NYILATKOZAT

Név: Károlyi Gellért Andor

ELTE Természettudományi Kar, szak: Matemetika

NEPTUN azonosító: DB6GYY

Szakdolgozat címe:
ORVOSI KÉPKLASSZIFIKÁCIÓ MÉLYHÁLÓKKAL

A **szakdolgozat** szerzőjeként fegyelmi felelősségem tudatában kijelentem, hogy a dolgozatom önálló szellemi alkotásom, abban a hivatkozások és idézések standard szabályait következetesen alkalmaztam, mások által írt részeket a megfelelő idézés nélkül nem használtam fel.

Budapest, 2021. 05. 31

a hallgató aláírása

EÖTVÖS LORÁND TUDOMÁNYEGYETEM

TERMÉSZETTUDOMÁNYI KAR

ORVOSI KÉPKLASSZIFIKÁCIÓ MÉLYHÁLÓKKAL

SZAKDOLGOZAT

Károlyi Gellért Andor

Matematika BSc

Alkalmazott matematikus szakirány

Témavezető: Lukács András

Számítógéptudományi Tanszék

Budapest

2021

Tartalomjegyzék

Köszönetnyilvánítás

Köszönöm a témavezetőmnek, Lukács Andrásnak, hogy megismertette velem a mesterséges neurális hálók témakörét és, hogy elegyengette a szakdolgozat írása során felmerülő problémákat. Hálával tartozom továbbá azért is, hogy aktuális cikkek ajánlásával időszerűvé tette a dolgozat témáját.

Külön meg szeretném köszönni a szüleimnek, hogy átolvasták a dolgozatot, és jelezték felém a helyesírási hibákat és fogalmazásbeli pontatlanságokat!

1. fejezet

Bevezetés

Az orvosok ideje (főleg a COVID-19 világjárvány ideje alatt) meglehetősen limitált, így fontos a gyorsabb, hatékonyabb döntéshozatal segítése a rendelkezésünkre álló számítástechnikai eszközök használatával. Ezt a funkciót korábban szakértői rendszerek látták el, azonban az elmúlt évtizedekben ezek szerepét elkezdte felváltani a gépi tanuláson alapuló mesterséges intelligencia (MI). Az MI elmúlt évekbeli előretörésével megnövekedtek az egészségügyi alkalmazásai is, melyek közt az egyik legfontosabb az orvosi döntések (diagnózis felállítása, kezelés megtervezése stb.) támogatása.

Mesterséges intelligencia alatt sokan különböző dolgot értenek. A dolgozatomban egy olyan algoritmust vagy függvényt értek alatta, amelynek nem specifikáljuk, hogy adott bemeneti adatokra pontosan mit adjon kimenetként, hanem nagyszámú tanító adat, vagyis (bemenet, kimenet) pár, alapján approximálja az adott bemenetre várt kimenetet.

A COVID-19 okozta súlyosabb megbetegedések szinte kivétel nélkül tüdőgyulladással járnak, így a fertőzés kezelésének tervezésében sokat segíthet, ha ki tudjuk mutatni a tüdőgyulladást, meg tudjuk állapítani a súlyosságát, az eredetét, és meg is tudjuk ezeket indokolni, például egy képen szegmentáljuk a diagnózis szempontjából fontos részeket. A rendelkezésemre álló adatok ennek a feladatnak egy relaxált változatát engedték megvalósítani, méghozzá tüdőröntgen képek klasszifikációját egészséges, vírus által okozott tüdőgyulladásos, és baktérium által okozott tüdőgyulladásos kategóriákba. A dolgozat célja tehát egy olyan algoritmus kidolgozása, amely minél nagyobb pontossággal tud tüdőröntgen-felvételeket klasszifikálni.

Természetesen nem csak tüdőröntgen-képek alapján lehet orvosi képeket osztályozni, számos ilyen témájú mesterséges intelligencia kutatás CT (computed tomography) felvételek kiértékelését tűzi ki célul, ám Magyarországon

a röntgengépek elterjedtebbek. A COVID-19-cel kapcsolatos jelenlegi MI kutatási irányokról és orvosi alkalmazásokról Shi és tsai. [16] írnak részletesebben. Konkrétan a koronavírus vagy egyéb vírusok klasszifikációjának problémájával tüdőröntgen alapján is többen foglalkoztak, például Wang és Wong [21] egy klasszifikációs modellt és egy nyílt forrású tesztelési adathalmazt mutat be, ám ennek és több hasonló eredménynek is nehézsége, hogy csupán néhány száz annotált koronavírusos tüdőröntgen kép van, ami gépi tanításhoz nagyon kevésnek számít.

Amint Litjens és tsai. [11] cikkének bevezetésében is utal rá, jelenleg a tanuláson alapuló képelemző algoritmusok közül a mély mesterséges neurális hálók, ezen belül pedig a konvolúciós neurális hálók (Convolutional Neural Network, vagy röviden CNN) a leghatékonyabbak. Ezek érik el a legmagasabb pontosságot, például az Imagenet Large Scale Visual Recognition Challenge (ILSVRC) versenyen (Russakovsky és tsai. [14]). Emiatt én is ezt a megközelítést használom a feladat megoldására.

A 2. fejezetben ismertetem a mesterséges neurális hálók felépítését, működését, kitérve a matematikai hátterükre. Ezután a 3. fejezetben vázolom a speciálisan képfeldolgozásra használt konvolúciós neurális hálók szerkezetét. A 4. fejezetben bemutatok egy módszert, mely más képklasszifikációs feladatokon tanított hálók segítségével javíthat a klasszifikáció pontosságán és rövidítheti a tanításhoz szükséges időt, majd ismertetek néhány erre a célra gyakran használt hálót. Végül a 5. és 6. fejezetekben bemutatom a betanított modelleket, és elemzem őket pontosságuk és általánosíthatóságuk szerint. A dolgozat eredeti, 2020-as megírása után is folytattam az ez irányú kutatást, és az eredményeim egy részét a 7. fejezetben ismertetem.

A dolgozat írása során igyekeztem következetesen magyar fogalmakat használni, ott is, ahol az angol nyelvű szakkifejezéseknek, műszavaknak még nincs magyar megfelelője. Félreértések elkerülése végett ilyen esetekben a kifejezések első használata után és esetenként később is zárójelben az eredeti angol kifejezést is leírtam.

2. fejezet

Neurális hálók

Ebben és a 3. fejezetben adott áttekintés egy jóval részletesebb leírása található Goodfellow, Bengio és Courville [3] Deep Learning című könyvében.

2.1. Hálók felépítése

A neurális hálókat felépítő alapvető egység a mesterséges neuron (a továbbiakban csak neuron), amely egy (aktivációs) függvényt $\sigma : \mathbb{R} \to \mathbb{R}$. Ezek a neuronok rétegekbe rendeződnek. Az első réteg a bemeneti (input) réteg, ez annyi neuronból áll, amennyi a feldolgozandó adatunk dimenziója. Például ha 28×28 pixeles fekete-fehér (egy színcsatornás) képeket szeretnénk feldolgozni, akkor ez a réteg $28 \times 28 \times 1 = 784$ neuront tartalmazna. Az input réteg σ függvényei jellemzően az identitásfüggvények $f(x) = x$.

Ezután néhány rejtett réteg (hidden layer) következik, melyek jellemzően valamilyen nemlineáris függvényből állnak. Eredetileg a *sigmoid* függvényt használták erre a célra, bár ez az utóbbi évtizedekben egyre kevésbé jellemző, mivel a mély (több tíz - több száz) réteges hálók, amelyek *sigmoid*-ot használtak, jellemzően nagyon lassan tanultak. A modernebb hálók jellemzően ReLU (Rectified Linear Unit) vagy Tanh függvényeket alkalmaznak.

Aktivációs függvények:

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

$$\text{ReLU}(x) = \max(0, x)$$

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$$

Az utolsó rétegben lévő neuronok száma attól függ, hogy hogyan szeretnénk elkódolni a háló kimenetét (output layer). Ha például klasszifikálni szeretnénk a képeket három kategóriába, akkor nem célszerű egy neuront alkalmazni, és a $0, 1, 2$ számokat a három kategóriának megfeleltetni, mivel ha a modellünk 50%-ban biztos abban, hogy az adott kép a 0 kategóriába tartozik, és 50%-ig, hogy a 2-be, akkor ezek súlyozott átlagaként 1-et fog eredményül adni, ami nyilvánvalóan hibás. Ezért érdemes a kimeneti réteg méretét úgy választani, hogy annyi neuron legyen benne, ahány képkategóriánk van, és minden kimeneti neuron egy kategóriához tartozzon. Tehát, ha az előző példabeli eset lép fel, akkor a háló a $[0.5, 0, 0.5]$ vektort adja vissza, míg ha úgy gondolja, hogy a kép biztosan az 1 kategóriába tartozik akkor a $[0, 1, 0]$ vektort. Ezt fogom az adatok bináris kódolásának nevezni (one-hot encoding). Ha azt is szeretnénk, hogy a háló kimenetét valószínűségként tudjuk értelmezni, akkor az is fontos, hogy a kimeneti vektor elemeinek összege 1 legyen, és minden koordinátája pozitív legyen. Erre való a softmax függvény.

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=0}^{n} e^{z_j}},$$

ahol z_i a z bemeneti vektor i-edik koordinátája.

2.2. Előrecsatolás

A neurális háló a neuronok rétegeiből épül fel. A bemeneti réteg kivételével minden neuron bemenete az előző réteg neuronjai kimenetének a súlyozott összege, plusz egy eltolás (bias) tag. Ezen súlyok és eltolások variálásával fog a modell majd tanulni. Jelöljük a rétegek számát n-el, az i-edik réteg neuronjainak számát m_i-vel, és az i-edik réteg j-edik neuronját és a kimenetét is $z^{(i,j)}$-vel. Ez remélhetőleg nem okoz majd félreértést. Ekkor tehát $z^{(i,j)}$ bemenete $= \sum_{k=1}^{m_{i-1}} (w^{(k)} z^{(i-1,k)}) + b^{(i,j)}$ ahol $w^{(k)}$ a súlyozott összeg megfelelő súlya, és $b^{(i,j)}$ az eltolás. Eszerint ezeket a súlyokat az i, és $i-1$ réteg közt egy $W^{(i)} \in \mathbb{R}^{m_i \times m_{i-1}}$ mátrixba rendezhetjük, ahol $W^{(i)}_{a,b}$ a $z_{i-1,a}$ és $z_{i,b}$ közötti súly. Tehát az i-edik réteg kimenete az $i-1$ edik réteg függvényében

$$Z^{(i)} = \sigma_i(W^{(i)} Z^{(i-1)} + B^{(i)}), \tag{2.1}$$

ahol σ az aktivációs függvény, $Z^{(i-1)}$ és $B^{(i)}$ pedig a kisbetűs megfelelőikből álló vektorok. Az előrecsatolási folyamat (forward propagation) egy bemeneti

adat (a feladattól függően általában egy vektor) esetén tehát ilyen függvények kompozíciója.

A bemeneti adatokat azonban nem célszerű egyesével a hálóba adni, ennél hatékonyabb, ha mátrix formájában egy egész csomagot (batch-et) küldünk végig a hálón. Ennek az oka, hogy a mátrixszorzásoknál jelentős optimalizációs lehetőségeink vannak, ha minden egyes művelet kiszámolása helyett egy vektorizációs könyvtárat használhatunk. A bemenetünk tehát (egy dimenziós bemenet esetén) $Z^{(0)}$, ahol $Z^{(0)}_{a,b}$ a b-adik tanulási adat a-adik komponense (tehát az egydimenziós bemeneti adatok képezik a mátrix oszlopait). Ilyen mátrixos leírásban a 2.1. képlet csak annyiban módosul, hogy $Z^{(i)}$ nem vektor hanem mátrix és a $b(i)$ vektort a mátrix minden oszlopához hozzáadjuk.

Amikor az előrecsatolásban elérünk az utolsó (n-edik) réteghez, szeretnénk mérni a hálónk becslésének (x) a távolságát a valós eredménytől (y). Ez a veszteségfüggvény, vagy hibafüggvény (loss function) n kategóriás klasszifikáció, és egy elemű tanító adat ((x, y) pár) esetén $L(x, y) : \mathbb{R}^{m_n} \times \{0, 1\}^{m_n} \to \mathbb{R}$, ezt szeretnénk majd minimalizálni. A legegyszerűbb veszteségfüggvények az átlagos abszolút hiba:

$$L(x, y) = ||x - y||_1 = \frac{\sum_{i=1}^{n} |x_i - y_i|}{n}$$

és az átlagos négyzetes hiba:

$$L(x, y) = ||x - y||_2 = \frac{\sum_{i=1}^{n} (x_i - y_i)^2}{n}.$$

Ha az output rétegen egy softmax függvényt használtunk, akkor ezek lassú tanulást eredményeznek, ezért kétkategóriás osztályozásnál bináris kereszt-entrópiát (binary cross-entropy),

$$L(x, y) = -(y_1 \log(x_1) + y_2 \log(x_2))$$

általánosan, m_n osztály esetén, pedig többkategóriás kereszt-entrópiát (categorical cross-entropy)

$$L(x, y) = -\sum_{i=1}^{m_n} y_i \log(x_i)$$

érdemes használni, ahol y_i annak az indikátora, hogy az adott adat az i-edik kategóriába tartozik. Több tanító adat esetén az egyes tanító adatok által generált hibák átlagát vesszük.

Bár csak fogalmi eltérés, de fontosnak tartom megjegyezni, hogy az angol irodalomban rendszerint nem az itt használt neuron fogalmat tekintik a neurális háló alapegységének, hanem a perceptront. Ez a dolgozatban használt fogalmakkal leírva a neuront, és a hozzá tartozó súlyokat és eltolást, vagyis egy speciális $P : \mathbb{R}^n \to \mathbb{R}$ függvényt jelent. Innen ered a multilayer perceptron kifejezés is, amely rendszerint sekély mesterséges neurális hálót jelent. Ebben a fogalomrendszerben a dolgozat neuron fogalma az aktivációs függvény megfelelője. Azért döntöttem a fogalmak ilyen tagolása mellett, mert véleményem szerint sokkal szemléletesebb a mesterséges neurális hálóra, mint egy irányított gráfra gondolni, ahol az irányított élek a háló súlyaival vannak súlyozva, és a gráf csúcsai az aktivációs függvényeknek felelnek meg.

2.3. Visszaterjesztés és a gradiens módszer

Ahhoz, hogy a hálónk jobban klasszifikáljon, csupán a hiba- vagy veszteségfüggvényt (cost function) kell minimalizálni. Tekinthetünk az egész hálóra úgy, mint a benne lévő súlyok és eltolások függvényére, amely egy valós számot (a hibát) ad, és az x bemeneti adatok és a hozzájuk tartozó y címkék fix paraméterek. Tehát a hibafüggvény minimalizálása egy sok (jellemzően több millió) változós valós értékű függvény minimalizálását jelenti. Erre analitikusan kevés esély van, ezért a gradiens módszert (gradient descent) lehet használni, vagyis az x,y paraméterek mellett ki kell számolni a háló deriváltját minden súlyra és eltolásra. A gradiens kiszámolásának módszerét nevezzük visszaterjesztésnek (backpropagation), míg ennek segítségével a hibafüggvény minimalizálását gradiens módszernek (gradient descent).

Mivel a háló sok egyszerű függvény kompozíciója, elég csak ezeknek kiszámolni a deriváltját, és a lánc szabály vektoriális alakjának ismételt alkalmazásával megkapjuk a kívánt deriváltakat (a gradienst).

Aktivációs függvények deriváltjai

$$\frac{\partial \text{sigmoid}(x)}{\partial x} = \frac{\partial \frac{1}{1+e^{-x}}}{\partial x} = \frac{e^{-x}}{(1+e^{-x})^2} = \text{sigmoid}(x)(1-\text{sigmoid}(x))$$

$$\frac{\partial \text{ReLU}(x)}{\partial x} = \begin{cases} 0 & \text{ha } x < 0 \\ 1 & \text{ha } x > 0 \end{cases}$$

$$\frac{\partial \tanh(x)}{\partial x} = 1 - \tanh^2(x) = 1 - \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Softmax és kereszt-entrópia deriváltja:

A softmax $\mathbb{R}^{m_n}$-ből $\mathbb{R}^{m_n}$-be képez, így a deriváltja a Jacobi-mátrixa. Legyen $\text{softmax}(x)_i = S_i$. Ha $i = j$, akkor $x_i = x_j$, ezért

$$\frac{\partial S_i}{\partial x_j} = \frac{e^{x_i}\left(\sum_{k=1}^{m_n} e^{x_k}\right) - e^{x_j}e^{x_i}}{\left(\sum_{k=1}^{m_n} e^{x_k}\right)^2} = \frac{e^{x_i}}{\sum_{k=1}^{m_n} e^{x_k}} \frac{\left(\sum_{k=1}^{m_n} e^{x_k}\right) - e^{x_i}}{\sum_{k=1}^{m_n} e^{x_k}} = S_i(1-S_i).$$

Ha viszont $i \neq x_j$, akkor

$$\frac{\partial S_i}{\partial x_j} = \frac{0 - e^{x_j}e^{x_i}}{\left(\sum_{k=1}^{m_n} e^{x_k}\right)^2} = -\frac{e^{x_j}}{\sum_{k=1}^{m_n} e^{x_k}} \frac{e^{x_i}}{\sum_{k=1}^{m_n} e^{x_k}} = -S_j S_i,$$

tehát

$$\frac{\partial S_i}{\partial x_j} = \begin{cases} S_i(1 - S_i), & \text{ha } i = j \\ -S_i S_j, & \text{ha } i \neq j. \end{cases}$$

A kereszt entrópia deriváltja pedig

$$\frac{\partial L(S, y)}{\partial S_i} = -\sum_{j=1}^{m_n} y_j \frac{1}{S_i}.$$

A kettőt összerakva, egyszerűsítve, és felhasználva, hogy az y vektornak pontosan egy (a j-edik) eleme egy, a többi nulla, kapjuk, hogy

$$\frac{\partial L(S, y)}{\partial x_j} = \sum_{i=1}^{m_n} \frac{\partial L(S, y)}{\partial S_i} \frac{\partial S_i}{\partial x_j} = S_i - y_j.$$

A láncszabály vektoriális alakja, ha $\mathbf{x} \in \mathbb{R}^m$, $\mathbf{y} \in \mathbb{R}^n$, $g : \mathbb{R}^m \to \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$ és $g(\mathbf{x}) = y$, $f(\mathbf{y}) = z$:

$$\nabla_x z = \left(\frac{\partial y}{\partial x}\right)^T \nabla_y z$$

ahol $\left(\frac{\partial y}{\partial x}\right)$ a g $m \times n$-es Jacobi-mátrixa. A fenti deriváltak és a láncszabály már elég ahhoz, hogy meghatározzuk a hibafüggvény minden súlyára és eltolására vett parciális deriváltját, azaz a gradiensét.

2.3.1. Gradiens módszer variációi

A célunk tehát a hibafüggvény minimalizálása. A 2.3. fejezet alapján már ki tudjuk számolni a háló gradiensét, a súlyok w és eltolások b függvényében, ahol a tanító adatok a paraméter θ. A feladat tehát

$$\min_{(w,b)} \frac{1}{N} \sum_{\theta} L(f_\theta(w, b), y),$$

ahol N a tanító adatok száma és $f_\theta(w, b)$ a háló tippje. Hívjuk a minimalizálandó függvényt $J_\theta(x)$-nek, ahol $x = (w, b)$. A gradiens azt az irányt mutatja, amerre a függvény a leggyorsabban nő, tehát a súlyokat az ezzel ellenkező irányban érdemes módosítani. Azt, hogy mennyivel, azt két dolog határozza meg. Az egyik magának a gradiensnek a nagysága (ezt azért nem normáljuk

le, mert ha minimumhely közelében vagyunk, akkor szeretnénk csak kicsit lépni), illetve egy választható λ tanulási ráta:

$$x := x - \lambda \nabla_x J_\theta(x).$$

Ha az egész tanító adathalmaz alapján teszünk meg egy lépést, akkor konvex függvény esetén garantáltan konvergálni fogunk egy globális minimumhoz, egyébként egy lokális minimumhoz. Ezzel a módszerrel az a baj, hogy nagy adathalmaz esetén nagyon lassú egy gradiens lépést meglépni, sőt a tanító adathalmaz lehet, hogy be se fér a számítógép memóriájába. Ez a batch gradiens módszer.

Ha nem számoljuk ki a teljes tanító adathalmazra a gradienst, csak egy részhalmazára, akkor beszélünk mini-batch gradiens módszerről. Elfajult esetben, amikor minden tanító adat esetén lépünk egyet, a módszert sztochasztikus gradiens módszernek nevezzük (SGD). A mini-batch és sztochasztikus gradiens módszerek jóval gyorsabbak, ám nem garantált, hogy a tanulási ráta módosítása (ütemezése) nélkül konvergálnak. Emellett az sem nyilvánvaló, hogy milyen tanulási rátával érdemes indulni, vagy mikor érdemes azt csökkenteni. Ez a feladattól és hálótól függő hiperparaméter, ki kell kísérletezni.

A tanulási ráta ütemezésére vagy a tanulási folyamat felgyorsítására számos, részben heurisztikus, ám gyakorlatban jól működő módszer van, melyet a gyakorlatban a tanulási folyamat során fellépő problémák motiválnak. Ezeket Ruder [13] gyűjtötte össze egy cikkbe, ám én is megemlítek néhány fontosabbat, melyet később használni fogok. Az első a momentum módszer, amely a lépések pályájának simítása céljából hozzáadja a gradiens i-edik lépéséhez az $i-1$-edik lépés γ-szorosát:

$$L_i = \gamma L_{i-1} + \lambda \nabla_x J_\theta(x)$$

$$x := x - L_i,$$

ahol L_i a gradiens módszer i-edik lépése. Ez a $\dim(x)$ dimenziós kanyonok gyorsabb navigálására szolgál. Itt γ 0 és 1 között bármilyen érték lehet, ám általában a 0,9 vagy ahhoz közeli értékek tűnnek a leghatékonyabbnak. Erre a módszerre épít a Nesterov gyorsítás módszere, amelyet az motivál, hogyha tudjuk, hogy a momentum módszer úgyis lép γL_{i-1}-et, akkor érdemes lehet előre tekinteni és már az új helyen számítani a gradienst:

$$L_i = \gamma L_{i-1} + \lambda \nabla_x J_\theta (x - L_{i-1})$$
$$x := x - L_i.$$

A következő módszerek mind a tanulási ráta módosítására szolgálnak. Az első az Adagrad, amelynek célja, hogy a tanulási rátát különbözőre állítsa be minden súlyra és eltolásra, hogy azok, amelyek ritkán változnak, mert az adathalmaz ritka tulajdonságaihoz tartoznak, nagyobb lépésekben változzanak, mint azok a súlyok és eltolások, melyek gyakran változnak. Ekkor a t-edik lépés új módosítási szabálya:

$$x_t := x_{t-1} - \frac{\lambda}{\sqrt{G_t + \varepsilon I}} g_t$$

ahol g_t a t-edik lépés gradiense, I az identitás mátrix, ε egy 0-hoz közeli szám (jellemzően 10^{-8}), melynek célja, hogy elkerüljük a 0-val való osztást, és

$$G_t = \operatorname{diag}\left(\sum_{i=1}^{t} g_t g_t^T \right),$$

ahol a $\operatorname{diag}(G)$ függvény a mátrix diagonálisán kívüli elemek kinullázása. A gyökvonást és az osztást G_t főátlójának elemeivel elemenként kell érteni. G_t diagonális mátrix elemei tehát a t-edik lépésig a gradiensek i-edik elemeinek négyzetösszege.

Az Adagrad módszer hátránya, hogy hosszabb tanítás esetén a gradiens eltűnik, mivel a nevezőben lévő érték folyamatosan nő. Erre egy megoldás lehet az RMSProp algoritmus, amely a négyzetösszeg tagjait exponenciálisan lecsengő súlyokkal súlyozza:

$$S_t = \beta S_{t-1} + (1 - \beta) g_t^2$$
$$x_{t+1} = x_t - \frac{\lambda}{\sqrt{S_t + \varepsilon}} g_t.$$

Általában 0,9 jó érék β-ra, míg 0,001 λ -ra jó kezdeti érték.

Végül az Adam algoritmus kombinálja az RMSProp és Adagrad módszert:

$$L_t = \beta_1 L_{t-1} + (1 - \beta_1) g_t,$$
$$S_t = \beta_2 S_{t-1} + (1 - \beta_2) g_t^2,$$

ahol L_t és S_t a gradiens első és második momentumára torzított becslés, innen kapta az algoritmus a nevét is (ADAptive Moment estimation). A becslést torzítatlanná téve:

$$L_t^* = \frac{L_t}{1-\beta_1^t},$$
$$S_t^* = \frac{S_t}{1-\beta_2^t}.$$

Végül ezekkel a t-edik lépés módosítási szabálya

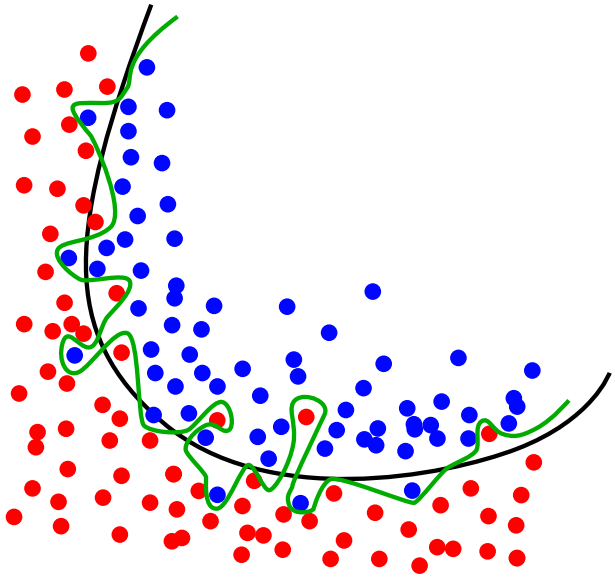
$$x_t := x_{t-1} - \frac{\lambda}{\sqrt{S_t^* + \varepsilon}} L_t^*.$$

2.3.2. Túltanulás és regularizáció

A háló tanítása során két probléma is felléphet. Az első az alultanulás (underfitting), a második a túltanulás (overfitting). Az alultanulás azt jelenti, hogy a modellünk nem elég bonyolult ahhoz, hogy jól illeszkedjen az adathalmazra. Ennek a jelenségnek egy egyszerűbb példája, amikor egyenest próbálunk illeszteni egy alapvetően parabolát követő adathalmazra. Erre a problémára szerencsére egyszerű a megoldás: bonyolultabb hálót kell alkalmazni! Ez a bonyolultság több paramétert jelent, vagyis csupán meg kell növelni a hálónk rétegeinek számát, vagy a rétegeken belül a neuronok számát.

Az érdekesebb probléma a túltanulás. Ez akkor fordul elő, ha a modellünknek túl nagy a szabadsági foka, és az adathalmazban lévő zajt kezdi megtanulni a háló. Ez akkor fordul elő gyakran ha nagy hálót próbálunk kisméretű adathalmazon tanítani. Egy intuitív kétdimenziós példa látható erre a problémára a 2.1. ábrán, ahol a zöld vonal egy túl bonyolult modellt reprezentál a kék és piros pontok elkülönítésére. A túltanulás azért probléma, mert ha a túltanult hálónak egy olyan példát adunk, amely nem szerepelt a tanító adathalmazban, akkor nem várható, hogy pontos becslést adjon rá, ha az adathalmazban lévő zajra tanult rá. Nem tud a háló jól általánosítani.

Az első lépés a túltanulás csökkentésére a mérése. Ehhez egy olyan adathalmaz kell, amelyen a háló nem tanult, hogy mérni tudjuk a modell által tanult valóban hasznos, általános információ mennyiségét. Ez a teszt adathalmaz. Viszont gyakori, hogy egy probléma megoldása során több háló architektúrát és még több hiperparamétert (rétegek száma, tanulási ráta, gradiens módszer változat stb.) próbálunk ki. Ha mindegyik esetében a teszt adathalmazzal mérjük a háló teljesítményét, akkor felmerül a hiperparaméterekre való túltanulás veszélye: hogy a sok hiperparamétert használó modell közül a legjobb csak véletlenül pont ezen a teszt adathalmazon teljesített jól. Emiatt a modellek teljesítményét tanítás során egy harmadik, úgynevezett validációs adathalmazon mérjük (melyen a háló szintén nem tanul), és a teszt



2.1. ábra. Példa túltanulásra. Forrás: By Chabacano - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=3610704

halmazt csak a tanítás végén használjuk a végső háló teljesítményének a mérésére. Azt, hogy a hálónk alul tanul tehát úgy vehetjük észre, hogy még a tanító adathalmazon sem teljesít jól, bár ennek számos más oka is lehet. Azt, hogy a háló túltanul könnyű észrevenni, ez akkor fordul elő, ha a háló sokkal rosszabbul teljesít a validációs adathalmazon, mint a tanítási adathalmazon.

A túltanulás mértékének csökkentésére számos lehetőségünk van. Talán a legegyszerűbb, hogy kisebb hálót tanítunk. Ez a tanítás sebességében is segít, ám nem mindig hatásos, mert nehéz kikísérletezni, hogy pontosan mekkora az ideális méretű modell. A második lehetőség, hogy nagyobb mennyiségű adaton tanítjuk a modellt, ám ez hosszabb tanítást jelent, és általában az adatgyűjtés még nehezebb feladat, mint a hálók tanítása, tehát gyakorlatban ritkán ez a jó megoldás. Az adatok mennyiségének növelésére jó stratégia lehet a tanító adathalmaz augmentálása. Ez például képek esetében átméretezéseket, tükrözéseket, forgatást zoomolást elmosást nyújtást és egyéb képmanipulációkat jelent. Ez hatásos eszköz lehet a túltanulás ellen, de az adatokban lévő hasznos információ mennyiségét is csökkentheti, rontva a modell teljesítményét.

A következő lehetőség a túltanulás ellen a regularizáció. Hagyományosan a két leggyakrabban használt regularizációs technika az L_1 és L_2. Ezek azon alapulnak, hogy a modell nagyszámú súlyait és eltolásait próbálják büntetni. Az ötletet az motiválja, hogy ha egy magas fokú polinom (mint a 2.1. ábra

példájában a zöld vonal) együtthatóit egy nagy számmal leosztjuk, akkor az így kapott polinom egy kisebb fokút (ha elég nagy számmal osztunk konstanst) közelít, és ez a kisebb fokú polinom nem elég bonyolult a túltanuláshoz. A polinomok analógiájában a hálók súlyai és eltolásai az együtthatóknak felelnek meg, tehát ezeket szeretnénk valahogy minimalizálni. Ezt úgy lehet megtenni, hogy a hibafüggvényt egy olyan taggal bővítjük, amely a nagy súlyokat bünteti. Az új hibafüggvényeket is L_1-el és L_2-vel jelölve tehát:

$$L_1(x, y) := L(x, y) + \lambda \left|\left|\theta\right|\right|_1 = L(x, y) + \lambda \sum_{\theta \in \Theta} |\theta| ,$$

$$L_2(x, y) := L(x, y) + \lambda \left|\left|\theta\right|\right|_2 = L(x, y) + \lambda \sum_{\theta \in \Theta} \theta^2,$$

ahol Θ a súlyok halmaza és λ állítható hiperparaméter. Általában Θ-ba az eltolásokat nem szokás belevenni, mivel ezek a gyakorlatban kevésbé járulnak hozzá a túltanuláshoz, de bele lehet.

Szokás regularizáció ellen korai megállást (early stopping) is használni, ám ennek az a hátránya, hogy fenn áll a veszélye, hogy azelőtt állítjuk le a tanítást, mielőtt a modell az összes releváns információt megtanulhatta volna a tanító adathalmazból. Végül Srivastava és tsai. [18] egy teljesen más megközelítést ajánlanak regularizációra: a kiesési réteget (dropout layer). Ezt két réteg közé rakva a következő réteg neuronjainak $\lambda \in [0, 1]$ részének kinullázza a bemenetét, ezzel gyakorlatilag kiejtve a hálóból. Ezt a megközelítést két dolog motiválja. Az első, hogy egy ilyen modell neuronjai nincsenek mindig összekötve, így nem hagyatkozhatnak egymásra. Ez megakadályozza, hogy a háló olyan bonyolult mintázatokat tanuljon meg, amely sok neuron együtt dolgozását igényli. Véleményem szerint a módszer hatékonysága mögötti másik érv az lehet, hogy egy ilyen háló tanítás során rákényszerül, hogy nagy mennyiségű redundáns információt tanuljon meg, ezzel csökkentve a kapacitását a túltanulásra.

3. fejezet

Konvolúciós hálók

3.1. Konvolúciós rétegek célja

Ha képeket szeretnénk elemezni neurális hálókkal, érdemes kihasználni azt az információt, hogy az egymáshoz közeli pixelek valamilyen értelemben szorosabban kapcsolódnak, mint az egymástól távoliak. Azonban, ha egy réteg összes neuronját a következő réteg összes neuronjával összekötjük, elveszítjük ezt a kétdimenziós információt. A másik probléma, hogy ha minden kép pixelt egy külön bemenetnek tekintünk, nehéz egy mély hálót tanítani, mert ha rétegről rétegre nem csökken drasztikusan a neuronok száma, akkor a modell tanítása nagyon lassúvá válik, sőt ami még rosszabb, hogy a modell nagy szabadsági foka miatt hajlamos lesz a túltanulásra (overfitting). Viszont ha drasztikusan lecsökkentjük a neuronok számát, akkor a képet gyakorlatilag tömörítjük, ami információvesztéssel járhat. A nagy képeken természetesen rengeteg redundáns információ van, de semmi garancia arra, hogy a tömörítés során a fontos információ marad meg. Ezért van szükség egy olyan háló-architektúrára, amely képes felismerni a kétdimenziós mintázatokat (például élek, sarkok, textúrák) egy képen, és a tömörítés során ezt az információt igyekszik megtartani.

3.2. Konvolúciós hálók felépítése

A konvolúciós hálók jellemzően három fajta rétegből állnak:

- a hagyományos mindent mindennel összekötő rétegből (dense layer)
- a konvolúciós rétegből (convolutional layer)
- összegző rétegből (pooling layer).

A konvolúciós réteg lényege a kernel, vagy filter, amely egy $\mathbb{R}^{n\times n\times k}$ - es súlytenzor, tanítható súlyokkal, ahol n jellemzően egy kis páratlan szám (3 vagy 5), és k a kép színcsatornáinak száma (fekete-fehér kép esetében 1, RGB kép esetében 3). Ezt a filtert az eredeti kép egy $n \times n \times k$ részletére helyezve, az ez által kimetszett résztenzort a filterrel elemenként szorozva, majd az így kapott n^2k darab számot és egy tanulható eltolást összeadva kapott valós szám lesz a következő réteg azon neuronjának (pixelének) a bemenete, amely az eredeti képen a filter közepére esett (ha n páratlan). A filternek fontos paramétere továbbá a lépésköz (stride), ami azt mondja meg, hogy hány pixelenként helyezzük le a filtert, amikor sorfolytonosan végigmegyünk a képen. Tehát a konvolúciós réteg csupán olyan neuron réteg, amelyben egy neuron az előző rétegbeli neuronok közül csupán a hozzá térben közeliekkel van összekötve.

Természetesen felmerül a kérdés, hogy mi legyen a szélső $\lfloor \frac{n}{2} \rfloor$ sorral és oszloppal. A legegyszerűbb megoldás, hogy ezekre nem illesztünk filtert, így a következő réteg $n-1$ sorral és oszloppal kisebb lesz. A második megoldás, hogy adunk a képünknek egy $\lfloor \frac{n}{2} \rfloor$ széles keretet. Ez állhat 0 értékű pixelekből, vagy feltölthetjük a valós kép legközelebbi pixelének értékével.

Egy filter tehát a kép valamilyen lokális tulajdonságát (feature) emeli ki az előző réteg (a második rétegtől kezdve immár tulajdonságokból álló) képének. Azonban egy filter általában nem elég a kép összes tulajdonságának megőrzésére, így több (hagyományosan kettőhatvány számú) különböző filtert kell alkalmazni, hogy ennyi tulajdonságot emeljünk ki rétegről rétegre. Ennek az az eredménye, hogy nem egy darab kétdimenziós neuronokból álló képünk van egy rétegben, hanem annyi, ahány filtert használtunk. Így a következő réteg filtereinek bemenete $n \times n \times k$ méretű, ahol k az előző réteg filtereinek száma. Ebben az értelemben a filterszám analóg a bemenet színcsatornáinak számával.

Az összegző réteg célja a rétegek méretének csökkentése, a feladat megoldásához szükséges információ minél nagyobb részének megőrzésével. Ez a

túltanulás, és a tanulási sebesség miatt is fontos. Ezt a célt úgy éri el, hogy egy $n \times n \times 1$-es tartományt a bemeneti rétegen leképez egy 1×1-es tartományra. Összegző rétegből a két leggyakoribb típus a maximum összegző réteg (max pooling layer) és átlagos összegző réteg (average pooling layer). A maximum réteg a bemeneti tartomány legnagyobb elemét, az átlagos réteg a bemeneti tartomány elemeinek átlagát adja kimenetként. A leggyakoribb összegző réteg a 2×2 maximum réteg 2-es lépésközzel (így a tartományok között nincs átfedés). Ez gyakorlatilag megfelezi az előző réteg képeinek szélességét és magasságát, míg megőrzi azt az információt, hogy egy adott tulajdonság jelen van-e. Az információ, aminek egy részét elveszti, az, hogy a réteg által elkódolt tulajdonság pontosan hol van jelen, ám ez még pozitívum is lehet, ha az alapján szeretnénk képeket osztályozni, hogy mi van a képen (például kutya vagy macska), és nem az alapján, hogy a képen belül hol. Tehát az összegző rétegek egy fajta eltolás-invarianciát biztosítanak a képeken lévő objektumoknak, mintázatoknak.

A konvolúciós hálók utolsó rétegeit jellemzően egy vagy több sűrű réteg képezi, amely a nagy számú, de kis szélességű és magasságú képeket, amelyek mindegyike valamilyen tulajdonságot kódol el, leképezi egy annyi dimenziós vektorra, ahány kategóriánk van. A konvolúciós hálók elejére tehát lehet úgy tekinteni, mint a háló tulajdonság-kifejtő (feature extractor) részére, míg az utolsó sűrű rétegek egy hagyományosabb hálót képeznek, amely immár nem a kép pixeleinek értéke alapján, hanem a kép valamilyen absztrakt tulajdonságai alapján végzi az osztályozást.

4. fejezet

Átviteli tanulás

4.1. Az átviteli tanulás fogalma

Embereknek általában nem okoz problémát a megszerzett tudás egy új környezetben való használata. Ha valaki megtanul biciklit vezetni, akkor nem kell szükségképpen előről kezdenie a motorkerékpár használatának megtanulását. Természetesen adódik tehát az igény, hogy a betanított modelljeinket valamilyen értelemben újra felhasználhatóvá tegyük, és itt nem csupán a modell architektúrájáról van szó, hanem a tanulás során megszerzett információról is. Ezt a megközelítést nevezzük átviteli tanulásnak (Transfer learning).

Az átviteli tanulás igénye már 1995-ben megfogalmazódott, és 2009-ben Pan és Yang [12] egy átfogó tanulmányt írt a témában. Az átviteli tanulás definíciója előtt azonban szükségünk van néhány jelölésre. Egy tartomány (domain) D két részből áll: egy tulajdonság alaphalmazból (feature space), ezt jelölje χ és egy eloszlásból $P(X)$, ahol $X = \{x_1, x_2, \ldots, x_n\}$, $x_i \in \chi$. Tehát a mi esetünkben x_i egy vektor, ami elkódolja a röntgenképeink tulajdonságait, és χ minden lehetséges kép halmaza, X pedig ebből vett minta.

$$D = \{\chi, P(x)\}$$

Egy feladat (T) egy címke alaphalmazból γ (label space) és egy $f(x) = P(Y \mid X)$ előrejelző függvényből áll, ahol $Y = \{y_1, y_2, \ldots, y_n\}$, $y_i \in \gamma$. A mi esetünkben $\gamma = \{\text{egészséges, bakteriális, vírusos}\}$.

$$T = \{\gamma, P(Y \mid X)\}$$

Az eredeti feladat tartománya és feladata legyen D_E és T_E, míg a cél feladaté legyen D_C és T_C. Ekkor az átviteli tanulás egy olyan módszer, amely $f_C(x)$

tanulását segíti D_C-ben, D_E és T_E-beli tudás segítségével, úgy hogy $D_E \neq D_C$ vagy $T_E \neq T_C$.

$D_E \neq D_C$ kétféleképpen lehet: ha $X_E \neq X_C$, vagy $P_E(X) \neq P_C(X)$. Hasonlóan $T_E \neq T_C$ azt jelenti, hogy $Y_E \neq Y_C$, vagy $P(Y_E \mid X_E) \neq P(Y_C \mid X_C)$

A konvolúciós neurális hálókkal való képklasszifikáció témakörében X_E és X_C megegyeznek, mivel mindkettő képek halmaza, és a képek manipulálásával könnyű elérni, hogy ugyan olyan felbontásúak és elkódolásúak legyenek. $P_E(X)$ és $P_C(X)$ lehet hasonló, vagy különböző, attól függően, hogy pontosan honnan származnak a képek az eredeti és cél tanulási feladatokban. Y_E és Y_C általában jelentősen különböznek, mivel ritka az, hogy pontosan ugyan azokba a kategóriákba szeretnénk sorolni a képeket. Ez magával vonja $P(Y_E \mid X_E)$ és $P(Y_C \mid X_C)$ különbözőségét is.

Az átviteli tanulás sikerességéhez Pan és Yang [12] három kérdés megválaszolását javasolja: mikor- mit- és hogyan- vigyünk át? Ezekkel a kérdésekkel én csupán a konvolúciós neurális hálók kontextusában fogok foglalkozni.

A mikor? kérdésre a válasz, hogy akkor, ha az eredeti tanítási folyamathoz nagy mennyiségű tanulási adat, és számítógépes erőforrás áll rendelkezésünkre, míg a célfeladathoz nem. A mit? kérdésre a válasz, hogy az eredeti modell által megtanult súlyok egy részét (első néhány réteg súlyait). Erről részletesebben a 4.2. fejezetben írok. Végül a hogyan? kérésre a válasz az, hogy az átvitt súlyoknak csak az eredeti háló kontextusán belül van értelme, ezért ha ezeket át szeretnénk ültetni az új hálónkba meg kell tartanunk az eredeti modell architektúrájának a súlyokhoz tartozó részét is. Az itt használt megközelítés Pan és Yang [12] cikkében tehát az induktív átviteli tanulás (inductive transfer learning) kategóriájába esik.

4.2. Átviteli tanulás konvolúciós hálókkal

A 3.2. fejezet végén utaltam rá, hogy a konvolúciós hálók első, konvolúciós részének nem annyira a klasszifikálás a feladata, mint inkább a képek klasszifikációjához fontos információ kinyerése. A második, sűrű réteg(ek)ből álló rész végzi a kinyert információ alapján az osztályozást. Mivel a sűrű rész nagyon feladat specifikus, függ a feladat típusától (klasszifikáció, regresszió) és klasszifikáció esetében a kategóriák számától, elkódolásától is, nem várható, hogy az általa megtanult információ egy teljesen új feladat esetében hasznos lenne. A gyakorlatban a képek közt lokálisan sok közös vonás lehet, élek, sarkok, mintázatok, tehát ha ezeket emeli ki a konvolúciós része a modellnek,

akkor várható, hogy ennek az átültetése az új feladat megoldásához használt modellbe segítheti annak tanulását.

Valóban, a tapasztalat azt mutatja, hogy a konvolúciós hálók első rétegei éleket és egyszerűbb mintázatokat emelnek ki, amik jobban általánosíthatók egy új feladatra [23], míg a későbbi rétegek bonyolultabb, absztraktabb, feladat-specifikusabb alakzatokat. Ez analóg azzal az elképzeléssel, hogy a mély neurális hálók azért ilyen hatékonyak, mert az egyre mélyebb rétegek hierarchikusan egyre magasabb szintű absztrakcióját tárolják a bemeneti adatoknak.

Az új feladat modelljének megalkotásakor el kell tehát dönteni, hogy a régi feladat modelljének melyik rétegéig emeljük át a rétegeket súlyokkal együtt. Ez nagyban függ az eredeti modell felépítésétől, és érdekes kutatási terület lehet, de az általános megközelítés az, hogy a határt a konvolúciós és sűrű rétegek határán húzzuk meg, ezért én is ezt teszem majd.

Az átvitel után a legfontosabb kérdés, hogy az áthozott rétegeket tanítsuk-e az új feladat adatain (fine-tuning) vagy pedig fagyasszuk meg az átemelt súlyokat és tekintsünk az átemelt részre úgy, mint egy fekete doboz, amely megalkotja a képek egy alacsonyabb dimenziós reprezentációját (feature extraction). Ebben az esetben az új feladat tehát nem a képek alapján való klasszifikálás, hanem a képek egy reprezentációja alapján való klasszifikálás. A kérdést Litjens és tsai. [11] is felveti, és utal rá, hogy egy papírban az első, egy másikban a második megközelítés bizonyult hatékonyabbnak, ám kevés az ez irányú kutatás.

A továbbtanítás (fine-tuning) előnye, hogy ha megfelelő méretű adathalmazunk és számítógépes erőforrásaink vannak, akkor a kifejtő háló is rá tud tanulni az új feladatra. Ezzel az új képek egy jobb reprezentációját tudja előállítani, növelve a modell teljesítményét. A hátránya, hogy ha nem megfelelő módon próbáljuk tanítani, (például túl nagy tanulási rátával) akkor elronthatjuk az eredeti feladat során óvatosan beállított súlyokat. A másik probléma ezzel a megközelítéssel, hogy kis adathalmaz esetén ez nagy szabadsági fokot ad a modellnek, amivel túl tanulhat a kevés adaton, ezzel jelentősen rontva a modell általánosító képességét. Én először az első módszert fogom követni, majd megnézem, hogy az így tanított modell teljesítményén javít-e ha egy alacsony tanulási rátával az összes réteget tanítom. A 7. fejezet kísérleteiben pedig a tanítás elejétől kezdve minden réteget közös tanulási rátával fogok továbbtanítani.

4.3. Gyakori modellek

Ahhoz tehát, hogy átviteli tanulást alkalmazhassunk, kell egy nagy és általános adathalmazon tanított modell. Szerencsére a Tensorflow és Keras ökoszisztémáján belül számos hatalmas adathalmazon tanított modell érhető el ingyenesen. A Keras egy nyílt forrású alkalmazásprogramozási felület (Application Programming Interface vagy API), amely az eddig leírt függvényeket, és sok más mélytanulási eszközt tartalmaz. A Tensorflow a Google által fejlesztett szimbolikus matematikai könyvtár, amely a mélytanulást helyezi középpontjába és tartalmazza Kerasnak egy implementációját.

Azonban mielőtt rátérek a konkrét modellekre, érdemes egy pár szót ejteni az adathalmazról, amin előtanították őket (ImageNet), és az ehhez tartozó ImageNet Large Scale Visual Recognition Challenge (ILSVRC) versenyről. Az ImageNet adathalmaz 2010.04.30.-án 14 197 122 képből állt, amelyek 21 841 kategóriához tartoztak és 1 034 908 képen határolókerettel (bounding box) jelölve van a képen található objektum helye (http://image-net.org/about-stats). Ezen képek egy részhalmazából rendezték meg évente 2010 és 2017 között az ILSVRC-et. A verseny klasszifikáció kategóriájához tartozó adathalmaz nagyjából 1,2 millió képből állt, melyek 1000 kategóriához tartoztak. Az adathalmazról és a versenyről Russakovsky és tsai. [14] írnak részletesebben. A továbbiakban is a versenynek csak a klasszifikáció kategóriával fogok foglalkozni.

- A 2010-es és 2012-es verseny győztese az Alexnet modell volt (Krizhevsky, Sutskever és Hinton [10]).
- A 2014-es verseny győztese az Inception (GoogLeNet) modell volt (Szegedy és tsai. [19]). Itt az inception a modell architektúrája, a GoogLeNet pedig a konkrét modell.
- A 2014-es verseny második helyezettje a VGG modell volt (Simonyan és Zisserman [17]).
- A 2015-ös verseny győztese a ResNet volt (He és tsai. [4]).

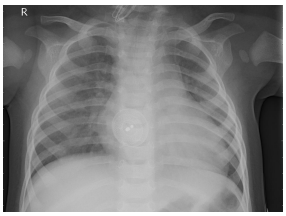
Az Alexnet kivételével a fenti modellek mindegyike (és még néhány) megtalálható a tensorflow.keras.applications modulban, ahonnan be lehet ezeket a modelleket tölteni az ImageNeten tanult súlyaikkal együtt. A modul sok kisegítő lehetőséget is tartalmaz, például a modellek sűrű rétegei nélkül való betöltést.

Egy másik hely, ahol számos nagy adathalmazon tanított modell található a Tensorflow hub. Itt található meg a 2020 májusában Kolesnikov és tsai. [9] által megosztott három modell közül a két kisebb adathalmazon tanított modell sorozat (BiT-M, és BiT-S). A BiT-S az ImageNet adathalmaz $\sim$ 1 millió képén volt tanítva, BiT-M az ImageNet-21k $\sim$ 14 millió képén. A harmadik, és egyben legnagyobb modell sorozat a BiT-L, amely a JFT-300M adathalmazon volt tanítva, amely $\sim$ 300 millió kissé zajosan címkézett képet tartalmaz. A JTF-300M adathalmazt és az ezen tanított modelleket azonban a Google Brain csapata nem tette elérhetővé. A BiT-S, BiT-M és BiT-L a ResNet architektúrának különböző méretű variációi. Ezeket a modelleket direkt átviteli tanulásra való felhasználással készítették, és több mint 20 különböző feladaton tesztelték. A készítők ajánlottak egy heurisztikus átviteli tanítási szabályt is (BiT HyperRule).

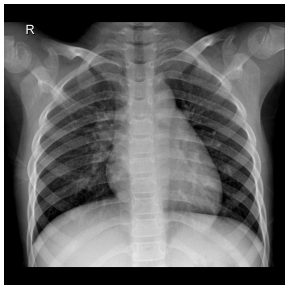
A Modellek, amiket az átviteli tanításhoz fogok kipróbálni a VGG-16 és a BiT-M R101x1.

5. fejezet

Adathalmaz és feladat



5.1. ábra. Röntgenfelvétel bakteriális eredetű tüdőgyulladásról.



5.2. ábra. Röntgenfelvétel egészséges emberről.

5.3. ábra. Röntgenfelvétel vírusos eredetű tüdőgyulladásról.

Amint már utaltam rá, a dolgozat feladata tüdőröntgen képek alapján való klasszifikáció három kategóriába:

- egészséges
- bakteriális tüdőgyulladás
- vírusos tüdőgyulladás.

Ehhez az általam felhasznált adathalmaz a Kaggle-ről származott:

CoronaHack -Chest X-Ray-Dataset

https://www.kaggle.com/praveengovi/coronahack-chest-xraydataset

ahol az adathalmaz egy részét Cohen, Morrison és Dao [2] által gyűjtött adatok képezték (https://github.com/ieee8023/covid-chestxray-dataset). Az adathalmazt 2020 áprilisának végén töltöttem le. Ez azért fontos, mivel azóta Cohenék adatbázisába több COVID-19-es kép is felkerült, ám a Kaggle adathalmazt nem frissítették ezzel együtt. A Kaggle adathalmaz leírásában az szerepel, hogy 5910 kép van benne, azonban letöltés után 624 tesztelési és 5309 tanítási, azaz összesen 5933 képem volt. A tanítási képek közül néhány kép nem tüdőröntgen felvétel, hanem CT felvétel volt, és ezek nem is szerepeltek a címkék közt. Emiatt kivettem az adathalmazból az összes olyan képet, amelynek nem volt meg a címkéje, és négy olyan képet is, amelyek címkézett tüdőröntgen felvételek ugyan, de nem szemből, hanem oldalról készültek. Ezek után 5904 képem maradt, amelyekből 624 tesztelésre el volt különítve.

Label	Label_1_Virus_category	Label_2_Virus_category	Image_Count
Normal			1576
Pnemonia	Stress-Smoking	ARDS	2
Pnemonia	Virus		1493
Pnemonia	Virus	COVID-19	58
Pnemonia	Virus	SARS	4
Pnemonia	bacteria		2772
Pnemonia	bacteria	Streptococcus	5

5.4. ábra. A Kaggle adathalmaz. Forrás: https://www.kaggle.com/praveengovi/coronahack-chest-xraydataset

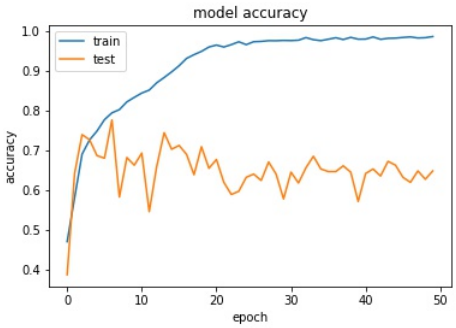
A tanító adatok közül 2535 bakteriális, 1342 egészséges és 1203 vírusos kép volt. A tesztelési adatok közt 242 bakteriális, 234 egészséges és 148 vírusos kép volt. Az 5.1., 5.2. és 5.3. képeken a három tanító adathalmaz egy-egy képe látható. Egy kis problémát jelentett, hogy a képek szinte mind különböző méretűek voltak. A magasságaik ∼500 és ∼2500 pixel közt szinte minden értéket felvettek, és a szélességük és képarányuk is hasonlóan szórt volt. Ezen kívül még így is túl nagyok voltak ahhoz, hogy egy hálót gyorsan lehessen velük tanítani. Emiatt az előfeldolgozás (preprocessing) és augmentáció végén képaránytól függetlenül minden képet 224×224-es képméretre zsugorítottam. Ezen kívül minden képpixel minden színcsatornájának az értékét elosztottam 255-tel, hogy 0 és 1 közé essen az értéke, ezzel normalizálva a tanulási adathalmazt.

6. fejezet

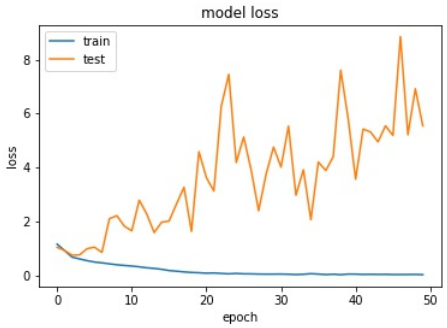
Kísérletek

A kísérletek során a Colabot, a Google Research jóvoltából ingyenesen használható online notebook környezetet használtam és a Google videokártyáin tanítottam a hálóimat. Először egy saját modellt próbáltam ki, amelynek az architektúrája hasonlít a VGG-16 architektúrájára, de annál jóval kisebb. Ennek a teljesítményét szántam alapvonalnak, azt várván, hogy az átviteli tanulással tanított modellek ezt a teljesítményt fogják megugrani. Ez az egyik oka annak, hogy a tesztelési adathalmazt használtam a tanítás során validációs adathalmazként, gondolván, hogy a több tanulási adattal a modell csak jobban teljesíthet. A másik ok emellett, hogy összesen öt hiperparaméter kombinációt próbáltam ki vele, így a validációs adathalmazra való rátanulásnak kevés az esélye. Az első kísérletben használt háló architektúrája a többivel együtt a függelékben található.

A függelékben lévő táblázat minden sora egy réteget reprezentál. Az első oszlop mutatja a réteg típusát. A második oszlop n-eseinek utolsó tagja a konvolúciós és kiesési (dropout) rétegek esetében a filterek számát jelöli, a sűrű (Dense) rétegek esetében pedig a neuronok számát. Az alap modell legelső konvolúciós rétege tanh aktivációs függvényeket használ, míg az összes többi ReLU-t. Ennek az az oka, hogy amikor egy ehhez nagyon hasonló modellt a CIFAR-10 adathalmazon tanítottam, akkor a modell el se kezdett tanulni, ha az első rétegének aktivációs függvénye ReLU volt. Aközött és az itteni modell között csupán annyi a különbség, hogy a CIFAR-10-en tanított modellben nyolc helyett csak két 128 filteres konvolúciós réteg volt és a kimaradt konvolúciós blokkok utáni összegző (MaxPooling) rétegek is hiányoztak. A 6.1. ábrán kékkel van jelezve a modell teljesítménye a tanulás során a tanító adathalmazon és narancssárgával a validációs adathalmazon (amely itt még megegyezik a teszt adathalmazzal).

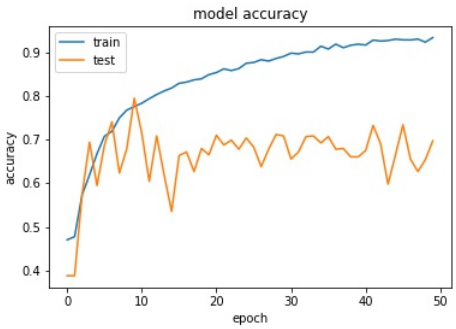


(a) Alapmodell pontossága

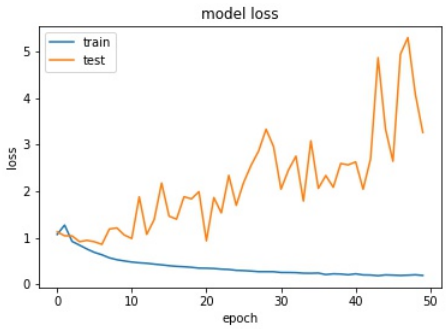


(b) Alapmodell veszteségfüggvénye

6.1. ábra. Az első kísérlet eredménye.



(a) Alapmodell pontossága

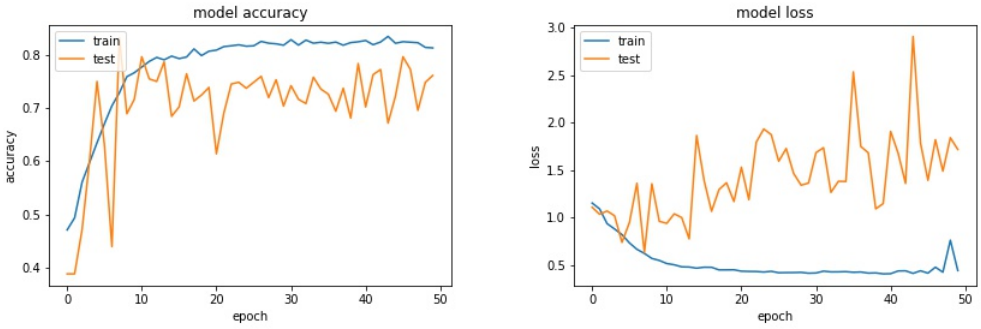


(b) Alapmodell veszteségfüggvénye

6.2. ábra. A második kísérlet eredménye az alapmodellel és 0,2-es kiesési rétegekkel.

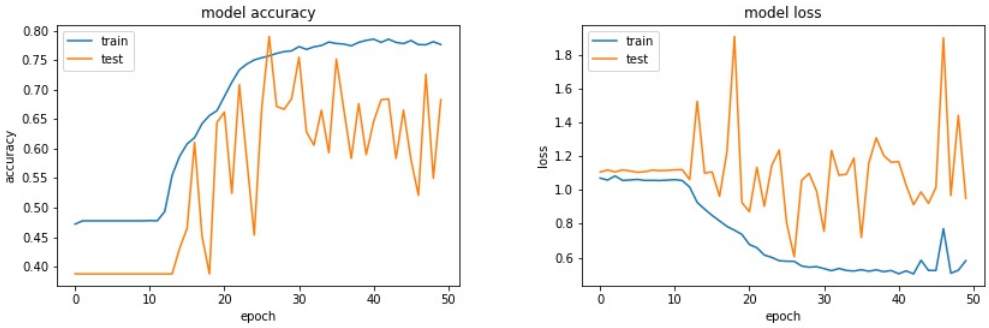
A 6.1. ábrán tisztán látszik, hogy a modell már az ötödik epoch környékén elér körülbelül 70% pontosságot, azonban regularizáció hiányában ezután elkezd a tanító adathalmazon túltanulni. A tesztelés grafikonja valószínűleg az adathalmaz kis mérete miatt ingadozik, ez a későbbi ábrákon is megfigyelhető lesz.

A következő négy kísérletem célja az alapmodell általánosító képességének növelése volt. A modell néhány rétege közé beraktam egy kiesési (dropout) réteget. A kiesési rétegek értékei egy modellen belül megegyeznek, csak a kiesési rétegek értékét változtattam 0,2-ről 0,4-re, majd 0,6-ra (ez $20\%, 40\%$ és 60% kiesést jelent). Ezeknek az eredménye látható a 6.2., 6.3. és 6.4. ábrákon. Ezen modellnek az architektúrája is a függelékben található.



(a) Alapmodell pontossága (b) Alapmodell veszteségfüggvénye

6.3. ábra. A harmadik kísérlet eredménye az alapmodellel és 0,4-es kiesési rétegekkel.



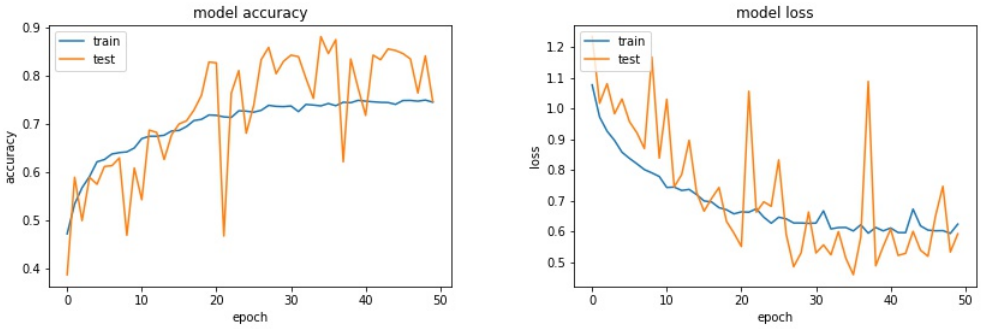
(a) Alapmodell pontossága (b) Alapmodell veszteségfüggvénye

6.4. ábra. A negyedik kísérlet eredménye az alapmodellel és 0,6-os kiesési rétegekkel.

Amint a 6.2., 6.3. és 6.4. ábrák grafikonján látható, a legjobb teljesítményt a 0,4-es kiesési rétegek érték el. A 0,6-os modellnek már nagyon nehezére esett a tanulás. Viszont a 0,4-es modell veszteségfüggvényéből látszik, hogy még mindig túltanul a modell tíz epoch után, még ha ez nem is jár a klasszifikálási pontosság romlásával a validációs adathalmazon. Emiatt döntöttem úgy, hogy augmentálom a tanulási adathalmazt. Az utolsó kísérlethez az alapmodellen és az összes átviteli tanulási kísérlethez a következő képaugmentációs technikát használtam a tanulási adathalmazon:

- legfeljebb 20°-os véletlen forgatást,
- legfeljebb 20%-os szélességi nyújtást,
- legfeljebb 20%-os magassági nyújtást,
- legfeljebb 20%-os zoomolást,
- véletlenszerű vízszintes tükrözés.

A képek fenti torzítását a tanítás során véletlenszerűen, minden képen külön elvégeztem mielőtt a képet a háló tanításához használtam. Az alapmodellel végzett utolsó kísérlethez tehát 0,4-es kiesési rétegeket és a fenti augmentációt használtam a tanító adathalmazon, ám a teszt adathalmazt csak átméreteztem 224×224-es méretűre, és lenormáltam a pixeleik értékét 0 és 1 közé. Ennek a kísérletnek az eredménye a 6.5. ábrán látható. Ez a modell körülbelül 80%-os pontosságot ért el a teszt adathalmazon. A pontossági grafikonon két érdekesség figyelhető meg. Az első a validációs pontosság grafikonjának ingadozása, melyre már korábban utaltam, hogy valószínűleg a validációs adathalmaz kis mérete az oka. A második érdekesség, hogy a validációs adathalmazon a modell pontossága a tizenötödik epoch után nagyobb, mint a tanító adathalmazon. Ez arra utal, hogy az augmentáció során valóban lényegi információvesztés történt. Ez azért jó jel, mert azt mutatja, hogy a modell a felvételek olyan tulajdonságaira tanult rá, amelyek valóban kapcsolódnak a vírusos vagy bakteriális fertőzés jelenlétéhez, és nem csupán az augmentált képeken lévő zajra.



(a) Alapmodell pontossága (b) Alapmodell veszteségfüggvénye

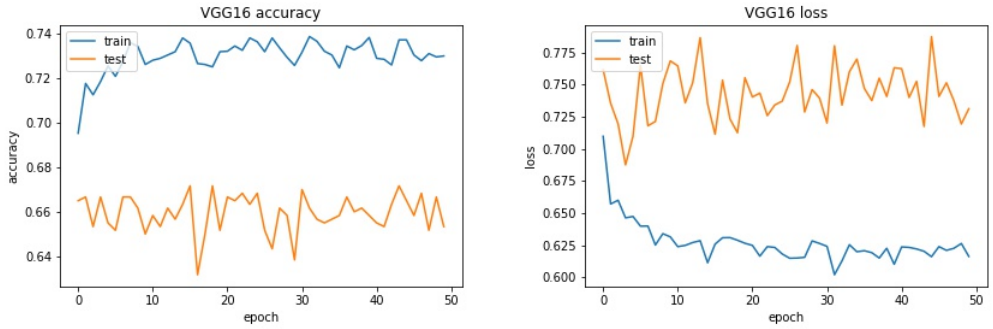
6.5. ábra. Az ötödik kísérlet eredménye az alapmodellel, és 0,4-es kiesési rétegekkel és augmentációval.

Ezután a VGG-16 modellel próbáltam átviteli tanulás segítségével javítani a klasszifikáció pontosságán. Az VGG-16 rétegei a függelékben találhatók. Annak megállapítására, hogy pontosan milyen architektúrájú sűrű rétegeket érdemes a háló végére rakni, egy véletlen keresést végeztem. Tíz különböző modellt próbáltam ki, mindegyiket 5 epochig tanítva. Három dolog változott egymástól függetlenül a kísérletek közt:

- a sűrű rétegek száma 1 és 4 között,
- a sűrű rétegek neuron száma (külön-külön) 32, 64, 128, 512, 1024 számok között,
- a gradiens módszer változata Adam és RMSProp közül.

A legjobb modellnek nagyon furcsák voltak a hiperparaméterei (négy sűrű réteg 32, 32, 1024, 32 neuronszámmal), ezért a második legjobban teljesítő modellt használtam a tanításra 0,2-es kiesési rétegekkel. Ennek a modellnek is a Függelékben található az architektúrája (négy sűrű réteg 512, 1024, 128, 512 neuronnal, és Adam módszerrel). Ennek a kísérletnek az eredménye a tanítás során a tanító és teszt adathalmazon a 6.6. ábrán látható. A modell nagyon gyorsan, már az első epoch alatt 65% pontosságot ért el, ám ezután nem sikerült többet tanulnia, ráadásul jelentős mértékben túltanult a tanító adathalmazra. A modell pontossága 78,20% míg a vesztesége 0,63 volt a tanítás végén a teszt adathalmazon.

A második modell, amelyet az átviteli tanuláshoz használtam a BiT-M R101x1 modell volt. Ennek a pontos felépítéséről és tanításáról Kolesnikov és tsai. [9] írnak részletesebben. A cikkben írnak a BiT-HyperRule-ról amely



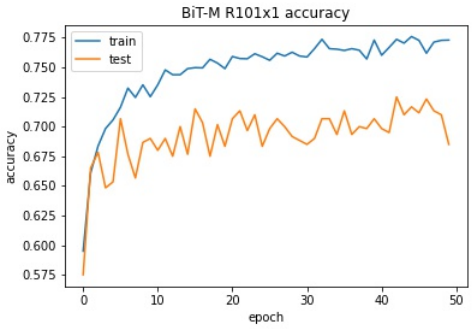
(a) VGG-16 pontossága (b) VGG-16 veszteségfüggvénye

6.6. ábra. A VGG-16, mint tulajdonság-kifejtő

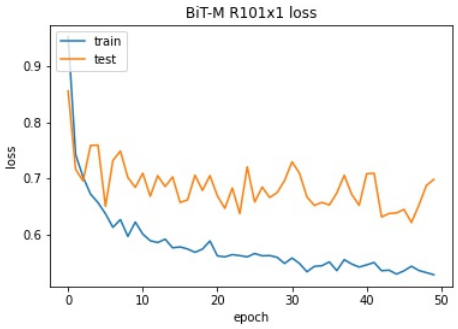
egy sok feladaton jól működő heurisztika a hiperparaméterek beállítására. Eszerint érdemes SGD (a dolgozatom fogalmai szerint mini-batch) gradiens módszert 0,003 kezdeti tanulási rátával és 0,9-es momentummal használni a tanításra, és a tanulási rátát tizedére csökkenteni a tanulási folyamat $30\%, 60\%$ és 90%-ánál. Ezen kívül augmentálásra az adatok átméretezését, véletlen részeik kivágását és tükrözését ajánlják. A 0,9 momentumos mini-batch gradiens módszert megtartottam, azonban az 512 batch méret túl nagynak bizonyult memória szempontjából, ezért itt is, ahogy az összes korábbi kísérletnél 16-os batch méretet használtam. A modell sűrű rétegei számának és alakjának meghatározásához hasonlóan a VGG-16 tanításához tíz modellt próbáltam ki a VGG-16-nál ismertetett hiperparaméter kombinációk közül, hogy lássam milyen hiperparaméterekkel tanulnak jól a modellek. Továbbá a tanulási rátát is mind a tíz alkalommal véletlenül választottam a 0,03, 0,003 és 0,0003 értékek közül.

A tíz hiperparaméter kísérlet után azt tapasztaltam, hogy minél kisebb volt a tanulási ráta annál jobban teljesítettek a modellek 5 epoch után. Emellett a 32-es sűrű rétegek túl keskenynek bizonyultak, vagyis azok a modellek, melyekben volt 32-es sűrű réteg rosszul teljesítettek. Ezért döntöttem három sűrű réteg mellett $1024, 512$ és 128 súlyokkal és 0,2-es kiesési rétegekkel. A teljes modell felépítése a függelékben található, ahol a KerasLayer réteg a teljes BiT-M R101x1 hálót tartalmazza. A tanítás során a tanulási és validációs adathalmazon a modell teljesítménye a 6.7. ábrán látható. Ez a modell A VGG-16-hoz nagyon hasonló 78,21%-os pontosságot és 0,55 veszteséget ért el a tanítás végén a teszt adathalmazon. A 6.7. ábrán az látszik, hogy a modell még az ötvenedik epoch környékén is tanult, ezért elképzelhető, hogy további tanítással néhány százalékkal jobb eredményt ért volna el.

Végül megpróbáltam a két átviteli tanuláshoz használt teljes modellt ta-

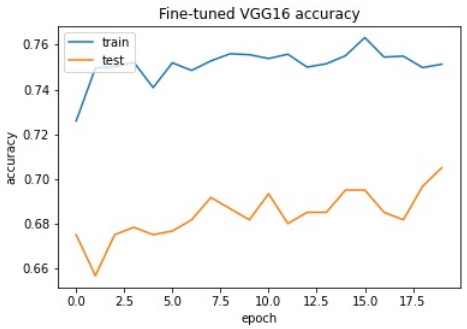


(a) BiT-M R101x1 pontossága

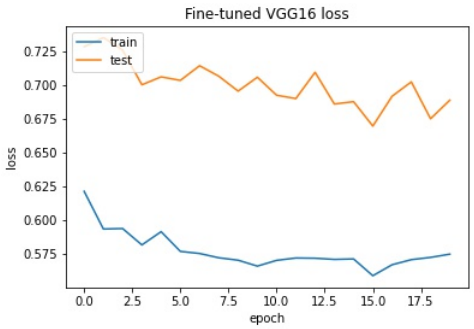


(b) BiT-M R101x1 veszteségfüggvénye

6.7. ábra. A BiT-M R101x1, mint tulajdonság-kifejtő



(a) VGG-16 pontossága



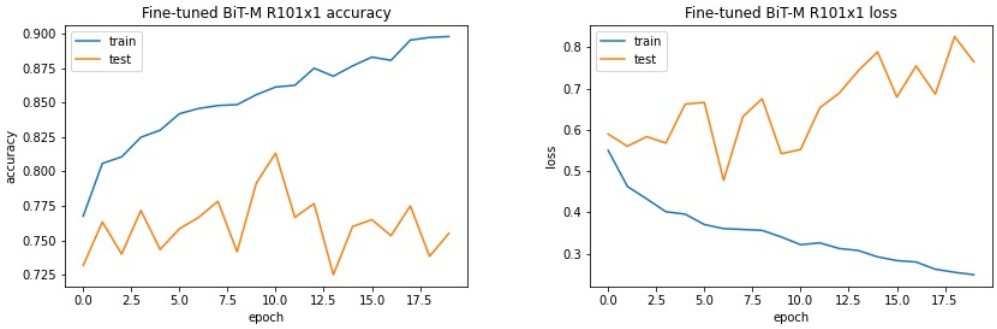
(b) VGG-16 veszteségfüggvénye

6.8. ábra. A VGG-16 eredeti súlyinak továbbtanítása

nítani. Ezeknek a mérete jóval nagyobb volt (a függelékben megtalálható a paraméter számuk), ezért jelentősen lassabb volt őket tanítani. Ez azért jelentett komoly problémát, mert a Colab hosszú videokártya használat után néha felszámolja a kapcsolatot, felszabadítva a videokártyát. Emiatt csupán 20 epochon át tudtam tanítani a teljes modelleket.

Először a VGG-16-on tanult modellt próbáltam továbbtanítani 0,9-es momentumos mini-batch módszerrel és 0,0003 tanulási rátával. Ezen kívül a korábbi túltanulás ellen L_2 regularizációt használtam $\lambda = 0{,}001$-el, de csak a sűrű rétegek súlyain. Ennek az eredménye a 6.8. ábrán látható. Itt a validációs pontosság néhány százalékkal nőtt, azonban a továbbtanított modell a teszt adathalmazon csupán 75,00%-ot és 0,72 veszteséget ért el. Tehát a továbbtanítás előttihez képest 3,2%-kal pontatlanabb volt.

Az utolsó kísérletem a BiT-M R101x1 továbbtanítása volt. Ehhez a VGG-16-hoz hasonlóan 0,9-es momentumos mini-batch módszert és 0,0003 tanulási



(a) BiT-M R101x1 pontossága (b) BiT-M R101x1 veszteségfüggvénye

6.9. ábra. A BiT-M R101x1 súlyainak továbbtanítása

rátát használtam. Ennek az eredménye a 6.9. ábrán látható. Ezen az látszik, hogy ugyan a modell a nagyszámú tanulható paraméter következtében azonnal elkezdett túltanulni a tanító adathalmazon. Érdekes módon a teszt adathalmazon ez teljesített a legjobban: A tanítás végén 80,12%-os pontosságot ért el 0,70 veszteséggel.

Egy érdekes megfigyelés, hogy a modellek jellemzően magasabb pontosságot értek el a teszt adathalmazon, mint a validációs adathalmazon. Ez véleményem szerint arra utal, hogy az adathalmaz egyes kategóriáinak tipikusabb példái kerültek a teszt adathalmazba.

7. fejezet

Mérések a CheXpert adathalmazon

A dolgozatom korábbi fejezeteinek 2020-as megírása után tovább foglalkoztam az ImageNet képekről tüdőröntgen képekre történő átviteli tanulással. Így a többi fejezettel ellentétben, ebben a fejezetben a 2021-es eredményeimet ismertetem. Két kérdést vizsgáltam Az első, hogy az ImageNeten való előtanulás tényleg javítja-e a tüdőröntgeneken elért eredményeket. A kérdés azért nem nyilvánvaló, mivel a röntgenképek fekete-fehérek (a színes ImageNet képekkel ellentétben), jóval kevesebb osztályba kell őket klasszifikálni, illetve az ImageNetes képek nagyon sokkal változatosabbak. A második, hogy igaz-e az állítás, hogy minél kisebb a rendelkezésre álló tanító adathalmaz, annál fontosabb, hogy elő legyen tanítva egy modell.

7.1. Egyéb tüdőröntgen adathalmazok

A három legnagyobb tüdőröntgen adathalmaz a CXR14, ami a CXR8-at [22], hozzávett képeket és 6 új címkét tartalmaz, a CheXpert [6], ami 2002 és 2017 közötti felvételeket tartalmaz a Stanford Hospital adatbázisából, és a MIMIC-CXR [7], amely a Beth Israel Deaconess Medical Center 2011 és 2016 közötti felvételeit tartalmazza. Mindhárom adathalmazhoz természetes nyelvfeldolgozási (Natural Language Prosessing) módszerekkel generálták a képekhez tartozó címkéket, a képekhez tartozó szöveges orvosi leletekből. Mivel a CXR-14-hez tartozik a legkevesebb (112.120) felvétel, és valószínűleg ehhez tartoznak a legzajosabb címkék, illetve a MIMIC-CXR-t nem sikerült beszereznem, ezért a CheXpert adatbázist használtam.

A CheXpert 224 316 képet tartalmaz, 13 egymástól nagyrészt független megfigyeléshez tartozó címkével, 1 külön címkével a normális képeknek. A megfigyelésekhez tartozó címkék mindegyike 0, -1, 1 értékeket vehet fel, vagy hiányozhat, ahol 1 jelez pozitívat, 0 negatívat, -1 bizonytalant, míg ha a címke hiányzik, az arra utal, hogy a lelet nem említette az adott megfigyelést. Az adathalmazban szerepelnek frontális, illetve oldalsó nézetből készült felvételek is, azonban én csak a frontálisokat használtam, mivel a két nézetből készült felvételek jelentősen különböznek egymástól, így a modelleknek gyakorlatilag két külön feladatot kellene egyszerre megoldaniuk.

A választott feladat, aminek segítségével az előtanítás hasznosságát vizsgálom a tüdőopacitás jelenléte. Azért ezt választottam, mivel sokkal több az ehhez tartozó pozitív kép, mint a tüdőgyulladáshoz, illetve a felvételeken látható információ gyakran nem elég annak eldöntéséhez, hogy a felvételen látható rendellenesség tüdőgyulladás-e. Ezzel szemben azt megállapítani, hogy van-e rendellenesség biztosabban meg lehet tenni pusztán a felvétel alapján. A adathalmazban az oldalsó nézetbeli képek kiszűrése után 94 211 tüdőopacitásra pozitív, 5051 negatív kép volt 16 974 teljesen egészséges képpel együtt. A negatív és egészséges képeket összevonva 20 170 negatív felvételem volt. Azért, hogy kiegyensúlyozott adathalmazzal dolgozhassak véletlenszerűen kiválasztottam 19 000 pozitív és 19 000 negatív példát a tanítóhalmazhoz, és 1000-1000 példát a validációs halmazhoz. Hogy a különböző méretű adathalmazokon való tanítások összehasonlíthatóak legyenek, a kisebb adathalmazokat egy epochon belül annyiszor megismételtem, ahányszor belefért a 38 000-es tanító halmazba. Ez azonban azt jelenti, hogy mivel minden epoch után megkevertem a tanítás során a tanító halmazt, így nem kizárt, hogy egy-egy batch-be ugyan az a kép kétszer is belekerüljön.

7.2. A hálók és hiperparamétereik

Mivel ezekhez a kísérletekhez hozzáférhettem az ELTE Ai Research Group RTX 2080 Ti GPU-ihoz, a batch és képméreteket úgy választottam, hogy még éppen beférjenek a GPU-k memóriájába. Sabottke és Spieler [15] szerint a 256×256-os és 448×448-as felbontás az optimális, és mivel a modellek az ImageNeten 224×224-es felbontáson tanultak, hogy ettől ne térjek el nagyon, $256 \times 256 \times 3$-as felbontást használtam. Itt a 3 a színcsatornák számára utal: a szürke tüdőröntgen képeket 3 azonos csatornára bontottam, hogy kompatibilisek legyenek az előtanított modellekkel. Emellett a felbontás mellett 64 kép még mindig belefért egy batch-be, így ezt a batch méretet választottam.

Optimalizálónak kipróbáltam a CoolMomentumot [1] ami előzetes mérések alapján nagyjából olyan jónak tűnt, mint az SGD. Ehhez $\rho_0 = 0{,}99$, $\alpha = 0{,}99997$ értékeket és 0,0001-es kezdeti tanulási rátát alkalmaztam. Ezután, a biztos konvergencia érdekében a tanulás során epochonként exponenciálisan csökkentettem a tanulási rátát, legfeljebb 100 epoch alatt legfeljebb a kezdeti tanulási ráta 100-ad részére. Azért csak legfeljebb ennyit, mivel, a kis adathalmazokat gyorsabban megtanulták a modellek, így, hogy fölöslegesen ne fussanak a GPU-k leállítottam a tanítást, ha 5 epoch alatt a tanítási halmazon vett pontosság nem javult legalább 1%-ot. Ez egy nagyon engedékeny korai leállási feltétel, így jellemzően tényleg csak akkor lépett életbe, amikor a modellek már interpolálták a tanító adathalmazt (100%-os pontosság).

Az átviteli tanulás alatt itt a 4.2. fejezetben leírt megközelítések közül a továbbtanítást (fine-tuning) választottam, tehát az egész modellt tanítottam, csak a súlyokat inicializáltam vagy véletlenszerűen, vagy ImageNetről előtanítva. mivel az egész modellt tanítottam, elég volt a modellek konvolúciós rétege után egyetlen sűrű klasszifikációs réteg, amit mindig véletlen súlyokkal inicializáltam.

Három különböző modellt is megvizsgáltam: a már korábban használt VGG16-ot, a ResNet50-et, ami egy kicsit kisebb tagja a korábban használt ResNet családnak, illetve az InceptionV3-at [20], az Inception egy továbbfejlesztett változatát. Ezek nagyjából összehasonlíthatóak a paraméter számukat tekintve,

- VGG16 [17] (14.715.201 tanítható paraméter és 14.715.201 összesen),
- ResNet50 [5](23.589.761 tanítható paraméter és 23.536.641 összesen),
- InceptionV3 [20] (21.804.833 tanítható paraméter és 21.770.401 összesen),

viszont az architektúrájukat tekintve nagyon különbözőek. Ke és tsai. [8] szerint az a fontosabb, hogy melyik modell családba tartozik egy adott modell, és kevésbé, az hogy a modell családon belül mekkora. Ezért használtam három különböző modell család egy-egy reprezentánsát.

Augmentációnak minden képen alkalmaztam véletlen vízszintes tükrözést. A legkisebb, összesen 200 képből álló tanító adathalmaznak ezen kívül a fényerejét változtattam, úgy hogy a képeken minden pixelhez legfeljebb $\pm 0{,}2$-t adtam. Az összesen 2000 képből álló tanító adathalmaz képeit legfeljebb $2 * \pi * 0{,}05 = 0{,}1 * \pi$-vel elforgattam, az üres pixeleket az eredeti kép tükrözésével kitöltve. Előzetes kísérletek alapján ezek tűntek a legjobb

augmentációknak és értékeknek (kipróbáltam még a véletlen kivágást is). A nagyobb, 8000 és 38000 képből álló adathalmazokon az előzetes kísérletezés során nem javítottak az egyéb augmentációk, ezért azokat csak tükröztem.

A mérések között tehát a következő három dolog változott:

- a modell architektúrája (VGG16, ResNet50, InceptionV3),
- a modell konvolúciós súlyainak inicializálása (véletlen, vagy ImageNeten előtanított),
- a tanító halmaz mérete (200, 2000, 8000 és 38 000, vagyis kategóriánként 100, 1000, 4000 és 19 000).

A tanulási ráta ütemezés, hosszú tanítási idő, és a kevés regularizáció hatására a modellek túltanultak, így az itt ismertetett eredmények nem a tanítás végiek, hanem a tanítás közbeni legjobb epoch utániak.

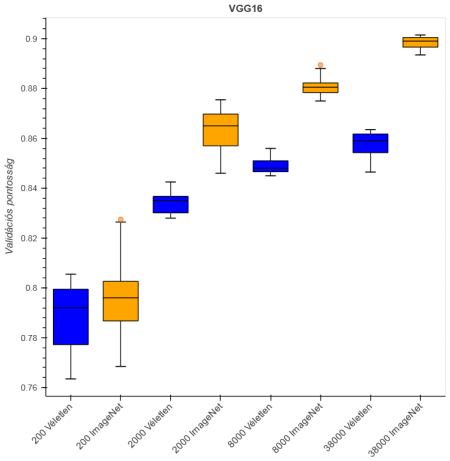
7.3. Mérési eredmények

A 7.2. fejezetben ismertetett hiperparaméterek közül (modell, kezdeti súlyok, tanulási halmaz mérete) minden kombinációt 10-szer kipróbáltam és a tanítás közbeni legjobb validációs eredményeket átlagoltam. A 10-szeres futtatásra azért volt szükség, mivel a tanítás során véletlenített folyamatok miatt (epochok előtt a tanítóhalmaz keverése, súlyok inicializálása, illetve a kisebb tanító adathalmazok esetében a teljes tanító halmaz felhasznált részhalmazának kiválasztása) az elért pontosságok futásról futásra változtak. A futtatások eredményeit a 7.1. ábra tartalmazza. Itt jól látszik, hogy a szórásban a legfontosabb tényező az adathalmaz mérete volt, látható, hogy a legkisebb, 200 tanító képből álló adathalmaz esetében volt a legnagyobb. Ez érthető, hiszen, amikor ilyen kevés képpel próbálunk tanítani, akkor a véletlenül beválogatott képek tartalmának minősége nagyban befolyásolhatja a modell teljesítményét és általánosító képességét. Az ábrán az is jól látszik, hogy nyilván minél nagyobb volt a tanító adathalmaz mérete, annál jobban tudott a modell általánosítani, és az is, hogy az ImageNetes súlyok használata mindig javított a modellek teljesítményén. Ami viszont meglepő, az a javulás mértéke, ugyanis azt várnánk, hogy minél kisebb az adathalmaz annál többet számít az előtanítás, hogy a kevés tanítóképen található információt ne az alacsonyabb rendű mintázatok (élek, sarkok stb.), hanem az egyéni feladat sajátosságainak megtanulására tudja használni. Ezzel szemben itt azt tapasztalhatjuk, hogy a 200 méretű tanító adathalmazon keveset, míg a

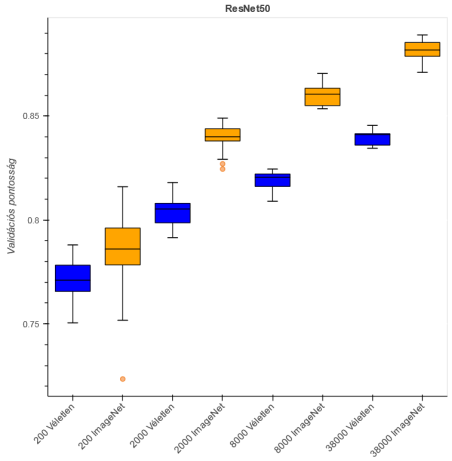
Tanító halmaz mérete	Modell	Átlagos pontosság véletlen súlyokkal (%)	Átlagos pontosság előtanított súlyokkal (%)	Különbség (%)
200	VGG16	78,78	79,59	+0,81
200	ResNet50	77,06	78,32	+1,26
200	InceptionV3	77,07	77,65	+0,58
2000	VGG16	83,46	86,33	+2,87
2000	ResNet50	80,41	83,90	+3,49
2000	InceptionV3	79,35	84,33	+4,98
8000	VGG16	84,91	88,09	+3,18
8000	ResNet50	81,87	86,06	+4,19
8000	InceptionV3	81,69	85,31	+3,62
38 000	VGG16	85,74	89,83	+4,09
38 000	ResNet50	83,97	88,13	+4,16
38 000	InceptionV3	84,57	88,31	+3,74

7.1. táblázat. Tanulási pontosság javulása ImageNeten előtanított súlyok hatására a validációs halmazon.

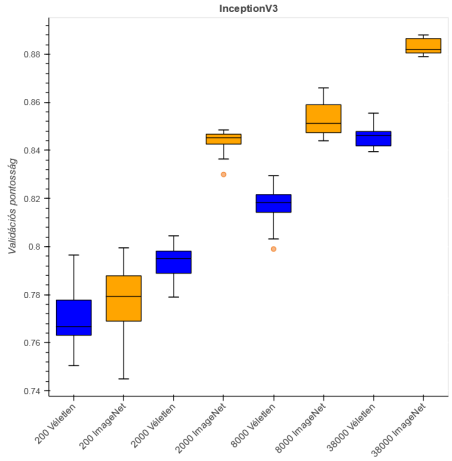
többi adathalmazon többet számított az előtanítás. A pontos mérési eredmények a 7.1. táblázatban olvashatók. A szemléletesség kedvéért a táblázat utolsó oszlopa a 7.2. grafikonon is ki van emelve. Ezen azonnal látszik, hogy a 200-as méretű tanító adathalmazon tanult modelleken keveset javított az előtanítás. Ennek az lehet az oka, hogy ahhoz, hogy az előtanított súlyokból nyert információt tényleg ki tudja használni egy modell, ahhoz előfeltétel, hogy elégséges adat legyen a feladat rendes megtanulására. Bíztató azonban, hogy az előtanítás hatása még a legnagyobb, 38 000-es tanító halmazon tanult modell esetében sem gyengült, ami azt jelenti, hogy még ilyen nagyságrendű adathalmaznál is érdemes előtanítást használni; tehát ahhoz, hogy csökkenjen az előtanítás fontossága legalább több százezres adathalmazon szükséges a modelleket tanítani.



(a) VGG16

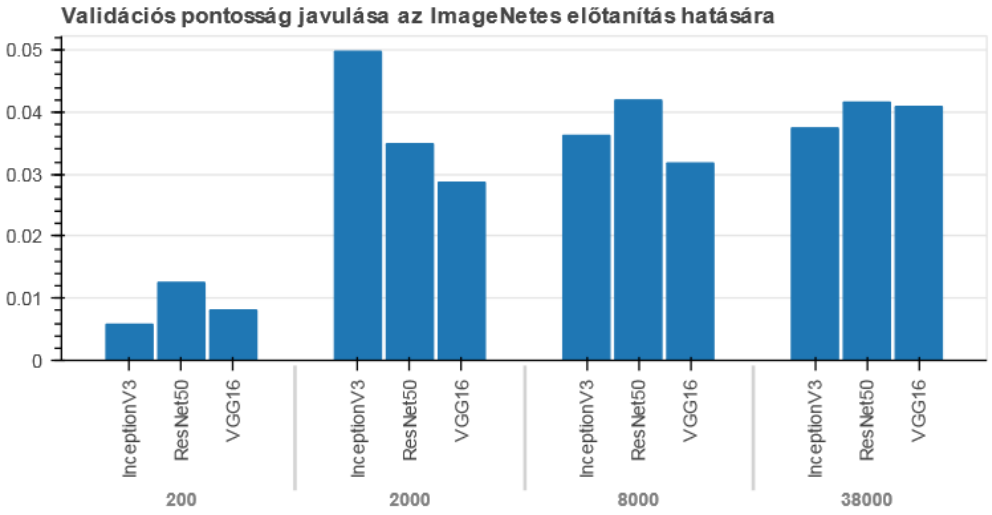


(b) ResNet50



(c) InceptionV3

7.1. ábra. Validációs pontosságok összesítése. Minden doboz 10 mérést jelöl: a középső csík a medián, a dobozok teteje, illetve alja az interkvartilis tartomány (azaz a 25. és 75. percentilis közti tartomány) határát jelöli. A kiugró értékek sárga körrel, míg a véletlen módon inicializált modellek kékkel, az ImageNeten előtanult modellek narancssárgával vannak jelölve.



7.2. ábra. Boost in accuracy

8. fejezet

Konklúzió

Az elvárásaim ellenére az első, 6. fejezetben bemutatott kísérlet sorozatban az átviteli tanulás nem javított lényegesen a modelljeim klasszifikációs pontosságán. Az összes modell 75 és 80 százalékos pontosság közötti eredményt ért el a teszt adathalmazon. Ez véleményem szerint két dolgot jelenthet. Az első, hogy a képek augmentációja és zsugorítása során túl nagy mértékű információvesztés történt, és az augmentált adathalmazból az összes fontos információt kinyertem. Ezt az támasztja alá, hogy még a Google által tanított, robosztus modell sem volt képes az eredmény jelentős javítására. A másik, hogy az átvitelhez használt módszer (feature extraction), vagy az adathalmaz nem volt megfelelő.

Amikor később egy nagyobb adathalmazon és több számítógépes erőforrás segítségével pontosabban kimértem az átviteli tanulás hatását és a tanítás elejétől engedtem az egész modellt tanulni (fine-tuning), akkor sokkal egyértelműbben kitűnt az előtanított súlyok pozitív hatása mind a validációs pontosságon, mind a tanításhoz szükséges időben.

Fontos megjegyezni azonban hogy a két kísérletsorozat eredményei nem hasonlíthatóak közvetlenül össze, nemcsak azért mert más adathalmazt használtak, vagy más módszerekkel tanítottam, hanem azért is, mert más volt a feladat, amit a modelleknek meg kellett tanulniuk. Az első kísérletsorozatban 3 kategóriába kellett a felvételeket klasszifikálni, a másodikban csak kettőbe, így már önmagában emiatt sem meglepő, ha ezek a modellek nagyobb pontosságot értek el.

Összességében pozitív eredménynek tekinthető, hogy a tüdőröntgen képek klasszifikálása konvolúciós hálók számára tanulható problémának tűnik. Bíztató, hogy még egy egyszerű modell is képes viszonylag jól tud teljesíteni.

9. fejezet

Függelék

Az első (alap) háló architektúrája dropout rétegek nélkül

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 224, 224, 32)      896
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 224, 224, 32)      9248
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 112, 112, 32)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 112, 112, 64)      18496
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 112, 112, 64)      36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 56, 56, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 56, 56, 128)       73856
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 56, 56, 128)       147584
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 28, 28, 128)       0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 28, 28, 128)       147584
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 28, 28, 128)       147584
_________________________________________________________________
```

```
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 14, 14, 128)       147584
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 14, 14, 128)       147584
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 7, 7, 128)         147584
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 7, 7, 128)         147584
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 3, 3, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 1152)              0
_________________________________________________________________
dense (Dense)                (None, 128)               147584
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 387
=================================================================
Total params: 1,320,483
Trainable params: 1,320,483
Non-trainable params: 0
_________________________________________________________________
```

Az első (alap) háló architektúrája dropout rétegekkel együtt

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 224, 224, 32)      896
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 224, 224, 32)      9248
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 112, 112, 32)      0
_________________________________________________________________
dropout_7 (Dropout)          (None, 112, 112, 32)      0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 112, 112, 64)      18496
```

```
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 112, 112, 64)      36928
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 56, 56, 64)        0
_________________________________________________________________
dropout_8 (Dropout)          (None, 56, 56, 64)        0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 56, 56, 128)       73856
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 56, 56, 128)       147584
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 28, 28, 128)       0
_________________________________________________________________
dropout_9 (Dropout)          (None, 28, 28, 128)       0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 28, 28, 128)       147584
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 28, 28, 128)       147584
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 14, 14, 128)       0
_________________________________________________________________
dropout_10 (Dropout)         (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 14, 14, 128)       147584
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 14, 14, 128)       147584
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 7, 7, 128)         0
_________________________________________________________________
dropout_11 (Dropout)         (None, 7, 7, 128)         0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 7, 7, 128)         147584
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 7, 7, 128)         147584
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 3, 3, 128)         0
_________________________________________________________________
dropout_12 (Dropout)         (None, 3, 3, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1152)              0
_________________________________________________________________
```

```
dense_2 (Dense)              (None, 128)               147584
_________________________________________________________________
dropout_13 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 387
=================================================================
Total params: 1,320,483
Trainable params: 1,320,483
Non-trainable params: 0
_________________________________________________________________
```

A VGG-16 háló architektúrája

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
```

```
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
global_max_pooling2d (Global (None, 512)               0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

A VGG-16 háló sűrű rétegekkel

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 512)               14714688
_________________________________________________________________
dropout_16 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_29 (Dense)             (None, 512)               262656
_________________________________________________________________
dropout_17 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_30 (Dense)             (None, 1024)              525312
_________________________________________________________________
dropout_18 (Dropout)         (None, 1024)              0
_________________________________________________________________
dense_31 (Dense)             (None, 128)               131200
_________________________________________________________________
dropout_19 (Dropout)         (None, 128)               0
```

```
_________________________________________________________________
dense_32 (Dense)             (None, 512)               66048
_________________________________________________________________
dense_33 (Dense)             (None, 3)                 1539
=================================================================
Total params: 15,701,443
Trainable params: 986,755
Non-trainable params: 14,714,688
_________________________________________________________________
```

A BiT-M R101x1 háló sűrű rétegekkel

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
keras_layer (KerasLayer)     (None, 2048)              42492480
_________________________________________________________________
dropout_6 (Dropout)          (None, 2048)              0
_________________________________________________________________
dense_11 (Dense)             (None, 1024)              2098176
_________________________________________________________________
dropout_7 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_12 (Dense)             (None, 512)               524800
_________________________________________________________________
dropout_8 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_13 (Dense)             (None, 128)               65664
_________________________________________________________________
dense_14 (Dense)             (None, 3)                 387
=================================================================
Total params: 45,181,507
Trainable params: 2,689,027
Non-trainable params: 42,492,480
_________________________________________________________________
```

Irodalom

[1] Oleksandr Borysenko és Maksym Byshkin. "CoolMomentum: A Method for Stochastic Optimization by Langevin Dynamics with Simulated Annealing". *arXiv preprint arXiv:2005.14605* (2020).

[2] Joseph Paul Cohen, Paul Morrison és Lan Dao. "COVID-19 image data collection". *arXiv 2003.11597* (2020). URL: `https://github.com/ieee8023/covid-chestxray-dataset`.

[3] Ian Goodfellow, Yoshua Bengio és Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[4] Kaiming He és tsai. "Deep Residual Learning for Image Recognition". *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016. jún.

[5] Kaiming He és tsai. "Deep residual learning for image recognition". *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, 770–778. old.

[6] Jeremy Irvin és tsai. "Chexpert: A large chest radiograph dataset with uncertainty labels and expert comparison". *Proceedings of the AAAI Conference on Artificial Intelligence*. 33. köt. 01. 2019, 590–597. old.

[7] Alistair EW Johnson és tsai. "MIMIC-CXR-JPG, a large publicly available database of labeled chest radiographs". *arXiv preprint arXiv:1901.07042* (2019).

[8] Alexander Ke és tsai. "CheXtransfer: performance and parameter efficiency of ImageNet models for chest X-Ray interpretation". *arXiv preprint arXiv:2101.06871* (2021).

[9] Alexander Kolesnikov és tsai. "Large Scale Learning of General Visual Representations for Transfer". *arXiv preprint arXiv:1912.11370* (2019).

[10] Alex Krizhevsky, Ilya Sutskever és Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". *Advances in neural information processing systems*. 2012, 1097–1105. old.

[11] Geert Litjens és tsai. “A survey on deep learning in medical image analysis”. *Medical image analysis* 42 (2017), 60–88. old.

[12] Sinno Jialin Pan és Qiang Yang. “A survey on transfer learning”. *IEEE Transactions on knowledge and data engineering* 22.10 (2009), 1345–1359. old.

[13] Sebastian Ruder. “An overview of gradient descent optimization algorithms”. *arXiv preprint arXiv:1609.04747* (2016).

[14] Olga Russakovsky és tsai. “Imagenet large scale visual recognition challenge”. *International journal of computer vision* 115.3 (2015), 211–252. old.

[15] Carl F Sabottke és Bradley M Spieler. “The effect of image resolution on deep learning in radiography”. *Radiology: Artificial Intelligence* 2.1 (2020), e190015.

[16] F. Shi és tsai. “Review of Artificial Intelligence Techniques in Imaging Data Acquisition, Segmentation and Diagnosis for COVID-19”. *IEEE Reviews in Biomedical Engineering* (2020), 1–1. old.

[17] Karen Simonyan és Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. *arXiv preprint arXiv:1409.1556* (2014).

[18] Nitish Srivastava és tsai. “Dropout: a simple way to prevent neural networks from overfitting”. *The journal of machine learning research* 15.1 (2014), 1929–1958. old.

[19] Christian Szegedy és tsai. “Going deeper with convolutions”. *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, 1–9. old.

[20] Christian Szegedy és tsai. “Rethinking the inception architecture for computer vision”. *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, 2818–2826. old.

[21] Linda Wang és Alexander Wong. “COVID-Net: A tailored deep convolutional neural network design for detection of COVID-19 cases from chest radiography images”. *arXiv preprint arXiv:2003.09871* (2020).

[22] Xiaosong Wang és tsai. “Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases”. *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, 2097–2106. old.

[23] Jason Yosinski és tsai. “How transferable are features in deep neural networks?”: *arXiv preprint arXiv:1411.1792* (2014).