|  |  |  |
| --- | --- | --- |
|  | **NEUMANN JÁNOS**  **INFORMATIKAI KAR** | NIK_cimer.jpg |

**SZAKDOLGOZAT**

|  |  |  |
| --- | --- | --- |
| **OE-NIK**  **2019** | Hallgató neve:  Hallgató törzskönyvi száma: | **Nagy Miklós Zoltán**  **T/005211/FI12904/N** |

Tartalomjegyzék

[1. Bevezetés 1](#_Toc40641044)

[2. Célkitűzés 2](#_Toc40641045)

[3. Irodalomkutatás 3](#_Toc40641046)

[3.1. A feladat és a tanítóminta megértése: 3](#_Toc40641047)

[3.1.1. Funkcionális Mágneses Rezonancia Képalkotás (fMRI) 3](#_Toc40641048)

[3.1.2. RDM 4](#_Toc40641049)

[3.2. Neurális hálózatok 5](#_Toc40641050)

[3.3. Aktivációs függvények 7](#_Toc40641051)

[3.3.1. A sigmoid aktivációs függvény 7](#_Toc40641052)

[3.3.2. A softmax aktivációs függvény 8](#_Toc40641053)

[3.3.3. A ReLU/LeakyReLU aktivációs függvény (Rectified Linear Unit) 8](#_Toc40641054)

[3.4. Konvolúciós Neurális Hálózatok 9](#_Toc40641055)

[3.5. Neurális Hálózati alkotó elemek, rétegek 10](#_Toc40641056)

[3.5.1. Konvolúciós Réteg (*Convolutional Layer*): 10](#_Toc40641057)

[3.5.2. Pooling Réteg (Pooling *Layer*): 11](#_Toc40641058)

[3.5.3. Dropout: 11](#_Toc40641059)

[3.5.4. Teljesen összekapcsolt réteg (Fully-Connected layer) 12](#_Toc40641060)

[3.6. Architektúrák 13](#_Toc40641061)

[3.6.1. LeNet-5 13](#_Toc40641062)

[3.6.2. Kiegészítés a hátralévő megvizsgált architektúrákhoz 13](#_Toc40641063)

[3.6.3. AlexNet 14](#_Toc40641064)

[3.6.4. VGG 16](#_Toc40641065)

[3.6.5. ResNet 17](#_Toc40641066)

[3.6.6. Sziámi hálózat 18](#_Toc40641067)

[3.7. Algonauts Kihívás Megvalósításai 19](#_Toc40641068)

[3.7.1. TOP-2: Aakash Agraval 19](#_Toc40641069)

[3.7.2. TOP-1 Agustin Lage-Castellanos, Federico De Martino 21](#_Toc40641070)

[4. Megvalósítás 23](#_Toc40641071)

[4.1 Naiv megoldás 23](#_Toc40641072)

[4.2. Klasszifikáció 24](#_Toc40641073)

[4.2.1 Miért van erre szükség? 24](#_Toc40641074)

[4.2.2 Klasszifikációs osztályok és tanítóminták 25](#_Toc40641075)

[4.2.3. Klasszifikáció – Szűkített osztályhalmazzal 27](#_Toc40641076)

[4.2.4. Klasszifikáció – Transfer learning segítségével 29](#_Toc40641077)

[4.3. A teljes modell 33](#_Toc40641078)

[4.3.1. Bemenet kezelése – Saját generátor 33](#_Toc40641079)

[4.3.2. Sziámi hálózat 34](#_Toc40641080)

[Források 37](#_Toc40641081)

[Mellékletek 40](#_Toc40641082)

[1. számú melléklet: Konzultációs napló 2019/20-I. félév 40](#_Toc40641083)

[2. számú melléklet: Algonauts Kihívás TOP-1 Eredmények 41](#_Toc40641084)

[3. számú melléklet: Sziámi hálózat abstract modellje 42](#_Toc40641085)

# 1. Bevezetés

Napjainkban a mélytanuló rendszerek az egyik legelterjedtebb kutatási és fejlesztési területnek számít. Bár a gépi tanulás már régebb óta létezett, a mélytanulást csak 1996-ban, Rina Dechter, valamint 2000-ben Igor Aizenberg és kollégái mutatták be az ezzel foglalkozó közösségek számára*.* [*[1][2]*](#_Források) A számítógépek taníthatósága az utóbbi évtizedben egyre nagyobb hangsúlyt kapott, mivel az információs korszaknak köszönhetően egyre nagyobb adatbázisok álltak rendelkezésünkre a különböző minták észrevételéhez. Fontos, hogy ez a technológia eddig korlátokba ütközött hardver szinten is, de az egyre erősebb, egyre több maggal rendelkező processzorok és egyre nagyobb memóriaterületű videókártyák széles körben való elérhetősége lehetővé tette akár a mindennapi fejlesztők otthoni, gyors és hatékony munkáját. Ennek köszönhetően pedig lehetőségünk nyílt, eddig egyesek által lehetetlennek titulált témákhoz tartozó architektúrák és programok megalkotására.

Az egyik ilyen téma az emberi agy működése, melyben az idegek csomópontjai, a neuronok milliói kommunikálnak egymássál. Ezeket az összekapcsolt neuronok csoportjait neurális hálózatnak nevezzük, melyen az utóbbi időben egyre inkább a mesterséges neurális hálózatot értjük. Ez a hálózat a biológiai működést és neurális hálózat néhány tulajdonságát modellezi, másolja le, mélytanulás alatt pedig a komplexebb neurális hálózatokat értjük. Ez a komplexitás igazából annak a függvénye, hogy mennyi információ tud átáramlani a modellen.

Köszönhetően az erősebb és gyorsabb hardvereknek láthatjuk, hogy az emberi intelligencia természetének megértése és a mesterséges intelligencia területén elért eredmények mindinkább összefonódnak. Az Algonauts Projekt összehozza a biológiai és a mesterséges intelligencia kutatóit egy közös színtérre, így pedig, az eddig egymással nem találkozó ötletek cseréjével fejlesztve ezeket a talán nem is olyan különböző kutatási területeket. [*[3]*](#_Források)

Az Algonauts Project, a Massachusetts Technológiai Intézet (MIT) vezetésével 2019 tavaszán létrehozott egy kihívást, melynek témája az Emberi Vizuális Agy megértése (Explaining the Human Visual Brain). Ehhez biztosítottak egy komplex adathalmazt a kutatók számára tizenöt kísérleti alany, orvosi műszerekkel rögzített, kilencvenkettő és száztizennyolc képre adott agyi reakcióból készített reprezentatív különbözőségi mátrix formájában.

# 2. Célkitűzés

A bevezetés végén olvasott Algonauts Project – Explaining the Human Visual Brain kihívásának teljesítése a szakdolgozatom témája. Megoldásához pedig az egyetemi tanulmányaim alatt megismert mélytanuló rendszereket fogom felhasználni, mint az emberi agy működésének megfigyeléséből nyert, majd korrelációs algoritmusok által átalakított adatok becslésére szolgáló rendszert.

Első sorban a számunkra biztosított adathalmazt fogom feldolgozni, megérteni a képek közötti lehetséges kapcsolatokat, majd pedig a mélytanuló rendszerekkel kapcsolatos ismereteim kibővítésével és a témában való kutatással szeretnék létrehozni egy olyan rendszert, ami minél pontosabb kimenetet tud prediktálni a képi bemenetek alapján.

A fenti prediktáláshoz a feladatom az, hogy egy egymással egyszerű adatokkal kommunikáló modulokból álló rendszert hozzak létre. Ezek a modulok igazából 4 fázisra lesznek oszthatók:

* Adatelőkészítés
* Mélytanulás előkészítése az adatok alapján
* A tényleges mélytanuló architektúra melynek bemenetei a képek kimenetei pedig a képekre adott agyi reakciók közötti korrelációk mértéke
* Predikció és a teljes képhalmazra adott becsült kimenetek teljes mátrixá alakítása kialakítása

Mivel az emberi agy működése nagyban hasonlít a mélytanuló rendszerek neurális hálózatainak működéséhez ( ez szolgált megtervezésének alapjául), így ezt a hasonlóságot felhasználva olyan eredményeket várunk, melyeket a későbbiekben akár az orvostudomány terén kamatoztatni tudunk. Hiszen, ha ismerjük egy átlagos ember agyának működését, annak számértékké alakítható reakcióit bizonyos képtípusok közötti különbség formájában, akkor egy esetleg valamilyen agybeli, idegi eredetű betegséggel rendelkező páciensnél szimulálni tudjuk a betegség lefolyása előtti működést. Összehasonlítva a kettőt pedig lehetőségünk van elemezni, és az adatok alapján valamilyen új típusú kezelést létrehozni.

# 3. Irodalomkutatás

## 3.1. A feladat és a tanítóminta megértése:

### 3.1.1. Funkcionális Mágneses Rezonancia Képalkotás (fMRI)

A bevezetés során olvashattunk agyi működésről, az erről készült adatok mérésére szolgáló folyamatról, valamint a szolgáltatott adatok formátumáról. Ezt követően ezekről fogunk kicsit részletesebben beszélni.

Az 1890-es években két kutató, Charles Roy és Charles Sherrington a Cambridge Egyetemen, egy kísérletként kötötték össze a világon először az agyi funkciókat az ottani vérellátás változásaival. A következő lépés az volt, hogy rá kellett jönniük hogyan mérjék meg pontosan ezeket a változásokat. 1936-ban egy másik kísérlet során megállapították, hogy az oxigén-dús vért és hemoglobinokat enyhén taszították a mágneses terek. Ezzel ellentétben pedig az oxigén csökkentett vért vonzotta a mágnesesség. Persze közel sem annyira, mint ha valamilyen fémet, például a vasat vonzaná.

1. ábra [18] Agyi aktivitás mérése fMRI-vel

Ezek alapján a kutatások alapján az AT&T Bell laboratóriumban felismerték, hogy ezzel a folyamattal a figyelésére fel tudnák használni az fMRI-t. A gép a vér mágneses tulajdonságainak változására produkált statikus változások alapján mérhető adatokkal tudott szolgálni az agy különböző területeinek aktivitásainak intenzitásáról. [*[5]*](#_Források)

### 3.1.2. RDM

Most, hogy az előző fejezetben egy kicsit pontosabb információkat szereztünk az emberi agy működéséről és az aktivitásának méréséről, nézzük meg pontosan milyen tanítómintát biztosítanak számunkra az Algonauts Kihívás szervezői.

A kutatásban résztvevő embereknek képeket mutattak, ezekre a képekre adott agyi reakciókat megmérték, ezekből az adatokból pedig kiszűrték a kutatás számára és a számunkra fontos Vizuális Agyi Lebeny működését reprezentáló adatokat. A felhasznált képeket az alábbi módon egy mátrix soraihoz és oszlopaihoz rendelték. Az ’A’ kép tartozik a mátrix első sorához és első oszlopához is, a B kép a mátrix második sorához és második oszlopához és így tovább. A mátrix egy adott cellájában a hozzá tartozó sorhoz és oszlophoz tartozó képekre adott vizuális agyi lebenyi reakciók közötti különbségek RSA (Reprezentatív Hasonlósági Analízis) segítségével feldolgozott 0 és 1 közé eső számértéket tároljuk. Ez egyszerűsítve úgy működik, hogy az agyi aktivitásból készítenek egy aktivitási vektort és az ezek közötti vektorok közötti különbségeket számolják ki az 1-korreláció képlettel. Ebből fakadóan, a mátrix főátlójában lévő értékel biztosan nullák lesznek, hiszen ugyan arra a képre adott agyi reakciókat összehasonlítva a különbség értéke nulla. Ezt a mátrixot nevezzük RDM-nek, vagyis **Reprezentatív Különbségi Mátrixnak**.

ábra 2 [[26]](#_Források) RDM kiszámítása

A kihíváshoz és a mélytanuló rendszerünk betanításához két ilyen mátrixot biztosítottak. Az egyik mátrixhoz kilencvenkét, a másikhoz pedig 118 kép tartozik – vagyis van egy 92x92-es és egy 118x118-as mátrixunk.

## 3.2. Neurális hálózatok

Az alap számítási egység az élőlények agyában a neuron. Ezekből körül-belül nyolcvanhat milliárd van az emberi neurális rendszerben, és ezeket a neuronokat nagyjából 104 – 105 szinapszis kapcsolja össze. A lentebb látható diagrammon egy emberi agyi neuronról készített rajz látható   
(1. ábra), utána pedig ennek egy általános matematikai neuron modellje (2. ábra). Minden neuron jeleket fogad az idegsejtnyúlványain keresztül és kimenő jelekké alakítja őket az (egyedüli) idegszálán. Ez az idegszál ezek után szétválik és szinapszisokkal kapcsolódik a többi neuron idegsejtnyúlványaihoz. A neuron számítási modelljében, a jelek végig utaznak az idegszálakon (pl.: **x0**) és többszörözve (pl.: **w0**\***x0**) kapcsolatba kerülnek más neuronokkal az idegsejtnyúlványokon keresztül, a szinapszis ereje alapján (pl.: **w0**).

A háttérben ez úgy néz ki, hogy a szinapszisok súlyai –pl.: **w** –, tanulhatóak és ezek kontrollálják a különböző bemenetek/kimenetek súlyozottságát és azok irányát (lehetnek pozitív és negatív súlyok is). Az alap modellben a nyúlványok eljuttatják az információkat az adott cellába, ahol összeadódnak. Az utolsó bemenet hozzáadása után, ha az ,,összeg” elért egy bizonyos küszöböt, a neuron elsül, kiküldve egy jelet az idegszálán.



ábra 2 [[24]](#_Források) Matematikai Neuron modell

ábra 3[] Biológiai Neuron modell

A számítási, matematikai modellben úgy kezeljük ezt, hogy ezeknek a továbbított jeleknek a pontos ideje lényegtelen és csak is a kilövellt jelek frekvenciája kommunikál információt a számunkra. Ennek a rátának az alapján, a neuron eltüzelését egy aktivációs függvénnyel modellezük (pl.: **f**), amely a jelek gyakoriságát reprezentálja az idegszálon.

Sokféle ilyen aktivációs függvény létezik ma már, melyek közül az egyik leggyakrabban használt talán a sigmoid függvény (pl.: **σ**), mivel ez fogja a valós bemeneti adatokat (a jelek erősségét az összeadás után) és normalizálja azokat a nulla-egy intervallumra. [*[4]*](#_Források)

## 3.3. Aktivációs függvények

Az aktivációs függvényeknek két csoportja van: a lineáris és a nem-lineáris függvények. A lineáris függvény értékei egy egyenesen helyezkednek el, vagyis a függvény egyes kimenetei fix értékkel rendelkeznek, nem pedig valamilyen nagyobb intervallumba tartoznak. Ebből már érezhető, hogy az aktivációs függvényeknek ez a verziója számunkra nem lesz megfelelő. A neurális hálózatok komplexitása és különböző paramétereinek mennyisége miatt akár a legegyszerűbb probléma megoldásához is bonyolultabb függvényekre lesz szükségünk.

A nem-lineáris függvényeket használják a legelterjedtebben az aktivációs függvények közül és segít az olyan gráfok elkészítésében is, mint amilyen az alábbi képen látható (3. ábra). Ez megkönnyíti a neurális hálózati modell számára, hogy általánosítson és adaptálódjon az bemeneti adatok alapján és hogy el tudja különíteni a kimeneteteket.

A nem-lineáris aktivációs függvényeknek a bennük lévő intervallumok és azok görbületeinek változataik alapján, több fajtájuk van.

### 3.3.1. A sigmoid aktivációs függvény

A legfontosabb tulajdonsága, és egyben az elterjedt használatának oka is, hogy a kimenete nulla és egy intervallumra képződik le. Ebből adódóan, ha egy olyan problémát kell megoldani, ahol a kimenetünk egy adott valószínűség megbecslése, értelemszerűen ezzel a függvénnyel fogunk a legtöbbször dolgozni, hiszen a valószínűség definíció szerint is egy nulla és egy közötti érték lesz.

ábra 4 [17] Sigmoid és ReLU aktivációs függvények

**Egyéb tulajdonságok:**

* differenciálható: ami azt jelenti, hogy a sigmoid görbe bármely két pontja között található görbület (slope)
* monoton növekvő, de a függvény deriváltja nem az
* előfordulhat, hogy a neurális hálózat tanítása lokális maximumba/minimumba kerül
* inkább bináris klasszifikációhoz alkalmas.

[[14]](#_Források)

### 3.3.2. A softmax aktivációs függvény

A softmax függvény szintén valószínűségekkel dolgozik, viszont ez kiszámolja az előfordulási valószínűségét minden eseménynek a többi N számú esemény függvényében. Később ezeknek a valószínűségeknek a felhasználásával könnyebb meghatározni a kimeneti osztályt a bemeneti adatok alapján.

Egyéb tulajdonságok:

* A kiszámolt valószínűségek mind a nulla-egy intervallumba tartoznak.
* A fenti leírásból adódóan az összes esemény valószínűségeinek összege egy.
* Gyakran használják többszörös klasszifikáció esetén.

### 3.3.3. A ReLU/LeakyReLU aktivációs függvény (Rectified Linear Unit)

A ReLU jelenleg a leggyakrabban használt aktivációs függvény világon. Ez amiatt lehetséges, mivel szinte az összes konvolúciós neurális hálózatban vagy deeplearning megoldásban felhasználják.

A fenti ábrából látható, hogy ez a függvény értékkészlete a nulla-végtelen intervallum, valamint, hogy a nulla-mínusz végtelen intervallumon konstans nullát ad. Mind a függvény, mind a deriváltja monoton növekvő.

A probléma ezzel az aktivációs függvénnyel, hogy minden negatív bemeneti értéket nullával helyettesít, ezáltal csökkenti a képességét a modellnek, hogy megfelelően tanuljon a bemenet alapján. Erre ad megoldást a ReLUnak egy másik változata, a Leaky ReLU. Az elnevezés a fenti (jobb oldali) ábrából értelmet is nyer. Lényege, hogy a függvény átengedi a negatív bemenetii adatokat, megszorozva egy nagyon alacsony konstans számmal és így lecsökkentve abszolút értéküket. Ez az ’a’ szám általában 0.01. Abban az esetben, ha ez a szám nem 0.01, akkor Véletlenszerű (*Randomised*) ReLU-ról beszélünk.

ábra 5 ReLU és LeakyReLU - Saját ábra

## 3.4. Konvolúciós Neurális Hálózatok

Az utóbbi időben az emberek és a gépek képességei közötti szakadék egyre inkább szűkülni látszódik. A sok terület közül, ahol kutatások és fontosabbnál fontosabb áttörések születnek, az a Számítógépes Látás. E területnek a célja az, hogy a számítógépek minél inkább úgy lássák a világot, ahogy azt az emberek teszik, valamint fel is tudják dolgozni a ,,látott” információt különbözőm feladatok elvégzésére, mint például a kép-/videó-felismerés, képek analizálása, klasszifikációja, vizuális anyagok megalkotása. Ebben a témában fejlesztéseket hoztak létre, majd tökéletesítettek, elsődlegesen egy adott algoritmus segítségével – a **Konvolúciós Neurális Hálózattal**.

A konvolúciós neurális hálózat egy olyan mélytanuló algoritmus, mely vesz egy képet bemenetnek, a rajta szereplő eltérő tulajdonságokhoz súlyokat rendel, hogy a bemeneteit meg tudja különböztetni egymástól. Az adathalmaz előzetes feldolgozásának ideje egy ilyen rendszerben sokkal alacsonyabb, összehasonlítva a többi klasszifikációs algoritmushoz képest. Amíg a primitívebb algoritmusok számára ezeket a szűrőket (amikkel megkülönböztetik a képeket) saját-kezűleg kell finomhangolnunk, addig elég tanítással, a konvolúciós hálózatoknak megvan a képessége, hogy ezeket a karakterisztikákat és filtereket megtanulja magától.

Ez a hálózat sikeresen felismeri a tér- és időbeli összefüggéseket a képeken belül az eltérő filterek alkalmazásával, köszönhetően a felhasznált paraméterek számának lecsökkentésével és a súlyok folyamatos újrafelhasználásával, amik így egyre pontosabbak lesznek. Más szavakkal: A hálózat megtanítható arra, hogy jobban megértse a kép teljes mivoltját. Nézzük meg akkor ennek a hálózatnak a különböző alkotóelemeit.

## 3.5. Neurális Hálózati alkotó elemek, rétegek

### 3.5.1. Konvolúciós Réteg (*Convolutional Layer*):

A konvolúciós réteg konvolúciós kernelekből épül fel (minden neuron egy kernelként viselkedik). Ezek a kernelek a feldolgozandó kép egy kis részével állnak kapcsolatban, amit receptív területnek hívunk. Ezek a kernelek, felosztják egymás között a képet blokkok formájában, majd konvolálják őket egy különleges súlykészlettel.

A képek ilyen kis blokkokra való felosztása és külön kezelése segít információt kinyerni az egymással kapcsolatban álló pixelcsoportokról. Ezt a csoportosított információhalmazt tulajdonság mintának is hívjuk. Az egymástól eltérő tulajdonság mintákat kinyerjük a képekből az által, hogy egy ilyen konvolúciós kernelt végig-csúsztatunk az egész képen, végig ugyan azokat a súlyokat felhasználva. A konvolúciós folyamatnak ez a súly megosztó tulajdonsága teszi a konvolúciós neurális hálózatot paraméter-hatékonnyá a teljesen összekapcsolt hálózatokhoz képest (*Fully Connected Neural Network*).

A fent említett konvolúciós kernel működését a következőképpen lehet megfogalmazni:

Ahol a bemeneti képet Ixy-al jelöljük, x és y mutatja meg a térbeli elhelyezkedést, Klk pedig az l-dik konvolúciós kernelt jelöli a k-adik rétegben.

### 3.5.2. Pooling Réteg (Pooling *Layer*):

A tulajdonsági minták, amiket kimenetként kapunk a konvolúciós folyamat során, a képen szétszórva jelenhetnek meg. Miután ezeket a tulajdonságokat kinyertük, a pontos helyzetük számunkra kevésbé lesz fontos információ, mint ezeknek a tulajdonságoknak az egymáshoz való relatív poziciója. A pooling vagy downsampling a konvolúcióhoz hasonlóan egy érdekes lokális folyamat. Összegzi a receptív területhez tartozó hasonló információkat, majd visszaadja ezek közül a legdominánsabbat az adott területen. Ez a folyamat segít kinyerni egy olyan kombinációját a tulajdonságoknak, melyek így már torzítás mentesek lesznek.

Miközben a folyamat során csökken a feature map mérete az irreleváns részek elhagyásával, nem csak a hálózat komplexitása csökken a kisebb bemenet miatt, de a hálózatunk is jobban fog általánosítani a túlillesztés csökkenése miatt.

A pooling folyamatnak különböző fajtái vannak, mint például a maximum és az átlag.

ábra 6 [[22]](#_Források) Pooling verziók bemutatása

### 3.5.3. Dropout:

A dropout behozz a regularizációt a hálózatunkba, mely hatékonyan feljavítja az általánosítást azáltal, hogy egy bizonyos (általában alacsony) valószínűséggel elhagyunk néhány egységet vagy kapcsolatot a neuronok között. A neurális hálózatokban, az olyan kapcsolatok, amelyek nem lineáris úton tanulnak egymás mellett, képesek akár egymáshoz adaptálódni, így növelve túlillesztés veszélyét.

Ez a véletlenszerű eldobása a kapcsolatoknak és egységeknek, több vékonyított hálózati architektúrát hoz létre, melyekből végül egy darab reprezentatív hálózat kerül kiválasztásra kis súlyokkal. Ezt az architektúrát pedig úgy értelmezzük, mint egy közelítését az összes többi felajánlott hálózatnak.

### 3.5.4. Teljesen összekapcsolt réteg (Fully-Connected layer)

A teljesen összekapcsolt rétegeket széleskörben használják a mai napig, olyan esetekben, mikor a bemeneti tulajdonságok adott helyhez/oszlophoz köthetőek. Például, ha az egyik népszerű esetet, a FIFA játékos ,,jóság” besoroló mélytanuló algoritmusokat vesszük figyelembe, akkor ott látható, hogy az adatbázisnak minden sora egy játékost reprezentál, és minden oszlopa az adott játékoshoz tartozó egy fix adatot ad meg nekünk. Ezek az adatok az adatbázis más oszlopában nem szerepelnek, hiszen az egy játékoshoz tartozó bemeneti adatok/tulajdonságok oszloponként hordozzák az információkat. Innen már sejthetjük is, mi lesz a fő tulajdonsága a teljesen összekapcsolt neurális hálózatoknak, és az ilyen rétegeknek.

A mélytanuló architektúrákban ez az a réteg, amely a tényleges megkülönböztető tanítást végzi el a teljes hálózatban. A bemeneti adatok teljes egészét tovább közvetíti a következő réteg számára, úgy, hogy a különböző helyen lévő tulajdonságokhoz súlyokat rendel.

Amennyiben veszünk egy képet, amely ugyebár pixelek egy csoportjaként fogható fel (pl.: 100\*100\*3 – a hárma szám itt az RGB színcsatornákra utal), minden pixelnek 0 és 255 közé eső értéke lesz, amelyet, ha egy egydimenziós sorrá szeretnénk alakítani, akkor 30000 oszlopunk lesz.

Mikor a kutatók elkezdtek ilyen egy-két réteges neurális hálózatokat összerakni a számítógépes látás fejlesztése céljából, olyan problémákba ütköztek, mint hogy a képen szereplő tárgyak nem ugyan ott voltak, a képek esetleg elfordítva, más szögből, más távolságból készültek. Vagyis a teljesen összekapcsolt rétegünknek biztosított képhez tartozó tulajdonságok nem rögzített pozícióban voltak – oszlopokhoz kötve –, hanem inkább csoportokban helyezkedtek el, a teljes képhez képest relatív helyeken. Így ez a fajta réteg a képfeldolgozáshoz hasonló esetekben önmagában nem használható, viszont a megfelelő típusú rétegekkel összhangban jó eredményeket produkálhat. [*[6]*](#_Források)

## 3.6. Architektúrák

Az elmúlt évtizedben több konvolúcós hálózati architektúra születését figyelhettük meg. Ezeknek nagy része már olyan mennyiségű réteget, információ, bonyolultságot tartalmaz, hogy a teljes modellt nehéz is elképzelnünk. Emiatt elkezdtük őket egyfajta fekete-dobozként kezelni. tudjuk mit csinálnak, milyen bemenetet várnak és milyen kimenetet biztosítanak, de az átfogó belső működésüket nem mindig szükséges teljesen átlátnunk. Ezek közül vizsgáljunk meg néhány közismert verziót*.* [*[7]*](#_Források)

### 3.6.1. LeNet-5

Négy kutató, Yann LeCun, Leon Bottou, Yosuha Bengio és Patrick Haffner mutattak be egy olyan neurális hálózati architektúrát, amely a kézzel írott és a gépi, nyomtatott karakterek felismerésére szolgált az 1990-es években. Ezt a modellt nevezték el **LeNet-5**-nek. Az architektúra elég egyértelmű és könnyen megérthető a működése, így a mai napig sokszor használják bevezetőként a konvolúciós neurális hálózatról való tanítás során.

A LeNet-5 felépítése a következő: két szett konvolúciós réteg és átlag pooling réteg, majd ezeket követve egy kisimító flatten konvolúciós réteg, két teljesen összekapcsolt réteg, végül pedig egy softmax klasszifikáló réteg.[*[8]*](#_Források)[*[9]*](#_Források)

7. ábra [21] LeNet Architektúra diagram

### 3.6.2. Kiegészítés a hátralévő megvizsgált architektúrákhoz

Mielőtt tovább haladnánk a modernebb architektúrák irányába, fontos megjegyezni egy olyan kihívás sorozatot az elmúlt évtizedből. Ezen a versenyen tűntek fel és bizonyítottak sorban a jobbnál jobb és gyakran egyszerűbb felépítéssel rendelkező neurális hálózatok és mélytanuló algoritmusok.

Ez pedig nem más, mint az **ImageNet Large Scale Visual Recognition Challenge** (ILSVRC), amely 2010 óta méretteti meg az algoritmusokat egymással szemben az objektum-felismerés és a kép-klasszifikáció területén. Hét év alatt, 2017-ig a kihíváson résztvevő algoritmusok egyre jobb eredményeket értek el. 2012 után a CNN-ek messze felülmúlták a többi algoritmus típust, végül átlépve az 5%-os hibahatárt (2015-ben), amelyet az emberek átlagosan tettek a kihívás során. [*[10]*](#_Források)

A kihívás tartalma 2017-ben:

1. Objektum lokalizálás 1000 kategóriára
2. Objektum felismerés 200 kategóriában
3. Objektum felismerés videó anyagon, 30 kategóriában

Most pedig nézzünk meg néhány olyan architektúrát mely a szakdolgozat témája szempontjából fontosabb lesz a későbbiekben, valamint, hogy időrendben milyen sikereket értek el az ImageNet kihíváson.

ábra 8: [[10]](#_Források) ImageNet-en elért eredmények 2010-2015

### 3.6.3. AlexNet

Bár a LeNet volt a kezdete a mély konvolúciós hálózatoknak, abban az időszakban a CNN-ek felhasználhatósága korlátozódott a szám és betű felismerésekre és sajnos nagyon rosszul teljesített a több osztályba tartozó képek felismerésében. Az **AlexNet**-et tartjuk az első igazi mély konvolúciós neurális hálózati architektúrának, amely akkoriban hihetetlennek számító eredményeket hozott magával a kép osztályozási és képfelismerési feladatokban.

Az AlexNet-et Alex Krizhevsky alkotta meg, aki architektúrájában felfejlesztette a tanulási kapacitását a konvolúciós neurális hálózatnak az által, hogy mélyebbé tette azt és számos paraméter optimalizáló stratégiát adott hozzá.

A LeNet-hez képest, ötről hétre növelte a tulajdonság kinyerő helyek számát, hogy még jobban el tudja különíteni egymástól a bemeneti képek osztályait. Bár a mélység növelésével a megkülönböztetésre használt tulajdonságok egyre általánosabbá válnak (avagy egyszerűbben különböztetjük meg az osztályokat), viszont ezáltal a túlillesztés problémája is egyre erősebbé válhat. Ez annyit jelent, hogy a tanító és teszt mintánkra bár nagyon jól fog működni az architektúránk, az egyéb, eddig nem látott tesztmintákra nem fog olyan jó eredményeket produkálni.

Erre a problémára adott megoldást a ReLU aktivációs függvény, amelynek használatával a tulajdonság kinyerés közben véletlenszerűen el-el hagytak elemeket, így terelve az architektúrát a robusztusabb tulajdonságok irányába. Fontos változtatás volt még a nagyobb filter méret használata a bemeneti rétegben (11x11 és 5x5), az eddigi neurális hálózatokhoz képest. Mindezekkel együtt a 16.4 %-os hibahatárt elérve az ImageNet kihíváson 2012-ben. [*[11]*](#_Források)

ábra 9: [[16]](#_Források) AlexNet absztrakt architektúra

### 3.6.4. VGG

A CNN architektúrák képfelismerésben elért sikerei után, Karen Simonyan mutatott be egy egyszerű, de annál hatékonyabb felépítési elvet egy új modellre. A **VGG** modulokra osztott réteg mintája tizenkilenc mély rétegből állt, jóval többől, mint elődjei, az AlexNet és a Zefnet. A Zefnet architektúrából tanultakat felhasználva, amely megmutatta a 2013-as ImageNet kihívás során, hogy a kisebb filterek használata javíthatja a teljesítményét a CNN-nek, a VGG lecserélte a 11x11-es és 5x5-ös filtereket egy adag 3x3-as verzióra. Ezzel pedig demonstrálta, hogy több 3x3-as konkurensen használt filter van olyan hatékony, mint a nagyméretű társaik. A kisebb filterméretnek köszönhetően csökkentek a számítási kapacitási igényei a modellnek az így keletkező kevesebb paraméterszám miatt.

A VGG a hálózat komplexitását szabályozta az által, hogy 1x1-es konvolúciókat helyezett el a konvolúciós rétegek közé, amelyek így a feature map-ek egy lineáris kombinációját tanulták meg az adott rétegek között. A hálózat finomhangolására a konvolúciós réteg után maxpooling rétegek következnek, miközben paddin segítségével megtartották a kétdimenziós méreteket. Bár ez az architektúra nem ért el győztes helyet a kihívás során, de hírnevet szerzett magának mélysége, egyszerűsége és hatékonysága miatt. Az fő hibája ennek a rendszernek viszont a magas számítási igény lett, még a kis méretű filterek felhasználásával is a paraméterszám közel 140 millióra rúgott.

ábra 10:[[19]](#_Források) VGG-16 architektúra felépítése

### 3.6.5. ResNet

A ResNet a mélyhálók egyfajta kibővítéséből jött létre. Forradalmasította a CNN architektúrák versenyét azáltal, hogy bevezetett egy új elképzelést a újrafelhasználó tanulás kapcsán és egy új, hatékonyabb metodológiát alkotott a mélyhálók tanításához.

Ez a modell 152 réteg mély konvolúciós neurális hálózatot jelentett, amely megnyerte a 2015-ös ILSVRC kihívást. A ResNet önmagában hússzor mélyebb volt, mint az Alexnet és nyolcszor mélyebb volt, mint a VGG, de mindeközben kisebb számítási komplexitást mutatott, mint bármely előző ilyen méretű neurális hálózat. Persze ezek még nem jelentették volna azt is, hogy hatékony és pontos munkára képes, de ez az 50/101/152 réteggel rendelkező architektúra kevesebbet hibázott a kép klasszifikáció területén, mint bármelyik 34 réteggel rendelkező hálózat addig. Továbbá jobb eredményt ért el a COCO nevű, híres képfelismerő dataset-en, 28%-kal pontosabb eredményt biztosítva az előző legjobbnál.

Ezek az eredmények bizonyították, hogy a CNN mélységének milyen központi szerepe van a képfelismerésben és a képeken való helymeghatározásban.

[[11]](#_Források)



ábra 11 [[20]](#_Források) ResNet „Újrafelhasználó” blokk

### 3.6.6. Sziámi hálózat

A sziámi hálózat két, egymással teljesen megegyező neurális hálózatból áll. Két különböző bemenete van abból a célból, hogy különböző mintákat tudjon összehasonlítani, aminek végül az eredményét a két hálózat közös kimenete adja meg. Két képi bemenet esetében a hálózat egyik ága az egyik, a másik ága pedig a másik képet dolgozza fel. Ezeknek a mellékágaknak a kimenete egy-egy, a programozó által meghatározott méretű tulajdonság vektor. Kimenetként ezek a vektorok kerülnek feldolgozásra. A vektorok által mutatott, a multidimenziós térben elhelyezkedő pontok helyzetét összehasonlítjuk, majd ezek távolságát kiszámolva tanítjuk a neurális hálózatunkat arra, hogy éppen például ugyan az az aláírás, arc, épület van-e két képen, avagy különbözőek.

Sejthető, hogy ez a távolság amennyiben ,,egy kategóriába tartozó” képekről beszélünk, akkor az ezekhez tartozó tulajdonságvektorok tartozó végpontok a tanítás után nagyjából ugyan arra a területre fognak korlátozódni. Így a rendszerünk szépen fokozatosan megtanulja elkülöníteni ezeket a képeket egymástól.

ábra 12 Sziámi hálózat elméleti felépítés

Fontos megjegyezni, hogy itt a rendszer nem az adott képosztályokhoz tartozó mintákat fogja megtanulni, mint egy hagyományos klasszifikáció során, hanem az egyes osztályok közötti eltérésekre fog koncentrálni.

[[27]](#_Források)

## 3.7. Algonauts Kihívás Megvalósításai

A nyáron véget ért kihívásra sok jelentkező több, különféle megoldást nyújtott be, voltak egyszerűbb algoritmikus megoldások is – ezek számunkra kevésbé érdekesek, ezért kihagyjuk őket, de volt több, kifejezetten kifinomult konvolúciós neurális hálózattal készült megoldás is. Ezekből nézzük meg most a kiemelkedőbbeket.

### 3.7.1. TOP-2: Aakash Agraval

Az indiai tudományos akadémia tagja, a IISc VisionLab számítógép erőforrásainak segítségével olyan hálózatot és megoldást alkotott, amely a 2019-es kihíváson a második helyet eredményezte.

A saját – emberi – vizuális rendszerünkhöz képest, a CNN-ek kategorizáló feladatokra való betanítása milliónyi tanító mintét követel meg és nem tudnak új kategóriákra általánosítani anélkül, hogy a modellt újra kellene tanítani. Egy alternatív megoldásként lehetőségünk van arra viszont, hogy egy kategorikus modell helyett, egy távolsági modellt tanuljunk meg. Ennek alapja, hogy az egy kategóriába tartozó képek közelebb helyezkednek el egymáshoz képest, mint a többi kategóriában lévőkhöz (lást alább a képen.)

ábra 13 Képekhez rendelt pontok a térben, Kategóriák elválasztása egymástól minták alapján[[12]](#_Források)

Ennek eléréséhez egy diszkriminatív loss függvényt használt egy Sziámi konvolúciós neurális hálózatban. A Sziámi hálózat, ahogyan azt a neve is sejteti, több, egymással párhuzamosan létező, neurális hálózatot jelent, amelyek közel azonosak és végül ezeknek az eredményei együtt befolyásolják a végső becslést. Egy új loss függvényt alkotott, amely kifejezetten ehhez a sziámi hálózathoz és a kihíváshoz tartozó különbségi értékek megbecsüléséhez készült. Agrawal finom-hangolt egy előre tanított AlexNet és VGG-16 modellt, használva a neurális különbözőségi értékeket az fMRI tanító mintákból.

Megoldásában, betanított egy sziámi hálózatot arra, hogy minimalizálja a különbséget a korrelációs távolságokon, amiket az adott neurális hálózati réteg és a megfigyelt különbözőségi értékből számolt. Ez a hálózat két olyan hálózatból épült fel, amelyek ugyan azokkal a súlyokkal dolgoznak.

ábra 14 [[13]](#_Források) Sziámi hálózat felépítése

A loss értéke a megbecsült index és a tényleges index között euklideszi távolsággal lett kiszámolva *𝐿(𝑜𝑏𝑠𝑒𝑟𝑣𝑒𝑑, 𝑝𝑟𝑒𝑑𝑖𝑐𝑡𝑒𝑑)=(||𝑑𝑝𝑟𝑒𝑑–𝑑𝑜𝑏𝑠||)2*. A két kép közötti különbséget pedig Pearson-korreláció segítségével számolja ki a hálózat által kiadott tulajdonságvektorokból.

Az fMRI adathalmazban szereplő RDM-eket finom-hangolt súlyokkal ellátott előre tanított AlexNet Sziámi hálózattal dolgozta fel. Fix tanuló rátát alkalmazott (0.005), ami mellé pedig 32-es batch méretet választott.

### 3.7.2. TOP-1 Agustin Lage-Castellanos, Federico De Martino

Az amerikai két fős csapat érte el idén a legjobb eredményeket a 2019-es Algonauts kihíváson. Ők emiatt külön ,,ajándékot” is kaptak, valamint a kihívás után rendezett workshop-okra is teljeskörűen finanszírozott meghívást is nyertek, tudományos munkájuk, módszereik és elért eredményük bemutatására.

Négy különböző megközelítést használtak megoldásuk során (kettőt az fMRI és kettőt a MEG adathalmaz felhasználásakor). Ezek közül főleg az fMRI-vel végzett munkájuk lesz számunkra a releváns. Az egyik megközelítés tisztán a képekből kinyert tulajdonságok alapján próbált becsülni, például a főbb vonalak, élek alapján, míg a másik kategorikus információkat próbál kinyerni a tanító mintából. Az ezek a modellek által becsült RDM-eket utána javították a biztosított neurális hálózatokból kinyert súlyozott átlagok segítségével.

Az első megoldási kísérletben az RDM becslésére használt modell kialakításánál 3 fő lépcsőfok volt. Először is a tanító és a teszt mintában kapott képeket rendszerezték és azonos méretűvé alakították. Másodszor, az éleket kinyerték, végül pedig gauss-simítást alkalmaztak a képeken (lásd lejjebb).

ábra 15 [[25]](#_Források) (fig1) Előfeldolgozás előtti és utáni kép

A megbecsült RDM-ket úgy számolták ki, hogy az előfeldolgozás után kapott képek közötti távolságot számolták ki, a képek közötti átfedésekből – vagyis egy mínusz SZUM(pixelek száma, melyek átfedésben vannak bármely két kép között).

A kép előfeldolgozáskor használt pixelszám, a küszöböt az élek kinyerésére és a gauss simító kernel mérete mind optimalizálásra került az alapján, hogy maximalizálják a korrelációt a becsült RDM- és a tanító mintából hozzájuk tartozó RDM-ek között.

Ezeknek az optimalizálásoknak az eredményei az fMRI adathalmazhoz:

* Felbontás: 166x166px
* Küszöb: 13
* Gauss kernel: 1

A kategorikus RDM becslésnél a kilencvenkét képből álló adathalmazba tartozó képeket manuálisan felcímkézték nyolc különböző kategóriába (saját megj.: Erre a kis adatmennyiség miatt volt lehetőség, bizonyos mennyiségű adatt felett ez manuálisan túlságosan időigényes feladat lenne.). Ezek a kategóriák: 1. objektumok-látványok, 2. állatok, 3. emberek, 4. gyümölcs-zöldség, 5. arcok, 6. kezek, 7. majom-arcok, 8. állati-arcok. (saját megj.: A majom arcok elkülönítésének biológiai oka van. Kutatások bizonyították, hogy a majmok arcát nézve az emberek hasonlóan reagálnak, mint mikor emberi arcot látnak. Lényegében ez a 3 arc-kategória könnyen elkülöníthető egymástól.) Ezek után egy Gauss Naiv Bayes klasszifikációt tanítottak meg ezeknek a kategóriáknak a teljes elkülönítésére, felhasználva alapként a képeket reprezentáló 1000 tulajdonságot visszaadó teljesen összekapcsolt neurális hálózati rétegéből a VGG-FC8 hálózatnak. Ez a módszer végül 90.22%-os pontosságot ért el a kereszt validációs tanítás során a 92-képes halmazon.

A 92-képes adathalmazt végül felhasználták egy (8x8) méretű RDM létrehozására, melyet a (92\*92) méretű RDM adataiból számoltak ki. A cellák tartalma, az adott kategóriákhoz tartozó képek közötti különbségek átlaga lett.

Ezután a teszt minta tartalmát besoroltuk a nyolc kategóriába a gauss naiv bayes algoritmusunk segítségével. Fontos észrevétel volt itt, hogy a más kategóriákba tartozó képek közötti emberi agyi reakciós különbség nagyban hasonlított az ugyan ahhoz a kategóriákhoz tartozó képek közötti különbségekre. Ezután, hogy megbecsüljük az tesztképekhez tartozó RDM-et, a két kép közötti távolságot definiáltuk úgy, mint a (8x8)-as RDM hozzátartozó cellájának tartalma. Itt lényeges, hogy a teszt adatokkal az algoritmus először a tényleges becsléskor találkozott.

A dolgozat mellékletei közül a hármas számú melléklethez tartozó képen látható a különböző megközelítésekkel elért eredmények.

# 4. Megvalósítás

## 4.1 Naiv megoldás

A feladat első tanításánál egy viszonylag egyszerű kérdésre kellett választ kapni. Vajon a számunkra biztosított tanítóminták, vagyis a 200 kép alapján, képes e egy neurális hálózat megtanulni pontosan megbecsülni a két kép közötti RDM cellák értékét. A próbálkozások kimeneténél fontos megjegyezni, hogy még nem a kihívás által számított pontértéket használtuk, valamint itt még az irodalomkutatás azon részénél jártam, ahol a kihíváson résztvevő dobogós helyezettek munkáit nem vizsgáltam át. Kizárólag az abszolút átlag különbséget (MAD) és a négyzetes különbséget vettük figyelembe a pontosságnál. Mivel a tanítás – a későbbi esetek során is– mindig két kép alapján történt egy számértékre, így a sziámi hálózat kialakítása szükséges feltétellé vált a projekt számára.

A naiv megoldásnak az alábbi ábrán látható a folyamatábrája:

ábra 16 - Naiv megoldás folyamatábra (saját)

A képek és az RDM-ek is a biztosított matlab-ban készült fájlokból lettek kinyerve. A képek ezután közösen 175x175x3px méretre lettek alakítva, végül pedig minden pixelérték normalizálva lett a nulla-egy intervallumra. Ezután a képeket párba állítottam a hozzájuk tartozó cella alapján, majd ezekhez az RDM-béli cella értékét is hozzárendeltem, így kaptam egy 3 elemből álló tanítómintát.

A következőkben több futtatást is teszteltem, viszont a tanításra használt adatmennyiség – bár a párszám magas volt (92 kép esetén is 4232 különböző képpárunk van) – kevésnek bizonyult, hiszen a rendszer nem 4232 teljesen különböző képpár alapján tanulta meg a különbségeket, hanem a 92 képből létrehozott képpárokon, melyekben így túl nagy volt az átfedés és az újra-felhasználtság.

A fenti okok miatt a feladat megoldásába bevettük a Transfer learning módszerét, melyet tesztelünk VGG16, MobileNetV2 és ResNet50 hálózatok, és az ImageNet-el elért súlyok felhasználásával. Előbbi kettő esetében az eredmény hasonlóan rossz lett. A minta mérete és a hálózat bonyolultsága miatt nem lehetett megoldani a tanítást overfitting nélkül, valamint még overfitting esetén is a rendszer egy átlageredmény becslését tanulta meg. Ezt egy egyszerű átlagszámítással ellenőriztem és a tizenöt páciens 92x92 méretű RDM értékeit átlagolva egy-két százados pontossággal becsülte ugyan azt az értéket a rendszer bemenettől függetlenül.

A különböző paraméterek, optimalizáló és aktivációs függvények kipróbálgatása alatt sikerült egy 0.3-as MSE-t (négyzet átlag különbséget) elérni a tanítás alatt, viszont a teszt adatokra való becsélésnél látszódott, hogy a rendszer rosszul működik, hiszen hiába lettek megközelítőleg jobbak és változatosak az eredmények, összességében a pontosság 0.3 és akár 0.8 között mozgott.

A harmadik lehetőséget, a ResNet hálózatait sajnos nem tudtam érdemben tesztelni, mivel a hálózat mélysége, az ebből adódó paraméterszám és a rendelkezésre álló hardverek mellett a tanítás túl lassú lett ahhoz, hogy érdemben tudjam tesztelni azt, így a későbbiekben a jobb eredményt elért VGG16 hálózatát használtam fel transfer learning esetén.

## 4.2. Klasszifikáció

### 4.2.1 Miért van erre szükség?

A kihíváson résztvevők megoldásait figyelembe véve, valamint egy orvosi egyetemre járó hallgatótársam ajánlására a kutatás azt az eredményt hozta, hogy az emberi agy hasonlóképpen működik egy képosztályozó programhoz.

Az emberi agy vizuális információfeldolgozása először megpróbálja besorolni a látóterünk által befogott ,,képet” valamilyen osztályba. Ezt úgy tudja megtenni, hogy a látott dolgot egyben dolgozza fel, majd ezen látótér egyes részeire fókuszálva aktiválja az agyunk más és más részeit. Mivel esetünkben az agynak kizárólag az fMRI-vel a vizuális agyi lebenyre fókuszált mérések álltak rendelkezésre, így a látás folyamatának első felére koncentráltam.

Ez a legjobban egy klasszifikációra hasonlít. Mikor ránézünk valamire, először felfogjuk, hogy kutyát, macskát, embert, vagy pontosabban mit is látunk, csak ezután kapcsolódik be a folyamatba a részletekre való fókusz és felismerés. Ehhez a klasszifikációhoz, hogy eldönthessük mi az, amit jól el tud különíteni az agyunk egymástól, a tanító képhalmazt vizsgálva próbáltam jól elkülöníthető csoportra osztani a képeket.

### 4.2.2 Klasszifikációs osztályok és tanítóminták

Először a projekt szempontjából releváns, különböző nehézségű képhalmazokkal próbálkoztam, hogy lássam, miként viselkedik egy neurális hálózat az tanítóminta mérete és a neurális hálózati modell bemenetét kiszolgáló generátor változásaira. A neurális hálózatot fokozatosan építettem fel aszerint, hogy hány osztályra való besorolásra tanítottam éppen. Mivel kezdetektől olyan osztályokkal próbáltam dolgozni, melyek megtalálhatóak az eredeti mintában szereplő képek között is így az alábbi osztályokra kezdtem el a tanítást először kettő, három, négy, majd pedig öt osztályra való betanítással.

**Osztályok:**

1. **human-face:** avagy emberi arc, ahol a képen felismerhetően emberi arc szerepel a fókuszban
2. **animals**: avagy állatok, ahol felismerhetően valamilyen állat szerepel a képen
3. **fruits-veggies:** itt gyümölcsökről és zöldségekről készültek a képek
4. **hands:** emberi kézfejekről készült képek
5. **objects-scenes:** kategória szempontjából azok a képek, amelyeken valamilyen tárgy vagy ,,látkép” látható és bár lehet szerepel az előző osztályok valamelyike rajtuk, az nincs fókuszban, vagyis elhanyagolható az első ránézésre való ,,felismerés” (például olyan esetben, mikor egy tengerpart, emberekkel a háttérben szerepel a fotón, a kép fókuszában maga a tengerpart van, nem pedig a rajta szereplő emberek a homokban)

A fenti osztályok azért kerültek kiválasztásra, mivel ahogy azt már említettük, jól elkülöníthetőek ránézésre egymástól, illetve ezekhez sikerült a kutatás alatt megfelelő minőségű és mennyiségű tanítóminta előkészítése ingyenes keretek között.

**Tanítóminták:**

A tanításhoz először néhány száz képet és transfer learning alapú neurális konvolúciós hálót használtam. Viszont a jobb és általánosabb pontosság eléréséhez szükségem volt a nagyságrendekkel nagyobb tanítómintára. Így miután a begyűjtött, különböző *adathalmazokhoz\** tartozó képeket szétválogattam az általam kijelölt osztályokhoz tartozó mappákba.

*\*A felsorolt osztályokhoz tartozó tanítómintákhoz tartozó linkek a beszámoló végén, a Források: Tantítóminta Források címszó alatt találhatóak.*

Ezután készítettem egy programot, mely az osztályok számától függetlenül végighalad az összes osztály összes képén és ezekből a legkisebb képszámmal rendelkező osztályhoz igazított méretű képmintát vételez random módon mindegyik osztályból.

1. num\_sample = getMinSample()
3. **for** key **in** dataset\_dir\_dict.keys():
4. **print**('\tCurrent Class:' + key)
5. key\_dir = dataset\_dir\_dict[key]
6. curr\_images = getImagesWithPath(r=key\_dir)
7. **print**('\tGot images:' + str(len(curr\_images)))
9. sample = getRandomFiles(files=curr\_images, k=num\_sample)
10. **print**('\tSamples.....DONE')
12. size = 175, 175
13. copyResizedFilesTrainValid(src=sample, dest\_key=key, size=size)
14. **print**('\*\*\*' + key + '\*\*\*')

A kódban az osztályokhoz tartozó elérési utakat egy szótár változóban tároltam el, így könnyítve meg az újabb osztályok hozzáadását a munka későbbi fázisaiban. A random kiválasztott képeket ezután a program által használt főkönyvtárba másoltam, átméretezve őket az eredeti tanítóminta alapján kiválasztott 175x175px méretre. A képek minden esetben RGB színkóddal lettek feldolgozva, mivel a transfer learning-el használt VGG16-os hálózat is eredetileg RGB képekre lett tervezve, így a maximális kihasználhatóság miatt szükség volt erre.

A képeken kezdetben alkalmaztam a kihívásban leadott megoldásokból a képelőfeldolgozás egy módszerét is. Ez a képek elhomályosítása volt Gauss-blurr segítségével, majd ezeken a képeken alkalmaztam egy élkinyerési algoritmust. A többszöri tesztelés során a legjobb eredményt az előfeldolgozott képekkel a gauss algoritmus esetén a 3x3-s kernel, az éldetektálásnál pedig a (100,100) küszöb. Az eredmény alább látható.

A képek átméretezésén kívül a többi előfeldolgozást végül a végső rendszerben nem alkalmaztam, mivel a tesztelések alapján a klasszifikáció pontosságát nem csak nem növelte, de egyes random képhalmaz-minták esetében negatívan hatott rá.

ábra 17- Gauss Blurr és Élkinyerés szemléltetése (saját)

### 4.2.3. Klasszifikáció – Szűkített osztályhalmazzal

A következőkben leírt klasszifikációra megtanított neurális hálózatot használó algoritmus egyfajta transfer learning-nek is felfogható. Lényeges különbség azonban, hogy ebben a megoldásban nem csak a hálózati rétegeket és azok tanított súlyait használtam fel, de majdnem a teljes prediktív modellt.



ábra 18- Szűkített osztályhalmazzal prediktáló klasszifikációs folyamatábra

Ahogy a fenti folyamatábrán is látható, ez a fajta algoritmus először a saját tanítóminta alapján az ImageNet kihívásban használt ezer osztályra becsült. Itt az eredmény nem maga az osztály becslése lett, hanem az utolsó többdimenziós rétegének kimenetét szedtük ki. Ezt a többdimenziós kimeneti vektort használtuk fel ezután tanító mintának a neurális hálózatunkhoz való tanításra. A saját modellünk így – már az általunk ,,leszűkített osztályhalmazra” tanult meg becsülni az előzőleg a VGG16 által becsült adatok alapján. Látható, hogy a VGG16 modelljét ebben az esetben nem is igazán a transfer learning módszerével használtam fel, mint inkább egyfajta, az adatokat előfeldolgozó algoritmusként.

1. # VGG16 modell betöltése
2. vgg16 = applications.VGG16(include\_top=False, weights='imagenet')
3. datagen = ImageDataGenerator(rescale=1. / 255)
5. batch\_size = 128
7. generator = datagen.flow\_from\_directory(
8. train\_data\_dir,
9. target\_size=(img\_width, img\_height),
10. batch\_size=batch\_size,
11. class\_mode=None,
12. shuffle=False)
14. nb\_train\_samples = len(generator.filenames)
16. bottleneck\_features\_train = vgg16.predict\_generator(generator, predict\_size\_train)
18. np.save('bottleneck\_features\_train.npy', bottleneck\_features\_train)
20. # Elmentett tulajdonságvektorok betöltése tanítómintának
21. train\_data = np.load('bottleneck\_features\_train.npy')
22. train\_labels = generator\_top.classes
24. # A tanítómintában szereplő ,,osztályvektorok" egyedi kategóriákká alakítása
25. train\_labels = to\_categorical(train\_labels, num\_classes=num\_classes)
27. # A saját neurális hálózat
28. model = Sequential()
29. model.add(Flatten(input\_shape=train\_data.shape[1:]))
30. model.add(Dense(100, activation=keras.layers.LeakyReLU(alpha=0.3)))
31. model.add(Dropout(0.4))
32. model.add(Dense(50, activation=keras.layers.LeakyReLU(alpha=0.3)))
33. model.add(Dropout(0.4))
34. model.add(Dense(num\_classes, activation='softmax'))
35. model.compile(loss='categorical\_crossentropy',
36. optimizer=optimizers.RMSprop(lr=1e-4),
37. metrics=['acc', 'loss'])
38. history = model.fit(train\_data, train\_labels,
39. epochs=120,
40. batch\_size=128,
41. validation\_data=(validation\_data, validation\_labels))

Ezt a megoldást 3, illetve 5 osztályra is teszteltem, mindkét esetben egy általam válogatott  
1660 kép/osztály tanítómintával. Az eredmény szinte már az első epoch lefutásának végére 90% feletti pontosság lett. Végül átlagosan a negyedik epoch után a validációs loss nem csökkent tovább a validációs pontosság pedig beállt 97%-ra.

Ezt a megoldást a későbbiekben viszont elvetettem, mivel bár a pontossága magas volt, új tanítómintára való betanítása, valamint a teljes algoritmussal (VGG16 előfeldolgozás és Saját modell-el) való becslés túl bonyolulttá vált ahhoz, hogy érdemben, lehetőleg az emberi hibát kizárva tudjam használni a projekt végleges formájához. Amint az az alábbi kódból is sejthető, az algoritmusnak a tesztelés fázisában volt a pontosságán kívül egy másik nagy előnye is.  
A VGG16-al-való előzetes becslést, a képek elő-feldolgozásához hasonlóan elég a tanítómintán egyszer alkalmazni, valamint új osztályok, illetve tanítóminták hozzáadásánál is csak az új részekre kell futtatni a predikciót, ezt hozzáfűzve a régebbi adatokhoz. Ez a tulajdonság azért fontos, mert egy-egy ilyen becslés már a 4 osztályba tartozó 4\*500 darab képre is 4-5 perc volt.

### 4.2.4. Klasszifikáció – Transfer learning segítségével

A különböző hálózatokkal való kísérletezés során arra jutottam, hogy a kiválasztott osztályokhoz és a hozzájuk tartozó általam összeválogatott képhalmazokhoz a VGG16 elő-tanított modellje szolgáltatja a legjobb eredményeket. Ezen felül pedig még a 175x175px képmérettel és három színcsatornával is elég gyorsan működött ahhoz, hogy több modell-architektúrát le tudjak tesztelni.

#### Fennakadás a tanítás közben

A transfer learning-el való tanítás első szakaszában 80% körüli eredményeket értem el három osztály esetében. Ezt követően megnöveltem az osztály számot ötre, vagyis az éles projekt osztályszámára. Ezzel párhuzamosan pedig megnöveltem a tanítóminta méretét is 12600 kép/osztályra, a validációs minta méretét pedig 1400 kép/osztályra.

Itt tapasztaltam a dolgozatban a legnagyobb elakadást, ugyanis a modell a képhalmaz megnövelése után leállt a tanulással, 20%-os pontosságot és hozzá tartozóan szinte konstans loss értéket produkált akár több száz epoch lefutását követően is.

A pontosságból és a loss értékéből azért következtethetünk arra, hogy a modell nem tanul, mert 5 osztály esetében 20%-os értéket kaptunk, 4 osztály esetén pedig 25%-ot, 3 osztály esetén pedig 33.33%-ot. Ezekből a tesztesetekből kiderült, hogy ilyen esetben a modell azt a valószínűséget produkálja, ami megmutatja, mennyi az esélye annak, hogy az N számú osztályhoz tartozó n számú képből pont az N1 osztályba tartozó képet húzzuk ki véletlenszerűen. Ez az érték a modell méretének változtatásával, a felhasznált ImageNet súlyok elhagyásával, illetve a modell különböző számú konvolúciós rétegének befagyasztásával sem változott.

A probléma megoldásához végül az optimalizáló függvény kicserélése hozta meg az első eredményeket. Tesztelés céljából felhasználtam egy egyszerű Stochastic Gradient Descent   
(avagy SGD) optimalizálót – ez egyébként nagyon rossz választás neurális hálóval való osztályokba sorolás esetén. A rendszer ebben az esetben elkezdett tanulni, és a tizedik epoch környékén minden futtatásnál elérte a 40%-os pontosságot. Mikor utánanéztem, hogy az SGD miért futhat jobban, mint a nála a feladattal sokkal kompatibilisebb társai – az ADAM vagy akár az RMSProp optimalizálók, rájöttem, hogy a megoldás a learning rate megváltoztatása lesz. A learning rate az az érték, amivel a rendszerünk a neuronokhoz tartozó súlyokat változtatja annak függvényében, hogy a loss értékét csökkentse.

A problémát az okozta, hogy ADAM és RMSProp esetében is a learning rate-et általában 0.01 és 0.005 között változtattam. Ezek az értékek általában megfelelőek egy tanítás során ehhez a fajta hálózathoz, valamint a VGG16 modelljét felhasználók is ezt az intervallumot ajánlották, mint best practice. Esetemben viszont azért kaptam semmilyen változást a tanítás pontoságában, mivel ez az intervallum is túl nagy értékeket tartalmazott. Kiderült, hogy a komplexebb optimalizálók esetében a megfelelő learning rate érték drasztikusan változik annak függvényében, mekkora tanítómintával dolgozunk. Alább látható, hogy a [0.01, 0.005] intervallumot a pontosság növeléséhez lejjebb kellett tolni a [0.0001, 0.00005] intervallumra.

Eredmények a learning rate drasztikus megváltoztatásával egy epoch után:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | Tanító | | Validáció | |
| Optimalitáló | Learningrate | loss | acc | loss | acc |
| ADAM | 0.00001 | 0.2219 | 0.9216 | 0.0704 | 0.9768 |
| RMSProp | 0.00001 | 0.2044 | 0.9313 | 0.1007 | 0.9647 |
| SGD | 0.01 | 0.1646 | 0.9442 | 0.0649 | 0.9776 |

#### A modell kész formája

A tanításhoz 175x175px méretű képeket használtam, így a modell mélysége és a tanítás sebessége is nagyban függött nem csak a tanítóminta méretétől, de a rendelkezésemre álló hardvertől is. A tesztelési időszakban a konvolúciós hálózat mélységét változtatva azt tapasztaltam, hogy a VGG16 eredeti hálózatánál kisebb hálózat nem tanul olyan jól, mint amennyivel gyorsabb a futtatás, a hozzáadott konvolúciós rétegekkel növel mélység esetében pedig bár a futtatási idő nagyban változott a megnövekedett paraméterszám miatt, érdemi javulást ez a lehetőség sem mutatott. Egy bizonyos mélység után a rendszer volt, hogy pontatlanabb eredményekkel szolgált a nála egyszerűbb hálózatoknál. Az utóbbi romlás betudható annak, hogy a felhasznált képek és a teljes adathalmaz nem volt olyan komplex, hogy az átlagosnál több konvolúciós réteget alkalmazó hálózatok jó eredményt produkáljanak.

A VGG16 modelljét betöltéskor úgy állítottam be a paramétereket, hogy az ImageNet-es becslésnél használt súlyokat alkalmazza. Mivel a neurális hálózatok – képek alapján való tanulás esetében – az első néhány rétegcsoportban főleg a fő tulajdonságokat, motívumokat tanulják meg felismerni, mint például a képeken szereplő élek, így a hálózat első tíz rétegénél a súlyok tanítását befagyasztottam. Ez azt jelenti, hogy ezeken a rétegeken az általam végzett tanítás végén is az eredeti súlyok maradnak meg és kizárólag azok a neuronok tanulnak majd a futtatás alatt, amelyek az ezt követő rétegekben szerepelnek. A modell így az alábbi rétegek szerint tanult, melyeknek konvolúciós hálózati része a VGG16 nem befagyasztott rétegei és az ennek a kimenetéhez hozzákötött saját teljesen kapcsolódó Dense rétegek. Az alábbi leírásban használt paraméterekkel kaptam a legegyértelműbb eredményeket.

block4\_conv1 (Conv2D) (None, 21, 21, 512) 1180160 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block4\_conv2 (Conv2D) (None, 21, 21, 512) 2359808 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block4\_conv3 (Conv2D) (None, 21, 21, 512) 2359808 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block4\_pool (MaxPooling2D) (None, 10, 10, 512) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block5\_conv1 (Conv2D) (None, 10, 10, 512) 2359808 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block5\_conv2 (Conv2D) (None, 10, 10, 512) 2359808 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block5\_conv3 (Conv2D) (None, 10, 10, 512) 2359808 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block5\_pool (MaxPooling2D) (None, 5, 5, 512) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten\_1 (Flatten) (None, 12800) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 256) 3277056 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_1 (Dropout) (None, 256) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_2 (Dense) (None, 256) 65792 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_2 (Dropout) (None, 256) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

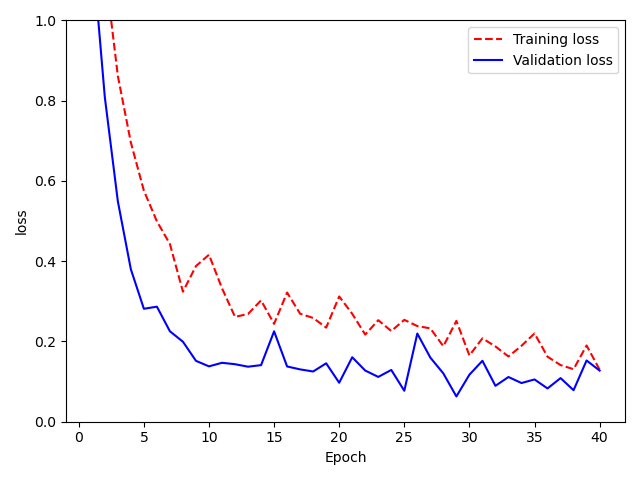
dense\_3 (Dense) (None, 10) 2570 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_3 (Dropout) (None, 10) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_4 (Dense) (None, 5) 55 =================================================================

Összes paraméter: 18,060,161

Tanítható paraméterek: 16,324,673

Nem tanítható paraméterek: 1,735,488

ábra 19- Loss értékének ábrázolása a klasszifikációs modell esetén

A felhasznált Dense rétegek midnegyikénél LeakyRelu aktivációs függvényt alkalmaztam 0.3-as alpha paraméterrel. Ez az alpha paraméter a szoró amivel a kapott x értékeket megszorozzuk minden x < 0 esetben. Ezek után minden Dense réteg közé beillesztettem egy Dropout réteget 0.25-ös értékkel.

A tanulás mértéke a nagy tanítóminta és a transfer learning miatt már az első epoch alatt túl nagy lett ahhoz, hogy ezt szemléltetni lehessen egy görbén epoch-onként. Emiatt szükség lett a tanítás átparaméterezésére. A megjelenítéshez az batch/epoch arányt, vagyis a tanító és a validációs mintánál is a step\_per\_epoch paramétert lecsökkentettem tízre az eredeti N/batch érték helyett, ahol N a tanítóminták teljes száma, az eredmény pedig megadja egész számra pontosan hány batch-nyi adat található a teljes mintában. Az itt látható eredmények persze torzítottak, mivel egyszerre csak a minta töredékével tanítunk, valamint a csökkentett érték miatt változó, hogy  
van-e átfedés a vizsgált minták között (ez azért fordulhat elő, mert a képeket minden batch esetében random választjuk ki a még nem felhasznált képek közül, viszont nem-felhasználtságra csak epoch-onként figyelünk).

A tanítást ellenőrzőpontokkal futtattam, ami azt jelentette, hogy minden epoch végén vizsgáltuk az aktuális loss-t, és ha ez kisebb volt, mint az utolsó ellenőrzőponton, akkor a modellt és a tanult súlyokat is elmentettük. Végeredményben a legalacsonyabb loss érték, amit el tudtam érni az 0.05 volt, így az ehhez tartozó modell került mentésre és későbbi felhasználásra.

## 4.3. A teljes modell

A teljes neurális hálózati modellhez szükség volt a bemenet és a kimenet megfelelő átalakítására is. Mivel a bemenet logikailag is két kép a kimenet pedig egy tőlük függő 0 és 1 közötti lebegőpontos szám, így a modell egy sziámi hálózat lett végül melyhez az előre tanított klasszifikációs modellemet használtam fel. Lássuk ennek a folyamatát, melynek az absztrakt folyamatábrája a harmadik számú mellékletben olvasható. [[M-3]](#_3._számú_melléklet:)

### 4.3.1. Bemenet kezelése – Saját generátor

Mivel a sziámi hálózatok az átlagostól eltérő bemenetet várnak tanítás közben, ezért az első lépés egy, a képeket a hozzájuk tartozó számértékkel párba rendező generátor megalkotása volt. A generátor egy olyan objektum, mely a benne található metódusok és adattagok segítségével átalakítja és biztosítja a neurális hálózatnak az általa várt formátumú adatokat. Esetemben mivel a sziámi hálózat egyszerre ,,két helyen fut” így a párban álló két képet és a számértéket is egyszerre kellett biztosítani számára.

Átlagos esetben ez a következő módon néz ki:

1. Adatok betöltése a generátorba (tanítóminta + prediktálni kívánt adat)
2. Adatok szétbontása véletlenszerűen a megadott batch méretének alapján
3. Batchek egymás utáni kiadása N-szer

A feladatot bonyolította, hogy bár csak 92 illetve 118 képem volt, vagyis a képeket akár egyszerre is lehetett volna tárolni a memóriában, viszont ha minden képpárt tárolni akartunk volna akkor összesen 4278 db képpár tárolása lenne szükséges. Látható, hogy ez így nem lenne lehetséges, valamint a képek folyamatos beolvasása fájlból szintén bonyolítana a dolgon. Szerencsére a képeket és a hozzájuk tartozó adatokat is megkaptuk a kihívás szervezőitől nem csak fájlként, külön eltárolva, de matlab által kezelt .mat típusú fájlként is. Ebből a fájlból így a beolvasott képeket, mint egész számokat tartalmazó tömböket tudtuk kezelni, bármilyen képbetöltő külső könyvtár használata nélkül.

A képpárokat viszonylag egyszerű volt létrehozni, egyszerű ismétléses kombináció segítségével meghatároztuk a lehetséges párokat index alapján ([0,0], [0,1], [1,1], [1,2] …stb). Ilyen módon el tudtuk tárolni az összes képpárt. Mivel ezeket az index párokat soronként generáltuk le, így nem kellett hozzáigazítani a kimeneti adatokat, elég volt azt egy egyszerű egy dimenziós tömbbé alakítani.

A kimeneti cellaértékeket elő kellett készíteni viszont a tanításhoz. Eredetileg a tervezés fázisában az ötlet az volt, hogy a klasszifikációban használt osztályokba való tartozás alapján nézzük meg a képpárok cellaértékeit, majd ezeket átlagoljuk az osztályok közötti különbségek alapján. Itt viszont a probléma sajnos az lett, hogy a viszonylag kevés osztály és az adathalmazban ezeknek a nem egyenletes jelenléte (az adathalmaz nagy része főleg az egyik osztályba tartozott) az átlageredményeket közel ugyan arra a 0.62-0,65 intervallumú eredményre hozta. Így a tanátás ezekkel szinte értelmetlen lett volna. Ezután azzal az elvvel haladtam tovább, hogy bár a klasszifikációban el tudjuk különíteni a képeket, az ezek közötti különbségekre inkább úgy tanítjuk meg a rendszert, hogy a 15 pácienshez tartozó RDM mátrixot teljes egészükben átlagoltuk. A tanító halmaz becslendő része így egy darab képszám\*képszám méretű RDM lett.

A metódust, mely az adatcsomagokat biztosítja a hálózatunk számára úgy alkottam meg, hogy minden futáskor először létrehozza az indexpárokból álló listát, ezekhez hozzárendeli a számértékeket, majd megkeveri az egész listát, ezzel biztosítva a véletlenszerűséget minden epoch alatt. Ezután a metóduson belül egy while típusú végtelen ciklus biztosítja minden futáskor az éppen aktuális batch számú adatcsomagot. Mivel a hálózat tanításakor a lépésszámot (step\_per\_epoch) úgy adjuk meg, hogy lehetőleg a teljes adathalmaz tanításra kerüljön minden epoch-ban, így nem kellett ezen kívül foglalkozni a while ciklus kilépési feltételével.

1. **def** generator(self, shuffle: bool):
2. """Recreate random shuffled triplets order"""
3. self.create\_triplets\_Index(shuffle)
4. **while** 1:
5. self.cur\_train\_index += self.batch\_size
6. **if** self.cur\_train\_index >= self.samples\_per\_train:
7. self.cur\_train\_index = 0
8. max\_train = self.cur\_train\_index + self.batch\_size
9. left = np.array(self.getLeftImages(self.cur\_train\_index, max\_train))
10. right = np.array(self.getRightImages(self.cur\_train\_index, max\_train))
11. y = np.array(self.getY\_RDMS(self.cur\_train\_index, max\_train))
12. **yield** tuple(([left, right], y))

Az adatokon a generátor a felhasználás véletlenszerűvé tételén kívül nem végez semmilyen transzformációt hiszen a ,,képek felismerésének” tanítását előzetesen megtettük a klasszifikációs neurális hálózat futtatása során.

### 4.3.2. Sziámi hálózat

A sziámi hálózatnak, mint ahogyan az irodalomkutatás során kifejtésre került, az a különlegessége, hogy több neurális hálózat párhuzamosan képes tanulni és közös kimenetet biztosítani. Eközben ezek a különálló hálózati ágban lévő, de az azonos mélységű rétegek megosztják egymás között a tanult súlyokat, ezzel biztosítva, hogy a hálózat minden ,,sziámi ága” ugyan azokkal a súlyokkal dolgozzon a predikció alatt.

A sziámi ágakat úgy készítettem el, hogy a saját klasszifikációval előre tanított modellemet használtam fel mindkét ágban a transfer learning alapjául. Ezt a modellt viszont nem teljes egészében illesztettem be a sziámi hálózatba. Mivel transfer learning-et használtam, így a klasszifikációs modellem konvolúciós rétegjeinek súlyait befagyasztottam, majd az ezeket követő teljesen összekapcsolt rétegek egy részét kicseréltem egy új hálózatra. Utóbbira azért volt szükség, hogy ne az osztályokba való soroláshoz kellő súlyokat tanulja ilyen mélyen a rendszer, hanem a tényleges RDM cellaértékre finomhangolódjon.

Alább látható a modell egyik kész változatának felépítése. A következő leírásban olvashatóak az új rétegek és a hozzájuk tartozó (a modell mentése miatt külön rétegben felvett) aktivációs függvények, melyek paraméterezett értékein az eredeti klasszifikációs modellhez képest nem változtattam. Ezeket a rétegeket a hálózat végén szereplő kettő darab Dense(256)-os réteg után fűztem újra. Az ezeket megelőző rétegek súlyértékeit pedig befagyasztottam.

Dense\_new\_1 (Dense) (None, 100) 25700

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

LeakyRelu\_new\_1 (LeakyReLU) (None, 100) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Dropout\_new\_1 (Dropout) (None, 100) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Dense\_new\_2 (Dense) (None, 100) 10100

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

LeakyRelu\_new\_2 (LeakyReLU) (None, 100) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Dropout\_new\_2 (Dropout) (None, 100) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Dense\_new\_3 (Dense) (None, 25) 2525

Az új teljesen összekapcsolt rétegeket úgy alakítottam ki, hogy jobban idomuljon a sziámi hálózatokat összefogó Lambda réteghez. A lambda réteg sajátossága az, hogy segítségével létrehozhatunk a saját függvényeink alapján új rétegeket. Esetemben ez azt jelentette, hogy a két képet egyszerre feldolgozó sziámi ágak, az általuk adott (a képen is látszódó) két darab 25 dimenziójú vektort kellett összehasonlítani. Az ágak ilyen módú összefonását kétféle megoldással is teszteltem. Az összehasonlítás kedvéért vegyük most mindkét esetben a loss értékének az átlagos abszolút különbséget(MAE) és az optimalizálónak Adam-et 0.00001 learning rate-el.

Első esetben euklideszi távolság-ot számoltam a bejövő két vektor között. Itt az alapelv az volt, hogy a két képet az osztályba sorolás segítségével a hálózat a térben is úgy helyezzel el a 25 dimenziós vektor alapján, hogy a végpontok közötti távolságból a rendszer következtessen az adott RDM cella értékére. Teszteltem egy olyan verziót is, ahol az euklideszi távolság közvetlenül a becsült adat lett, ebben az esetben azonban egy nagyságrenddel rosszabb loss értékeket kaptam.

A másik tesztesetnél a két vektor közötti különbség abszolútértékét számoltam ki. Ehhez a keras.backend könyvtárának használtam fel az abs (abszolútérték) függvényét, mivel a Lambda réteg nem kompatibilis külső könyvtárakkal, illetve különböző paramétereket kell hozzá beállítani. A tesztelés során a következő eredményeket kaptam.

|  |  |  |  |
| --- | --- | --- | --- |
| *Legjobb Validációs Loss eredmények* | | tanító loss | validációs loss |
| Euklideszi távolság | közvetlenül | 0.2292 | 0.1525 |
| közvetve | 0.1384 | 0.0310 |
| Abszolút különbségvektor | | 0.0357 | 0.0813 |

Látható, hogy a legjobb eredményt a közvetett euklideszi távolság számolásával kapott hálózattal kaptam. Feltűnhet, hogy a validációs loss értéke két esetben is sokkal alacsonyabb, mint a tanítómintáé. Ez akkor fordulhat elő gépi tanuláskor, ha a validációs adatok egyszerűbben a tanító adatoknál. Mindhárom esetben a teszteléskor a tanító a 92 képből álló, míg a validációs halmaz a 118 képből álló képminta volt. Ebből látszódhat, hogy a 118 képből álló halmaz ezek szerint egyszerűbbnek számít a 92 képből álló társánál, ami abból a szempontból is érdekes, hogy a 92-es halmazban körbevágott, leegyszerűsített képek vannak, míg a 118 képből álló halmaz életszerű, nem előkezelt képeket tartalmaz. Fontos megjegyezni, hogy ez a halmazok közötti komplexitás-beli különbség kizárólag az aktuális modell szempontjából fontos, amire a bizonyíték a következő tesztelés alkalmával még jobban látszódik.

Az eddigi modell felépítésén annyit változtattam, hogy az eredeti klasszifikációs modell konvolúciós részét követő teljesen összekapcsolt rétegjei közül nem csak egy darab Dense(256) réteget vittem át, hanem mindkettőt a hozzájuk tartozó LeakyReLU és Dropout rétegekkel együtt. Ebben a hálózatban viszont kizárólag a konvolúciós hálózati rétegek súlyait fagyasztottam be, hogy a rendszer jobban tanuljon.

Ezzel a hálózattal sikerült a legáltalánosabban is jó, képhalmaz komplexitástól függetlenül jó eredményt elérnem, amely a következő paraméterekkel rendelkezett:

|  |  |  |  |
| --- | --- | --- | --- |
| Optimalizáló | learning rate | tanító loss (92) | validációs loss(118) |
| ADAM | 0.0005 | 0.0400 | 0.0385 |

Mivel a további változtatások alatt sem tudtam ennél jobb eredményt elérni, valamint a learning rate változtatásával is csak azt tudtam ebben az esetben változtatni, hogy hány epoch alatt közelíti meg ezt az eredményt a hálózat, így ezt az új euklideszi távolságot figyelő hálózatot és az abszolút vektorkülönbséggel dolgozó hálózatot vittem tovább a teljes mátrixvizsgálathoz.

## 4.4. Végső eredmény számítása

Az eredmény kiszámításánál felhasználtam az átlageredményekre tanított abszolút vektorkülönbséggel és a közvetett euklideszi távolsággal dolgozó sziámi hálózatomat is. Ezek eredményeit fogjuk összehasonlítani.

### 4.4.1. Az eredmény kiszámítása

A kihívás által biztosított eredmény mátrix adatait először meg kell vizsgálnunk ahhoz, hogy tudjuk miért pont ezt a fajta korrelációs algoritmust alkalmazták az Algonauts Challenge.ben elért pont kiszámításához. Alább néhány tulajdonsága az adatoknak:

* Minden cellaérték 0 és 1 közé esik
* A cellaértékek függenek a másiktól (ha az egyik sorhoz tartozó képet kicseréljük, akkor az a teljes sorra hatással lesz, így látható, hogy a cellaértékek egymással együtt változnak amennyiben a mindegyikre hatással lévő kép változik)
* A cellaértékek a térben nem alkotnak lineáris egyenletet (vagyis a pontokat, ha ábrázoljuk sorrendben egy koordinátarendszerben, akkor nem közelíthetőek pontosan semmilyen lineáris egyenlettel)

Ezek a tulajdonságok vezettek arra, hogy a Spearman korrelációt használják fel az eredmények kiszámításához.

### 4.4.2. Spearman korreláció

A Spearman korreláció azt mutatja meg, hogy milyen mértékben határozza meg az egyik változó nagysága a másik változó nagyságát, illetve az összefüggés irányát és erősségét is. Az ok-okozati összefüggések feltárására azonban nem alkalmas. Ez azt jelenti, hogy csak azt tudjuk megmondani, hogy a két vizsgált változó összefügg-e, de arra nem tudunk választ kapni, hogy mi is pontosan ez az összefüggés. Ez a fajta korrelációvizsgálat a jelenlegi feladatban azért hatékony, mert az RDM-ek közötti vizsgálatnál is számunkra csak az eltérések és az egymásra való hasonlóság mértéke fontos. Az, hogy az adatok közötti különbségeket milyen ok-okozati összefüggések vezérlik, számunkra itt irrelevánsok.

A Spearman korreláció egy fajta rang korreláció(angolul: Rank Correlation). A nemparaméteres eljárások csoportjába tartozik. Minél nagyobb számú a minta, annál pontosabb lesz az értéke. Alkalmazását olyan esetekben ajánlják, ahol legalább ordinális mérési szintű változók szerepelnek, vagyis a számoknak nem csak az értékei, de a sorrendjük is számít; ahol az adatok nem folytonosak, az adathalmaz tartalmazhat extrémen kiugró értékeket is (esetünkben például a teljesen nulla mátrix átló); és ahol az adatok nem normál eloszlásúak a mintában.  
[[28]](#_Források)

### 4.4.3 A hálózatokkal elért eredmények

Az eredmény kiszámítását elvégeztem a prediktált értékek és az általam készített átlagmátrix, valamint az eredeti RDM mátrix között is, hogy lássam az eredmények változását (várhatóan az átlagmátrixra tanított hálózat természetesen jobb eredményt adott az azzal való korrelációval, mint az eredeti mátrixxal valóra).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Hálózat** | **Optimalizáló** | **learning rate** | **tanító loss** | **validációs loss** |
| Közvetett Euklidesz | ADAM | 0.0005 | 0.1384 | 0.0310 |
| Abszolút vektorkülönbség | ADAM | 0.0005 | 0.0357 | 0.0813 |
|  |  |  |  |  |
|  | **Átlag RDM** | | **Valós RDM** | |
| **Hálózat** | **92 kép** | **118 kép** | **92 kép** | **118 kép** |
| Közvetett Euklidesz | 5.15 | 7.69 | 0.03 | 0.07 |
| Abszolút vektorkülönbség | 11.13 | 6.65 | 9.66 | 1.22 |

### 4.4.4. Következtetés

Ahogyan az a mért eredményeken is látszódik, a becslés sokkal pontosabb lett a tanítómintára mint a validációs mintára, ez persze várható volt. Ami viszont jobban szembe ötölhet, hogy miközben az euklideszi hálózat lényegesen jobb eredményt ért el a validációs halmazon, vagyis lényegesen könnyebnek tartotta azt, annyival rosszabban teljesített a tényleges korreláció során. Ebből levonható a következtetés, miszerint a hálózat tanítása közbeni loss és a végső korreláció értéke összefügg, de a loss átlagos abszolút különbség mivolta miatt előfordulhat egy olyan típusú overfitting, ami miatt a rendszer már rosszul prediktálja a teljes mátrixot.

# Források

**[1]** Rina Dechter (1986). Learning while searching in constraint-satisfaction problems. University of California, Computer Science Department, Cognitive Systems Laboratory.

**[2]** Igor Aizenberg, Naum N. Aizenberg, Joos P.L. Vandewalle (2000). Multi-Valued and Universal Binary Neurons: Theory, Learning and Applications. Springer Science & Business Media

**[3]** The Algonauts Project: Explaining the Human Visual Brain – <http://algonauts.csail.mit.edu/#about> – megtekintve: 2019.11.19.

**[4]** CS231n: Convolutional Neural Networks for Visual Recognition – <http://cs231n.github.io/neural-networks-1/> – megtekintve: 2019.11.20.

**[5]** "Magnetic Resonance, a critical peer-reviewed introduction; functional MRI". European Magnetic Resonance Forum. Retrieved 17 November 2014.

**[6]** Abhishek Mishra: Why do you need a Fully Connected Layer? -<http://numahub.com/articles/why-do-you-need-fully-connected-layer> - utoljára megtekintve: 2019.12.01.

**[7]** Raimi Karim: Illustrated: 10 CNN Architectures - <https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d#e2761> - utoljára megtekintve: 2019.12.10.

**[8]** Yann LeCun, Leon Bottou, Yosuha Bengio and Patrick Haffner: Gradient\_Based Learning Applied to Document Recognition - <http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf> - utoljára megtekintve: 2019.12.01.

**[9]** Muhammed Rizwan: LeNet-5 – A classic CNN Architecture: -   
<https://engmrk.com/lenet-5-a-classic-cnn-architecture/>- utoljára megtekintve: 2019.12.01.

**[10]** Kien Nguyen, Clinton Fookes, Arun Ross, Sridha Sridharan:  
Iris Recognition with Off-the-Shelf CNN Features: A Deep Learning Perspective -  
<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8219390>- utoljára megtekintve: 2019.12.02.

**[11]** Asifullah Khan, Anabia Sohail, Umme Zahoora, Aqsa Saeed Qureshi:  
A Survay of the Recent Architectures of Deep Convolutional Neural Networks  
<https://arxiv.org/ftp/arxiv/papers/1901/1901.06032.pdf> - utoljára megtekintve: 2019.12.02.

**[12]** Karel Lenc: CNN MNIST Siamese network - <https://github.com/lenck/siamese-mnist>  
- utoljára megtekintve: 2019.12.04.

**[13]** Aakash Agrawal: Dissimilarity learning via Siamese network predicts brain imaging data - <https://arxiv.org/ftp/arxiv/papers/1907/1907.02591.pdf>  
- utoljára megtekintve: 2019.12.04.

**[14]** Sagar Sharma: Activation Functions in Neural Networks - <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>  
- utoljára megtekintve: 2019.12.10.

**[15]** 10-es forrás: Figure 1, 18850. oldal - utoljára megtekintve: 2019.12.02.

[**[16]**](#_Források) 11-es forrás: Fig. 5, 19. oldal - utoljára megtekintve: 2019.12.02.

**[17]** 14-es forrás: 3. ReLU: Fig: ReLU v/s Logistic Sigmoid  
- utoljára megtekintve: 2019.12.10.

**[18]** Kihívás által generált agyi kép az eredmények alapján, saját pillanatkép: <http://algonauts.csail.mit.edu/agustin_results.html> - utoljára megtekintve: 2019.12.10.

**[19]** 7-es forrás: Fig. 3 - utoljára megtekintve: 2019.12.10.

**[20]** 7.6. Residual Networks (ResNet): Fig. 7.6.2  
 <https://d2l.ai/chapter_convolutional-modern/resnet.html> - utoljára megtekintve: 2019.12.10.

**[21]** 8-as forrás: 7. oldal, Fig. 2. - utoljára megtekintve: 2019.12.10.

**[22]** Stanford University - <https://stats385.github.io/poolinglayers>   
- utoljára megtekintve: 2019.12.10.

**[23]** Avinash Navlani January: Neural Network Models in R - <https://www.datacamp.com/community/tutorials/neural-network-models-r>  
- utoljára megtekintve: 2019.12.11.

**[24]** Affine Analitycs: Deep Learning Demystified - <https://www.affineanalytics.com/deep-learning-demystified/> - utoljára megtekintve: 2019.12.11.

**[25]** Agustin Lage-Castellanos ésFederico De Martino:   
Predicting stimulus representations in the visual cortex using computational principles  
- <http://algonauts.csail.mit.edu/reports/agustin.pdf> - utoljára megtekintve: 2019.12.11.

**[26]** Algonauts Challenge: Fig1. Generating RDMs - <http://algonauts.csail.mit.edu/rsa.html> - utoljára megtekintve: 2019.12.11.

**[27]** Jane Bromley et al:   
Signature Verification using a "Siamese" Time Delay Neural Network <http://papers.nips.cc/paper/769-signature-verification-using-a-siamese-time-delay-neural-network.pdf> - utoljára megtekintve: 2019.12.12.

**[28]** SPSSABC.hu – Spearman Korreláció - <https://spssabc.hu/ketvaltozos-elemzes/spearman-korrelacio/> - utoljára megtekintve: 2020.05.18.

# Mellékletek

## A képen szöveg, rajztábla látható Automatikusan generált leírás1. számú melléklet: Konzultációs napló 2019/20-I. félév

## 2. számú melléklet: Algonauts Kihívás TOP-1 Eredmények

ábra 20 Kihívás által biztosított eredmény kalkuláló algoritmus által adott eredmények [[25]](#_Források) Table 1.

## 3. számú melléklet: Sziámi hálózat absztrakt modellje