XXI. reál- és humántudományi Erdélyi Tudományos Diákköri Konferencia (ETDK)
Kolozsvár, 2018. május 24–27.

Sniff!

Kutyafajta azonosító alkalmazás gépi tanulással

Szerző:

Ráduly Zalán
Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év

Témavezetők:

Sulyok Csaba, doktorandusz,
Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar
Vadászi Zsolt, szoftverfejlesztő,
Codespring Kft.
Zölde Attila, szoftverfejlesztő,
Codespring Kft.

CODESPRING
OUR CODE YOUR SUCCESS

Kivonat

A Sniff! egy gépi tanuláson alapuló alkalmazás, amely segít a felhasználóknak beazonosítani egy kutya fajtáját. Adatait a Stanford Dogs korpusz képezi, melynek segítségével a világon elterjedt százhúsz különböző kutyafajtát képes felismerni. A különböző típusok megkülönböztetésére a szoftverrendszer konvolúciós neurális hálózatokat használ, amelyek napjainkban egyre több elismerést nyernek a képfeldolgozási és -osztályozási kutatások körében.

Jelen dolgozat a Sniff! projektet mutatja be, amely két fontos komponensből áll: egy központi szerverből valamint egy okostelefon-alkalmazásból. A felhasználók a mobil alkalmazásban tudnak készíteni egy képet egy kutyáról, beazonosítani a képen megjelenő kutya fajtáját és a felismert kutyafajtáról részletes információkat szerezni.

Tartalomjegyzék

Bevezető **1**

1. Elméleti áttekintés **3**

1.1. Neurális hálózat 3

1.2. Konvolúciós neurális hálózat 5

2. Adathalmaz **7**

2.1. Előfeldolgozás 7

3. A tanítás részletei **8**

3.1. Felhasznált technológiák 8

3.2. CNN Architektúrák 9

3.3. Előfeldolgozás 9

3.4. Tanítás és hiperparaméterek 9

3.5. Fagyasztott gráf 11

4. Eredmények **12**

5. A szoftverrendszer **17**

5.1. Architektúra 17

5.2. A szerver 18

5.2.1. Képfelismerő modul 19

5.2.2. Előfeldolgozás és kiértékelés 19

5.3. A mobil kliens 20

5.3.1. Felhasznált technológiák 20

5.3.2. Komponensek 20

5.3.3. Képfelismerő modul 21

5.3.4. Előfeldolgozás és kiértékelés 21

5.3.5. Megjelenített adatok 22

5.3.6. A mobil alkalmazás működése 22

6. Következtetések és továbbfejlesztési lehetőségek **24**

Bevezető

Napjainkban a gépi tanuláson belül a neurális hálózatok több különböző kutatási és felhasználási területen el vannak terjedve: képfelismerés [19], detektálás [38], beszédfelismerés [1], adatgenerálás [12].

A képfelismerés több hagyományos módszere is el van terjedve, ezek közé tartozik a Scale-Invariant Feature Transform (SIFT) [20], a Histogram of Oriented Gradients (HoG) [5] tulajdonságok osztályozása hiba alapú lineáris vagy nem lineáris klasszifikálókkal: Support Vector Machine (SVM) vagy Softmax + Cross Entropy veszteség. [21]. A tulajdonságok kinyerésére újabban konvolúciós neurális hálózatokat használnak. Jelen dolgozat több elismert konvolúciós neurális hálózat továbbtanítási módszereit és eredményeit mutatja be a Stanford Dogs [17] adathalmazon. A megközelített probléma a képosztályozás és a „fine-grained image recognition" csoportjába tartozik, ahol egyes osztályok kinézete között nincs sok különbség.

A projekt ötlete a Kaggle Dog Breed Identification versenyéből [16] keletkezett. A Kaggle egy gépi tanulással és Big Data-val foglalkozó weboldal, ahol az említett területekkel kapcsolatos versenyek, illetve információk találhatóak meg. A verseny által biztosított adathalmaz volt kiértékelve a weboldalon megtalálható pontozási rendszerrel. Jelen dolgozatban továbbtanítási módszereket és eredményeket taglalunk. Hasonló problémát klasszikus megközelítésekkel, különböző tulajdonságok kinyerésével és főkomponens-analízissel[28] oldottak meg míg más dolgozatok különböző architektúrájú konvolúciós neurális hálózatokat használnak. Alternatív tanítási módszerekről nyújt leírást Liu et al. [22] 2016-ban. 2017-ben Howard et al. [15] egy MobileNet architektúrájú konvolúciós neurális hálózat tanítását mutatja be a Stanford Dogs adathalmazt kibővítve zajosabb adatokkal. A jelen dolgozatban bemutatott képosztályozási eredmények a változatlan Stanford Dogs adathalmazon vannak, amelynek eredményei reprodukálhatóak a jelen dolgozatban említett módszerekkel.

A dolgozatban nemcsak a konvolúciós neurális hálózatok tanítási módszereiről és eredményeiről van szó, hanem azok felhasználása is érintve van a modern technológiákban - mint egy külön modul -, ami egy szoftverrendszer által van bemutatva. A szoftverrendszeren belül egy mobil alkalmazás is található, amely a kutyabarát felhasználók számára van elkészülve. Az alkalmazás lehetőséget ad szórakoztató módon a különböző kutyafajták megtanulására, több információ szerzésére a felhasználó által készített vagy megadott kép segítségével. A szoftverrendszeren belül a kép osztályozását interneten keresztül vagy a telefon erőforrásait használva lehet elvégezni.

A Sniff! projekt fejlesztése 2017 októberében kezdődött a Codespring Mentorprogram és a csoportos projekt egyetemi tantárgy keretein belül. Kezdetben a fejlesztő csapathoz csatlakozott: Kurkó Mihály-Zsolt, Péter Ottó, Maier Kurpé Erik és Mester Attila, akik besegítettek

a fejlesztésbe egy egyetemi félév erejéig. A projekt fejlesztését a Codespring cég nagyszerű csapata erőforrásokkal és szakértéssel nagymértékben támogatja.

A projekt bemutatásra került a Mobile World Congress 2018 esemény keretein belül a Softech (Codespring) cég által.

A dolgozat 1. fejezete egy rövid áttekintővel kezdődik a neurális hálózatoktól egészen a konvolúciós neurális hálózatokig. A 2. fejezet bemutatja a felhasznált adathalmazt és annak előfeldolgozását. A 3. fejezet bemutatja az előretanított konvolúciós neurális hálózatok továbbtanítási módszereit a Stanford Dogs adathalmazon, míg a 4. fejezet ezen módszerek eredményeit tartalmazza. Részletesebb információk a szoftverrendszer elkészítéséről a 5. fejezetben olvasható. A dolgozatot a következtetések, illetve a továbbfejlesztési lehetőségek bemutatása zárja, ami a 6. fejezetben található meg.

1. Elméleti áttekintés

A fejezet egy áttekintést tartalmaz a képfelismerésben használt neurális hálózatoktól egészen a konvolúciós neurális hálózatokig.

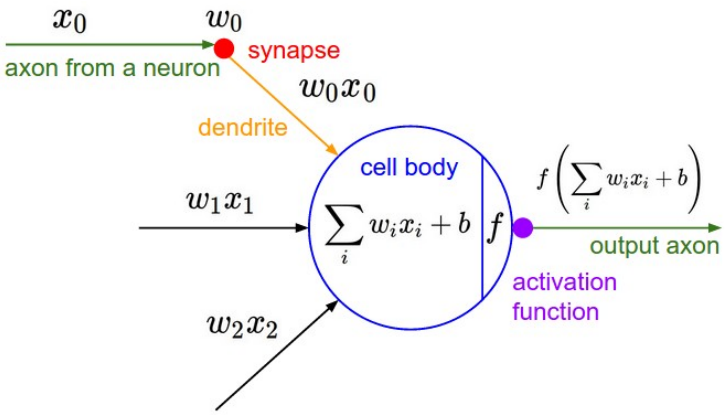
1.1. Neurális hálózat

A neurális hálózat egy modell, ami a biológiai neuron működéséből inspirálódott [23]. Napjainkban ezen hálózatok sok érdeklődést keltettek a gépi tanulás kutatási és felhasználási területein.

A neuron a neurális hálózatokban egy alapegység, ami a bemenetekből (numerikus bemenetek, súlyok, illetve bias) egy aktivációs függvény alapján kiszámolja a kimenetet (1. ábra[1]). Ezen neuronok összekötése aciklikus gráfot alkotva a neurális hálózatot képezik. A szokásos neurális hálózatoknál gyakran használt a teljesen összekötött (fully-connected) réteg vagy másnéven sűrű (dense) réteg, amelyben minden bement összeköttetésben áll minden kimenetellel egy súllyal (w) összekötetve egy nem lineáris aktivációs függvényt követve (pl. Sigmoid, Tanh, ReLU, Leaky ReLU, Maxout [2] stb.). A leggyakrabban használt aktivációs függvény az utóbbi években a Rectified Linear Unit (ReLU) [18], amely kiszűri a negatív értékeket és azt nullával helyettesíti: $f(x) = max(0, x)$ (2. ábra[1]).

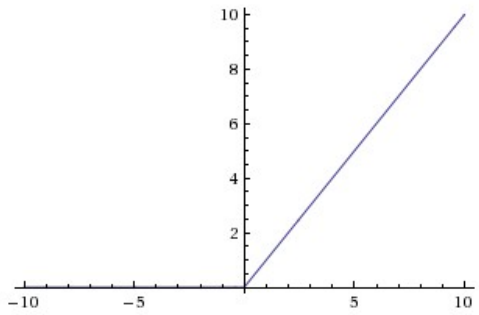
A neuronok súlyai kezdetben több lehetséges módszerrel lehetnek incializálva véletlenszerű számokkal: általában Xavier [10], MSRA [24] vagy Gaussian [50] inicializálással.

A neurális hálózatok a rétegeken keresztül könnyen kiértékelhetőek mátrix műveletek se-



1. ábra. A neuron szerkezete. Külső neuronoktól több bemenetet is kaphat, amelyekből a neuron kiszámítja a kimenetet. Minden bemenetnek van numerikus paramétere (x), egy súlya (w), amely a neurális hálózat létrejöttekor inicializálódik és a tanítás alatt beállítódnak, illetve egy bias (b), ami a súlyok értékeinek az eloszlását szabályozza. Ezekből a bemeneti adatokból a neuron kiszámítja egy aktivációs függvény alapján (jelen esetben egy lineáris osztályozó) a kimenetet.

[1]Forrás: http://cs231n.github.io/neural-networks-1, utolsó megtekintés dátuma: 2018-04-10



2. ábra. Az utóbbi években legelterjedtebb Rectified Linear Unit (ReLU) aktivációs függvény $f(x) = max(0, x)$ ábrázolása.

gítségével. Ez a folyamat az előrefele történő (feed-forward) kiértékelés, ami pontosan annyi kimenetet tartalmaz, amennyi különböző címke (reprezentálva a különböző osztályokat) van. Ezt a kimeneti réteget (output layer) más néven logits layernek is nevezik. Az utolsó rétegre egy veszteségfüggvény (SVM, Softmax + Cross Entropy) van használva aminek a kimenete egy numerikus szám, ami a veszteséget (loss) képezi. Ezt a veszteséget a súlyok függvényében minimalizálni kell, ezt a problémát gradiens leereszkedés (gradient descent) optimalizációs problémának nevezik, ami megkeresi egy függvény lokális minimumát. A gradiens egy differenciáloperátor, ami a függvények deriválásának egy általánosítása a többváltozós függvényekre. Ezeket a gradienseket a hiba-visszaterjesztéses (backpropagation) algoritmus segítségével számítódnak ki a láncszabály (chainrule) alapján a veszteség függvény kimenetelével kezdődve visszafelé egészen az első rétegig. Az utóbbi években több különböző optimalizációs eljárás is elterjedt: sztochasztikus gradiens leereszkedés (SGD), Momentum, Nesterov momentum, RMSprop, Adam és még más módszerek [39]. Az általános súlyok frissítése a következőképpen néz ki:

$$w_{\text{új}} = w_{\text{régi}} - \alpha \nabla f(w_{\text{régi}})$$

ahol w a súly, α a tanulási arány, $f(w)$ az aktivációs függvény és $\nabla f(w)$ a függvény gradiense. Ezen módszereknél közös hiperparaméter a tanulási arány (learning rate), amely a gradienst befolyásolja a frissítés során. Ha ez a hiperparaméter túl nagy vagy túl kicsi, akkor nagyon lassan fogja elérni a függvény a minimumát, ezért ajánlott a véletlenszerű keresés $(0, 1)$ intervallumban.

A neurális hálózatok tanítása lépésenként zajlik, egy lépés alatt az előrefele történő kiértékelés és a visszaterjesztéses algoritmus hajtódik végre egy konkrét optimalizációs eljárás és a megadott hiperparaméterek (tanulási arány, tanulási arány csökkenés stb.) segítségével így elérve egy teljesítményt egy adott képekből álló adathalmazon. Egy adathalmazon történik a tanítás és egy másik adathalmazon zajlik a validálás egy százalékos hatékonyságot elérve, reprezentálva, hogy hány adatot klasszifikált helyesen a neurális hálózat az adott adathalmazból.

A hatékonyság mellett még különböző metrikákat szoktak mérni: precizitás, recall (vissza-

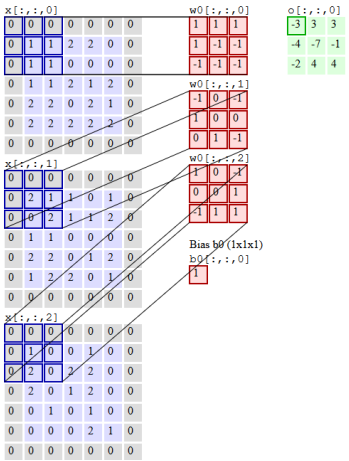
hívás), konfúziós mátrixok, stb. A precizitás a helyesen kiértékelt adatok és a helyesen és a helytelenül kiértékelt adatok összegének az aránya. A recall a helyesen kiértékelt adatok és a helyesen, illetve a helytelenül elutasított adatok aránya. A recall az adott neurális hálózat adott halmazon levő magabiztosságát is mutatja. A konfúziós mátrix, vagy más néven klasszifikációs tábla az elvárt osztályok és a ténylegesen kiértékelt osztályok besorolását jeleníti meg. Az elvárt osztályok a táblázat bal oldalán (y tengely) fentről lefelé szerepelnek és a ténylegesen kiértékelt osztály a táblázat balról jobb felé haladva (x tengely) helyezkednek el (pl. 10. ábra a 4. fejezetben).

1.2. Konvolúciós neurális hálózat

A konvolúciós neurális hálózatok hasonlóak, mint az előző alfejezetben említett hagyományos neurális hálózatok. Az optimalizációs eljárások, a klasszifikáló függvények és a tanítási módszerek ugyanúgy megjelennek itt is. A konvolúciós neurális hálózatok architektúrálisan térnek el a megszokott neurális hálózatoktól, amelyek kevesebb súly paraméterrel rendelkeznek.

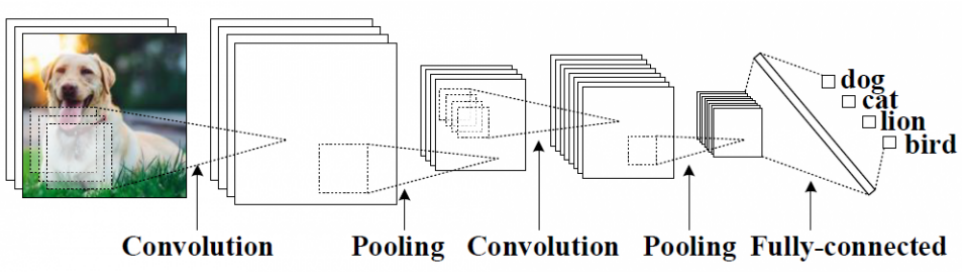
Az innovatív neurális hálózatok konvolúciós rétegekkel rendelkeznek, amelyek a bemenő adatból bizonyos tulajdonságokat nyernek ki kernelek segítségével feldolgozva az egész bemenő adatot. A kernelek vagy más néven szűrők (filterek) tanítható súly paraméterekkel rendelkező mátrixok. Egy konvolúciós rétegben több szűrő található, amelyek egyenként végig konvolválnak a bemenő adaton (3. ábra[2]).

A konvolúciós réteg egy $W_1 \times H_1 \times D_1$ dimenziójú bemenetet fogad el és négy fontos hiperparamétere van: K a szűrők száma, F a szűrő mérete, S a konvolúció során a kerettel való lépés és P a bemenő adat keret szerű kiterjesztése nullás értékekkel (a 3. ábrán a bemenetnél ez



3. ábra. A konvolúció működése. A három bemeneti csatorna, x (bal oldalt kék színnel) és az ezen konvolvált három dimenzós w szűrő (az ábrán középen piros színnel) súlyozott összege és a hozzáadott bias adja meg a kimeneti kétdimenziós mátrixot (jobb oldalt zöld színnel). A szűrő mélységét a bemeneti csatornák száma határozza meg.

[2]Forrás: http://cs231n.github.io/assets/conv-demo, utolsó megtekintés dátuma: 2018-04-10



4. ábra. Egy konvolúciós neurális hálózat szerkezete. A konvolúciós rétegek után általában egy összevonási réteget helyeznek el, ezzel csökkentve a paraméterek számát és a szükséges számításokat a hálózatban. Ezek kombinációja végén egy teljesen összekötött réteg helyezkedik el, amelynek a kimenete megegyezik az osztályok számával.

az érték 1 és keretszerűen nullás értékkel kiterjeszti az x mátrixot). Kimenete $W_2 \times H_2 \times D_2$ dimenzióval rendelkezik, ahol

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$
$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$
$$D_2 = K$$

[4] összefüggéssel számítódik ki. A kernelek lehetnek kvadratikus mátrixok vagy nem kvadratikus mátrixok, az utóbbi esetben az utóbbi összefüggés a kernelnek megfelelő méretével változik.

A konvolúciós rétegek közé általában egy összevonási réteget (pooling layer) helyeznek (4. ábra[3]), ami csökkenti a paraméterek számát és a nagymértékű számítást a hálózatban [27].

Az utóbbi években több különböző konvolúciós neurális hálózat architektúra is elterjedt: AlexNet [19], GoogLeNet (Inception modulok kifejlesztése) [44], VGGNet [42], Residual Network (ResNet) [14]. A különböző konvolúciós neurális hálózatok különböző megközelítésekkel bővítenek egy kitanulmányozott architektúrát az ImageNet [6] adathalmazra betanítva. A jelen kutatás két architektúrát tanít tovább: Inception Resnet V2 [43], illetve NASNet-A mobil [53] (amelyekről részletesebben a 3.2. fejezetben olvasható).

[3]Forrás: https://ren-fengbo.lab.asu.edu/sites/default/files/styles/panopoly_image_full/public/fpga_accel_bcnn_fig_1.png, utolsó megtekintés dátuma: 2018-04-10

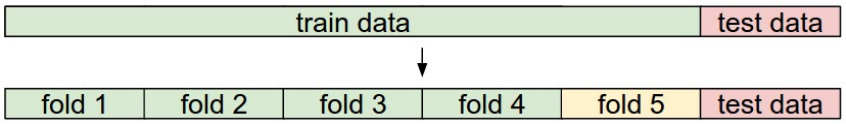
2. Adathalmaz

A jelen dolgozatban használt konvolúciós neurális hálózatok a Stanford Dogs [17] adathalmazra vannak betanítva, ami a világon elterjedt 120 különböző kutyafajtát tartalmaz és az ImageNet [6] részhalmaza. Az adathalmaz két részből tevődik össze: egy tanítási halmazból, illetve egy teszt adathalmazból. Mindkét adathalmazban különböző méretű képek jelennek meg és minden képhez egy címke van társítva, képviselve az adott képen megjelenő kutyafajtát. A tanítási adathalmaz 12.000 képet tartalmaz és minden kutyafajta ugyanolyan eloszlásban jelenik meg (fajtánként 100 adat). A teszt adathalmaz 8.580 képből tevődik össze, amelyek nem egyenletes eloszlásúak a kutyafajtákra nézve.

2.1. Előfeldolgozás

Az előfeldolgozás első lépése az adathalmaz felosztása tanítási és tesztelési halmazokra a Stanford Dogs konvenciója szerint. A felosztás után a képek a nekik megfelelő címkék mappájában találhatóak, így egy átláthatóbb mappaszerkezet keletkezik. Ezt a mappaszerkezetet a felhasznált külső könyvtárak ismerik. A felosztás során a megadott képek át vannak méretezve a felhasznált konvolúciós neurális hálózatok bemenetének megfelelően.

A konvolúciós neurális hálózat hiperparamétereinek kitapasztalására egy 5 részes keresztvalidáció (5-fold cross-validation, 5. ábra[4]) algoritmus van használva, így kapva öt különböző tanítási és validációs adathalmazt. Az öt különböző tanítási adathalmaz 9.600 adatból és a hozzá tartozó validációs adathalmaz 2.400 adatból áll. A hiperparaméterek kitapasztalása után az egész tanítási adathalmaz van használva tanításra, míg a validációs adathalmaz a teszt adatok lettek.



5. ábra. Egy gyakran használt adathalmaz felosztás, ahol a tanítási és teszt adathalmaz adott. A tanítási adathalmazt több kisebb részre (fold) osztódik fel (jelen esetben öt részre). A felosztott részek közül az egyik fold validációs adathalmaz és a többi a tanítási adathalmazt képezi, amelyek közül minden lehetséges változatot figyelembe kell venni. Ez a módszer az 5-fold cross-validáció és a hiperparaméterek megtalálására van használva. Ha megvannak a hiperparaméterek, akkor a modell kiértékelődik az ábrán jobb oldalon látható teszt adathalmazon.

[4]Forrás: `http://cs231n.github.io/classification`, utolsó megtekintés dátuma: 2018-04-10

3. A tanítás részletei

A megfelelő adatok előállítása után a konvolúciós neurális hálózatok tanítása következett. Ebben a fejezetben a tanításhoz felhasznált technológiákról, architektúrákról, módszerekről, hiperparaméterekről és a betanított konvolúciós neurális hálózat felhasználásáról lesz szó.

Ez a képfelismerés a „fine-grained recognition" osztályába tartozik, ahol az osztályok között nem sok az eltérés (például a husky és a malamut között), emiatt nem csak külső formabeli, hanem apróbb különbségekre is kell figyeljen a neurális hálózat. A fejezet bemutatja az előretanított konvolúciós neurális hálózatok továbbtanítási módszereit és eredményeit a Stanford Dogs adathalmazon.

3.1. Felhasznált technológiák

Az adatok felosztásra Python 3.5.2 [29] verziójú programozási nyelv van használva, illetve a képek gyors átméretezésére az OpenCV (Open Source Computer Vision Library) [26] 3.3.0 verziójú Python külső könyvtár nyújtott segítséget. A tanítási és validációs adathalmazokat külön-külön TFRecord [49] elnevezésű formátumban van előállítva a TensorFlow-Slim [47] Python külső könyvtár segítségével, ami a TensorFlow [48] által ajánlott bináris fájl.

A konvolúciós neurális hálózatok tanításához a TensorFlow [48] 1.4.1 verziójú Python külső könyvtára nyújtott segítséget. A TensorFlow Python, C++ és CUDA programozási nyelvben írt nyílt forráskódú gépi tanulásos keretrendszer, amelyet a Google Brain csapata fejleszti és fontos jellemzője, hogy adatfolyam programozási paradigmát használ, amelyben a program egy irányított gráfként van modellezve.

A TensorFlow-n belül konkrétan a tanításhoz a TensorFlow-Slim [47] képosztályozási könyvtár van használva, amely lehetőséget ad tanítani és kiértékelni gyakran használt konvolúciós neurális hálózatok modelljeit. Ez a könyvtár azért hasznos, mert itt megtalálhatóak előre definiált modellek implementálása és az ImageNet-en [6] betanított neurális hálózatok lementett bináris változata, amelyeket tovább- vagy újratanítani lehet.

A tanítás vizualizálásához a TensorBoard [45] nyújtotta lehetőségek vannak felhasználva, amely egy webes felületen jeleníti meg a tanulás során megadott bizonyos metrikákat (veszteség, hatékonyság stb.). A dolgozatban megjelenített neurális hálózatokkal kapcsolatos grafikonok ezzel a vizualizációs eszközzel készültek.

A neurális hálózatok tanítása kezdetben több különböző számítógépen történt, a könnyű hordozhatóság és kitelepíthetőség érdekében elkészült egy publikus Docker [7] image[5], amin megtalálhatóak a fent említett technológiák. Ahhoz, hogy a Docker konténerek az Nvidia videokártyát tudjanak használni nvidia-docker-re [25] van szükség. A Docker konténeren belüli

[5] Az image letölthető a `https://hub.docker.com/r/rzalan/tensorflow-gpu/` címről

Python állományok és a keresztvalidációs tanítás futtatására megfelelő szkriptek vannak biztosítva.

3.2. CNN Architektúrák

A dolgozat során két különböző publikus előretanított konvolúciós neurális hálózat van tovább-, illetve újratanítva: NASNet-A mobile [53] és Inception-Resnet V2 [43].

A NASNet-A architektúrája a Neural Architecture Search (NAS) keretrendszer [52] egyik megközelítésével, a Google AutoML [13] segítségével generálódott. A dolgozatban a NASNet-A architektúra kisebb továbbtanított változata van felhasználva a mobil kliensben a képek offline osztályozására.

Az Inception-Resnet V2 egy nagyon mély (több, mint 300 rétegből álló) neurális hálózat, amelyet a Google csapata készítette. A dolgozatban az Inception-Resnet V2 architektúra egy továbbtanított változata van felhasználva a szerver oldalon a képek online klasszifikálásra egy REST [9] API-n keresztül.

3.3. Előfeldolgozás

A tanítás során egy külön előfeldolgozásra van szükség, különböző neurális hálózatok architektúrájánál bizonyos előre kitapasztalt előfeldolgozási módszerek hatékonyak. Ezen módszerek két nagy kategóriába sorolhatóak: VGG, illetve Inception előfeldolgozás. A NASNet mobil, illetve Inception-Resnet V2 Inception előfeldolgozást használ:

$$f(x) = (\frac{x}{255.0} - 0.5) * 2.0$$

ahol x a bemeneti kép. Az Inception előfeldolgozás előtt a kép a distorted bounding box algoritmus [40] segítségével vágódik ki.

Tanítási előfeldolgozásként a kép véletlenszerű elforgatása ($\pm$[0-15] fok között), véletlenszerű képre közelítés vagy véletlenszerű 87.5% részben való kivágása nem segített a hatékonyság növelésén.

Validálási előfeldolgozásként az alapértelmezett 87.5% képre való közelítés, illetve Inception előfeldolgozás van használva.

3.4. Tanítás és hiperparaméterek

A neurális hálózatok kezdetben saját eszközökön történt, viszont a végső tanítása egy GeForce GTX 1080-as videokártyával, Intel(R) Core(TM) i5-6400 @ 2.70GHz processzorral és 64 GB RAM-al rendelkező Codespring cég által támogatott számítógépen történt.

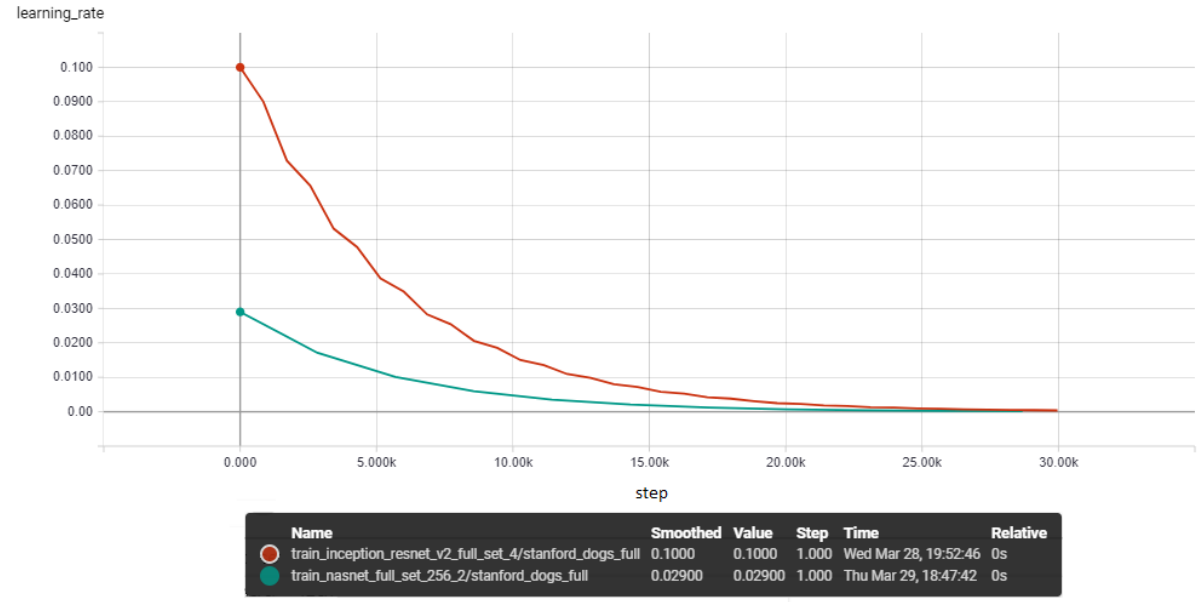
A tanítás során Softmax Cross-Entropy [21] veszteségfüggvény és Nesterov momentum [39] optimalizációs eljárás van használva a NASNet-nél és az Inception-Resnet V2-nél a továbbtanításhoz. Mivel az előretanított modellek már be voltak tanítva az ImageNet [6] adathalmazon és a Stanford Dogs [17] adathalmaz részét képezi, ezért ezek a modellek továbbtanítása szükséges az utolsó teljesen összekötött rétegre, a többi réteg fagyasztva marad, melyeknek paraméterei nem változnak.

A neurális hálózatok tanítása első sorban a hiperparaméterek kitapasztalása érdekében először a keresztvalidáció által előállított adathalmazon történt, ez után a teljes adathalmaz tanításra került és a teszt adathalmazon lett kiértékelve.

A NASNet a következő hiperparaméterekkel van betanítva: 64 batch size (egy lépés alatt ennyi képet tanít a hálózat), exponenciálisan csökkenő tanulási arány 0.029 értékkel kezdődően 3 epoch-onként (egy epoch a teljes adathalmaz tanításának megfelelő lépés, ez megközelítőleg 563 lépésnek felel meg az adathalmaznál) 0.9 csökkenéssel (6. ábra), 0.0001 súlycsökkenés (weight decay) és 30.000 lépés (step). Egy NASNet továbbtanítása megközelítőleg 2 óráig tart az említett hiperparaméterekkel és erőforrással.

Az Inception-Resnet V2 a következő hiperparaméterekkel van betanítva: 64 batch size, exponenciálisan csökkenő tanulási arány 0.1 értékkel kezdődően 3 epoch-onként 0.9 csökkenéssel (6. ábra), 0.0001 súlycsökkenés (weight decay) és 30.000 lépés (step). Egy Inception-Resnet továbbtanítása megközelítőleg 6 óráig tart az említett hiperparaméterekkel és erőforrással.

A fent említett hiperparaméterekkel lehet elérni a 4. fejezetben megjelenő hatékonysággal rendelkező neurális hálózatokat. Több különböző hiperparaméter van még kipróbálva: rögzített



6. ábra. A tanítás alatt a tanulási arány (learning rate) lépésenként exponenciálisan csökkenő grafikonja. A fenti 0.1 értékkel kezdődő az Inception-Resnet V2 neurális hálózat tanulási aránya, amíg az alsó a 0.029 értékkel kezdődő tanulási arány a NASNet tanulási aránya.

tanulási arány (0.01, 0.001 stb.), exponenciálisan csökkenő tanulási arány (0.031), alapértelmezett súlycsökkenés (0.0004) és különböző lépésszámok (15.000, 20.000, 20.500). A tanítás során a hiperparaméterek keresése véletlenszerűen történik, viszont a rácskeresés (grid search) is egy hatékony, viszont több időt igénylő módszer.

3.5. Fagyasztott gráf

A neurális hálózatokat hogy használni tudjuk API-k segítségével le kell fagyasztani, azaz egy bináris állapotba kerül, amely tartalmazza a gráf szerkezetét és a paramétereit. A tanítás során a gráf paraméterei és szerkezete gyakran nem egy fájlban tárolódik, ezért egy bináris fájl készül el a TensorFlow beépített metódusaival, amely a változókat kicseréli konstanssá, illetve kiszedi a nem használt csomópontokat a gráfból. Ez a bináris fájl könnyen kitelepíthető, illetve felhasználható a TensorFlow számára.

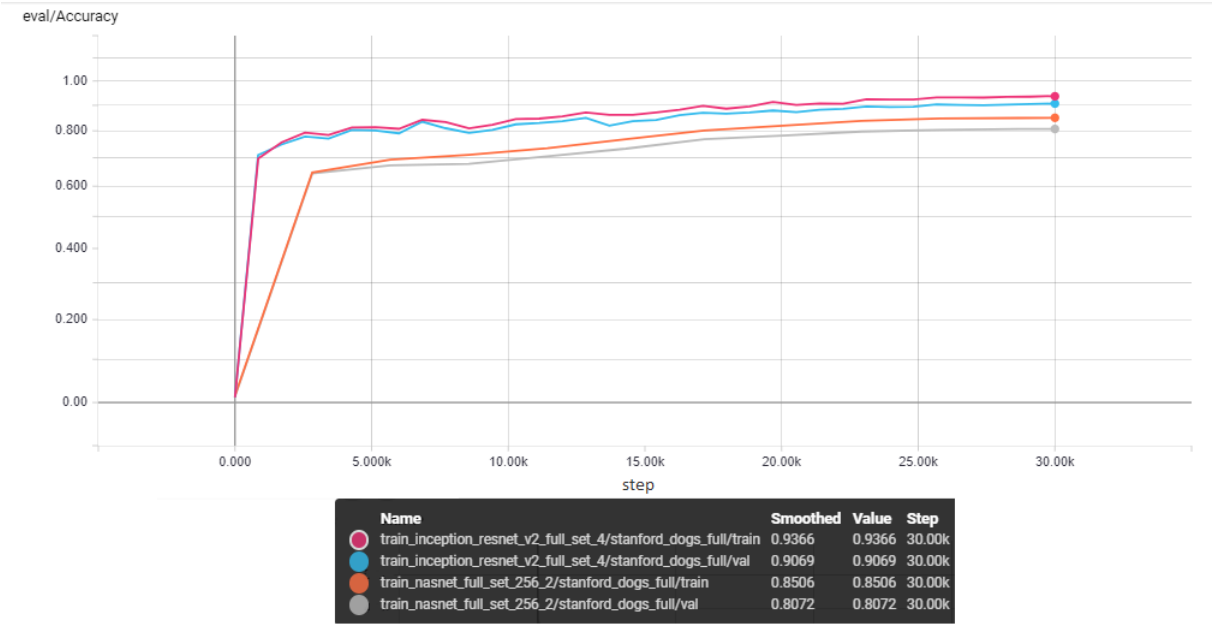
4. Eredmények

A neurális hálózatok tanítása után a hálózatok kiértékelése, tesztelése következik. Ezen értékelés bizonyos időközönként történik és különböző metrikák vannak figyelve: hatékonyság (7. ábra), precizitás (8. ábra), visszahívás (recall, 9. ábra) és konfúziós mátrixok (10. ábra).

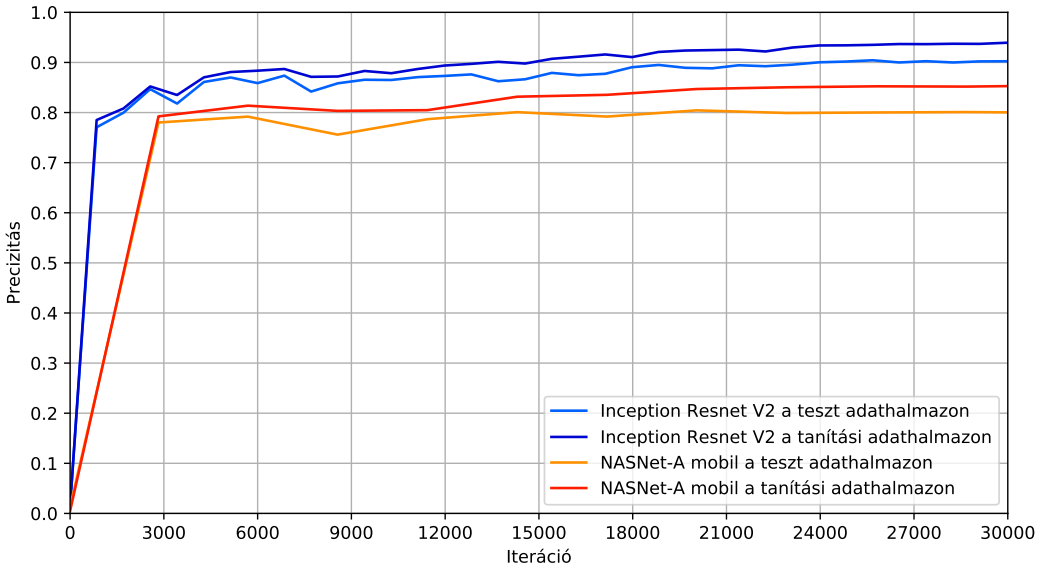
A hatékonyság a tanítási, illetve a teszt adathalmazon van mérve, amely a helyesen osztályozott képek számát jelképezi az adott adathalmazon. A NASNetnél a végső állapotban mért hatékonyság a tanítási adathalmazon 85.06%, míg a teszt adathalmazon 80.72%. Az Inception-Resnet V2-nél ezen értékek a tanítási adathalmazon 93.66%, míg a teszt adathalmazon 90.69% (7. ábra).

A precizitás és a visszahívás (recall) szintén a tanítási és a teszt adathalmazon van kiértékelve adott időközönként. Az ezen metrikák mellé szolgáltatott konfúziós mátrixok a neurális hálózat utolsó lépésében a teszt adathalmazon a tíz legrosszabban kiértékelt osztályt vizsgálják (10. ábra). Mindkét neurális hálózat nehezen tud különbséget tenni az olyan hasonló párosítások esetén, mint az eszkimó és a husky, illetve a játék uszkár és a miniatűr uszkár között. A konfúziós mátrixokból az is kiderül, hogy a NASNet-A mobil verziója nem éri el azt a hatékonyságot, amit az Inception Resnet V2, ezért több adatot osztályoz rosszul a teszt adathalmazból.

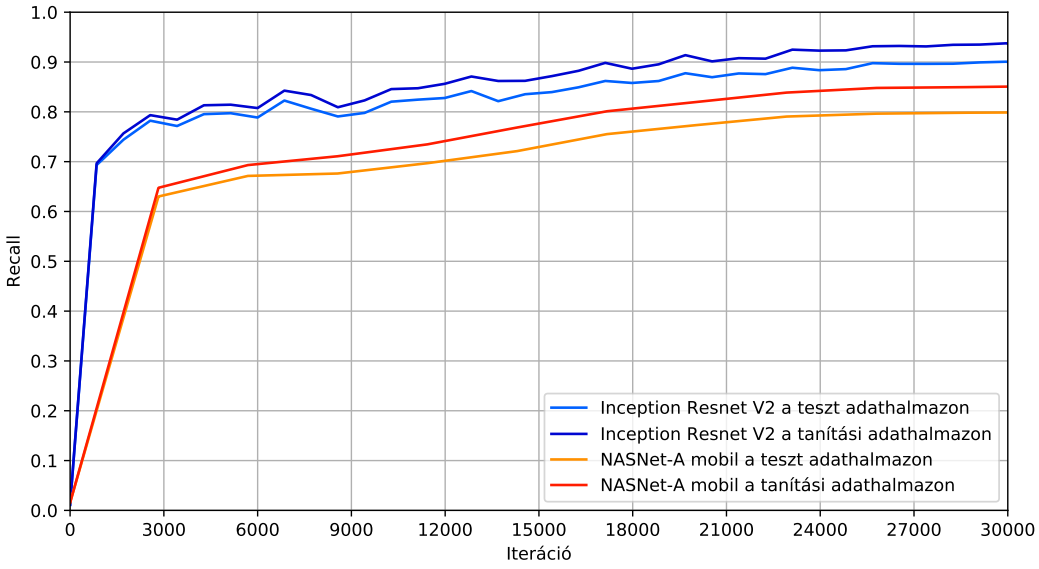
A betanított konvolúciós neurális hálózatok legjobban kiértékelt osztályait is vizsgáljuk. Az Inception Resnet V2 10 osztályt értékel ki 100%-san a teszt adathalmazból: brittany spaniel, chow, afghan hound, african hunting dog, keeshond, schipperke, bedlington terrier, sealyham terrier és curly. Ezzel szemben a NASNet-A mindössze 1 osztályt értékel ki 100%-san a teszt adathalmazból, ami a sealyham terrier.



7. ábra. Az Inception Resnet V2 és a NASNet-A mobil archetúrájú neurális hálózatok hatékonysága egyes lépésekben a tanítási és a teszt adathalmazon (fentről lefelé az említett sorrendben).

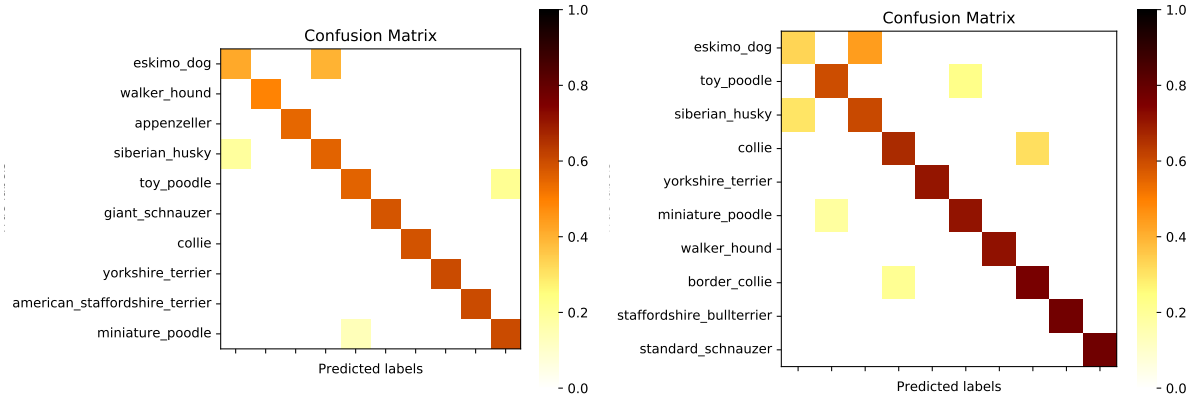


8. ábra. Az Inception Resnet V2 és a NASNet a tanítási és teszt adathalmazon levő precizitása (fentről lefelé az említett sorrendben).

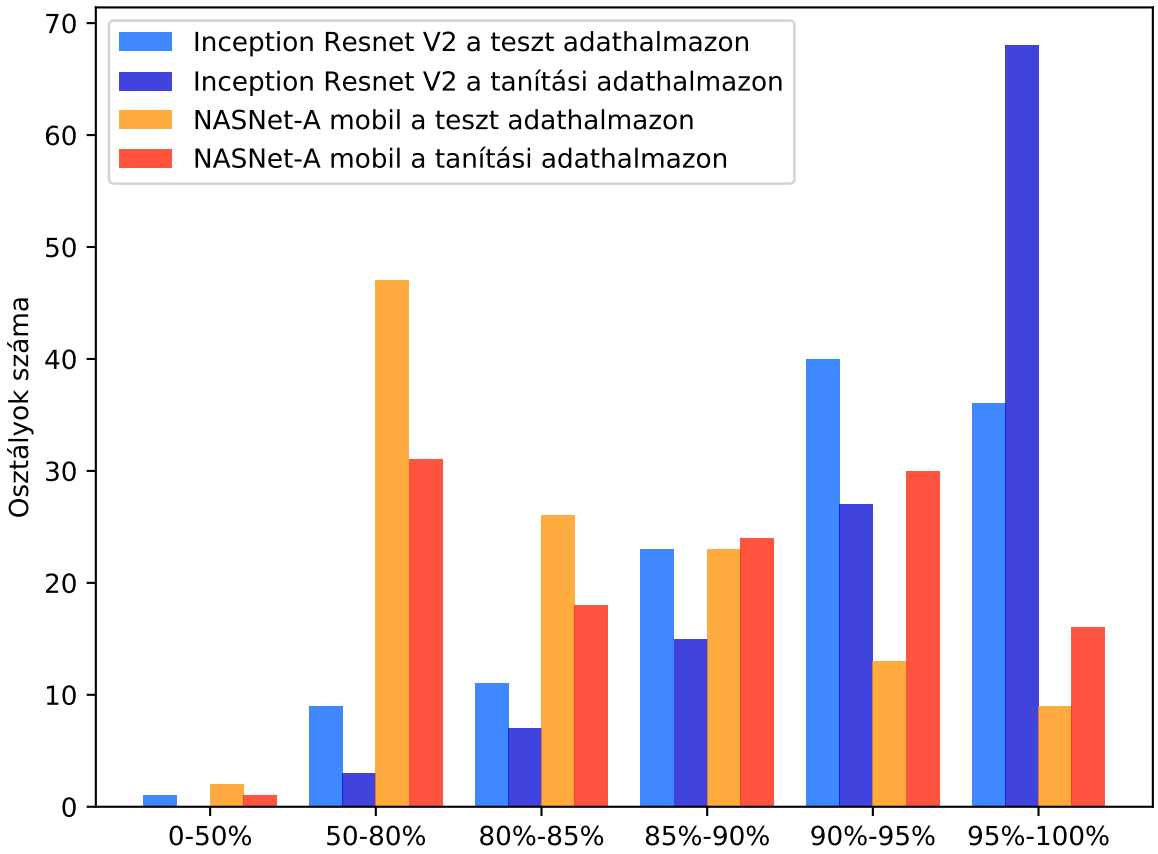


9. ábra. Az Inception Resnet V2 és a NASNet recallja a tanítási és a teszt adathalmazon (fentről lefelé az említett sorrendben).

A konfúziós mátrixok alapján végzett statisztikai elemzés kideríti, hogy hány különböző kutyafajtát képes felismerni egy bizonyos százalék intervallumban a teszt, illetve tanítási adathalmazon egy bizonyos architektúrájú konvolúciós neurális hálózat (11. ábra). A hisztogramon észrevehető, hogy az Inception Resnet V2 architektúrával rendelkező neurális hálózat a legtöbb osztályt a tanítási adathalmazból (95%, 100%] intervallumban értékeli ki, amíg a teszt adathalmazból a legtöbb osztályt a (90%, 95%] intervallumban értékeli ki. A teszt adathalmazt is jól



10. ábra. A 10 legrosszabban kiértékelt osztály a teszt adathalmazból. Bal oldalon a NASNet-A, illetve jobb oldalon az Inception Resnet V2 normalizált konfúziós mátrixa látható. Minden konfúziós mátrix y tengelyen az elvárt címkék jelennek meg és x tengelyen pedig a ténylegesen kiértékelt osztályok vannak ugyanolyan sorrendben, mint ahogy az y tengelyen fel van tüntetve.



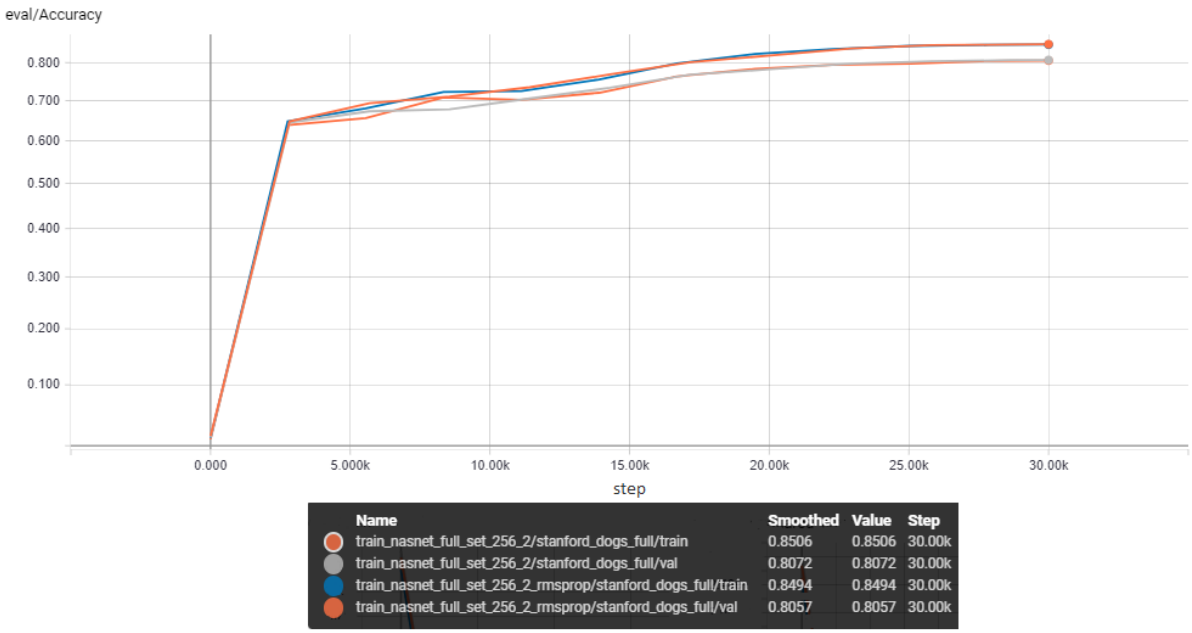
11. ábra. A különböző konvolúciós neurális hálózatoknál az osztályok kiértékelésének az eloszlása százalékos intervallumokra leosztva. Egy sáv azt mutatja meg, hogy hány osztály szerepel a kiértékelt százalék intervallumában az adott neurális hálózat architektúra és a kiértékelt adathalmaz szempontjából.

értékeli ki, viszont az ábra megmutatja, hogy vannak adatok amikre téved a hálózat. Az Inception Resnet V2 és a NASNet-A között nem csak architektúrai eltérés van, hanem hatékonyság különbség is. Az Inception Resnet V2 egy mély háló, míg a NASNet-A egy jóval kisebb architektúrával rendelkező neurális hálózat. A NASNet-A a tanítási és teszt adathalmazt tekintve kiértékelés szempontjából jobban eloszlik, mint az Inception Resnet V2 kiértékelése, több osztályt értékel ki (50%, 80%] intervallumban. Kisebb CNN architektúráknál is megfigyelhető, hogy a konvolúciós neurális hálózat képes kiértékelni bizonyos osztályokat (95%, 100%] intervallumban. Az 11. ábra nem csak a kiértékelt osztályok közötti különbséget ábrázolja, hanem a konvolúciós neurális hálózatok hatékonyságát is egy adott adathalmazra tekintve.

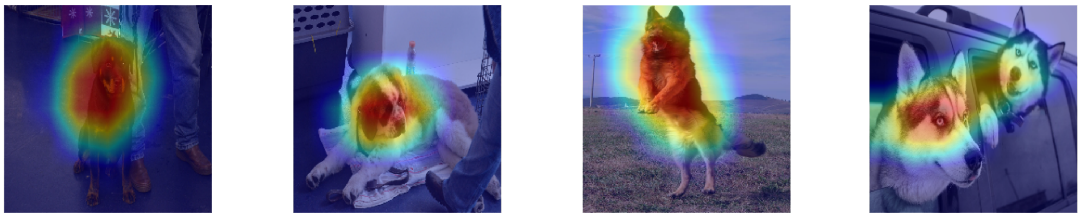
A tanítás a Nesterov momentum optimalizációs eljárással érte el a legjobb hatékonyságot, viszont más optimalizációs eljárás is ki van próbálva: RMSprop, amivel hasonló eredmények születtek (12. ábra).

A neurális hálózat kiértékelése után egy Grad-CAM [41] hőtérképes vizualizálás is megtörtént a NASNet-re. A neurális hálózat a 13. ábrán látható területekre "figyel". Grad-CAM segítségével bizonyos rétegek érdeklődési területeit lehetséges megtekinteni egy adott képen. Jelen esetben az utolsó réteg érdeklődési területei vannak egy hőtérkép segítségével ábrázolva, ahol a hideg színek a kevésébe érdekelt területek és az egyre melegebb színek a leginkább érdekelt területek.

A NASNet-A neurális hálózat a hőtérképes képek alapján inkább a kutyák fejét veszi figyelembe. A németjuhász pozíciója alapján nem csak a fejére figyel, hanem a testének különböző



12. ábra. Két különböző optimalizációs eljárással betanított NASNet hatékonysága a tanítási és a teszt adathalmazon. Jól látható az eredmény nagyon hasonló egymáshoz a tanítási és a teszt adathalmazon is.



13. ábra. Grad-CAM [41] segítségével készített hőtérkép a NASNet neurális hálózat utolsó rétegétől. A neurális hálózat a képen látható dolgokra "figyel" (kék színnel a kevésbé érdekelt területek vannak jelölve, míg a melegebb színek jobban felkeltik a neurális hálózat "figyelmét"). A megjelenített képek nem része a Stanford Dogs adathalmaznak.

pontjaira is. Ha egy képen több kutya jelenik meg, akkor a neurális hálózat bizonyos részben mindkét kutyafajtát figyelembe veszi és az alapján osztályozza a képet. Ez alapján kijelenthető az, hogy az osztályozás hatékonyságát tekintve ajánlatos egy kutyát tartalmazó képet klasszifikálni, hogy pontosabb legyen a kiértékelés. Mivel a neurális hálózat nem csak a kutyák fejére tekint, hanem egyes kutyafajtáknál bizonyos részekre is, ezért a jobb osztályozás érdekében érdemes egy jó pozícióban levő (ahol a kutyának a feje is jól látható) kutyás képet klasszifikálni. Az ábrán megtekinthető képeket a neurális hálózat helyesen osztályozza.

5. A szoftverrendszer

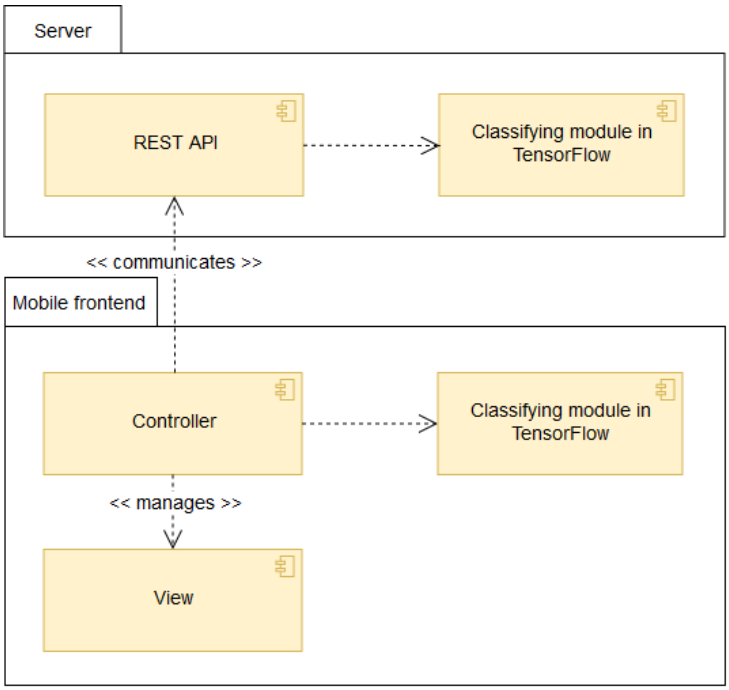
A betanított konvolúciós neurális hálózatok felhasználását tekintve egy szoftverrendszer van elkészítve, amely segítségével a felhasználók nem csak egy kép osztályozását próbálhatják ki, hanem bővebb információkat szerezhetnek a Stanford Dogs adathalmazban megjelenő százhúsz különböző kutyafajtáról.

5.1. Architektúra

Az alkalmazás architektúrája egy egyszerű szerkezeten alapszik. A rendszer fő komponensei: a szerver és a mobil alkalmazás (14. ábra). Ezek kommunikációja HTTP protokollon keresztül JSON formátumban valósul meg.

A szerver komponens tartalmazza a REST (Representational State Transfer) API megvalósítását és a osztályozó egységgel való kommunikációt, amit a TensorFlow (részletek a 3.1. fejezetben olvashatóak) keretrendszer biztosít. A REST API-n keresztül a szerver megkapja az adatot, amit továbbít a képfelismerő egység felé és ugyanezen a csatornán keresztül történik meg a válasz visszaküldése a kliens felé.

Az mobil alkalmazás felépítésében MVC minta érvényesül, ahol a vezérlő állítja a komponensek nézeteit, illetve szükség esetén használja a TensorFlow képfelismerő modulját.



14. ábra. A rendszer komponens diagramja. A szerver egy REST API-n keresztül kommunikál a kliensekkel használva a képfelismerő egységet. A mobil alkalmazás felépítésében MVC minta érvényesül.

5.2. A szerver

A Sniff! rendszer fontos komponense a Go-ban írt szerver, amely egy bonyolult és mély szerkezetű neurális hálózat segítségével tud osztályozni egy képet rövid időn belül. Ehhez egy jó videokártyával rendelkező szervergépre van szükség. Az alkalmazás idő szempontjából a leggyorsabban szerver kommunikációval működik.

A szerver Go [11] programozási nyelvben van írva. A Go egy nyílt forráskódú, a Google csapata által fejlesztett programozási nyelv, mely a C-hez hasonlít, viszont ez memória biztonságosabb, van memóriaszemét-gyűjtő mechanizmusa, illetve struktúra típusú.

A REST API megalkotására a `net/http` csomag van felhasználva, amely bizonyos elérési útvonalaknak egy handler metódust kell megfeleltetni. Szintén ennek a külső csomag segítségével lehet megmondani, hogy a szerver milyen hoszton és porton figyeljen. A hibakezelésre szintén lehetőséget nyújt ez a könyvtár, mivel lehetőséget nyújt az adott tartalom típusának ellenőrzésére képek esetében. A képek előfeldolgozására és osztályozására a TensorFlow Go API-ja [46] van használva.

A könnyű kitelepíthetőség érdekében elkészült egy-egy publikus Docker image különböző processzortípusokra: CPU[6], GPU[7] és ARM. Ezek a szerver kitelepítéshez és futtatáshoz szükséges technológiákat gyűjtik össze. A Tensorflow és Go-t társító publikus Docker image-ek hiánya diktálta a saját image-ek elkészítésének szükségességét. A gyors neurális hálózat klasszifikálás érdekében a GPU-s image az ajánlott és felhasznált.

A szerver a folyamatos integráció érdekében GitLab CI-t használ, amely során ha egy push üzenet érkezik, akkor először lefut a Golint statikus kódellenőrző. Siker esetén egy build folyamat indul el. Ha a főágon történik a folyamatos integráció és a build folyamat is sikeres, akkor a szerver automatikusan kitelepítődik és elindul a szerver a régi folyamatot leállítva `docker-compose` segítségével.

A kliensek egy REST [9] API-n keresztül kommunikálnak a szerverrel. A képek feltöltésére egy POST kérés szükséges a `host:port/image` elérési útvonalra, ahol a szerver JSON formátumban válaszol az adott kérésre. Más elérési útvonalak nem működnek, illetve ha egy nem megfelelő kérést (nincs kép vagy nem megfelelő tartalmú a küldött fájl) kap a szerver, akkor üres választ ad vissza a szerver. A hibakezelés során nem csak a fájl kiterjesztése van tekintve, hanem a POST kérésben küldött fájl tartalma is.

A szerver könnyen konfigurálható, ami egy konfigurációs fájl segítségével van megvalósítva. A konfigurációs fájlban testreszabható a szerver hosztja, hogy milyen porton figyeljen a kérésekre, a JSON formátum címkéinek az elnevezése, amely a kiértékelést válasz konvencióját határozza meg, alapértelmezett beállításként ez `label` és a hozzá tartozó `prob`.

[6] Letölthető a `https://hub.docker.com/r/rzalan/golang1.9.2-tensorflow-cpu1.4.0/` címről
[7] Letölthető a `https://hub.docker.com/r/rzalan/golang1.9.2-tensorflow-gpu1.4.0/` címről

5.2.1. Képfelismerő modul

A szerver egyik fő komponense a klasszifikáló egység, amely a Go számára elkészített TensorFlow könyvtárat használja a neurális hálózat betöltésére, a kép előfeldolgozására és kiértékelésére.

Az alapértelmezett konfiguráció alapján az Inception-Resnet V2 modellt használja a szerver az indítás során egy GPU Docker konténerben futtatva a gyors osztályozás érdekében.

Szerver oldalon két betanított neurális hálózat is be van konfigurálva: Inception-Resnet V2 és NASNet. A bekonfigurált és kiválasztott neurális hálózat architektúra a szerver indításakor betöltődik a memóriába, hogy ne a kliens kérésekor kelljen várni, míg a sok réteggel rendelkező neurális hálózat betöltődik a memóriába.

A neurális hálózatok tárolására a Google Drive szolgáltatás nyújtott segítséget. A szerver indításakor, ha még nem volt letöltve az dolgozatban említett eredményekkel rendelkező neurális hálózat, akkor az letöltődik a Google Drive-ról egy publikus címről tömörítve. Azt kicsomagolja és betölti a számítógép memóriájába.

A könnyen bővíthetőség és módosíthatóság érdekében a neurális hálózatok általánosítva van egy konfigurációs fájl segítségével, ahol egy betanított neurális hálózatot a következő tulajdonságok írnak le: a modell elnevezése kiterjesztéssel együtt, a hozzá tartozó címke fájl, a tömörített fájl elnevezése kiterjesztéssel együtt, a modell Google Drive-os elérési útvonala, a modell utolsó réteg elnevezése, a modell első réteg bemeneti mérete, egy `minThreshold` és egy `diffThreshold` ami a neurális hálózat küszöbértékeit reprezentálja egy egyszerű algoritmus számára.

Szintén ezen a konfigurációs fájlon belül van megadva, hogy maximum egy képhez hány címkét társíthat az osztályozás során egy neurális hálózat és az, hogy melyik neurális hálózat beállításait használja a szerver elindítása során.

5.2.2. Előfeldolgozás és kiértékelés

A REST API által átadott kép átalakítódik a TensorFlow-nak megfelelő `Tensor`-nak és utána végrehajtódik az Inception előfeldolgozás. Az előfeldolgozás után a már memóriába betöltött neurális hálózaton az előrefele történő módszer segítségével kiértékelődik a kép.

Az osztályozás után minden egyes kutyafajta egy bizonyos százalékban (a legtöbb 0 százalékban) jelenik meg, és ezek közül kiválasztódik a legjobb n érték és a hozzá tartozó címke, ahol n a konfigurációs fájlban van megadva. A kiválasztás után ezek az értékek rendeződnek a megfelelő címkével együtt és egy szűrésen megy keresztül, ahol a legjobban kiértékelt százalékok legalább `minThreshold` értékűek kell legyenek, illetve a legalább `diffThreshold` arányban szabad különbözzön az adott érték az előző értéktől. Az utolsó esetben az első leg-

jobb érték, ami legalább `minThreshold` ahhoz képest lesz a következő érték viszonyítva. A `minThreshold` és `diffThreshold` egy lebegőpontos konstans, ami az adott neurális hálózathoz társítva a konfigurációs fájlban található meg.

A kiválasztott érték és címke térítődik vissza a kliensnek JSON formátumban ugyanazon a csatornán keresztül, amelyen érkezett a kérés.

5.3. A mobil kliens

Az Sniff! egy informatív és tanító jelleggel rendelkező telefonos alkalmazás, amely segíti a kutyabarát érdeklődőket több információt szerezni egy-egy kutyafajtáról. Az applikáció egy érdekes módszert kínál a felhasználói számára az információszerzésre: egy adott kép alapján azonosítja be a kutyafajtát konvolúciós neurális hálózatok segítségével és megjeleníti a vele kapcsolatos információkat.

5.3.1. Felhasznált technológiák

A mobil kliens React Native [30] programozási nyelvben van írva, amely lehetőséget ad cross-platform mobil alkalmazások (Android és iOS) elkészítésére JavaScript és React segítségével. A React Native natív kódban írt változata van használva, mivel jelenleg egyes függőségek kizárólag natív változatban elérhetőek.

Az alkalmazás kinézetét a React Native Material UI [33] könyvtár segíti különböző megjelenített flexibilis komponensekkel. A React Native Camera-t [31] használja a telefon kamerájának az elérésére, míg a képek galériából való kiválasztására és azok kivágására egy adott méretben a React Native Image Crop Picker-t [32]. Ezen méret egy konfigurációs fájlban van megadva, ami alapértelmezetten 350. A beazonosított kutyafajták közötti navigációt a React Native Swiper [35] segíti, a beazonosított kutya százalékos megjelenítését a React Native Progress Circle [34] könyvtár teszi látványosabbá, illetve a kutyafajta információk mellett megjelent ikonokat a React Native Vector Icons [37] külső nyílt forráskódú könyvtár szolgálja.

A neurális hálózat betöltésére és a képek osztályozására a React Native TensorFlow [36] 0.0.1-es verziójú könyvtár egy módosított változata[8] van használva.

5.3.2. Komponensek

A React Native egy komponens alapú programozási nyelv, amelyekkel könnyen testreszabható nézeteket hozhatunk létre alkalmazásunknak. Az applikáció legfontosabb komponense a `ViewController`, amely a nézetet változtatja az applikáció működése során a felhasználó irányítása által. A kontroller öt különböző nézetet különböztet meg: kamera, a kép megjelenítése,

[8] A módosítás egy hibajavítás és optimalizáció miatt történt meg

várakozás, hiba, illetve az eredmények nézetet. Az applikáció a felsorolt nézetek között tud navigálni a felhasználó igénye szerint.

Az alkalmazás elindításakor az alapértelmezett nézet a kamera, ez a `CameraScreen` komponenssel valósul meg, amely tartalmazza a kamerát, illetve egy képet tud készíteni a segítségével. Tartalmaz még egy navigációs sávot, amelyen keresztül a galériából lehet egy képet kiválasztani.

A kiértékelés a kontroller segítségével történik a következő módon: először lekéri, hogy a telefon rendelkezik-e internetes kapcsolattal. Igenleges válasz esetén felküldi a képet a szervernek kiértékelésre, illetve várja a választ. Ha valamilyen úton-módon nem érhető el a szerver vagy sok ideig tart a kiértékelés, akkor a `ErrorScreen` komponens jelenik meg. Ha nincs internetes kapcsolata a felhasználónak, akkor a kép kiértékelése automatikusan az offline klasszifikáláson keresztül hajtódik végre a telefon erőforrásait használva. Klasszifikáció után a az `ResultScreen` jelenik meg.

5.3.3. Képfelismerő modul

Az alkalmazás fontos része az osztályozó egység, ami a React Native TensorFlow [36] külső könyvtárral valósul meg, amely a TensorFlow Java API-ját használja.

Az applikációnál két különböző klasszifikálásról lehet szó: online, illetve offline. Az online kiértékelés szerver segítségével történik, míg az offline osztályozás a telefonon történik.

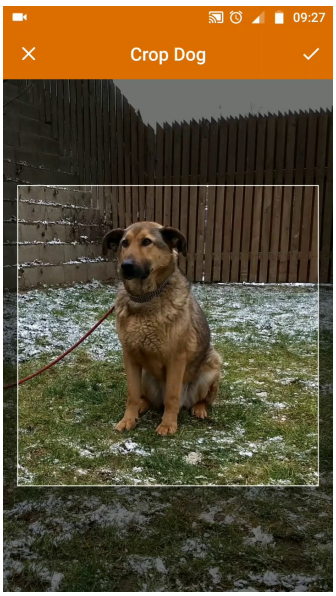
A telefonon futtatott konvolúciós neurális hálózat kis architektúrával rendelkezik és a NASNet betanított modelljét használja. A kiértékelés során a NASNet modellje betöltődik a telefon memóriájába és a telefon az erőforrásaitól függő időn belül kiértékel egy adott képet. Ezután a modell memóriája felszabadul, így nem foglalja folyamatosan a telefon memóriáját, ha háttérben futó állapotban van.

A könnyen módosíthatóság érdekében, hasonlóan a szerverhez, a CNN modellt egy konfigurációs fájl írja le. A következő tulajdonsággal rendelkezik a neurális hálózat: címke fájl, a konvolúciós neurális hálózat fájl neve, az előfeldolgozási műveletek konstansával, a legjobb n visszaterített eredménnyel, a bemeneti réteg méretével, annak nevével és az utolsó réteg nevével.

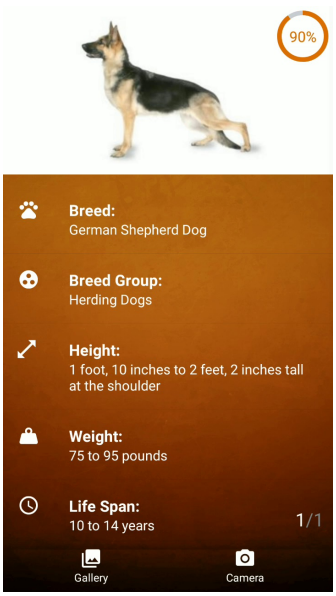
5.3.4. Előfeldolgozás és kiértékelés

A kiértékelés előtt az előfeldolgozási műveletek hajtódnak végre. A React Native TensorFlow [36] könyvtár standard normalizálás előfeldolgozási műveleteket használ két konstans segítségével: std és $mean$. A neurális hálózatok Inception előfeldolgozást használnak, viszont az Inception előfeldolgozás könnyen átalakítható standard normalizálás előfeldolgozásra egy

15. ábra. Kamera



16. ábra. Kép kivágása



17. ábra. Eredmények

egyszerű két ismeretlenes egyenlet megoldásával:

$$\frac{x - mean}{std} = (\frac{x}{255.0} - 0.5) * 2.0$$

ahol x a megadott kép, a megoldás során a következő eredmény lesz: $mean = std = 127.5$. Ez a módszer lecsökkenti egy matematikai művelettel az Inception előfeldolgozást.

Az előfeldolgozás után a könyvtárral kiértékelődik a konfigurációs fájlban beállított neurális hálózat segítségével a megadott kép és visszaadja az eredményeket, ami feldolgozásra kerül.

5.3.5. Megjelenített adatok

Az applikációban megjelenített kutyás képek és adatok a A-Z Animals [3], illetve a dogtime.com [8] weboldalakról vannak összegyűjtve web scraping módszerrel. A megjelenített adatok módosítatlanok, illetve a kiértékelt kutyafajták alján mindig megjelenik egy hivatkozás, hogy az információk pontosan melyik weboldalról vannak leszedve.

A begyűjtött adatok egy JSON formátumú fájlba vannak rendszerezve és az alkalmazás innen keresi elő a megfelelő kutyafajtához tartozó információkat.

5.3.6. A mobil alkalmazás működése

A mobil alkalmazás jelenleg Android felületen elérhető és használható. Az alkalmazás támogatja a különböző méretű Android eszközöket.

Az alkalmazás indításakor megjelenik egy nyitókép, ameddig betöltődik maga az alkalmazás. A betöltődés után megjelenik a telefon által használt kamera, illetve egy navigációs sáv, ahol a felhasználó tetszés szerint kiválaszthatja, hogy egy képet készít vagy a telefon galériá-

jából szeretne kiválasztani egy meglévő képet (15. ábra). A kép elkészülése vagy kiválasztása után a felhasználó tetszés szerint vághat a képből (16. ábra). A következő nézetben a felhasználó megtekintheti a kivágott és átméretezett képet és eldöntheti egy navigációs sáv segítségével, hogy mit szeretne csinálni. Ha a képet eldobja, akkor a kamera nézetre vált. Ha a felhasználó a kép osztályozását választja, akkor az alkalmazás automatikus eldönti, hogy interneten keresztül vagy internetkapcsolat nélkül osztályozza a képet. A osztályozási folyamat alatt az alkalmazás egy várakozási képernyőt jelenít meg. Az osztályozás után az eredmény, a hozzá tartozó információk és egy navigációs sáv jelenik meg (17. ábra).

6. Következtetések és továbbfejlesztési lehetőségek

Két különböző konvolúciós neurális hálózat architektúra volt betanítva: NASNet-A mobil architektúra, illetve egy szerverek számára kezelhető sokrétegű Inception Resnet V2 architektúra. A képfelismerés kisebb számítási igényeket használva közel 10%-os hatékonyságban különbözik a nagyobb architektúrával rendelkező konvolúciós neurális hálózat hatékonyságától.

A kitűzött céloknak megfelelően sikerült megvalósítani egy rendszert, amely képes egy adott képről felismerni egy vagy több kutyafajtát (akár internetkapcsolat nélkül is). Létrejött egy kis, illetve egy nagyobb architektúrájú konvolúciós neurális hálózat, amelynek adatbázisát a Stanford Dogs adathalmaz képezi. A nagyobb architektúrájú CNN a gyors kiértékelés érdekében egy szerveren keresztül bárki számára elérhető. A kisebb architektúrájú CNN-t a felhasználók egy mobil alkalmazásban próbálhatják ki.

A konvolúciós neurális hálózatok több lehetséges módon fejleszthetők tovább: Generative Adversarial Nets (GAN) [12] használata a tanuló adatok kibővítésére, más hibafüggvény használata, pl. center loss [51], más konvolúciós neurális hálózat architektúrák kipróbálása, a tanuló adatok kibővítése a világon megtalálható más kutyafajtákkal, detektor használata az egy képen megjelenő több kutyafajták azonosítására, szerveren és mobilon való osztályozás optimalizálása.

A mobil alkalmazás továbbfejlesztési tervei közé tartozik a funkcionalitások kibővítése: a kiértékelt képek előzményeinek eltárolása és böngészése, a különböző kutyafajta információk böngészése egy külön nézetben, lokalizáció, többnyelvűsítés, illetve az alkalmazás működőképes változatának elkészítése iOS-re.

Hivatkozások

[1] O. Abdel-Hamid et al. „Convolutional Neural Networks for Speech Recognition”. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 22. évfolyam, 10. szám (2014. okt.), 1533–1545. oldal. ISSN: 2329-9290. DOI: 10.1109/TASLP.2014.2339736.

[2] *Activation Functions*. URL: http://cs231n.github.io/neural-networks-1/#actfun (utolsó elérés dátuma: 2018. ápr. 09.)

[3] *A-Z Animals*. URL: https://a-z-animals.com/ (utolsó elérés dátuma: 2018. márc. 09.)

[4] *Convolutional Layer*. URL: http://cs231n.github.io/convolutional-networks/#conv (utolsó elérés dátuma: 2018. ápr. 10.)

[5] Navneet Dalal és Bill Triggs. „Histograms of oriented gradients for human detection”. *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. 1. évfolyam, IEEE. 2005, 886–893. oldal.

[6] J. Deng et al. „ImageNet: A Large-Scale Hierarchical Image Database”. *CVPR09*. 2009.

[7] *Docker*. URL: https://www.docker.com/ (utolsó elérés dátuma: 2018. ápr. 12.)

[8] *dogtime.com*. URL: https://a-z-animals.com/ (utolsó elérés dátuma: 2018. márc. 09.)

[9] Roy T Fielding. *Architectural styles and the design of network-based software architectures*. Évfolyam 7. University of California, Irvine Doctoral dissertation, 2000.

[10] Xavier Glorot és Yoshua Bengio. „Understanding the difficulty of training deep feedforward neural networks”. *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, 249–256. oldal.

[11] *Go programozási nyelv*. URL: https://golang.org/ (utolsó elérés dátuma: 2018. ápr. 15.)

[12] Ian Goodfellow et al. „Generative adversarial nets”. *Advances in neural information processing systems*. 2014, 2672–2680. oldal.

[13] *Google AutoML*. URL: https://research.googleblog.com/2017/05/using-machine-learning-to-explore.html (utolsó elérés dátuma: 2018. ápr. 12.)

[14] Kaiming He et al. „Deep Residual Learning for Image Recognition”. *CoRR* abs/1512.03385. évfolyam, (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[15] Andrew G. Howard et al. „MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”. *CoRR* abs/1704.04861. évfolyam, (2017). arXiv: 1704.04861. URL: http://arxiv.org/abs/1704.04861.

[16] Kaggle. *Dog Breed Identification*. URL: https://www.kaggle.com/c/dog-breed-identification (utolsó elérés dátuma: 2018. márc. 09.)

[17] Aditya Khosla et al. „Novel Dataset for Fine-Grained Image Categorization”. *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*. Colorado Springs, CO, 2011. jún.

[18] Alex Krizhevsky, Ilya Sutskever, és Geoffrey E Hinton. „ImageNet Classification with Deep Convolutional Neural Networks”. ().

[19] Alex Krizhevsky, Ilya Sutskever, és Geoffrey E Hinton. „ImageNet Classification with Deep Convolutional Neural Networks”. *Advances in Neural Information Processing Systems 25*. Szerkesztette F. Pereira et al. Curran Associates, Inc., 2012, 1097–1105. oldal. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[20] T. Lindeberg. „Scale Invariant Feature Transform”. *Scholarpedia* 7. évfolyam, 5. szám (2012). revision #153939, 10491. oldal. DOI: 10.4249/scholarpedia.10491.

[21] *Linear classification*. URL: http://cs231n.github.io/linear-classify/ (utolsó elérés dátuma: 2018. ápr. 08.)

[22] Xiao Liu et al. „Fully Convolutional Attention Localization Networks: Efficient Attention Localization for Fine-Grained Recognition”. *CoRR* abs/1603.06765. évfolyam, (2016). arXiv: 1603.06765. URL: http://arxiv.org/abs/1603.06765.

[23] Michael London és Michael Häusser. „Dendritic computation”. *Annu. Rev. Neurosci.* 28. évfolyam, (2005), 503–532. oldal.

[24] Dmytro Mishkin és Jiri Matas. „All you need is a good init”. *arXiv preprint arXiv:1511.06422* (2015).

[25] *nvidia-docker*. URL: https://github.com/NVIDIA/nvidia-docker (utolsó elérés dátuma: 2018. ápr. 12.)

[26] *OpenCV*. URL: https://opencv.org/ (utolsó elérés dátuma: 2018. ápr. 11.)

[27] *Pooling Layer*. URL: http://cs231n.github.io/convolutional-networks/#pool (utolsó elérés dátuma: 2018. ápr. 10.)

[28] P. Prasong és K. Chamnongthai. „Face-recognition-based dog-breed classification using size and position of each local part, and PCA”. *2012 9th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*. 2012. máj., 1–5. oldal. DOI: 10.1109/ECTICon.2012.6254212.

[29] *Python 3.5.2*. URL: https://www.python.org/downloads/release/python-352/ (utolsó elérés dátuma: 2018. ápr. 11.)

[30] *React Native*. URL: https://facebook.github.io/react-native/ (utolsó elérés dátuma: 2018. ápr. 15.)

[31] *React Native Camera*. URL: https://github.com/react-native-community/react-native-camera (utolsó elérés dátuma: 2018. ápr. 15.)

[32] *React Native Crop Picker*. URL: https://github.com/ivpusic/react-native-image-crop-picker (utolsó elérés dátuma: 2018. ápr. 15.)

[33] *React Native Material UI*. URL: https://github.com/xotahal/react-native-material-ui (utolsó elérés dátuma: 2018. ápr. 15.)

[34] *React Native Progress Circle*. URL: https://github.com/MrToph/react-native-progress-circle (utolsó elérés dátuma: 2018. ápr. 15.)

[35] *React Native Swiper*. URL: https://github.com/leecade/react-native-swiper (utolsó elérés dátuma: 2018. ápr. 15.)

[36] *React Native TensorFlow*. URL: https://github.com/reneweb/react-native-tensorflow (utolsó elérés dátuma: 2018. ápr. 15.)

[37] *React Native Vector Icons*. URL: https://github.com/oblador/react-native-vector-icons (utolsó elérés dátuma: 2018. ápr. 15.)

[38] Shaoqing Ren et al. „Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". *CoRR* abs/1506.01497. évfolyam, (2015). arXiv: 1506.01497. URL: http://arxiv.org/abs/1506.01497.

[39] Sebastian Ruder. „An overview of gradient descent optimization algorithms". *CoRR* abs/1609.04747. évfolyam, (2016). arXiv: 1609.04747. URL: http://arxiv.org/abs/1609.04747.

[40] *Sample distorted bounding box*. URL: https://www.tensorflow.org/api_docs/python/tf/image/sample_distorted_bounding_box (utolsó elérés dátuma: 2018. ápr. 12.)

[41] Ramprasaath R. Selvaraju et al. „Grad-CAM: Why did you say that? Visual Explanations from Deep Networks via Gradient-based Localization". *CoRR* abs/1610.02391. évfolyam, (2016). arXiv: 1610.02391. URL: http://arxiv.org/abs/1610.02391.

[42] K. Simonyan és A. Zisserman. „Very Deep Convolutional Networks for Large-Scale Image Recognition". *CoRR* abs/1409.1556. évfolyam, (2014).

[43] Christian Szegedy, Sergey Ioffe, és Vincent Vanhoucke. „Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning". *CoRR* abs/1602.07261. évfolyam, (2016). arXiv: 1602.07261. URL: http://arxiv.org/abs/1602.07261.

[44] Christian Szegedy et al. „Going Deeper with Convolutions". *CoRR* abs/1409.4842. évfolyam, (2014). arXiv: 1409.4842. URL: http://arxiv.org/abs/1409.4842.

[45] *TensorBoard*. URL: https://www.tensorflow.org/programmers_guide/summaries_and_tensorboard (utolsó elérés dátuma: 2018. ápr. 11.)

[46] *TensorFlow API Go programozási nyelvben*. URL: https://www.tensorflow.org/install/install_go (utolsó elérés dátuma: 2018. ápr. 15.)

[47] *TensorFlow-Slim*. URL: https://github.com/tensorflow/models/tree/master/research/slim (utolsó elérés dátuma: 2018. ápr. 11.)

[48] *TensorFlow-Slim*. URL: https://www.tensorflow.org/ (utolsó elérés dátuma: 2018. ápr. 11.)

[49] *TFRecord*. URL: https://www.tensorflow.org/programmers_guide/datasets (utolsó elérés dátuma: 2018. ápr. 11.)

[50] *Weight Initialization*. URL: http://cs231n.github.io/neural-networks-2/#init (utolsó elérés dátuma: 2018. ápr. 08.)

[51] Yandong Wen et al. „A discriminative feature learning approach for deep face recognition". *European Conference on Computer Vision*. Springer. 2016, 499–515. oldal.

[52] Barret Zoph és Quoc V. Le. „Neural Architecture Search with Reinforcement Learning". *CoRR* abs/1611.01578. évfolyam, (2016). arXiv: 1611.01578. URL: http://arxiv.org/abs/1611.01578.

[53] Barret Zoph et al. „Learning Transferable Architectures for Scalable Image Recognition". *CoRR* abs/1707.07012. évfolyam, (2017). arXiv: 1707.07012. URL: http://arxiv.org/abs/1707.07012.