönhetően a modell ezt követően jobban teljesített. A képek mennyiségének kiegyensúlyozatlanságának köszönhetően viszont a modell tud tévedni, de a valóságban is adott táblákból több van, adottakból pedig kevesebb. Például síkságon kevesebb valószínűséggel találkozunk kőomlást jelző táblát, ahogyan városon belül is kevesebb eséllyel látunk háziállatokat jelző táblákat.
Bár a kapott példa kódban a képek dúsítására is rendelkezésre állt egy kód, ezt használva a modell rosszabbul teljesített, így ezt a részt nem használtam fel a kezdeti modelleknél. Ellenben a végső modellnél javította az eredményeket.
A felhasznált képek többsége 640x640-es mérettel rendelkezett, valamint a Roboflow oldalán talált adathalmazokhoz tartozó README fájlokban pontosabb leírást is lehetett kapni arról, hogy hogyan dolgozták fel őket. Kezdetben ezeket a képeket 150x150-es mérettel alkalmaztam, ellenben ez nem volt megfelelő, mivel nem volt a legoptimálisabb a Pooling rétegekre nézve, ezért végül a 128x128-as képméret volt az, amivel a modell optimálisan tudott működni, hiszen mind a Pooling rétegekkel, mind a dúsítási folyamattal összhangban volt.

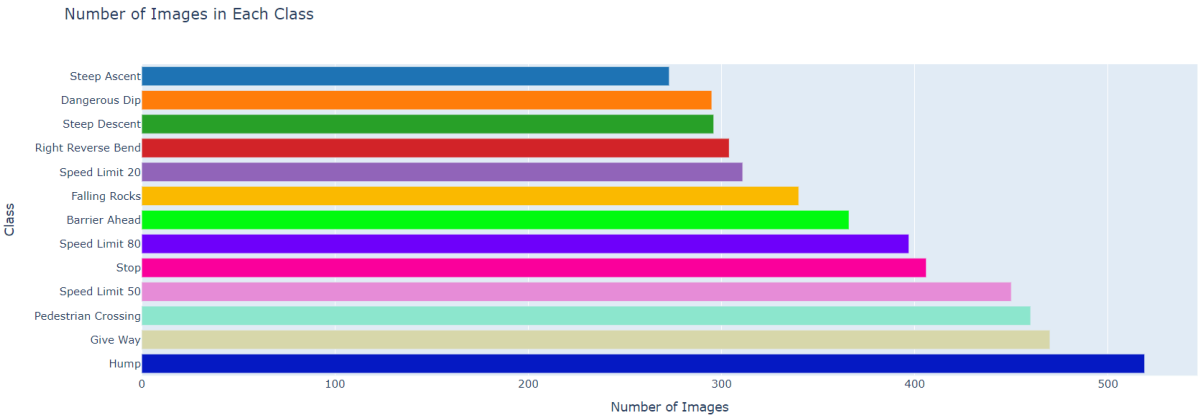
A modell felépítéséhez szükséges folyamatok elvégzésére létrehoztam egy Jupyter Notebook fájlt, ami tartalmazza a képek preprocesszálását is.
Előre meghatároztam a képeket tartalmazó mappák elérési útvonalát, valamint a 128-as képméretet. Ezután a képeket a megadott útvonalakról RGB formában, színes képként olvastam be, valamint címkéztem fel a megadott nevek alapján, amik a csoportok nevei voltak. Ezt a OpenCV segítségével tettem, ami könnyed kezelést biztosít a képek és videók feldolgozására. A beolvasási és címkézési folyamat során

a képek mennyiségét is követhetjük. Ezt követően a képek eloszlását is ábrázoljuk, ezt az 1. és 2. Ábrán láthatjuk.



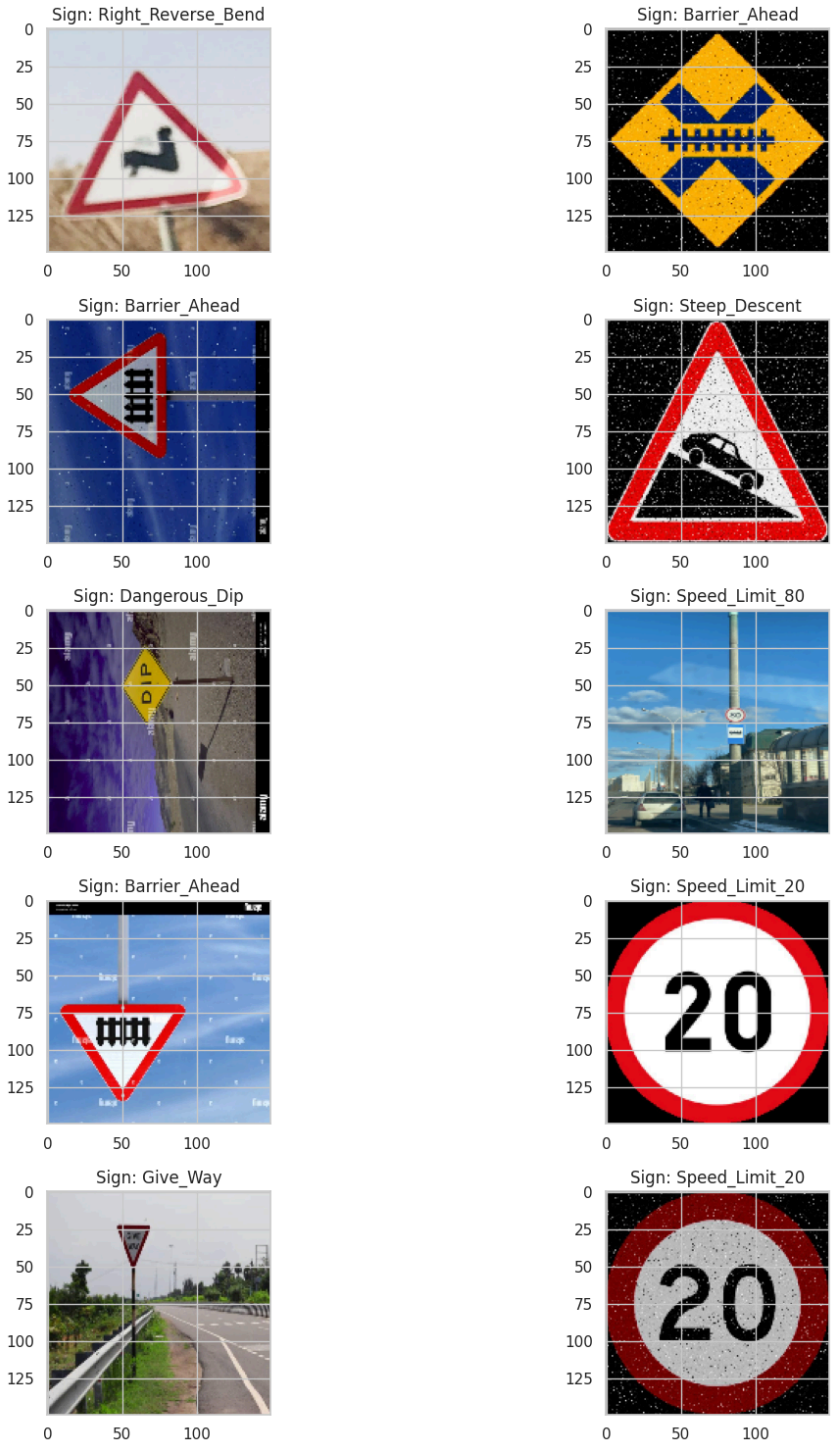
1. Ábra - A kezdeti kép mennyiség eloszlása

Két kezdeti csoportnevet, a “*Steep Ascent*”-et “*Stop*”-ra, a “*Dangerous Dip*”-et pedig “*Speed Limit 50*”-re cseréltem, természetesen a hozzájuk tartozó képek korrigálásával együtt, mivel az eredeti két adathalmazban olyan képek voltak, amik nagyon hasonlítottak más csoportban lévő képekre, ezért megtévesztőek voltak a modell számára. Az új csoportok több képpel is rendelkeztek a korábbi csoportoktól, már az adatok bővítése előtt, tehát már a legelső adathalmaznál. Bár a Google Drive-on fennmaradtak, ahogy az a 2. Ábrán is látható, az adatok betöltésekor nem adtam meg az eredeti két csoportot.



2. Ábra - A módosított kép mennyiség eloszlása

A képek ellenőrzésének érdekében egy pár képet randomizált módon kiválasztva a *matplotlib.pyplot* csomag segítségével jelenítettem meg. Ezzel ellenőriztem le, hogy megfelelő képeket töltöttem-e fel, illetve hogy azok megfelelően lettek-e felcímkézve. Egy futtatás során a 3. Ábrán látható képeket kaptuk, amik jelzik, hogy jó vagy rossz képeket tettünk fel, illetve azok jól lettek felcímkézve. Mint látható a képek találnak a címkékkel.



3. Ábra - Példák a betöltött képekre

Ezt követően One Hot Encoding (OHE) segítségével átalakítottam a címkéket, hogy megkapjam a modellhez szükséges X és Y tömböket, amikbe a már felcímkézett adatok kerülnek bele. A *LabelEncoder* segítségével hoztam létre a tömböket, amikbe a 11 osztály szerint csoportosulnak be az elemek.
Ezután az adathalmazt tanítási és tesztelési csoportokra bontottam fel, 75-25 százalékos arányban.

CNN modell építése

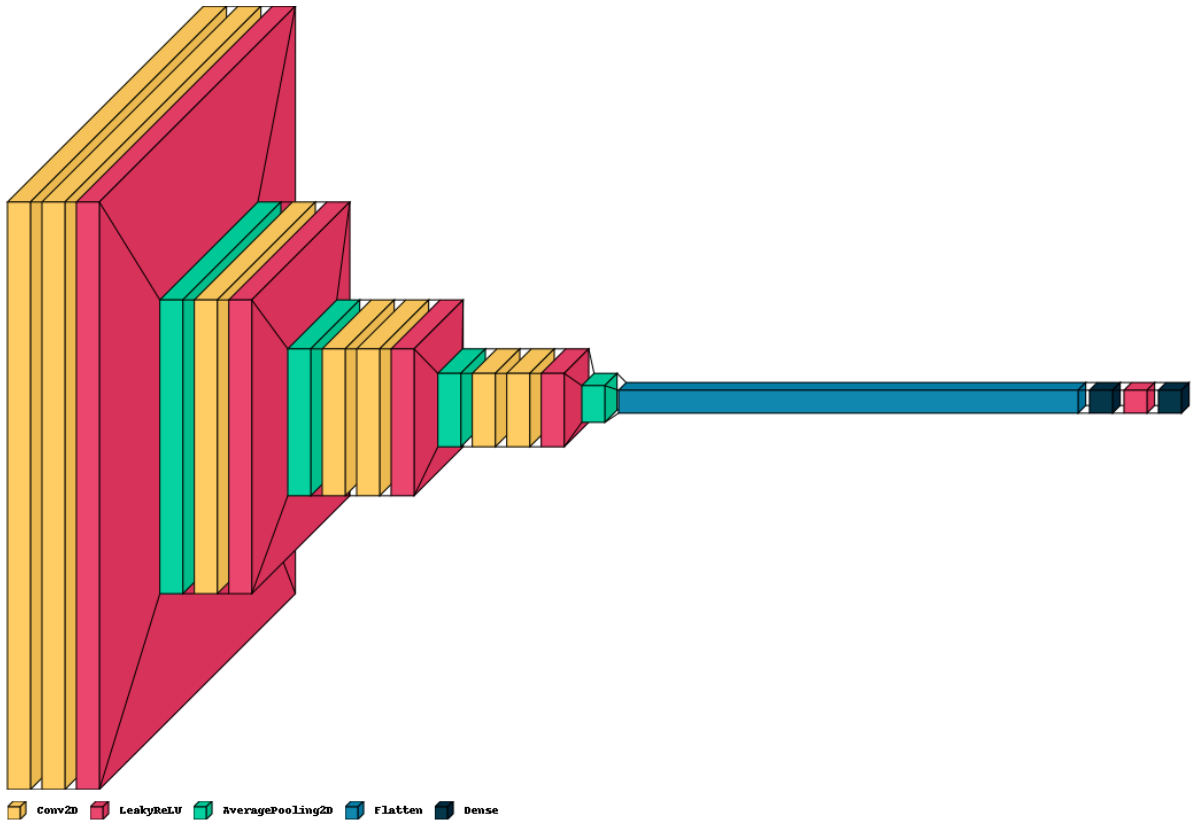
A modell felépítéséhez a korábban említett Lab7_CNN_Flowers_Recognition fájlként kapott példa kódot használtam fel, ami megfelelő alapot biztosított a kód megírásához. Azonban egyértelműen a feladatra és az adathalmazra kellett igazítani. Legelőször is a képek importálásához 128-as képméretet kellett megadjunk, hogy erre a méretre skálázzuk át a képeket a beolvasás során. Ezt követően, mivel 5 osztály helyett 11-el dolgoztunk, így ehhez is át kellett írni az adott részeket. Ezt követően a modell csak 1 osztályt ismert fel, ami valljuk be, messze állt a jótól. Ennek okán az epoch számokkal kezdtem kísérletezni, de sem 20, sem 30 epochra nem mutatott változást. Így elkezdtem a modellel kísérletezni, a *Dropout* értéket kevesebbre, illetve többre állítani, *relu* helyett *LeakyRelu*-t használni, *MaxPooling* helyett *AveragePooling*-ot, s bár látható volt fejlődés, az eredmények még mindig igen rosszak voltak. Ebből kifolyólag arra a következtetésre jutottam, hogy az adatok feldolgozásával lehet a baj. Így *fit_generator* helyett a sima *fit* metódust használtam, mivel előbbi már elavultnak számít, így nagymértékben javultak az eredmények, de még mindig nem kellő mennyiségben.

A végső modellnél a *Dropout* értéket kivettem, mivel nem okozott változást. Egyszerű konvolúció helyett dupla konvolúciót használtam 3 rétegnél, egy rétegnél pedig egyszerű konvolúciót hagytam, ami az előtte megjelenő filterszámmal rendelkezett. A dupla konvolúció nagymértékben javította a modell működését az egyszerű konvolúcióhoz képest. A LeakyReLU az egyszerű ReLU-hoz képest szignifikáns javulást ért el, mivel a komplexebb adatokat is könnyen felismerte a modell és könnyebben meg tudta őket különböztetni. Bár a Max Pooling és az Average Pooling esetében is a kép jellemzői vannak kiemelve, a Max Pooling jobban belemegy a részletekbe, így könnyen el tud veszni bennük, ezért nem hozott olyan jó eredményeket, mint az Average Pooling, ami inkább összességében tekinti a képet és annak jellemzőit. A neuronok számát 128-ra állítottam a Dense keretein belül, ami megfelelőbbnek bizonyult a rendelkezésre álló adatmennyiséggel, mint a 256 neuron. Végül pedig beállítottam, hogy megkapjam a 11 osztályt. A kapott modell látható a 4. és 5. ábrákon. Emellett, bár kipróbáltam különböző alpha értékekre a LeakyReLU rétegen a modell működését, az eredmények nem javultak. A modell kompilálásakor használt beépített Adam függvényt az optimizer tulajdonság Adam függvényére cseréltem, tanulási ráta megadása nélkül, mivel a kipróbált modellek esetében ez jobb eredményt hozott.

Megfigyelhető továbbá, hogy a bemeneti értékek között nem talált olyan paramétert, amelyet nem tudott volna felhasználni.

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_20 (Conv2D)          (None, 128, 128, 32)      2432

 conv2d_21 (Conv2D)          (None, 128, 128, 32)      9248

 leaky_re_lu_14 (LeakyReLU)  (None, 128, 128, 32)      0

 average_pooling2d_11 (Aver  (None, 64, 64, 32)        0
 agePooling2D)

 conv2d_22 (Conv2D)          (None, 64, 64, 64)        18496

 leaky_re_lu_15 (LeakyReLU)  (None, 64, 64, 64)        0

 average_pooling2d_12 (Aver  (None, 32, 32, 64)        0
 agePooling2D)

 conv2d_23 (Conv2D)          (None, 32, 32, 64)        36928

 conv2d_24 (Conv2D)          (None, 32, 32, 64)        36928

 leaky_re_lu_16 (LeakyReLU)  (None, 32, 32, 64)        0

 average_pooling2d_13 (Aver  (None, 16, 16, 64)        0
 agePooling2D)

 conv2d_25 (Conv2D)          (None, 16, 16, 128)       73856

 conv2d_26 (Conv2D)          (None, 16, 16, 128)       147584

 leaky_re_lu_17 (LeakyReLU)  (None, 16, 16, 128)       0

 average_pooling2d_14 (Aver  (None, 8, 8, 128)         0
 agePooling2D)

 flatten_3 (Flatten)         (None, 8192)              0

 dense_6 (Dense)             (None, 128)               1048704

 leaky_re_lu_18 (LeakyReLU)  (None, 128)               0

 dense_7 (Dense)             (None, 11)                1419

=================================================================
Total params: 1375595 (5.25 MB)
Trainable params: 1375595 (5.25 MB)
Non-trainable params: 0 (0.00 Byte)
```

4. Ábra - A végső modell architektúrája



5. Ábra - A végső modell felépítése

A modell képzése és validálása

A modell tanítására a Google Colab környezet által kínált T4 GPU környezetet használtam, ami, amennyiben a csillagok úgy által, engedett GPU-hoz való csatlakozást. Nos, a csillagok általában nem úgy álltak, hogy ez lehetséges legyen, így társítva egy “csodálatos” internetkapcsolattal, a tanítás elég hosszú időt vett igénybe a kezdeti modelleknél, mivel CPU-s környezetben történt. Azonban több fiókkal történő belépésnek köszönhetően párhuzamosan tudtam több modellt futtatni, így volt, amikor GPU-s környezetet tudtam használni. A végső modell esetében is szerencsém volt, ahogy a túltanított modell esetében viszont nem, így csak CPU-s környezetben tudtam tanítani a végső túltanított modellt.

A példa kódban beállított 0.1-es tanítási ráta nem bizonyult optimálisnak, így a korábbi feladatokból tanulva, kisebb értékeket is kipróbáltam, amik közül a tanításhoz a 0.1-es érték jobbnak bizonyult a 0.001-es értékkel szemben. Ellenben ezek az értékek a frissített modellnél rosszabb eredményekhez vezettek, mint az optimizer Adam-je, ami önszabályozza a tanulási rátáját. A ReduceLROnPlateau csomag a tanítási ráta optimalizálásában segített, biztosította, hogy amennyiben egy adott pont után nem történik változás, a tanítási ráta csökkenjen. A várakozási idejét 3 epochra állítottam. Emellett, bevezettem egy ModelCheckpoint-ot, amivel adott modelleket mentettem ki, a validációs pontosság függvényében. Továbbá,

bevezettem egy EarlyStopping-ot, aminek a várakozását 5-re állítottam a validációs veszteség függvényében, így, amennyiben 5 epochon keresztül nem változik a validációs veszteség, leállítja a validálást.

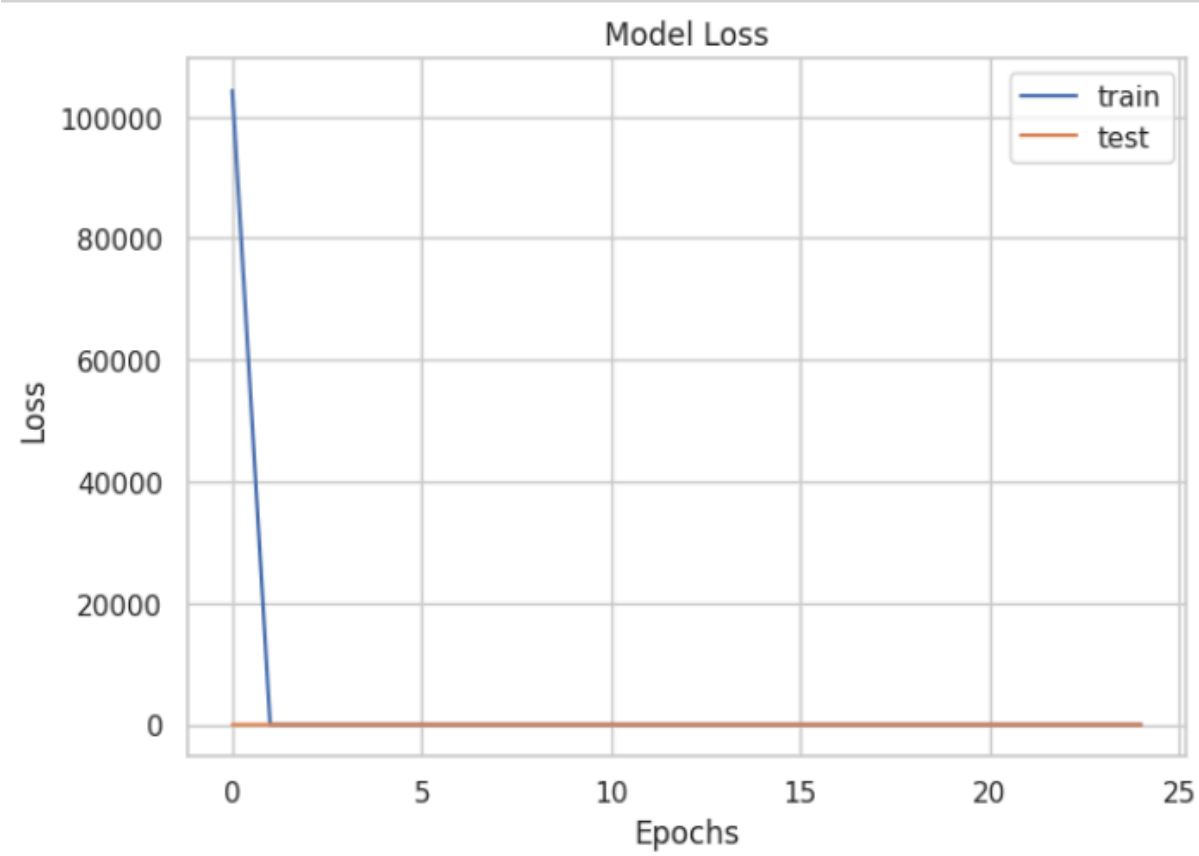
Az ImageDataGenerator csomagot kipróbálva a képek kezdetben nem bizonyultak megfelelőnek, ezt a 6. Ábra is bizonyítja, ahol randomizált módon kiválasztott képeket láthatunk. Bár emberi lény számára kivehető, hogy mit ábrázol a kép, egy modell tanítására, véleményem szerint nem megfelelőek ezek a képek. Azonban, némi finomhangolás után a képek olyan formátumra generálódtak, ami már a modell számára is megfelelő minőségű volt. Ezt a 7. Ábra mutatja be. Így végső soron az ImageDataGenerator hasznosnak minősült.

6. Ábra - ImageDataGenerator képek a példaként kapott értékekkel

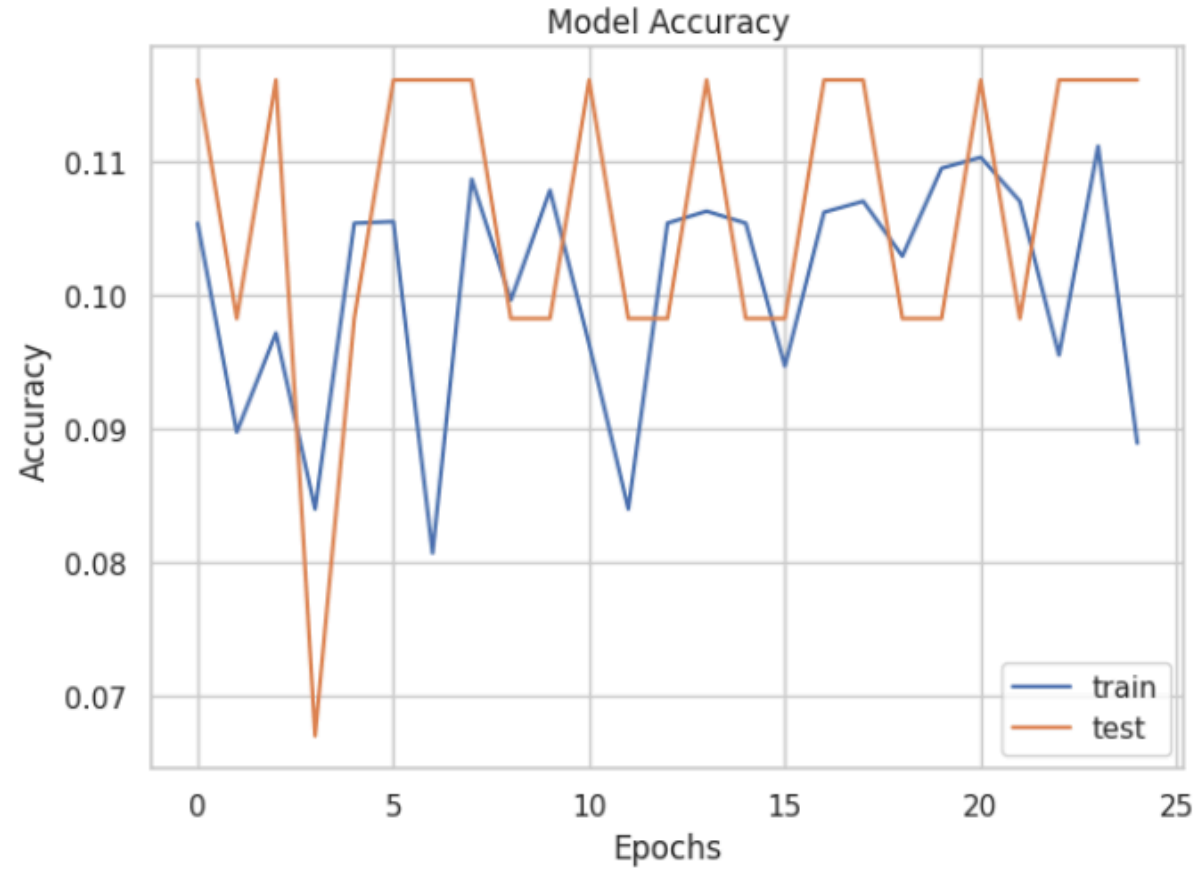
7. Ábra - ImageDataGenerator képek finomhangolás után

A modellt a futtatás után kimentettem a *save* függvénnyel a megadott helyre a Google Drive-on *cnn_model_vegso.keras* néven. A túltanított modellt pedig *cnn_model_tultanitott_vegso.keras* néven.

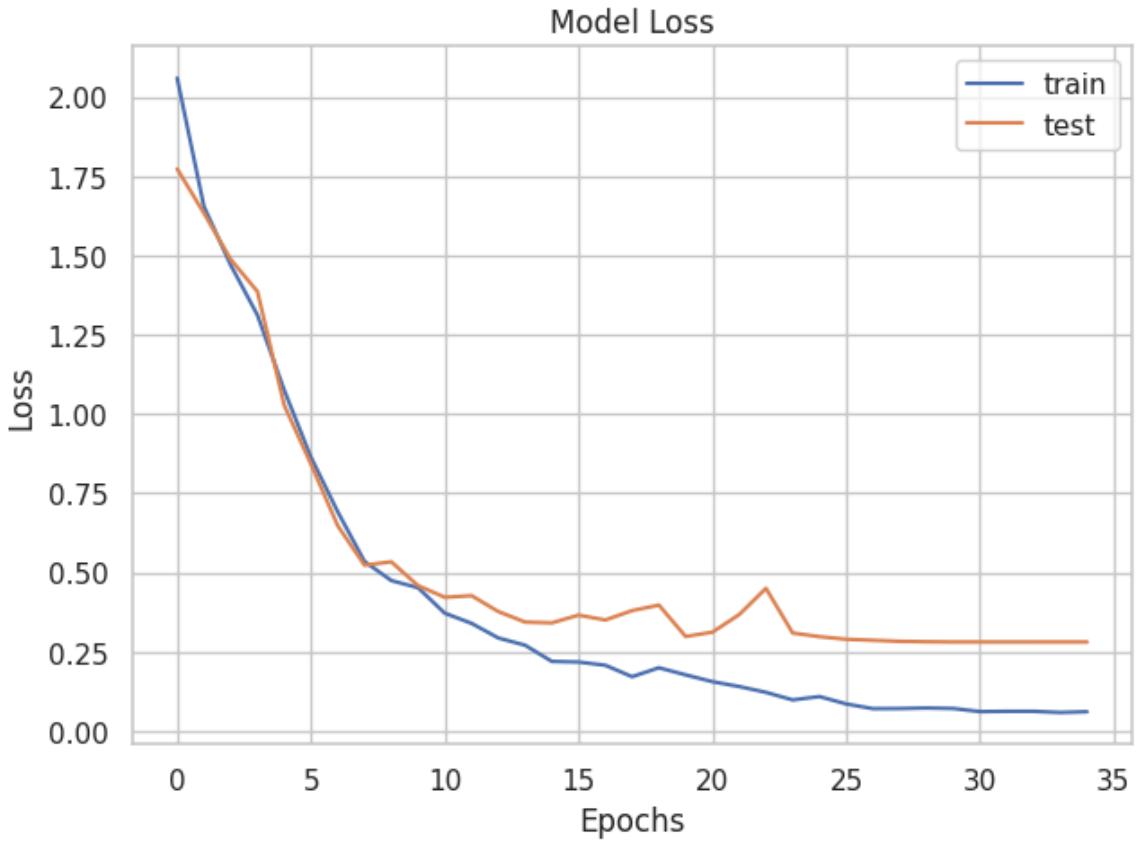
Ezt követően a History-ba kimentett adatok segítségével ábrázoltam a kapott eredményeket. A 8. és 9. ábrán láthatjuk a kezdeti modell tanulási és tesztelési hiba értékét és pontossági értékét, ami nem teljesített jól. A 10. és 11. ábrákon pedig a végső modell tanulási és tesztelési hiba értékét és pontossági értékét láthatjuk, ami szignifikánsan jobb.



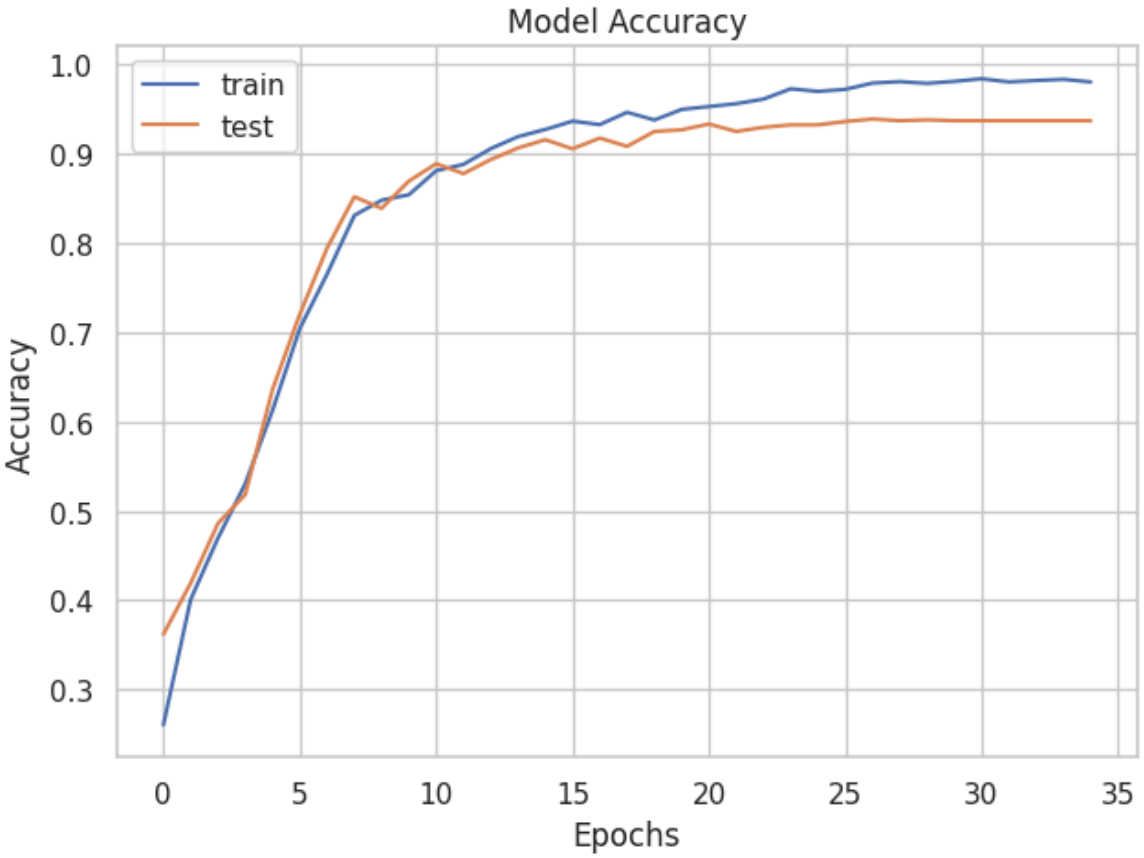
8. Ábra - A kezdeti modell validációs és tesztelési hibája



9. Ábra - A kezdeti modell validációs és tesztelési pontossága



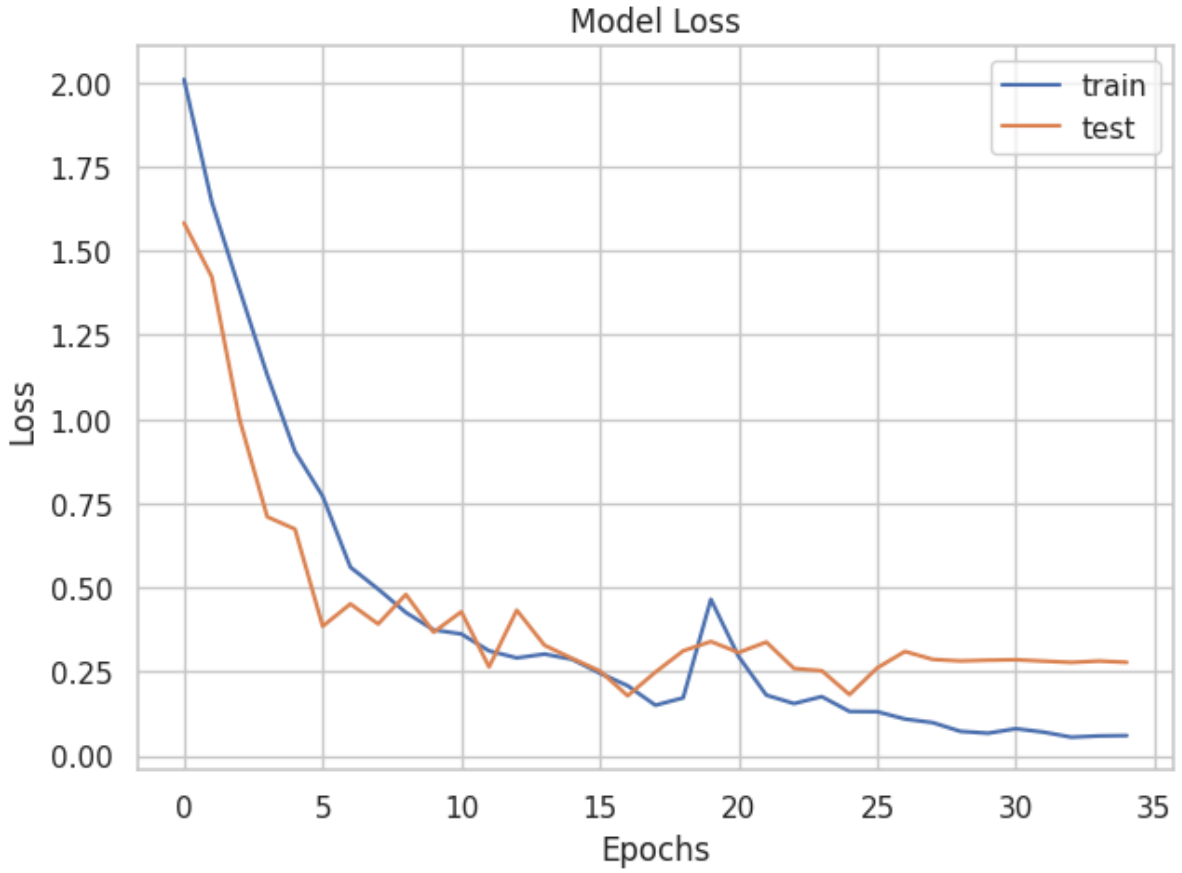
10. Ábra - A végső modell validációs és tesztelési hibája



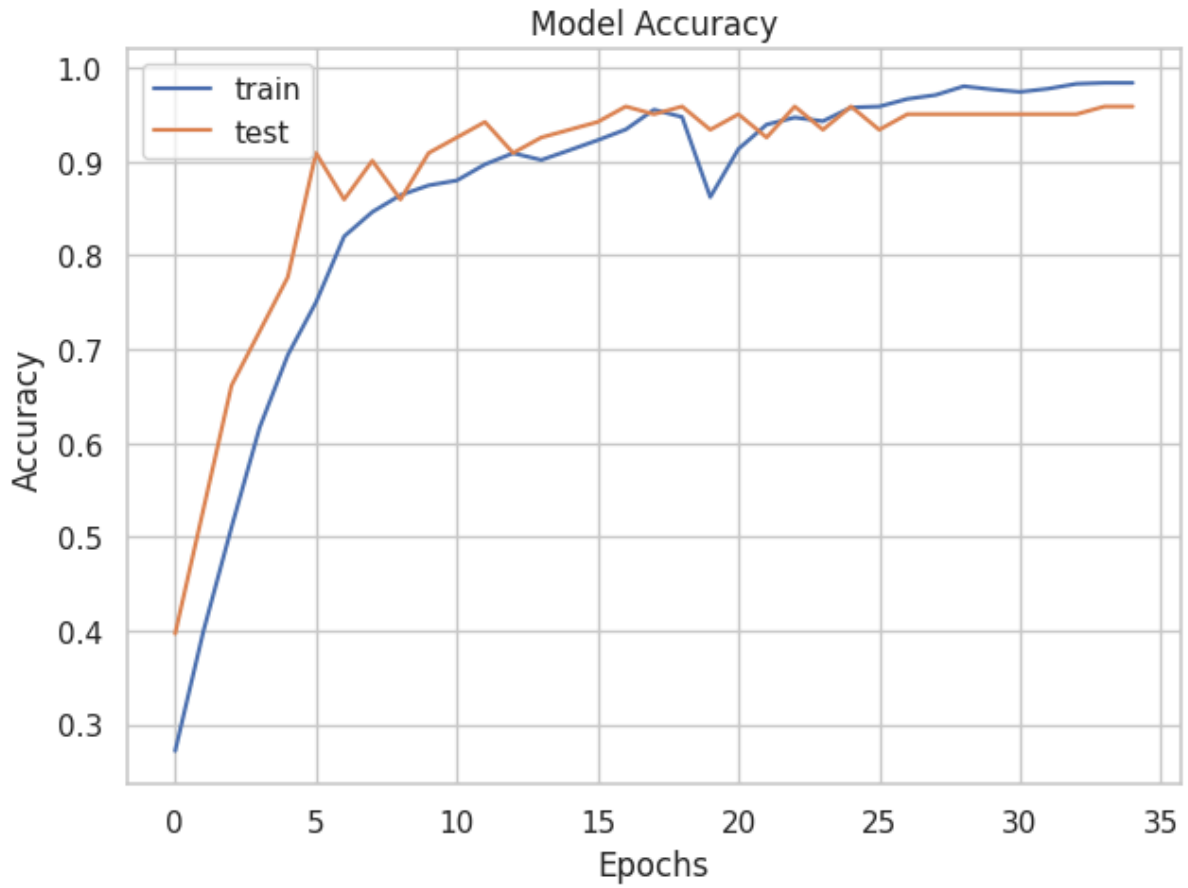
11. Ábra - A végső modell validációs és tesztelési pontossága

Az 8. és 9. Ábrán látható, hogy a modell tanulása instabil, míg 10. és 11. ábrákon stabil tanulást figyelhetünk meg.

A modell túltanulásának elérésének érdekében az eredeti Jupyter Notebookról másolatot készítettem, amelyben különböző paramétereket módosítottam, majd mivel ez telt a legtöbb időbe, így bár nem 100%-ban túltanult, de 11-ből 5 osztályt teljes mértékben túltanult. Ezt az adathalmaz 97 és 3 százalékra való osztásával értem el, valamint az epochok lecsökkentésével a végső túltanított modellnél. Az ezt megelőző modelleknél maximum 2 osztályt tudott 100%-osan megtanulni. Az eredményeket a 12. és 13. Ábrákon láthatjuk.



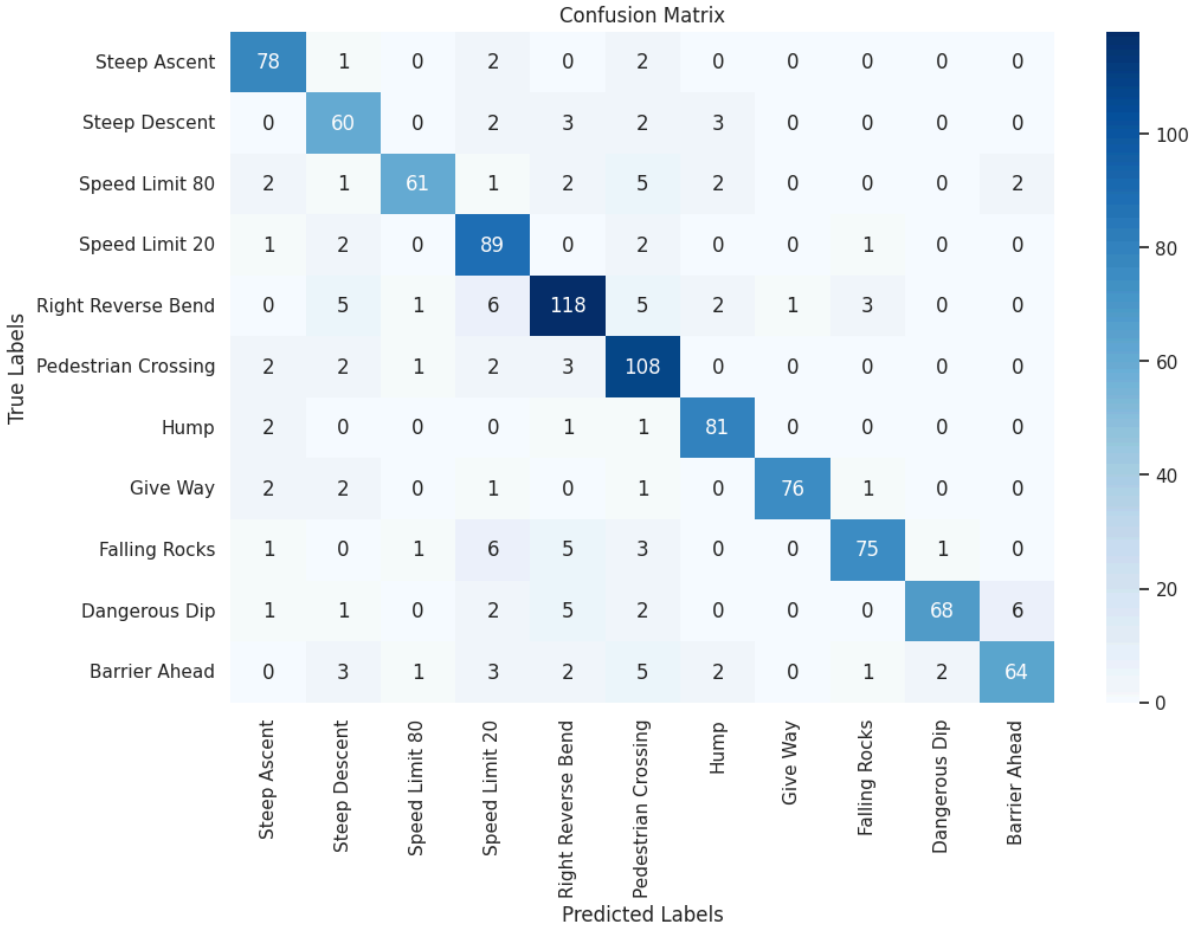
12. Ábra - A túltanult modell validációs és tesztelési hibája



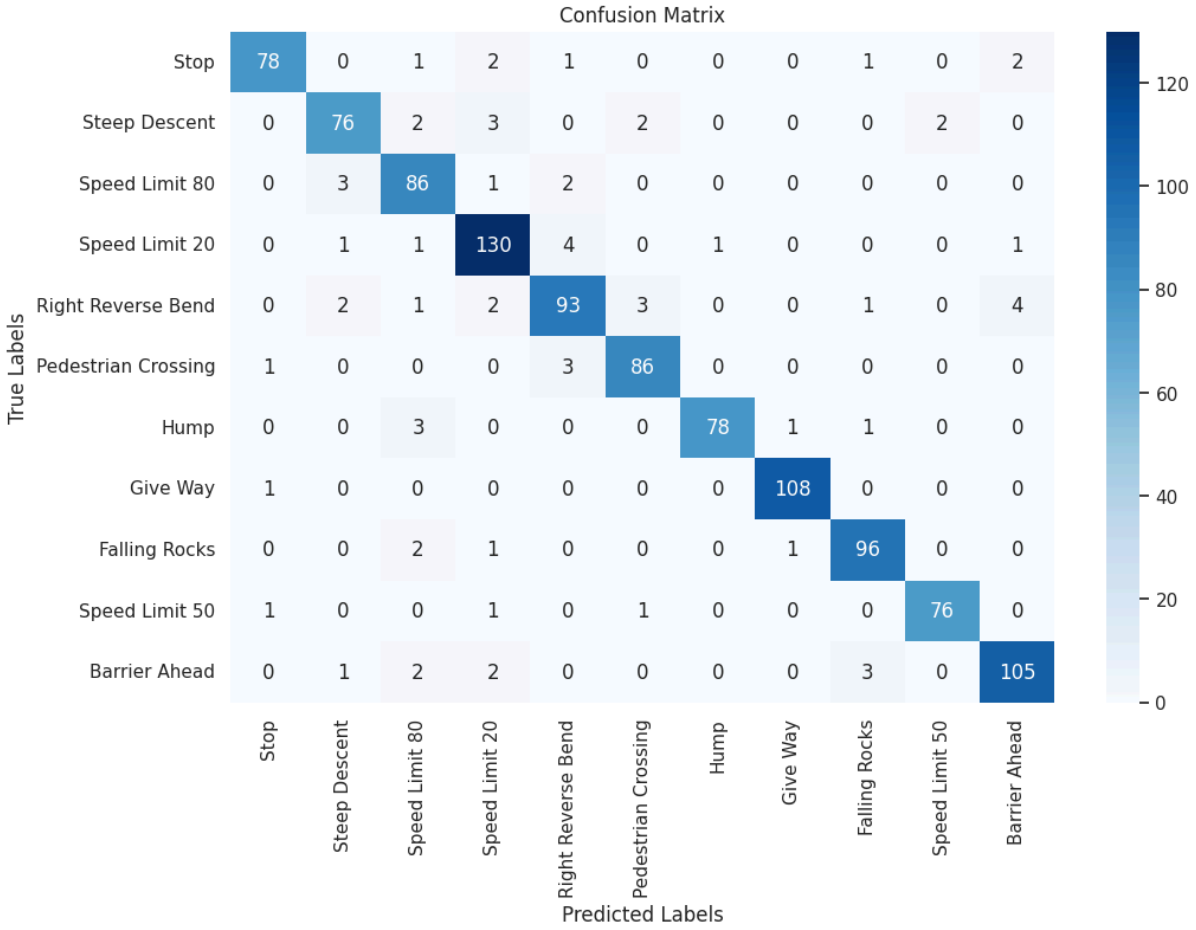
13. Ábra - A túltanult modell validációs és tesztelési pontossága

A modell tesztelése

A modell teszteléséhez, a korábban említett 75-25 százalékos arányban felosztott adathalmaz 25%-t használtam fel.
Ahogy az a 14. Ábrán található konfúziós mátrixból látható, az esetek többségében a megjósolt címke megegyezik a valódi címkével, ám vannak esetek, amik feltehetően a képek hasonlóságából eredően, vagy a képmennyiségek eltérése miatt nem jól kategorizálja be a képet. Ez volt megfigyelhető egy korábbi modell kiértékelésénél. Voltak olyan modellek, ahol a "*Dangerous Dip*" és a "*Barrier Ahead*", valamint a "*Steep Ascent*" és "*Steep Descent*" csoportokat nagymértékben összekeverte. Erről sajnos nem rendelkezem képpel, viszont ezen összekeverés volt az, ami miatt a "*Dangerous Dip*" és "*Steep Ascent*" csoportokat "*Stop*" és "*Speed Limit 50*" csoportokra cseréltem le. A 15. Ábrán látható, hogy a végső modell merően jobban teljesít és nem téveszti össze a csoportokat.

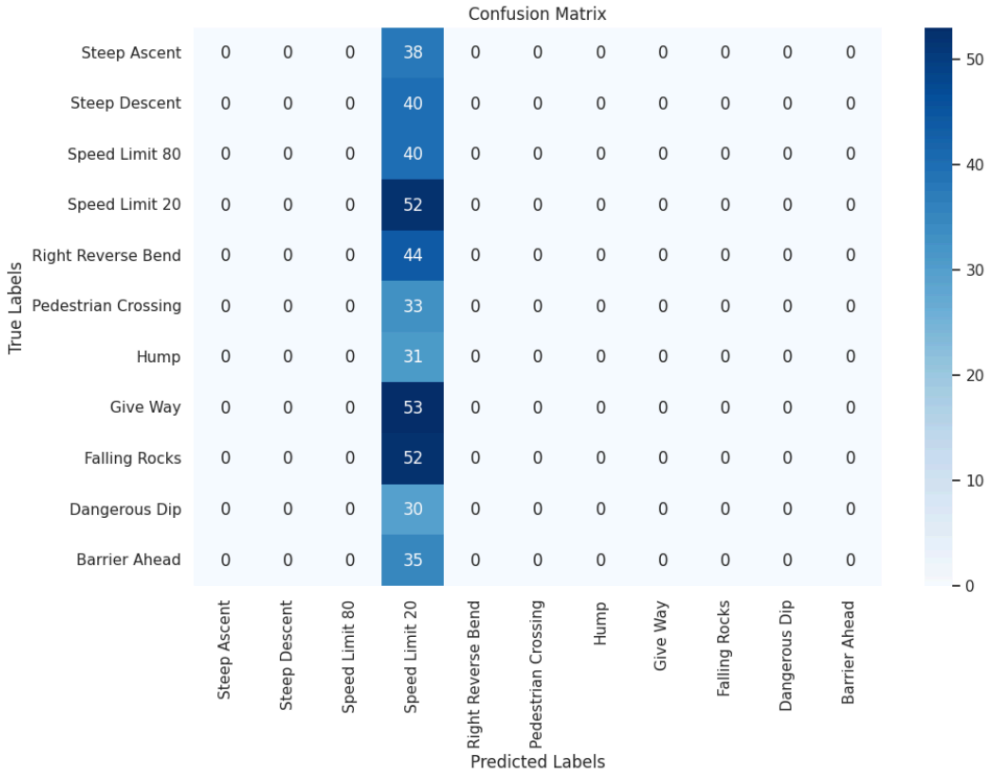


14. Ábra - Konvolúciós mátrix kezdeti modell minőségének kimutatására



15. Ábra - A végső modell konvolúciós mátrixa

Emellett olyan kezdeti modell változat is volt, ami alultanulásra utalt a konfúziós mátrix által, mivel csak egy osztályt tudott pontosan felismerni. Ez látható a 16. Ábrán.



16. Ábra - Kezdeti modell konvolúciós mátrixa alultanulásra utalva

A 17. Ábrán a végső modell kimentett értékei közül a Dice Score látható, ami minden osztályra 0.88 feletti értéket ért el. Ez is utal arra, hogy a modell jól működik.

Dice Score (DS)
0.9397590361
0.9047619048
0.9052631579
0.9285714286
0.8899521531
0.9450549451
0.962962963
0.9863013699
0.9504950495
0.9681528662
0.9333333333

17. Ábra - A végső modell Dice értékei

Statisztikai adatok kinyerése és excelbe kiírása

A statisztikai adatok kinyerésére a feladatleírásban megadott képleteket használtam fel, majd egy CSV fájlba mentettem ki, amit a már említett Google Drive részre mentettem ki, ahova a többi adatot is. A következő adatokat rögzítettem: "*True Positive (TP)*", "*False Positive (FP)*", "*False Negative (FN)*", "*True Negative (TN)*", "*Positive Predictive Value (PPV)*", "*True Positive Rate (TPR)*", "*True Negative Rate (TNR)*", "*Negative Predictive Value (NPV)*", "*Accuracy (ACC)*", "*Dice Score (DS)*". A 18. Ábrán látható értékek egy korábbi modell statisztikáit mutatja be, ahol szembetűnő, hogy bár nem rosszak az értékek, lehetnének jobbak. A 19. Ábrán a végső modell értékei láthatóak, amik jobbnak bizonyultak. Bár nincs képem róla, de volt olyan modell, ahol a korábban említett "*Dangerous Dip*" és "*Barrier Ahead*" osztályok Dice Score értékei 0.60 körül voltak.

Label	True Positive (TP)	False Positive (FP)	False Negative (FN)	True Negative (TN)	Positive Predictive V	True Positive Rate (T	True Negative Rate (	Negative Predictive '	Accuracy (ACC)	Dice Score (DS)
Steep Ascent	78	11	5	917	0.8764044944	0.9397590361	0.9881465517	0.9945770065	0.9841740851	0.9069767442
Steep Descent	60	17	10	924	0.7792207792	0.8571428571	0.9819341126	0.9892933619	0.9732937685	0.8163265306
Speed Limit 80	61	4	15	931	0.9384615385	0.8026315789	0.9957219251	0.9841437632	0.981206726	0.865248227
Speed Limit 20	89	25	6	891	0.7807017544	0.9368421053	0.9727074236	0.9933110368	0.9693372898	0.8516746411
Right Reverse Bend	118	21	23	849	0.8489208633	0.8368794326	0.975862069	0.9736238532	0.9564787339	0.8428571429
Pedestrian Crossing	108	28	10	865	0.7941176471	0.9152542373	0.9686450168	0.9885714286	0.962413452	0.8503937008
Hump	81	9	4	917	0.9	0.9529411765	0.9902807775	0.9956568947	0.9871414441	0.9257142857
Give Way	76	1	7	927	0.987012987	0.9156626506	0.9989224138	0.9925053533	0.9920870425	0.95
Falling Rocks	75	6	17	913	0.9259259259	0.8152173913	0.9934711643	0.9817204301	0.9772502473	0.8670520231
Dangerous Dip	68	3	17	923	0.9577464789	0.8	0.9967602592	0.9819148936	0.9802176063	0.8717948718
Barrier Ahead	64	8	19	920	0.8888888889	0.7710843373	0.9913793103	0.9797657082	0.9732937685	0.8258064516

18. Ábra - Korábbi modell statisztikai értékei

Label	True Positive (TP)	False Positive (FP)	False Negative (FN)	True Negative (TN)	Positive Predictive V	True Positive Rate (T	True Negative Rate (	Negative Predictive '	Accuracy (ACC)	Dice Score (DS)
Stop	78	3	7	992	0.962962963	0.9176470588	0.9969849246	0.992992993	0.9907407407	0.9397590361
Steep Descent	76	7	9	988	0.9156626506	0.8941176471	0.9929648241	0.9909729188	0.9851851852	0.9047619048
Speed Limit 80	86	12	6	976	0.8775510204	0.9347826087	0.987854251	0.9938900204	0.9833333333	0.9052631579
Speed Limit 20	130	12	8	930	0.9154929577	0.9420289855	0.9872611465	0.9914712154	0.9814814815	0.9285714286
Right Reverse Bend	93	10	13	964	0.9029126214	0.8773584906	0.9897330595	0.9866939611	0.9787037037	0.8899521531
Pedestrian Crossing	86	6	4	984	0.9347826087	0.9555555556	0.9939393939	0.995951417	0.9907407407	0.9450549451
Hump	78	1	5	996	0.9873417722	0.9397590361	0.998996991	0.995004995	0.9944444444	0.962962963
Give Way	108	2	1	969	0.9818181818	0.9908256881	0.9979402678	0.9989690722	0.9972222222	0.9863013699
Falling Rocks	96	6	4	974	0.9411764706	0.96	0.993877551	0.9959100204	0.9907407407	0.9504950495
Speed Limit 50	76	2	3	999	0.9743589744	0.9620253165	0.998001998	0.997005988	0.9953703704	0.9681528662
Barrier Ahead	105	7	8	960	0.9375	0.9292035398	0.9927611169	0.9917355372	0.9861111111	0.9333333333

19. Ábra - A végső modell statisztikai értékei

Összefoglaló

Véleményem szerint a rendelkezésre álló adatmennyiség és minőség, az eszközök hatékonyságának, illetve annak hiányának függvényében a modell jól teljesített. Valószínüleg pontosabb képekkel, illetve nagyobb adatmennyiséggel hatékonyabban tudta volna megtanulni a közlekedési táblák felismerését, ennek ellenére igazán jó eredményeket ért el. Ezen és az ehhez hasonló technológiák optimális tanulást biztosítanak, amivel az önvezető autókat is be lehet tanítani a biztonság növelésének és a balesetek elkerülésének érdekében. (Vagy éppen ennek ellenkezőjének elérésében, amennyiben valakinek rossz szándékai vannak.)

A képek listája.