Mélytanulási módszerek az orvosi képalkotó diagnosztikában

Szakdolgozat

Katona Réka
Matematikai elemző szakirány

Témavezető:

Csiszárik Adrián
MTA Rényi Alfréd Matematikai Kutatóintézet

Belső konzulens:

Lukács András
Számítógéptudományi Tanszék
Eötvös Loránd Tudományegyetem, Természettudományi Kar

Eötvös Loránd Tudományegyetem
Természettudományi Kar
2018

Tartalomjegyzék

Ábrák jegyzéke

Köszönetnyilvánítás

Szeretném megköszönni témavezetőmnek, Csiszárik Adriánnak, hogy megismertette velem a témakört, és tudásával, szakértelmével biztosította, hogy minden felmerülő kérdésemre választ kapjak. Köszönöm a rugalmasságát, türelmét, a konzultációkat, a mérhetetlen időt és munkát, a rengeteg tanácsot, amit a hónapok során kaptam. Hálás vagyok érte, hogy ilyen sok és értékes új ismerettel lehettem gazdagabb. Köszönöm Lukács Andrásnak, hogy belső konzulensként tanácsaival segítette munkámat. Hálás vagyok a családomnak és barátaimnak, hogy a szakdolgozatom írása során türelemmel, bizalommal támogattak. Emellett köszönöm az MTA Rényi Alfréd Matematikai Kutatóintézetnek, hogy biztosították számomra a szükséges infrastruktúrát.

1. fejezet

Bevezetés

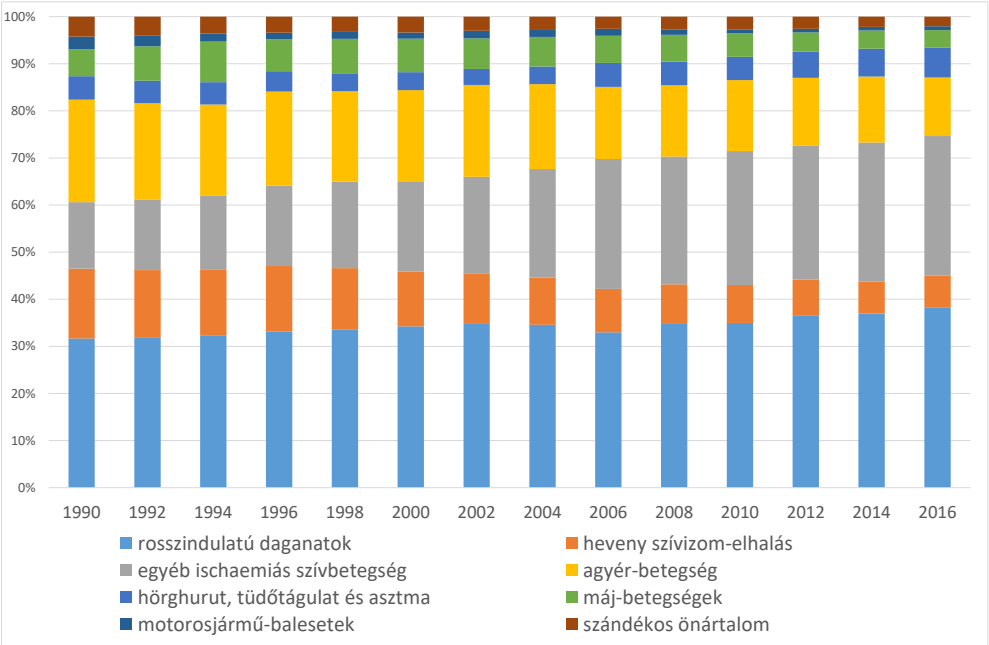
Az elmúlt években a gépi tanuló módszerek használata robbanásszerű fejlődést hozott a mindennapi és a tudományos életbe. A közelmúltban számos olyan áttörés történt, amelyeknek köszönhetően olyan feladatok váltak könnyen és gyorsan megoldhatóvá, amelyek korábban nagyon lassan vagy egyáltalán nem lettek volna kivitelezhetők. A fejlődés hátterében az egyre nagyobb mennyiségben keletkező adat, az adatokat hatékonyan kezelni képes modellek és a számítástechnikai megoldások állnak. A kapacitáshiány problémájának megoldása lehetővé tette az algoritmusok további fejlődését. Napjainkban és a közeljövőben a mesterséges intelligencia hatalmas területe és azon belül a gépi tanulás olyan feladatok megoldására is használhatók lesznek, amelyekre mindeddig nem volt lehetőség.

A gépi tanuló módszercsalád fontos, társadalmilag hasznos alkalmazási lehetősége az orvosi diagnosztika, amely területen már bizonyított [1, 9]. A technológiák fejlődésével egyre több és részletesebb diagnosztikai célú felvétel készül a páciensekről. Ezért kiemelten fontos az, hogy olyan eljárásokat, módszereket alakítsunk ki, amelyekkel ez a nagy mennyiségű adat gyorsan, és hatékonyan feldolgozható lehet a betegségek minél korábbi felismerésének érdekében.

Dolgozatom célja az, hogy bemutassam, hogy a képi adatok feldolgozásában oly sikeres gépi tanuló algoritmusok, azon belül is mesterséges neurális hálózatok módszercsaládja hogyan használható képalkotó orvosi diagnosztikai eljárások esetében.

A számos felhasználási lehetőség közül (pl. bőrelváltozások, röntgen- és CT-felvételek kielemzése) a rákos elváltozások detektálását tűztem ki célként CT-felvételekről. Azért tartom fontosnak ezt a témát, mert a világ legtöbb országához hasonlóan Magyarországon is vezető helyen állnak a rosszindulatú daganatok a halálozási okok listáján 1.1.

A késői felismerésből fakadó nagyfokú halálozási arány egyértelműen csökkenthető lenne a rák korai stádiumban való felismerésével és diagnózis felállításával. A



1.1. ábra. Kétévenkénti halálozások száma a leggyakoribb halálokok szerint Magyarországon (`http://www.ksh.hu/docs/hun/xstadat/xstadat_eves/i_wnh001.html`)

mesterséges neurális hálók segítségével orvosi képalkotó berendezések által létrehozott adatok pontosabban feldolgozhatók, csökkenthető a hibásan egészségesnek (false negative) és a hibásan betegnek (false positive) diagnosztizált találatok aránya.

A szakorvosi munkát nagyban támogathatja az automatizált eljárás, hiszen sokkal kevesebb időt kellene diagnosztikával tölteniük, kapacitásuk nagy részét az előzetesen pontosan diagnosztizált, valóban kezelésre szoruló betegek kezelésére fordíthatják. [3]

A nagyfokú fejlődés tehát nem fogja a szakorvosok tömeges elbocsátását okozni, hiszen az összetettebb kórképek elemzéséhez elengedhetetlen az emberi szakértelem. A gépek nagyban segíthetik többek közt az egészségügyi dolgozók munkáját, de elvenni nem fogják azt, hiszen minél több betegről állapítják meg időben a diagnózist, annál hamarabb kerülhet az orvosok látóterébe.

A fejlesztések legnagyobb nehézségét a neurális hálók tanítására alkalmas címkézett adatok előállításának költségessége jelenti. Hiába rendelkezünk nagy mennyiségű, könnyen elérhető adattal, ha a tanító halmaz előállításához szükséges nagyfokú szakértelem és tapasztalat nehezen, vagy csak nagyon költségesen érhető el. A hasonló projektek költségvetésének jelentős részét emészti fel a szakképzett radiológusok

óradíja, vagy a már létező címkézett adatok ára. A jövőbeli feladatok közé tartozik a meglévő módszerek javítása, a kevesebb adattal való tanítás kidolgozása, és az értelmezhetőség biztosítása.

A 2. fejezetben röviden bemutatom a mesterséges neurális hálók felépítését, illetve a mély konvolúciós neurális hálókat, hiszen ezek az algoritmusok kiemelkedően jól teljesítenek a képfeldolgozás területén. Ezután a 3. fejezetben a két feladattípus matematikai modelléjen bemutatása következik, a képklasszifikáció és illetve a képszegmentáció. A választott adatban nagyon kicsi a keresett csomók mérete, ezért a klasszifikációt meg kell előznie a szegmentációnak. A 4. fejezetben bemutatom a választott feladatot és annak részeit. Felvázolok egy lehetséges megoldási módszert, majd a kiválasztott részfeladatot közelebbről is megvizsgálom. A fejezet végén a kísérletek során kapott eredményeket elemzem. Az 5. fejezetben kerül sor a tapasztalatok és eredmények összegzésére, a konklúzió levonására.

2. fejezet

Mesterséges neuronhálók

A mindennapi élet során egyre nagyobb mennyiségű, változatos formátumú adatot állítunk elő. Lehetnek ezek ipari, pénzügyi, egészségügyi adatok, amelyeket nem kizárólag emberek, hanem gépek is generálnak. Az ilyen mennyiségű adat emberi ésszel szinte felfoghatatlan és átláthatatlan.

A gépi tanulás (machine learning) fogalma olyan rendszereket foglal magába, melyekben a gépek emberi segítséggel (vagy anélkül) képesek példa adatok alapján mintázatokat, szabályszerűségeket észlelni, és ezen általánosításokat alkalmazni új, még ismeretlen és feltáratlan adatokra. Ez a törekvés arra irányul, gyorsabban és hatékonyabban készíthessünk modelleket, amik segíthetik, esetenként helyettesíthetik is az emberi szakértelmet.

A gépi tanulás az orvosi diagnosztika területén is számos eljárásban nyújthat segítséget, például az elkészített CT felvételeket gépi algoritmus elemezheti, átvéve a szakorvosoktól az egyszerűbb lépéseket, és előkészíthet majd egy diagnózist, amit a szakorvos ellenjegyezhet vagy további vizsgálatokat végezhet.

Számos való életbeli probléma modellezhető függvényként, a módszerek kidolgozása során gyakran ezeket a leképezéseket próbáljuk a lehető legjobban közelíteni gépi tanulással. A deep learning módszercsalád elég gazdag ahhoz, hogy sokféle feladatra találhassunk megoldást, emellett megfelelően specifikálható az adott problémára. Dolgozatomban a rengeteg problémakör közül a képklasszifikációs feladatokat választottam, a továbbiakban a mélytanulási módszereket ezen feladatokra vetítve vizsgálom.

A mély előrecsatolt hálók (deep feedforward networks) számos képfeldolgozási feladatban nagy áttörést értek el. A modelleket azért nevezzük előrecsatoltnak, mert nincsenek visszacsatolások, amin keresztül a kimenetek vissza lennének vezetve a modellbe. A következőkben a Stanford [8] kurzusa alapján lépésről-lépésre áttekintem az előrecsatolt konvolúciós mesterséges neurális hálók felépítését.

2.1. A neurális hálók felépítése

2.1.1. Felügyelt tanítás

A felügyelt tanítás (supervised learning) esetében szükséges, hogy a tanító halmazon rendelkezzünk a bemenetekkel és a hozzájuk tartozó valódi kimenetekkel, tehát az (x, y) formában előálló párokra, ahol x a bemenő adat (input), y pedig az elvárt kimenet (output). A célunk az, hogy az input-output párokkal megadott függvényt jól közelítsük úgy, hogy eközben a modell általánosítóképessége a lehető legjobb legyen. Ez utóbbi azért fontos, mert abban az esetben, ha a modell túlságosan „rátanul" a már ismert adatokra, akkor az új, még ismeretlen bemenő adatok esetén értékelhetetlen eredményeket kapunk.

Az általánosítóképesség vizsgálatához a meglévő címkézett adatokat három csoportra bontjuk, ezek a tanító, validációs és teszt halmazok. A tanító halmazt értelemszerűen tanításra használjuk, a modell ezek alapján építjük fel. A validációs halmaz használata elhagyható, de ha az adatok mennyisége megengedi, akkor ajánlott elkülöníteni, hiszen segítségével hangolhatjuk a modellt, így minimalizálva a lehetőséget a túltanulásra. A teszthalmazon értékeljük ki a modellünk teljesítményét, ezért ideális esetben ezeket az adatokat egyetlenegyszer használjuk bemenetként, a tanítási folyamat legvégén.

2.1.2. Mesterséges neuronok

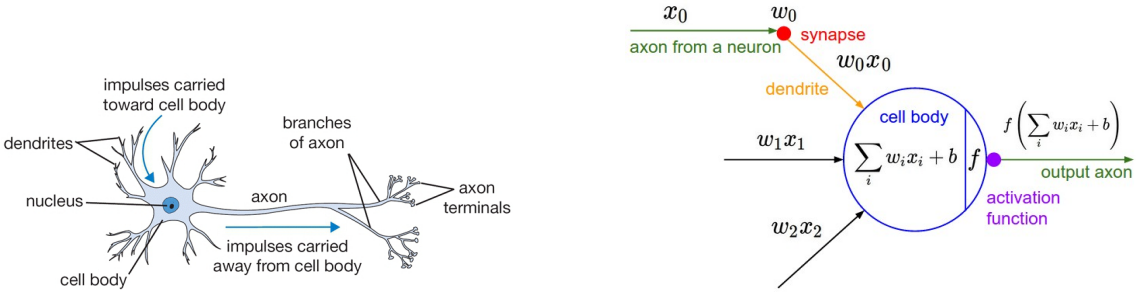
2.1.1. Definíció. *Mesterséges neuron:*

$$\sigma\left(\sum_{i=1}^{n} w_i x_i + b)\right),$$

ahol $x_i \in \mathbb{R}$ egy bemeneti érték, $w_i \in \mathbb{R}$ az ehhez tartozó súly $i \in [1, .., n]$, $b \in \mathbb{R}$ az eltolás (bias), σ pedig egy adott $\mathbb{R} \to \mathbb{R}$ aktivációs függvény.

A neuron paraméterei tehát w_i és b.

A mesterséges neurális hálók építőkövei a mesterséges neuronok. A mesterséges neuronokra tekinthetünk az emberi agy neuronjainak analógiájaként. Ez a szemlélet nem egyértelmű megfeleltetést jelent, hanem csupán hasonlóságot takar az ingertovábbítás terén. Ennek az az oka, hogy az emberi agy bonyolultsága nem teszi lehetővé azt, hogy pontos mását alkossuk meg mesterségesen. A jelenlegi tapasztalatok alapján erre nem is lesz szükség, hiszen a mesterséges neurális hálók jelenlegi formájukban is hatékonyan használhatók bizonyos feladatok megoldására. A mesterséges neuronokban a bemeneti értékek súlyozott összegét átvezetik egy aktivációs



(a) (b)

2.1. ábra. Egy biolgiai neuron egyszerűsített rajza (a) és a mesterséges neuron matematikai modellje (b) (`http://cs231n.github.io/neural-networks-1/`).

függvényen, és a kapott eredménytől függően továbbítják az „ingert”.

2.1.3. Az aktivációs függvény

A leggyakrabban használt aktivációs függvények:

2.1.2. Definíció. *Szigmoid függvény:*
$sigmoid(x) = \dfrac{1}{1+e^{-x}}$, *ahol* $\sigma\colon \mathbb{R} \to \mathbb{R}$, x *pedig a bemenet.*

2.1.3. Definíció. *ReLU (rectified linear unit):*
$ReLU(x) = \max(0, x)$, *ahol* $ReLU\colon \mathbb{R} \to \mathbb{R}$, x *pedig a bemenet.*

2.1.4. Definíció. *Tanh függvény:*
$\tanh(x) = \dfrac{1-e^{-2x}}{1+e^{-2x}}$, *ahol* $\tanh\colon \mathbb{R} \to \mathbb{R}$, x *pedig a bemenet.*

2.1.4. A mesterséges neuronok rétegekbe szervezése

A mesterséges neuronokat rétegekbe szervezhetjük úgy, hogy egy bemenethez több neuront csatlakoztatunk, így megvalósítható egy $\mathbb{R}^n \to \mathbb{R}^m$ leképezés, ahol n a bemeneti vektor hossza, m pedig a neuronok száma.

2.1.5. Definíció. *Neurális réteg:*

$$\sigma(A\underline{x} + \underline{b})$$

ahol $\underline{x} \in \mathbb{R}^n$ *a bemeneti vektor,* $A \in \mathbb{R}^{m \times n}$ *a súlymátrix,* $\underline{b} \in \mathbb{R}^m$ *az eltolásvektor,* σ *pedig az adott aktivációs függvény.*

Ezután a különböző rétegeket úgy helyezzük el, hogy az egyik réteg kimenete lesz a következő réteg bemenete, és így előáll az előrecsatolt mélytanuló neurális háló.

Az előrecsatolt mesterséges neurális hálóra tekinthetünk parametrizált függvénykompozícióként is, a következőképpen:

2.1.6. Definíció. *Előrecsatolt neurális hálózat parametrizált függvénykompozícióként:*

$$f_\Theta(\underline{x}) = f_L \circ \cdots \circ f_3 \circ f_2 \circ f_1(\underline{x})$$

ahol $\Theta = (\theta_1, \theta_2, \ldots, \theta_K)$ *a neurális háló paraméterei,* $f_1, f_2, \ldots, f_L$ *pedig a rétegek leképezései.*

Az aktivációs függvény tipikusan nemlineáris, hiszen lineáris aktivációs függvény esetén a háló egy lineáris függvénnyé fajul.

2.1.5. Veszteségfüggvény

Az általánosítóképesség méréséhez meg kell határoznunk egy függvényt, amit számszerűen megadja, hogy a modellünk kimenete mennyire különbözik az előre megadott mintától. A különbségre tekinthetünk távolságként is, az ezt megadó függvényt pedig veszteségfüggvénynek nevezzük.

2.1.7. Definíció. *Veszteségfüggvény:*

$$L(f_\Theta(\underline{x}), \underline{y}) = \left\| f_\Theta(\underline{x}) - \underline{y} \right\|_2,$$

ahol $L : \mathbb{R}^p \times \mathbb{R}^q \to \mathbb{R}$, $f_\Theta(x)$ *a neurális háló kimenete,* $\underline{y}$ *pedig az elvárt kimenet,* p $f_\Theta(x)$ *hossza,* q *pedig* y *hossza.*

2.1.8. *Megjegyzés.* A függvénykompozíció differenciálhatósága miatt a veszteségfüggvény is differenciálható lesz.

A mesterséges neurális hálózatot a veszteségfüggvény minimalizálásával tanítjuk, tehát

$$\min_\Theta \frac{1}{N} \sum_{i=2}^{N} L(f_\Theta(\underline{x}), \underline{y}).$$

Így skalár értékben kapjuk meg, hogy a hálónk kimenete mennyire felel meg az elvártnak.

2.1.6. A hálózat tanítása

A hálózat tanítása a gradiens módszerrel történik. A hálózat rétegeit alkotó függvények differenciálhatók, tehát a hibafüggvény is differenciálható. Így megállapítható az adott függvény súlyai szerinti gradiense, ami megmutatja, hogy merre kell kis lépést tennünk a hálózat paraméterei mentén ahhoz, hogy a hiba csökkenjen. A lépéseket iterálva megkaphatjuk a függvény egy lokális minimumát.

2.1.9. Definíció. *Gradiens lépés*

$$W \leftarrow W - \lambda \frac{\delta L(x, y, W)}{\delta W}$$

ahol L a veszteségfüggvény,x a hálózat bemenete, y a hálózat kimenete, W a hálózat súlymátrixa, λ pedig a tanulási ráta.

A tanulási rátával (λ) állíthatjuk be azt, hogy az adott helyzetben mekkorát lépünk a negatív gradiens irányába.

2.1.10. *Megjegyzés.* A gyakorlatban ez a szám kezdetben nagyobb, majd a tanulási folyamat előrehaladtával egyre kisebb lesz.

A hálózat kimenete gyakran egy vektor, aminek hossza a lehetséges kategóriák száma. „One-hot encoding" esetén a kimenet alakja:

2.1.11. Definíció. *One-hot encoding:*

$$\underline{y} = [y_1, y_2, \ldots, y_i, \ldots, y_{n-1}, y_n],$$

ahol $\underline{y}$ a kimeneti értékek vektora, n a lehetséges kategóriák száma és $y_i = 1$, $y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_n = 0$.

2.1.12. Példa. *Ha a bemeneti kép egy kutyát ábrázol, és az y vektor címkéi rendre [macska, ház, kutya, repülő, hajó], akkor a megfelelő y vektor [0, 0, 1, 0, 0] lesz.*

2.2. A konvolúciós neurális hálók

A konvolúciós neurális hálók (Convolutional Neural Networks), ezentúl CNN, áttörést hoztak számos természetes adathalmazzal kapcsolatos feladatban, többek között a képi formátumú adatok körében [11]. A jó teljesítmény mögött az áll, hogy a CNN kihasználja az adat kétdimenziós tulajdonságát, illetve a képi adathalmazokban lévő eltolásinvarianciát, ezáltal számottevően csökkenthető a hálózat paramétereinek száma. A csökkentett paraméterszám lehetővé teszi olyan hálózatok létrejöttét, amik a képklasszifikáció területén jelenleg a legjobb megoldásokat nyújtják. A

CNN-ek, mint az előbbiekben részletezett neurális hálók, tanulható paraméterekkel (súlyok és eltolás) és veszteségfüggvénnyel rendelkeznek. Képklasszifikációs feladatok esetében a bemenetet a képek pixelei adják, a kimenet leggyakrabban a korábban említett „one-hot encoding” típusú vektor.

A sűrűn kapcsolt (dense) neurális hálók alkalmazása nem praktikus nagyobb méretű képek esetében. Enne az az oka, hogy a hatalmas paraméterszám könnyen túltanuláshoz vezethet.

Ezzel szemben a CNN kihasználja a képi adathalmaz strukturális tulajdonságait, amik segítségével a neuronokat háromdimenziós (szélesség, magasság, mélység) tömbként kezelheti. Ennek megvalósítását a konvolúciós réteg leírásában tárgyalom.

2.2.1. A konvolúciós neurális háló rétegei

Az egyszerű CNN előállításához általában a következő fajta rétegeket használjuk:

- konvolúciós réteg (CONV),
- pooling réteg (POOL),
- teljes, azaz fully-connected réteg (FC).

A neurális hálóban az i-edik réteg teljes, azaz **fully-connected**, ha úgy épül fel, hogy az i-1-edik réteg neuronjainak mindegyike össze van kötve az i-edik réteg minden neuronjával.

A **konvolúciós réteg** paraméterei tanulható filterekből épülnek fel. Szélességét és magasságát tekintve minden filter kis kiterjedésű, de mélységében mindig megegyezik a bemenetével.

Így az adott réteg neuronjai az előző réteg egésze helyett csak egy részével lesznek összekötve. Ezt a filtert végigvezetve a bemenet teljes magasságán és szélességén kapunk egy 2 dimenziós aktivációs térképet, ami az adott helyen mutatja a filter válaszát.

Tehát a konvolúciós réteg egy $\mathbb{R}^{W_1 \times H_1 \times D_1} \rightarrow \mathbb{R}^{W_2 \times H_2 \times D_2}$ leképezés, ahol a réteg bemenete:
$W_1 \times H_1 \times D_1$ (width = szélesség, height = magasság, depth = mélység);
a paraméterei:
K (kernel) a filterek száma, minden ponton ugyanazokat a súlyokat használva,
F a filterek térbeli kiterjedése,
S (stride) a lépésnagyság, ami megmutatja, hogy hány pixelt lépünk, amikor csúsztatunk,

P (zero-padding) a kép szélét kipárnázandó nullák mennyisége.
A kimenet ekkor $W_2 \times H_2 \times D_2$ lesz, ahol:
$W_2 = (W_1 - F + 2P)/S + 1$,
$H_2 = (H_1 - F + 2P)/S + 1$,
$D_2 = K$.

Adott pozícióban a filterekhez a következőképpen számolható a lineáris kombináció 3D-s esetben: $\sum x_{i,j,k} W_{l,m,n}$ Az ilyen filterek képesek megtanulni bizonyos térbeli struktúrákat, mint sarkok, szélek vagy a magasabb szintű rétegekben akár méhsejt jellegű minták, illetve a lokális korrelációkat.

A **pooling réteg** elhelyezésének célja általában a dimenziócsökkentés. 2D-ben pooling réteget megelőző réteg kimenetét egymást nem fedő téglalap alakú (legtöbbször 2×2-es) klaszterekre bontjuk. Az egyes klaszterek kimenetének kombinációi lesznek az egyes neuronok bemenetei. Leggyakrabban max pooling vagy average pooling réteget alkalmazunk.

2.2.1. Definíció. *Max pooling: A max pooling réteg neuronjainak bemenete az előző réteg klasztereinek maximuma.*

2.2.2. Definíció. *Average pooling: A max pooling réteg neuronjainak bemenete az előző réteg klasztereinek átlaga.*

A réteg bemenete $W_1 \times H_1 \times D_1$, a paraméterei: F a filterek térbeli kiterjedése, S a lépésnagyság (stride). A kimenet ekkor $W_2 \times H_2 \times D_2$ lesz, ahol:
$W_2 = (W_1 - F)/S + 1$,
$H_2 = (H_1 - F)/S + 1$,
$D_2 = D_1$.

2.2.3. Példa. *A VGG16 hálózat: A VGG16 hálózat felépítése a Keras dokumentáció (https://keras.io/applications) alapján a következőképpen áll elő.*

2.1. táblázat. A VGG16 hálózat architektúrája

Layer	**Function**	**Size**	**Activation**	**Input**
Inputs	Input	(1,512,512)		
Conv1	Convolution2D	(64,3,3)	ReLU	Input
Conv1	Convolution2D	(64,3,3)	ReLU	Conv1
Pool1	MaxPooling2D	(2,2)		Conv1
Conv2	Convolution2D	(128,3,3)	ReLU	Pool1

Conv2	Convolution2D	(128,3,3)	ReLU	Conv2
Pool2	MaxPooling2D	(2,2)		Conv2
Conv3	Convolution2D	(256,3,3)	ReLU	Pool2
Conv3	Convolution2D	(256,3,3)	ReLU	Conv3
Conv3	Convolution2D	(256,3,3)	ReLU	Conv3
Pool3	MaxPooling2D	(2,2)		Conv3
Conv4	Convolution2D	(512,3,3)	ReLU	Pool3
Conv4	Convolution2D	(512,3,3)	ReLU	Conv4
Conv4	Convolution2D	(512,3,3)	ReLU	Conv4
Pool4	MaxPooling2D	(2,2)		Conv4
Conv5	Convolution2D	(1024,3,3)	ReLU	Pool4
Conv5	Convolution2D	(1024,3,3)	ReLU	Conv5
Conv5	Convolution2D	(1024,3,3)	ReLU	Conv5
Pool5	MaxPooling2D	(2,2)		Conv5
Include_top = True (classification block)				
Flatten	Flatten		ReLU	Pool5
Dense1	Dense	4096	ReLU	Flatten
Dense1	Dense	4096	ReLU	Dense1
Dense2	Dense	classes	Softmax	Dense1
Include_top = False; pooling = 'avg'				
Pool6	GlobalAveragePooling2D			Pool5
Include_top = False; pooling = 'max'				
Pool6	GlobalMaxPooling2D			Pool5

2.2.2. Regularizációs eszközök

A regularizáció magában foglal minden olyan módszert, amely segít az általánosításban és amivel igyekszünk megakadályozni a modell túltanulását.

A CNN-ek általában tízmilliós nagyságrendű paraméterből és 10-20 rétegből állnak. A rétegek számának meghatározásakor figyelembe kell venni azokat a tényeket, hogy minél több neuron használunk, annál nagyobb lesz a teljesítmény, ám egy bizonyos szint felett a modell hajlamos lesz a túltanulásra. Ennek a problémának a kiküszöbölésére nem kell feltétlenül kicsi hálókat használnunk, egyéb regularizációs technikákkal a túltanulás veszélye minimalizálható, miközben megtartható a háló mérete.

2.2.4. Példa (L2 regularizáció). *A leggyakoribb regularizációs forma. A lényege az, hogy minden paramétert négyzetesen „büntetünk", tehát minden w súly helyett $\frac{1}{2}\lambda w^2$-t veszünk. Ezzel az egyenetlen súlyvektorokat jobban, míg a diffúz vektorokat kevésbé büntetjük. Ezzel arra ösztönözzük a hálót, hogy az összes bemenetet használja egy kicsit, ahelyett, hogy csupán néhány értéket használna kiemelkedően.*

2.2.5. Példa (L1 regularizáció). *Az L1 regularizáció nagyon hasonló az L2-höz, azzal a különbséggel, hogy w súly helyett ebben az esetben $\lambda|w|$-t használjuk.*

2.2.3. Veszteségfüggvény képklasszifikációs feladatokhoz

A veszteségfüggvény a konvolúciós neurális hálók esetében is azt méri, hogy a predikció és az ismert valós kimenet mennyire egyezik. Az összes veszteség az egyedi veszteségek átlagaként számolható ki, tehát $L = \frac{1}{N}\sum_i L_i$, ahol N a tanító adatok száma. Legyen $f = f(x_i; W)$ a kimeneti réteg aktivációja. A képklasszifikációs feladatoknál az egyik leggyakrabban használt veszteségfüggvény az SVM (Support Vector Machine):

$$L_i = \sum_{j \neq y_i} \max(0, f_j - f_{y_i} + 1),$$

amit használhatunk squared hinge loss-ként ($\max(0, f_j - f_{y_i} + 1)^2$).

A másik gyakran használt függvény a softmax klasszifikátor, ami cross-entropy-t használ:

2.2.6. Definíció. *Softmax:*

$$p_j = \frac{e^{a_i}}{\sum_{k=1}^{N} e_k^a},$$

ahol a_i a i-edik logit aktivációját adja meg az $x \in \Omega$, $\Omega \subset \mathbb{Z}^2$ helyen. N az osztályok darabszáma, p_j pedig a közelített maximum-függvény.

Tehát a maximum aktivációjú j-re: $p_j \approx 1$. Az összes többi j-re: $p_j \approx 0$.

Ekkor a cross-entropy az mutatja, hogy mekkora a különbség a modell által generált kimenet és a valóság között.

2.2.7. Definíció. *Cross-entropy:*

$$H(y, p) = -\sum_i y_i \log(p_i),$$

ahol y egy becslése a $p(x)$-nek, ami x valós eloszlása.

A teljes veszteséget az L_i-k átlaga adja meg, amit a tanító adathalmazra számolunk ki. Az előbbi peremértékeket (margin scores) ad meg, utóbbi viszont valószínűségeket, tehát hogy az adott bemenet milyen valószínűséggel esik az egyes kategóriákba. Ebből következik, hogy a különböző kategóriákra adott számok 1-re összegződnek. Emellett gyakran (pl. ImageNet esetén) figyeljük meg a konvolúciós neurális háló rank-5 pontosságát, ami azt jelenti, hogy ellenőrizzük, hogy az elvárt kimenet szerepel-e az első 5 predikció között. Ehhez az eljáráshoz nagyon jól illeszkedik a valószínűségek számítása.

2.2.8. Példa. *A cross-entropy deriváltja:*

$$
\begin{aligned}
L &= -\sum_i y_i \log(p_i) \\
\frac{\partial L}{\partial o_i} &= -\sum_k y_k \frac{\partial \log(p_k)}{\partial o_i} \\
&= -\sum_k y_k \frac{\partial \log(p_k)}{\partial p_k} \frac{\partial p_k}{\partial o_i} \\
&= -\sum y_k \frac{1}{p_k} \frac{\partial p_k}{\partial o_i}
\end{aligned}
$$

A softmax *függvény deriváltjából:*

$$
\begin{aligned}
\frac{\partial L}{\partial o_i} &= -y_i(1-p_i) - \sum_{k \neq i} y_k \frac{1}{p_k}(-p_k p_i) \\
&= -y_i(1-p_i) + \sum_{k \neq 1} y_k p_i \\
&= -y_i + y_i p_i + \sum_{k \neq 1} y_k p_i \\
&= p_i \left(y_i + \sum_{k \neq 1} y_k \right) - y_i \\
&= p_i \left(y_i + \sum_{k \neq 1} y_k \right) - y_i
\end{aligned}
$$

Mivel y egy one hot encode-olt vektor, ezért $\sum_k y_k = 1$ és $y_i + \sum_{k \neq 1} y_k = 1$. Ebből:

$$
\frac{\partial L}{\partial o_i} = p_i - y_i.
$$

3. fejezet

A CT orvosi képalkotó eljárás feladatai

3.1. A CT-ről röviden

A gépi tanuló algoritmusok alkalmazása során nagyon fontos az, hogy pontosan ismerjük a bemenő és kimenő adatokat, hiszen ezen ismeret nélkül könnyen félreértelmezhetjük a kapott eredményt, vagy olyan eredményt kaphatunk vissza, ami a szakértők számára triviális.

Az orvosi képalkotó eljárások megértéséhez Az orvosi képalkotás fizikája [16] című könyvet használtam.

A rengeteg orvosi képalkotó eljárás (röntgen, ultrahang, lézer, CT, MR stb.) közül a rák diagnosztikájában a CT (Computed Tomography) bizonyult a leghatékonyabbnak. Ez az első olyan eljárás, amely lehetővé teszi az emberi test „szeletről-szeletre" történő vizsgálatát, műtéti beavatkozás nélkül. Kifejlesztéséért Godfreyv Hounsfield és Allan Cormac 1979-ben orvosi Nobel-díjat kaptak.

A CT eljárások során is használt röntgensugarakat 1895-ben Wilhelm Conrad Röntgen fedezte fel, a matematikai alapelveit pedig Johann Radon, osztrák matematikus dolgozta ki, 1917-ben. Az első CT felvétel 1971-ben készült, amely minőségét tekintve messze elmarad mai utódaitól, de így is hatalmas áttörést jelentett. A képek felbontása 80×80 pixeles (3 mm-es pixelméret mellett), a mérési idő szeletenként 4,5 perc, míg rekonstrukciós idő 1,5 perc volt, így egyetlen képkocka rögzítése több mint 5 percet igényelt. A mai gépek 1024×1024 pixeles felbontással dolgoznak, és egy szelet képe akár 0,3 másodperc alatt elkészül.

A dolgozat témájának szempontjából az eljárás kiemelkedően fontos része a digitális képi megjelenítés. A CT-képek általában 12 bites színmélységet használnak, tehát a CT-szám (Hounsfield Unit, HU) teljes tartománya $2^{12} = 4096$ különböző ér-

Levegő	-1000
Zsír	-120-tól -90-ig
Víz	0
Vér	+13-tól +50-ig
Tüdő sejtjei	-700-tól -600-ig
Koponya és arc csontjai	+200
Szivacsos szerkezetű csontok	+700
Tömött csontállomány	+3000

3.1. táblázat. A Hounsfield-skála néhány lényeges értéke.

ték lehet (−1000-től +3085-ig). A számítástechnika fejlődésének köszönhetően ennek a színmélységnek a feldolgozása már nem jelent problémát.

Az emberi szem kontrasztfelbontó képessége csupán 30-90 szürkeségi árnyalat, ami függ a pupilla átérőjétől, fényviszonyoktól és további tényezőktől. Ha a CT-felvételeken megjelenő nagyjából 4000-es szürkeségi fokozatot lefednénk az emberi szem által is megkülönböztethető 90 fokozattal, akkor rengeteg információ veszne el, például különböző szövetek vagy azonos szövet különböző struktúrái ugyanolyan árnyalattal lennének jelölve. Ezen probléma megoldására használják az úgynevezett ablakolást. Az ablak szélessége (W = width) határozza meg a kép kontrasztját úgy, hogy a szűkebb ablak nagyobb kontrasztot jelent. Az ablak helyzetét (szöveti tartományát) a centrum (L = level) adja meg.

$$P_1 = L - \frac{1}{2}W$$

$$P_2 = L + \frac{1}{2}W$$

P_1 és P_2 azokat a töréspontokat adja meg, amelyeken kívüli tartományokat homogén fekete illetve fehér színnel jelenítik meg. Ez az eljárás nyilvánvalóan adatvesztéssel jár, de így a lényeges területeken kiemelhető a kontraszt.

A felvételek 2D-s képekként készülnek el, de ahhoz, hogy az orvos megfelelően értékelhesse, szükséges a képek 3D-s megjelenítése.

3.2. Képklasszifikáció

3.2.1. A képklasszifikáció feladata

A képklasszifikáció feladata az, hogy egy adott képhez hozzárendeljen egy kategóriát (vagy az adott kategória valószínűségét) az előre meghatározott címkék közül.

3.2.1. Definíció. *Klasszifikációs feladat: A klasszifikációs feladatra tekinthetünk úgy, mint $f : x \to y$ függvény megtanulására, ahol x a bemeneti attribútumok halmaza (magyarázó változók), $y = \{y_1, y_2, \ldots, y_t\}$ pedig a kimeneti diszkrét osztálycímkék halmaza (célváltozó).*

Figyelembe véve a képek tulajdonságait, a gépi látásnak a következő kihívásoknak kell megfelelnie:

- **Nézőpont variancia:** egy adott objektumról végtelen számú állásból készíthetünk felvételt, és így ugyanazon objektum képei nagyon eltérhetnek egymástól.
- **Skála variancia:** ugyanazon objektum előfordulhat különböző méretekben a természetben. A modellnek képesnek kell lennie elvonatkoztatni a méretbeli különbségektől.
- **Deformáció:** a megfigyelhető objektumok nagyon része nem szilárd, többféle formában is megjelenhet a felvételeken.
- **Elnyelés:** elfordulhat, hogy a keresett objektumnak csupán egy kis része jelenik meg a felvételen, nem az egésze.
- **Megvilágítási kondíciók:** a kép pixeleit nagyban befolyásolhatják a felvétel készülésekor uralkodó fényviszonyok.
- **Beleolvadás a háttérbe:** sok esetben a háttér hasonló színű lehet, mint a keresett objektum, ezzel nehezítve a felismerést.
- **Osztályon belüli variancia:** egy adott osztályba tartozó objektumok nagyon különbözőek lehetnek, ezért az általánosítás nehézzé válik.

A megfelelő képklasszifikációs algoritmusnak tudnia kell kezelni ezen problémák akár együttes fennállását is, megtartva a jó általánosítás képességét. A legjobban működő modellek a gépi tanulás segítségével jönnek létre. A folyamatot a következőképpen formalizálhatjuk:

- **Bemenet:** N darab képet tartalmaz, mindegyikhez egy címkét rendelve a K darabos listából. Ez az adathalmaz a tanítóhalmaz.
- **Tanulás:** ebben a fázisban kell megtanulni minden egyes címkéről, hogy hogyan néz ki. Ekkor tanítjuk a modellt.
- **Kiértékelés:** a folyamat végén szükség van arra, hogy meghatározzuk, hogy mennyire jól működik az adott modell. Ennek érdekében olyan adatokat kap bemenetként, amit még sosem látott, és ezekre kell megadnia a címkéket. Ezután a modell által adott predikciót összehasonlítjuk az ismert valósággal. Ehhez előre meghatározott távolságmérést alkalmazunk, általában $L1$-et vagy $L2$-t. Az az elvárásunk, hogy a távolság minél kisebb legyen.

A klasszifikációs modellt az alábbi tényezők alapos vizsgálata után választjuk meg:

- **Az előrejelzés teljesítménye:** a modell „jósága" azt takarja, hogy mennyire jól becsüli meg a modell a célváltozót a rendelkezésre álló magyarázó változók alapján.
- **Robosztusság:** mennyire jól általánosít a modell, mennyi érzékeny az átlagtól jelentősen eltérő adatokra (outlierek).
- **Értelmezhetőség:** megismerhető-e a modell belső működése, a döntések okai feltérképezhetőek-e.
- **Skála-invariancia:** jelen esetben azt jelenti, hogy a képeket felnagyítva vagy lekicsinyítve is felismeri a modell a rákos betegek felvételeit.

A konkrét adathalmazon tehát olyan modellt építünk, amely a CT-felvételek egyes attribútumait és a rákos megbetegedések jelenlétét hozza összefüggésbe. A CT-felvételek esetében x a bemeneti képek halmaza lesz, $y \in \{0, 1\}$, ahol 0 érték esetén a vizsgált személy egészséges, 1 érték esetében pedig rákos. A feladat megvalósításakor a tanító halmazon ismertek a kimeneti értékek. A modell építésekor feltérképezzük, hogy az osztálycímkék hogyan függenek a többi változótól. A megalkotott modellt ezután használhatjuk új adatokra.

3.2.2. Képklasszifikációs architektúrák

A képklasszifikációs architektúrák feltérképezéséhez a „A Survey on Deep Learning in Medical Image Analysis” [13] című cikket használtam.

Az első konvolúciós neurális hálót használó klasszifikációs architektúra az 1998-as LeNet [12]. A 2012-ben publikált AlexNet [11] szintén fontos előrelépést jelentett. Mindkét modell sekély, mindössze 2 és 5 konvolúciós réteget használnak, a kernelek receptív területe bemenethez közeli rétegeken nagy, míg a kimenethez közeli rétegeken egyre kisebb. Az AlexNet már ReLU-t használ aktivációs függvényként.

2012 után egye több új architektúra került napvilágra, amelyek jellemzően mély modellek voltak. Ennek oka az, hogy a kisebb kernelek halmozásával kevesebb paraméterrel reprezentálható ugyan az a függvény, amit egyetlen réteggel és nagy receptív területtel adhatnánk meg.

Ilyen architektúrák a következők:

- VGG19 vagy OxfordNet: ImageNet Challenge 2014 győztese, 19 réteg [19],
- GoogLeNet vagy Inception: 22 réteg [20],
- ResNet: ImageNet Challenge 2015 győztese [7].

Jelenleg az orvosi képfeldolgozás területén a GoogleNet Inception v3 nevű verziója a legnépszerűbb [4, 6, 14].

3.2.3. Képklasszifikáció az orvosi diagnosztikában

Az orvosi diagnosztikában a képklasszifikáció bemenete a páciensről (szervről) készített felvétel, a kimenete pedig egy diagnózis, például: a betegség jelen van vagy nincs jelen.

Gyakori megoldás az, hogy egy természetes képeken előre tanított hálózatot használnak a jellemzők kiválasztására vagy a hálózat finomhangolására orvosi adatokon. A két eljárás között megoszlanak a preferenciák, és a kutatási eredmények [2, 10] sem döntik el egyértelműen, hogy melyik felhasználás a hatékonyabb. A döntésben ugyanakkor segíthet, hogy az Inception v3 finomhangolásával már megközelítették [4, 6] az emberi teljesítményt.

3.3. Képszegmentáció

A képszegmentációs feladat célja az, hogy a bemeneti teljes képet felosszuk olyan pixelhalmazokra, melyeket később címkézhetünk és klasszifikálhatunk. A szegmentációs feladat megközelíthető a gráfok irányából is [18].

Legyen $G = (V; E)$ egy tetszőleges, a tulajdonság változók terében választott súlyozott, irányítatlan gráf. A gráf csúcsai a tulajdonság változók terének pontjai, és páronként minden csúcs között legyen él. Az élek súlyai, $\omega(i, j)$, az i és j csúcsok közötti hasonlóság függvényei.

A csoportképzésnél, azaz szegmentációnál arra törekszünk, hogy a csúcspontok halmazát $V_1, V_2, \cdots, V_m$ diszjunkt halmazokra bontsuk fel, ahol a V_i beli csúcspontok közötti hasonlóság nagy, viszont V_i-ből és V_j-ből választva egy-egy tetszőleges csúcsot mindig kicsi hasonlóságot kapunk.

Ezen felül a szemantikus (jelentéstani) szegmentáció igyekszik megérteni minden egyes pixel szerepét a képben, eldöntve, hogy például kutyát, macskát vagy valamilyen egyéb osztályt lát éppen. Akár jelölhetjük a különböző kutyafajtákat különböző színekkel a képen.

Nagy előrelépést jelentett a 2015-ben kifejlesztett DeepMask (FAIR). A DeepMask kezdeti szegmentiációként nyers maszkokat képez az objektumokra. 2016-ban bemutatták a SharpMask-et, ami finomítja a DeepMask által biztosított maszkokat, így kevesebb részletet veszítve és javítva a szegmentációt. Ezután a MultiPathNet végzi el a maszkon belül körülhatárolt objektumok azonosítását.

Az eljárás használatának egyik célja az, hogy előfeldolgozást hajtsunk végre a bemeneten, például szegmentálva az objektumok hátterét, így a következő algoritmusoknak már csak a lényeges képrészletekkel kell törődniük. Az eljárás másik felhasználási módja a bemeneti kép megváltoztatása úgy, hogy az eredmény jobban értelmezhető, illetve könnyebben kezelhető legyen. A CT-felvételek esetében az utóbbi felhasználási mód vezethet inkább eredményre olyan formában, hogy a felvétel alapján rákos elváltozást mutató pixelekre maszkot használnánk.

Már az orvosi diagnosztikában is felismerték a képszegmentációban rejlő lehetőségeket, hiszen ezzel az eljárással a gépek kiemelhetik az adott vizsgálat szempontjából kritikus területeket, így a szakorvosoknak már csak ezekre a pontokra kell koncentrálniuk. Ezzel jelentősen rövidülhet a diagnózis felállításához szükséges idő hossza, illetve csökkenthető a hibák mennyisége.

A következőkben bemutatom a régió-alapú és az éldetektáló szegmentációt [21].

3.3.1. Régió-alapú szegmentáció

A legegyszerűbb és leggyakrabban használt terület-alapú szegmentációs algoritmus a küszöbérték szerinti szegmentáció. Ebben az aljárásban a pixeleket a célként kitűzött objektumok szürke árnyalata szerint szegmentáljuk. Ha globális küszöbértéket használunk, akkor a képet két részre osztjuk: egy célterület és egy háttérterület lesz az eredmény. Ha helyi küszöbértéket használunk, akkor meg kell határoznunk több lokális küszöbértéket, ennek megfelelően pedig az eredmény több részre osztott célterület és háttérterület lesz.

A területnövelő algoritmus mögött az az ötlet rejlik, hogy az egymáshoz közeli, azonos tulajdonsággal rendelkező pixeleket rendezzük egy régióba. Ehhez szükséges egy „magot” kijelölni, amely pixel a körülötte lévő, azonos tulajdonságú pixelekkel alkot egy régiót. A magpont kijelölése után a környező pixeleket akkor veszi bele a régióba, ha a szürke árnyalatuk különbségének abszolút értéke nem halad meg egy előre rögzített T küszöbértéket. Az eljárás előnye, hogy nagyon hatékonyan elkülöníti az összetartozó régiókat, hátránya az, hogy nagyon nagy a számítási igénye, és érzékeny a zajra.

3.3.2. Éldetektáló szegmentáció

A képeken az objektumok széle mindenképpen a lokális tulajdonságok folytonosságának hiányaként értelmezhető. Az objektumok szélénél megváltozik a szürkeárnyalat, a szín és a textúra is. Ha ezen tulajdonságok alapján sikerült meghatároznunk az objektumok széleit, azzal jelentősen megkönnyíthető a szegmentáció. Két szomszédos régió között mindig megfigyelhető egy szürke határvonal, de sokszor a szürke árnyalat nem folytonos. A folytonosságot vagy annak hiányát leggyakrabban deriváltakkal lehet detektálni, a deriváltakhoz pedig különböző operátorokat használhatunk.

4. fejezet

A választott feladat és a kísérletek

4.1. Az adathalmaz

Az adathalmaz forrása a Kaggle 2017-es Data Science Bowl versenye. A fő feladat a páciensekről rögzített képi adatok alapján eldönteni, hogy az illető beteg-e vagy nem. Az adatszett 2101 pácienstől tartalmaz címkézett felvételt, rögzítve azt, hogy a páciens beteg („cancer = 1”), vagy egészséges („cancer = 0”).

A verseny kiírása szerint csak az USA-ban évente 250 000 embert regisztrálnak tüdőrákkal, ami 12 milliárd dollár kiadást jelent az egészségügyi rendszernek. Nagyon fontos a korai stádiumú felismerés, hiszen nagyban hozzájárulhat a beteg túlélési esélyeinek javításához, illetve az intézmények kiadásainak csökkentéséhez.

Az adatok a páciensek tüdejéről CT-felvételek formájában elérhetőek, amit 2 dimenziós szeletek alkotnak. Ezeket a szeleteket egymásra helyezve kaphatjuk meg a 3 dimenziós képet a mellkasüregről. A páciensekhez tartozó felvételek változó számú (100-400 db) 512×512 pixeles szeletből állnak össze.

Az adathalmazból egyértelműen következik a képklasszifikáció feladata, aminek a nehézségét az okozza, hogy a CT-felvételeken (nagyjából $512 \times 512 \times 200 > 52$ millió pixel) a rákos csomók csak néhány pixel nagyságúak. Emiatt a klasszifikációs eljárás önmagában gyengén teljesít, hiszen a sikeres alkalmazásához minél jobb minőségű és minél gazdagabb szupervíziós jelre lenne szükségünk.

A minél pontosabb eredmények érdekében fontos, hogy képszegmentáció segítségével kiszűrjük a potenciálisan rákos csomókat (nodules), az így keletkezett maszkot annotálva felhasználhatjuk a későbbiekben bemenetként a képklasszifikációhoz. Így a CT-felvételek szegmentálásával támpontot adunk a következő feladat megoldásához.

A 2017-es Kaggle Data Bowl címkézett adatai nem tartalmaznak szegmentált képeket, ezért a szegmentációt elvégző U-Net tanítására a LUNA16 (Lung Nodule Analysis 2016) verseny adathalmazát használom. A LUNA16 célja az, hogy minél

4.1. ábra. Egy szelet a CT-felvételből.

hatékonyabb CAD (Computer-Aided Detection) rendszereket fedezzenek fel, amik automatikusan detektálják a rákos csomókat a tüdőben. A felhasznált adatok teljesen nyilvánosak, az LIDC/IDRI adatait négy radiológus szakorvos annotálta.

A LUNA16 adatszettben 888 páciens felvételei érhetők el. A Kaggle Data Bowl adataihoz hasonlóan ezek is változó szeletszámú 512×512 pixeles képekből állnak.

4.1.1. A DICOM formátum

A képek DICOM (Digital Imaging and Communications in Medicine) formátumban vannak tárolva, amely egy nemzetközi szabvány az egészségügyi adatok továbbítására, tárolására, visszakeresésére, nyomtatására, feldolgozására és megjelenítésére. A mindenkori aktuális kiadás online elérhető (`https://www.dicomstandard.org/`).

A széleskörű használatnak köszönhetően a páciensekhez tartozó felvételek könnyebben, gyorsabban visszakereshetők, illetve csökkenthető a leletek elcserélésének valószínűsége.

A formátum úgy lett kialakítva, hogy a különböző gépekkel készült fájlok könnyen kezelhetők, tárolhatók és megoszthatók. A DICOM fájlok tartalmazzák a páciensről szóló információkat, a képi adatokat, és az utóbbihoz kapcsolódó metaadatokat. Ilyen metaadat lehet például a pixelek való életbeli kiterjedése minden irányban, ami akár felvételenként is eltérő lehet.

A képek kezelése könnyen megoldható a Pydicom elnevezésű Python csomag segítségével. Ha a formátum tulajdonságait már az előfeldolgozás során megfelelően kezeljük, akkor jelentősen javíthatunk a CNN teljesítményén.

4.2. A képek előfeldolgozásának jelentősége

A páciensekről változó CT-beállítások mellett készültek képek, ezért a felvételek száma, paraméterei eltérők lehetnek. Mielőtt átadnánk az adatokat bemenetként a konvolúciós neurális hálónak, szükséges lesz az adatok előfeldolgozása. Az előfeldolgozás két feladat adódik: a tüdő szegmentálása, illetve a tüdőn belüli potenciálisan rákos csomók szegmentálása. Az alábbiakban mindkét eljárást bemutatom, a kísérletek során a csomók szegmentációjára került sor.

A LUNA16 adatszetten történő szegmentációt és a potenciálisan rákos csomók keresését a következő Kaggle Notebook alapján dolgoztam fel: Candidate Generation and LUNA16 preprocessing [1].

4.2.1. A tüdő szegmentációja

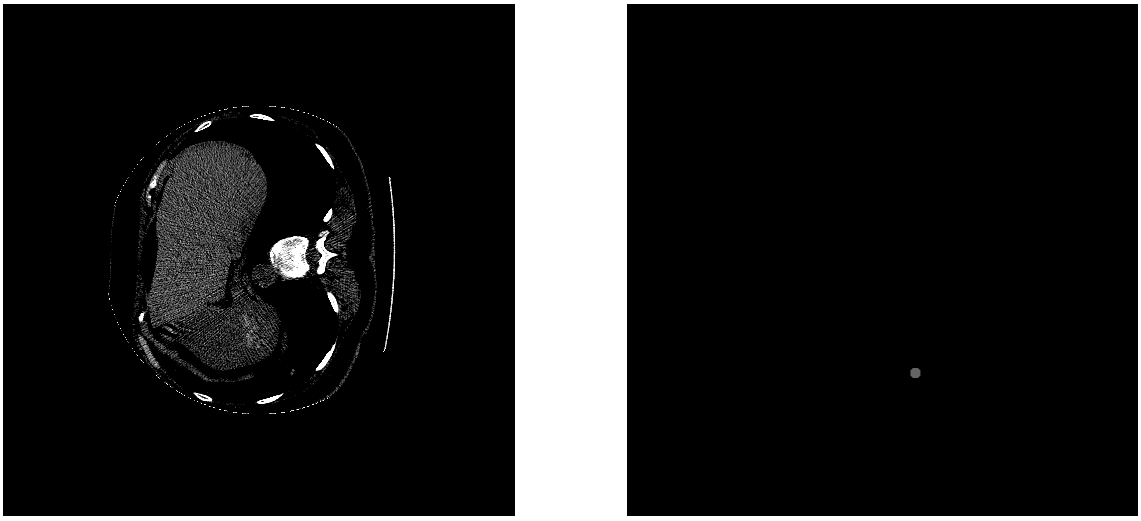
A felvétel minden páciens esetében n darab 512×512 pixeles képből áll össze, n annak a függvénye, hogy milyen felbontású a CT-felvételt készítő gép. Ez a képméret még túl nagy ahhoz, hogy CNN bemeneteként szolgáljon, ezért a HU-értékek alapján szegmentáció segítségével ki kell szűrni azokat a szövetrészeket, amik elhanyagolhatók, ilyenek lehetnek például a csontszövet, a levegő, és egyéb olyan adatok, amik zajossá teszik a felvételt.

Megfigyelhető, hogy a felvételeken megjelennek az előző fejezetben tárgyalt Hounsfield-skálán kívüli értékek is. Ennek az lehet az oka, hogy több CT szkennernek hengeres felvevőegysége van, míg a készített felvételek négyzet alakúak. A felvevőegységen kívül eső pixelek alapbeállításként a -2000-es értéket kapják, ezt érdemes átállítani 0-ra, tehát úgy kezeljük, mintha levegő lenne.

Minden 3D-s CT-felvétel sok 2D-s szeletből áll, és minden képen szerepel egy Instance Number, ami megmutatja, hogy az adott kép hányadik a sorban felülről nézve. Ezen index alapján a beolvasáskor könnyen sorbarendezhetők az egy páciens-hez tartozó felvételek, megalkotva a 3D-s képet.

A beolvasás után következő lépés a tüdő szövetének szegmentációja, hiszen az elváltozásokat a tüdőben keressük. Ez könnyen megoldható, ha a -400 HU alatti értékeket minden felételen eldobjuk. Így azok a területek maradnak meg, amik a tüdőhöz tartoznak.

[1] `https://www.kaggle.com/arnavkj95/candidate-generation-and-luna16-preprocessing/notebook`



(a) (b)

4.2. ábra. A tüdőről készült CT-felvétel (a) és a rákos csomó szegmentálva (b).

4.2.2. A lehetséges csomók generálása

A tüdő szegmentálása után meg kell találni a potenciális csomókat, amik daganatok lehetnek. A kiemelkedően fontos területek HU-értéke rendszerint nagyobb mint -400 HU, tehát ezzel a küszöbértékkel kiszűrhetők a sötétebb régiók. Ezután a meglévő pontokat klasszifikálhatjuk a False Positive-ok (FP-k) csökkentése érdekében. Ezt követően még maradhatnak a képeken zajos részek, amiket az erek okoznak. Ezt úgy lehet kiküszöbölni, hogy a két legnagyobb összefüggő területet eltávolítjuk.

Az 4.2 számú ábrán a teljes tüdőről készült szegmentálatlan CT-felvétel mellett látható a generált csomó.

A feladathoz tartozó kód megtalálható a Függelékben.

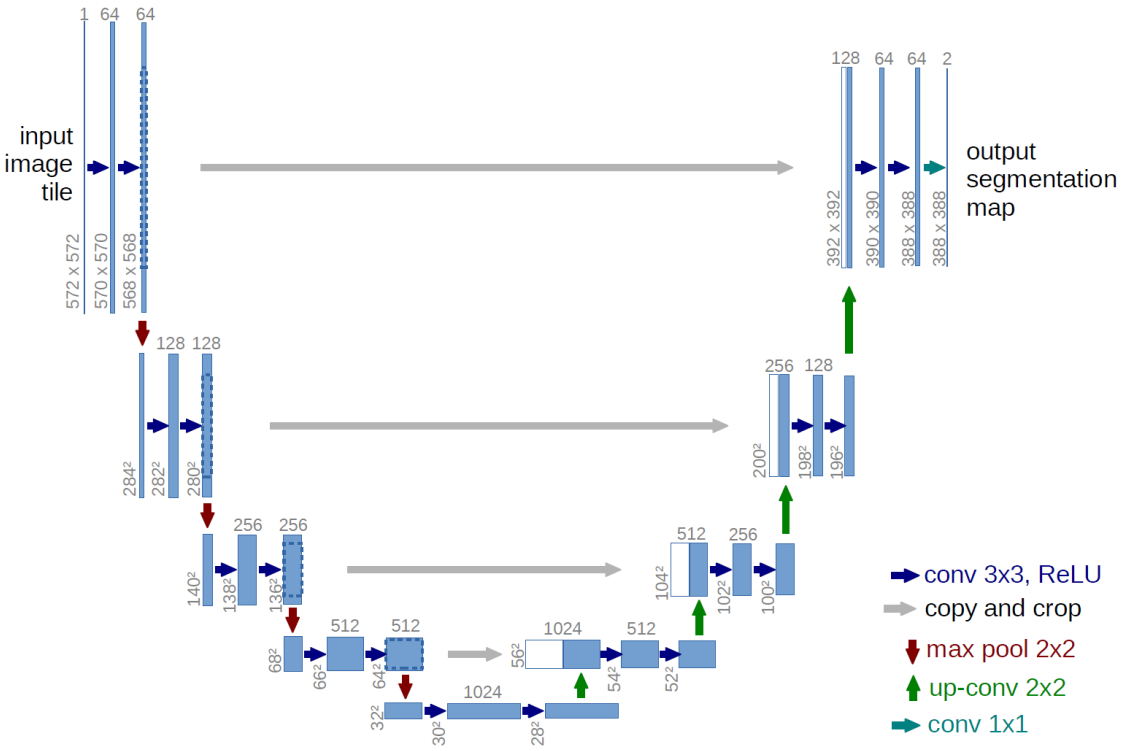
4.3. A felhasznált CNN

A modell teljes megvalósítása az a következő lépéseket foglalja magába: szegmentációval kiválasztjuk a potenciálisan rákos csomókat, és csupán ezeket a területeket adjuk át bemenetként a klasszifikátornak. Ezzel az eljárással hatékonyabban tanítható a CNN.

A megoldás során a szegmentációs feladatrészt valósítottam meg.

4.3.1. U-Net tanítása a LUNA16 adataival

Mivel a mélytanuló módszerek nagyon jó eredményeket érnek el az orvosi képalkotás szegmentációs feladatainak körében, ezért hasznos lehet az U-Net felhasználása



4.3. ábra. Az U-Net architektúrája

a potenciális daganatok megtalálásában. A hálózat tanításához az előre címkézett LUNA16 adatszettet használjuk.

Az U-Net feldolgozásához a „U-Net: Convolutional Networks for Biomedical Image Segmentation” [17] című cikket használtam fel.

Az U-Net segítségével történő szegmentáció előnye az, hogy jelentősen lecsökkenti a klasszifikációval vizsgálandó adat mennyiségét, így növelve az eredményességet. Ezt úgy érhetjük el, hogy olyan „dobozokat” emelünk ki a 3D-s felvételből, amik tartalmazzák a leginkább rák-gyanús csomókat.

A címkézett adat költségessége és gyakran elégtelen mennyiségű adat miatt, érdemes olyan tanító stratégiát választani, amely az augmentációra alapul az adat hatékony felhasználásának érdekében. Erre az eljárásra kiemelkedő példa a U-Net.

A felhasznált hálózat az U-Net LUNA16 adaton módosított változata.[17]

A LUNA16 adatszettben minden CT-felvételhez adottak a csomók középpontjai és a csomó átmérője, így egy bináris maszkot alkotva. A 3D-felvételek beolvasása után először szegmentálhatjuk a tüdőket, majd bináris maszkokat generálhatunk, végül elmenthetjük ezeket. Az előfeldolgozás után be kell tanítani a modellt a szegmentációra. Néhány irányelvet szem előtt kell tartani a tanítás során:

- Nem használunk olyan szeleteket, amiken nincs potenciális csomó, amire taníthatnánk.

- Az adat augmentációja nagyon fontos, hiszen a csomók általában gömb alakúak, de különböző az átmérőjük.
- Mivel nagyon kevés csomót tartalmazó terület van, így az adatszett nem kiegyensúlyozott. Ezért a veszteségfüggvényt megfelelően kell súlyozni.
- A túltanulás megakadályozása érdekében Dropout-ot használunk.

4.3.2. Szegmentáció az U-Net architektúrával

A hálózat architektúrája egy tömörítő (contracting) és egy kiterjesztő (expanding) részből áll össze, innen kapta a nevét is. A tömörítő rész két 3×3-as konvolúciót alkalmaz, mindegyik után egy ReLU-val és egy 2×2-es max pooling operátorral, 2-es lépéstávolsággal a mintavételezéshez. Minden mintavételező lépésnél megduplázzuk a filterszámot. A kiterjesztő szakaszban minden lépés egy filter upsample-lel kezdődik, majd 2×2-es konvolúció következik, ami megfelezi a filterek számát, összefűzve a tömörítő szakasz megfelelően körülvágott részével, majd kettő 3×3-as konvolúció, amelyek mindegyikét ReLU követi. Az utolsó réteg egy 1×1-es konvolúció, ami azért szükséges, hogy az összes 64 komponensű filter-vektort a kívánt számú osztályba képezze. Így összesen 25 konvolúciós rétege lesz a hálózatnak.

4.1. táblázat. A módosított U-Net architektúrája

Layer	**Function**	**Size**	**Activation**	**Input**
Inputs	Input	(1,512,512)		
Conv1	Convolution2D	(64,3,3)	ReLU	Input
Conv1	Dropout(0.2)			Conv1
Conv1	Convolution2D	(64,3,3)	ReLU	Conv1
Pool1	MaxPooling2D	(2,2)		Conv1
Conv2	Convolution2D	(128,3,3)	ReLU	Pool1
Conv2	Dropout(0.2)			Conv2
Conv2	Convolution2D	(128,3,3)	ReLU	Conv2
Pool2	MaxPooling2D	(2,2)		Conv2
Conv3	Convolution2D	(256,3,3)	ReLU	Pool2
Conv3	Dropout(0.2)			Conv3
Conv3	Convolution2D	(256,3,3)	ReLU	Conv3
Pool3	MaxPooling2D	(2,2)		Conv3
Conv4	Convolution2D	(512,3,3)	ReLU	Pool3
Conv4	Dropout(0.2)			Conv4

Conv4	Convolution2D	(512,3,3)	ReLU	Conv4
Pool4	MaxPooling2D	(2,2)		Conv4
Conv5	Convolution2D	(1024,3,3)	ReLU	Pool4
Conv5	Dropout(0.2)			Conv5
Conv5	Convolution2D	(1024,3,3)	ReLU	Conv5
Up6	Merge	(2,2)		Conv5, Conv4
Conv6	Convolution2D	(512,3,3)	ReLU	Up6
Conv6	Dropout(0.2)			Conv6
Conv6	Convolution2D	(512,3,3)	ReLU	Conv6
Up7	Merge	(2,2)		Conv6, Conv3
Conv7	Convolution2D	(256,3,3)	ReLU	Up7
Conv7	Dropout(0.2)			Conv7
Pool7	Convolution2D	(256,3,3)	ReLU	Conv7
Up8	Merge	(2,2)		Conv7, Conv2
Conv8	Convolution2D	(128,3,3)	ReLU	Up8
Conv8	Dropout(0.2)	(2,2)		Conv8
Pool8	Convolution2D	(128,3,3)	ReLU	Conv8
Up9	Merge	(2,2)		Conv8, Conv1
Conv9	Convolution2D	(64,3,3)	ReLU	Up9
Conv9	Dropout(0.2)			Conv9
Pool9	Convolution2D	(64,3,3)	ReLU	Conv9
Conv10	Convolution2D	(1,1,1)	Sigmoid	Conv9

A modell tanításakor a következő paramétereket módosíthatjuk a jobb teljesítmény érdekében:

- **Epochok száma:** azon fordulók száma, amelyek alatt az egész adatszett áthalad a hálón. Mivel egyszerre nem lehetséges ekkora mennyiségű adatot átfuttatni a hálón, ezért batch-ekre bontjuk a teljes inputot.
- **Batch mérete:** azon inputok száma, melyek egyszerre haladnak át a hálón egy batch-ben.
- **Filterek száma:** az egy rétegben elhelyezkedő mesterséges neuronok száma.
- **Wideness:** a filterek szélessége.
- **Iterációk:** ennyi batch-re lesz szükség az epoch teljesítéséhez.
- **Optimizer:** gradiens, sztochasztikus gradiens vagy Adam.

- **Learning rate schedule:** a tanítás alatt egy előre definiált rend (schedule) szerint csökkentjuk a learning rate-et.

A feladathoz tartozó kód megtalálható a Függelékben.

4.3.3. A csomók klasszifikálása a tüdőben

A képeken a csomók mellett ekkor még mindig nagy zaj van jelen, ezért érdemes klasszifikálni a meglévő pontokat: csomó vagy nem csomó. Ezzel az eljárással csökkenthető az FP-k száma.

A számítási kapacitást figyelembe véve a csomók lehetséges nagyságát 36 pixelnyi méretben limitáljuk. Az adathalmazban 1187 csomópont van, ami a mély neurális hálózat tanításához alacsony szám. Ez a probléma elhárítható azzal, hogy a csomók középpontja körül nagyon sok pixel kivágható, így növelhető a tanító adat. Ezzel megegyező számú negatív példát mintavételezünk a képről a tanításhoz.

- Nem szabad véletlenszerűen mintavételezni negatív mintákat a CT-felvételekről. Érdemes a potenciális csomókból válogatni, vagy az FP-kből, így a modell sokkal jobban megtanulja majd a nem-csomókat elválasztani.
- Az adatszett augmentációjára még a tanítás közben sort kell keríteni, mert a csomók szimmentrikus régiók válzozó méretben.

A következő lépésben 3D CNN-nel Kerasban tanítjuk a klasszfikátort. A rétegek a következők:

- Convolution3D (16);
- Activation (ReLU);
- MaxPooling3D;
- Convolution2D (32);
- Activation (ReLU);
- MaxPooling3D;
- Convolution3D (64);
- Activation (ReLU);
- MaxPooling3D;
- Dropout (0.25);

- Flatten;
- Dense (512);
- Activation (ReLU);
- Dropout (0.5);
- Dense (128);
- Activation (ReLU);
- Dropout (0.5);
- Dense (2);
- Activation (SoftMax);

A pontosság növeléséhez használható további feladatok:

- Multi-scale CNN-ek használata, hogy még több jellemzőt találjunk.
- Kivonjuk a jellemzőket a fully-connected rétegekből és XGBoost-ot használunk rajtuk a pontosság növelésének érdekében.

Tehát összefoglalva először U-Net segítségével létrehozzuk a lehetséges csomókat, majd minden potenciális csomó körül pixeleket vágunk ki és klasszifikáljuk őket. A pontokat DBScan segítségével klasztereztük, és egy klaszter valószínűségét a klaszteren belüli összes pont átlagvalószínűsége adja.

4.3.4. A modell jóságának mérése

A modell jóságának méréséhez szükséges a mérőszámok pontos definiálása [5]. A konkrét feladatra a 0-s érték jelenti az egészségeset, az 1-es érték a beteget.

4.2. táblázat. Igazságmátrix (confusion matrix)

	Valós kimenet		
Prediktált kimenet		1 (pozitív)	0 (negatív)
	1 (pozitív)	**TP**	**FP**
	0 (negatív)	**FN**	**TN**
Összesen		**P**	**N**

A táblázat alapján:

- **P, Positive:** $TP + FN$, az összes beteg száma a valóságban.

- **N, Negative:** $FP + TN$, az összes egészséges száma a valóságban.
- **TP, True Positive:** a valós és a prediktált kimenet is 1, tehát a modell helyesen prediktált.
- **FP, False Positive:** a valós érték 0, de a modell 1-est predikált.
- **FN, False Negative:** a valós érték 1, de a modell 0-t predikált.
- **TN, True Negative:** a valós és a prediktált érték is 0.

Gyakran használt mérőszámok:

- TP rate $= \dfrac{TP}{P}$ = Helyesen klasszifikált pozitívak / Összes pozitív
- FP rate $= \dfrac{FP}{N}$ = Helytelenül klasszifikált negatívok / Összes negatív
- Pontosság $= \dfrac{TP + TN}{P + N}$
- Precizitás $= \dfrac{TP}{TP + FP}$
- Szenzitivitás $= \dfrac{TP}{TP + FN}$

A jóság mérésére a Sørensen-Dice veszteségfüggvényt használjuk. Ennek oka az, hogy ez a veszteségfüggvény nagyon jól teljesít kiegyensúlyozatlan adathalmazokon, azaz olyan esetekben, amikor a pozitív minták száma jóval kevesebb a negatívaknál.[15]

4.3.1. Definíció. *Sørensen-Dice coefficent: Legyen y a valós címkék halmaza, $\hat{y}$ pedig a modell által prediktált címkék halmaza. Ekkor*

$$dice(y, \hat{y}) = \frac{2|y \cap \hat{y}|}{|y| + |\hat{y}|},$$

ahol $|y|$ és $|\hat{y}|$ rendre y és $\hat{y}$ számossága.

A Sørensen-Dice coefficent megadható a true positives (TP), false positives (FP) és false negatives (FN) segítségével is:

$$dice(y, \hat{y}) = \frac{2TP}{2TP + FP + FN}.$$

A Sørensen-Dice veszteségfüggvény a következő alakban áll elő:

$$L_{dice}(y, \hat{y}) = 1 - dice(y, \hat{y})$$

4.4. A kísérletek hardveres és szoftveres környezete

4.4.1. Hardver

Az MTA Rényi Alfréd Matematikai Kutatóintézet jóvoltából hozzáférést kaptam az alábbi hardverekhez. A nagy teljesítmény hatalmas segítséget jelentett a kísérletek elvégzésekor.

CPU - Intel Xeon Processor E5-1650 v4

- Processor Base Frequency: 3.60 GHZ

RAM: 64 GB

GPU - GeForce GTX 1080 Ti

- Memória: 11 GB
- Boost clock: 1582 MHz

4.4.2. Szoftver

A python környezet remek keretrendszert biztosít a deep learning modellek gyakorlati megvalósításához. A választott API (Application Programming Interface, alkalmazásfejlesztési környezet) a Keras. A dokumentációban is olvasható, hogy a fejlesztésének célja kifejezetten a gyors kísérletezés lehetőségének biztosítása[2].

A használt csomagok és verziószámaik:

- keras 2.1.5
- tensorflow 1.4.0.rc0 (backend)
- numpy 1.14.1

4.5. Az eredmények kiértékelése

A kísérletekhez a szegmentáció, feladatát választottam. Ez a teljes feladat egy relaxált változata, az eredmény pedig később felhasználható lesz a klasszifikációs feladathoz.

[2] `https://keras.io/`

4.5.1. Első kísérlet

A szegmentáció tanítása az alábbi paraméterekkel történt:

- Number of epochs = 20;
- Wideness = 2;
- Batch size = 8;
- Starting learning rate = 0.01;
- Optimizer = Adam.

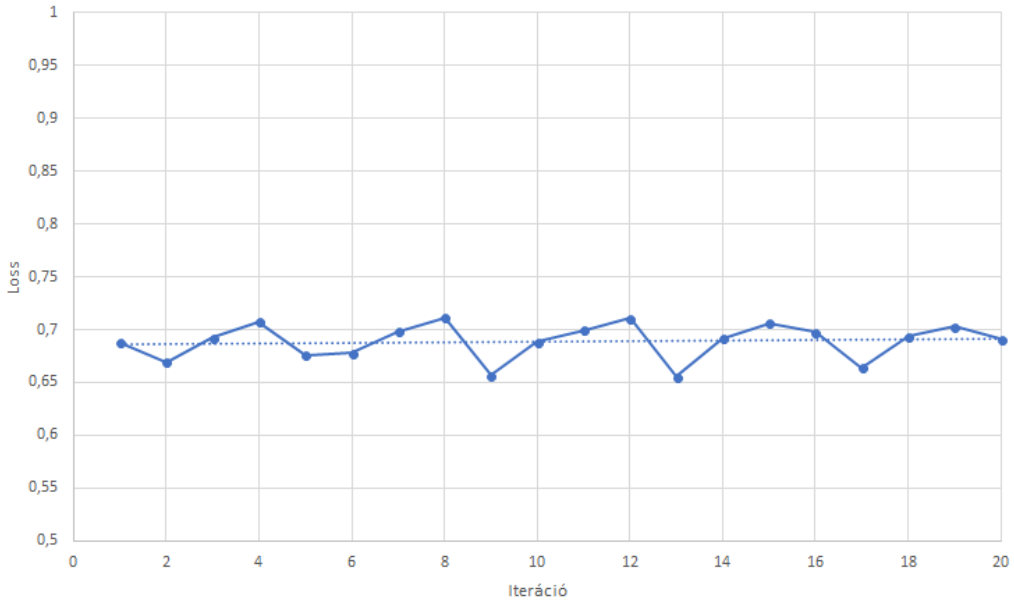
A veszteségfüggvény alakulása a 4.4 ábrán látható, a learning rate pedig a 4.5 ábrán.

4.5.2. Második kísérlet

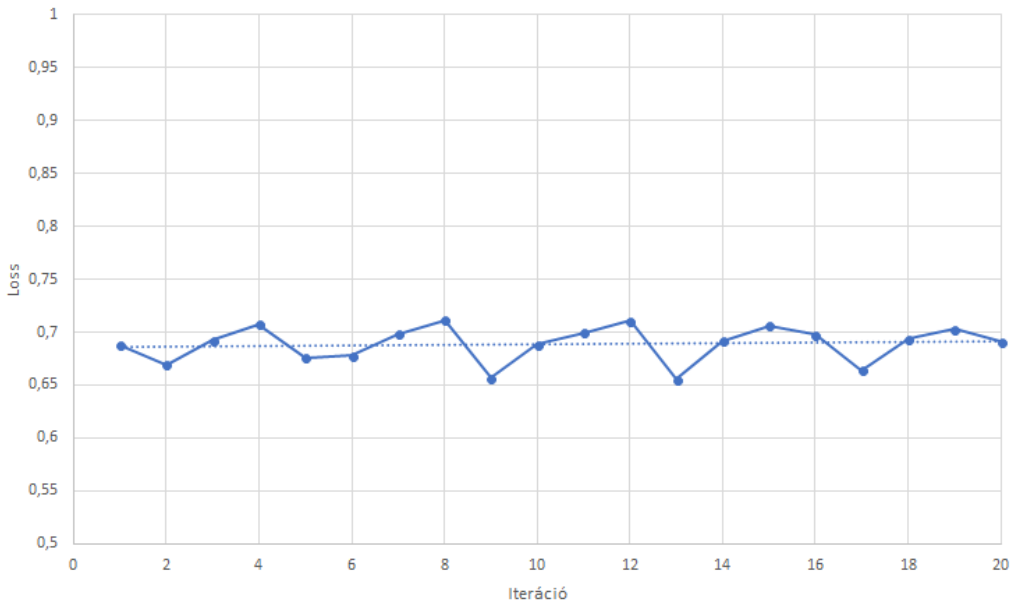
A szegmentáció tanítása az alábbi paraméterekkel történt:

- Number of epochs = 1;
- Number of iterations = 1000;
- Wideness = 1;
- Batch size = 8;
- Starting learning rate = 0.001;
- Optimizer = Adam.

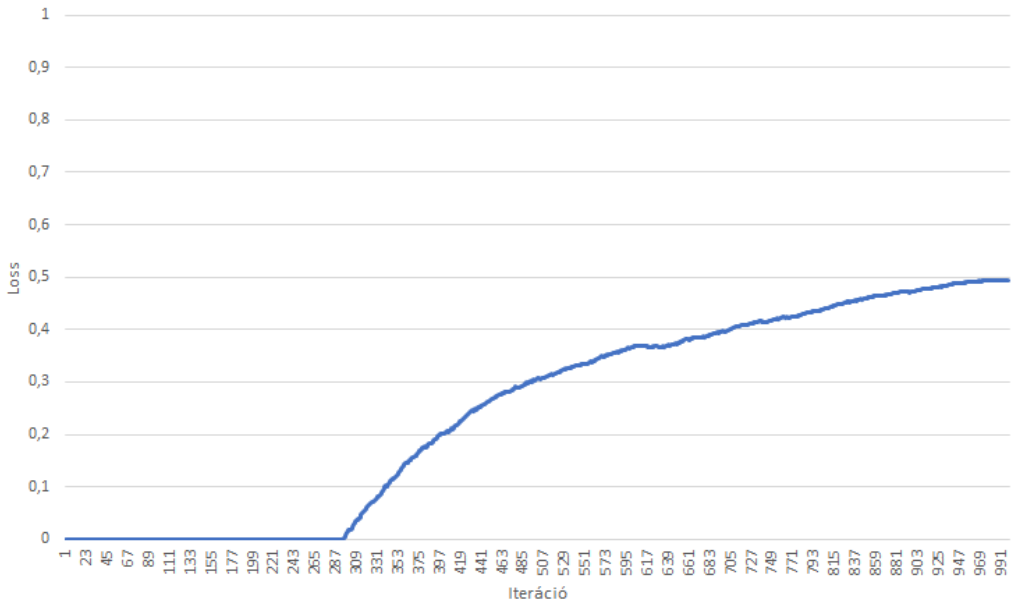
A veszteségfüggvény alakulása a 4.6 ábrán látható.



4.4. ábra. A veszteségfüggvény alakulása alakulása az első kísérlet alatt.



4.5. ábra. A learning rate alakulása az első kísérlet alatt.



4.6. ábra. A veszteségfüggvény alakulása a második kísérlet alatt.

5. fejezet

Konklúzió

Dolgozatom írása közben rengeteg új ismeretet szereztem a konvolúciós neurális hálókról és néhány kapcsolódó területről. Az adatok felderítése során kihívást jelentett az adatszettek hatalmas mérete, de ennek köszönhetően elsajátíthattam néhány módszert a hasonló problémák megoldására. A feladat bonyolultsága miatt csak a szegmentációs rész megvalósításáig jutottam el, de ez a megoldás nagyban segítette számomra a témakör jobb megértését.

Az eredmények alapján látható, hogy a szegmentáció szegmentáció elvégezhető konvolúciós neurális hálózatokkal, természetesen további kísérletekkel a hatékonysága jobban alátámasztható lenne.

Összességében látható, hogy a konvolúciós neurális hálók már rengeteg alkalommal kiemelkedően teljesítettek a képfeldolgozás, illetve az orvosi képfeldolgozás területén. Az algoritmusokat fejlesztve a közeljövőben hatékonyabbá és gyorsabbá tehetjük a diagnosztikát, aminek következménye várhatóan a rákos daganatok miatti halálozás arányának csökkenése lesz.

Irodalomjegyzék

[1] Filippo Amato, Alberto López, Eladia María Peña-Méndez, Petr Vaňhara, Aleš Hampl, and Josef Havel. Artificial neural networks in medical diagnosis, 2013.

[2] Joseph Antony, Kevin McGuinness, Noel E O'Connor, and Kieran Moran. Quantifying radiographic knee osteoarthritis severity using deep convolutional neural networks. In *Pattern Recognition (ICPR), 2016 23rd International Conference on*, pages 1195–1200. IEEE, 2016.

[3] dr. Firtkó Szilveszter. Hogyan fogja a mesterséges intelligencia befolyásolni a radiológiát az elkövetkezendő 20 évben? 2017.

[4] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115, 2017.

[5] Tom Fawcett. An introduction to ROC analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[6] Varun Gulshan, Lily Peng, Marc Coram, Martin C Stumpe, Derek Wu, Arunachalam Narayanaswamy, Subhashini Venugopalan, Kasumi Widner, Tom Madams, Jorge Cuadros, et al. Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. *Jama*, 316(22):2402–2410, 2016.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[8] Andrej Karpathy. Cs231n convolutional neural networks for visual recognition. *Neural networks*, 1, 2016.

[9] Javed Khan, Jun S Wei, Markus Ringner, Lao H Saal, Marc Ladanyi, Frank Westermann, Frank Berthold, Manfred Schwab, Cristina R Antonescu, Carsten Peterson, et al. Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks. *Nature medicine*, 7(6):673, 2001.

[10] Edward Kim, Miguel Corte-Real, and Zubair Baloch. A deep semantic mobile application for thyroid cytopathology. In *Medical Imaging 2016: PACS and Imaging Informatics: Next Generation and Innovations*, volume 9789, page 97890A. International Society for Optics and Photonics, 2016.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[12] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[13] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen AWM van der Laak, Bram van Ginneken, and Clara I Sánchez. A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88, 2017.

[14] Yun Liu, Krishna Gadepalli, Mohammad Norouzi, George E Dahl, Timo Kohlberger, Aleksey Boyko, Subhashini Venugopalan, Aleksei Timofeev, Philip Q Nelson, Greg S Corrado, et al. Detecting cancer metastases on gigapixel pathology images. *arXiv preprint arXiv:1703.02442*, 2017.

[15] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In *3D Vision (3DV), 2016 Fourth International Conference on*, pages 565–571. IEEE, 2016.

[16] Bogner Péter, Walter Norbert, Barta Miklós, Vandulek Csaba, Harkányi Zoltán, Morvay Zita, Bódi Péter, omosi Gábor, Kovács Árpád, Emri Miklós, Mikecz Pál, Trón Lajos, and Balkay László. Az orvosi képalkotás fizikája az orvosi laboratóriumi és képalkotó diagnosztikai analitikus alapszak hallgatói részére. Medicina Könyvkiadó Zrt., 2014.

[17] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[18] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000.

[19] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[20] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, et al. Going deeper with convolutions. Cvpr, 2015.

[21] Song Yuheng and Yan Hao. Image segmentation algorithms overview. *arXiv preprint arXiv:1707.02051*, 2017.

Függelék

A LUNA16 adatszetten végzett előfeldolgozás kódja

A kód forrása elérhető online: `https://github.com/dreamlxt17/luna_lung_detection_code/blob/master/creat_same_nodule_data.py`.

```
import glob
import numpy as np
import os
import SimpleITK as sitk
import skimage.transform
import scipy.ndimage
import pandas as pd
import cPickle as pickle
import gzip
import matplotlib.pyplot as plt
import time

from xyz_utils import load_itk, world_2_voxel, voxel_2_world
from joblib import Parallel, delayed

RESIZE_SPACING = [1, 1, 1]
SAVE_FOLDER_image = '1_1_1mm_slices_lung_filtered'
SAVE_FOLDER_lung_mask = '1_1_1mm_slices_lung_masks_filtered'
SAVE_FOLDER_nodule_mask = '1_1_1mm_slices_nodule_filtered'


def seq(start, stop, step=1):
    n = int(round((stop - start) / float(step)))
    if n > 1:
        return ([start + step * i for i in range(n + 1)])
```

```
    else:
        return ([])


def draw_circles(image, cands, origin, spacing):
    image_mask = np.zeros(image.shape)
    for ca in cands.values:
        radius = np.ceil(ca[4]) / 2
        coord_x = ca[1]
        coord_y = ca[2]
        coord_z = ca[3]
        image_coord = np.array((coord_x, coord_y, coord_z))
        image_coord = world_2_voxel(image_coord, origin,
           spacing)
        noduleRange = seq(-radius, radius, RESIZE_SPACING
           [0])

        for x in noduleRange:
            for y in noduleRange:
                for z in noduleRange:
                    coords = world_2_voxel(np.array((coord_x
                        + x, coord_y + y, coord_z + z)),
                       origin, spacing)
                    if (np.linalg.norm(image_coord - coords)
                        * RESIZE_SPACING[0]) < radius:
                        image_mask[int(np.round(coords[0])),
                            int(np.round(coords[1])), int(np
                           .round(coords[2]))] = int(1)

    return image_mask


def create_slices(imagePath, maskPath, cads):

    img, origin, spacing = load_itk(imagePath)

    try:
```

```
        mask, _, _ = load_itk(maskPath)
        print mask.shape
except:
        print 'mask is missing!'
        mask = np.zeros(img.shape)


imageName = os.path.split(imagePath)[1].replace('.mhd',
    '')
image_cads = cads[cads['seriesuid'] == imageName]

resize_factor = spacing / RESIZE_SPACING
new_real_shape = img.shape * resize_factor
new_shape = np.round(new_real_shape)
real_resize = new_shape / img.shape
new_spacing = spacing / real_resize

lung_img = scipy.ndimage.interpolation.zoom(img,
    real_resize)
lung_mask = scipy.ndimage.interpolation.zoom(mask,
    real_resize)

lung_mask[lung_mask > 0] = 1

nodule_mask = draw_circles(lung_img, image_cads, origin,
     new_spacing)

sliceList = []
for z in range(nodule_mask.shape[2]):
        if np.sum(nodule_mask[:, :, z]) > 0:
            sliceList.append(z)
print(sliceList)

for z in sliceList:
    lung_slice = lung_img[:, :, z]
    lung_mask_slice = lung_mask[:, :, z]
    nodule_mask_slice = nodule_mask[:, :, z]
```

```
original_shape = lung_img.shape
lung_slice_512 = np.zeros((512, 512)) - 3000
lung_mask_slice_512 = np.zeros((512, 512))
nodule_mask_slice_512 = np.zeros((512, 512))

offset = (512 - original_shape[1])
upper_offset = np.round(offset / 2)
lower_offset = offset - upper_offset

new_origin = voxel_2_world([-upper_offset, -
    lower_offset, 0], origin, new_spacing)

lung_slice_512[upper_offset:-lower_offset,
    upper_offset:-lower_offset] = lung_slice
lung_mask_slice_512[upper_offset:-lower_offset,
    upper_offset:-lower_offset] = lung_mask_slice
nodule_mask_slice_512[upper_offset:-lower_offset,
    upper_offset:-lower_offset] = nodule_mask_slice

savePath = imagePath.replace('original_lungs',
    SAVE_FOLDER_image)
directory = os.path.dirname(savePath)
if not os.path.exists(directory):
    os.makedirs(directory)
file = gzip.open(savePath.replace('.mhd', '_slice{}.
    pkl.gz'.format(z)), 'wb')
pickle.dump(lung_slice_512, file, protocol=-1)
pickle.dump(new_spacing, file, protocol=-1)
pickle.dump(new_origin, file, protocol=-1)
file.close()

savePath = imagePath.replace('original_lungs',
    SAVE_FOLDER_lung_mask)
directory = os.path.dirname(savePath)
if not os.path.exists(directory):
    os.makedirs(directory)
```

```
        file = gzip.open(savePath.replace('.mhd', '_slice{}.
            pkl.gz'.format(z)), 'wb')
        pickle.dump(lung_mask_slice_512, file, protocol=-1)
        pickle.dump(new_spacing, file, protocol=-1)
        pickle.dump(new_origin, file, protocol=-1)
        file.close()

        savePath = imagePath.replace('original_lungs',
            SAVE_FOLDER_nodule_mask)
        directory = os.path.dirname(savePath)
        if not os.path.exists(directory):
            os.makedirs(directory)
        file = gzip.open(savePath.replace('.mhd', '_slice{}.
            pkl.gz'.format(z)), 'wb')
        pickle.dump(nodule_mask_slice_512, file, protocol
            =-1)
        pickle.dump(new_spacing, file, protocol=-1)
        pickle.dump(new_origin, file, protocol=-1)
        file.close()

        print 'done!'



def createImageList(subset, cads):

    imagesWithNodules = []
    subsetDir = '/mnt/g2big/lung-luna/original_lungs/subset
        {}'.format(subset)
    imagePaths = glob.glob("{}/*.mhd".format(subsetDir))  #
        path for every image
    for imagePath in imagePaths:
        imageName = os.path.split(imagePath)[1].replace('.
            mhd', '')  # get pure name

        tmp = cads[cads['seriesuid'] == imageName]
```

```
        if len(tmp.index.tolist()) != 0:  # tmp.index.tolist
            (): get a list , shows there's nodule
             imagesWithNodules.append(imagePath)


    return imagesWithNodules
if __name__ == "__main__":

    cads = pd.read_csv("/mnt/g2big/lung-luna/CSVFILES/
        annotations.csv")

    for subset in range(10):
        start_time = time.time()
        print '{} - Processing subset'.format(time.strftime(
            "%H:%M:%S")), subset
        imagePaths = createImageList(subset, cads)
        print(imagePaths)
        Parallel(n_jobs=8)(delayed(create_slices)(imagePath,
             imagePath.replace('original_lungs/subset{}'.
            format(subset), 'seg-lungs'), cads) for imagePath
             in imagePaths


        print '{} - Processing subset {} took {} seconds'.
            format(time.strftime("%H:%M:%S"), subset, np.
            floor(time.time() - start_time))
```

A szegmentáció tanítása

Az U-Net architektúra gyakorlati megvalósításában és a kód létrehozásában témavezetőm, Csiszárik Adrián nyújtott segítséget.

```
from os import listdir
from os.path import isfile, join
import numpy as np
import math

import matplotlib
```

```
matplotlib.use('Agg')

from keras.models import Model
from keras.layers import Input, merge, Dropout, Conv2D,
    MaxPooling2D, UpSampling2D, BatchNormalization
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint,
    LearningRateScheduler
from keras import backend as K

import gzip
import pickle


import matplotlib.cm as cm
import matplotlib.pyplot as plt
from PIL import Image

from sklearn.preprocessing import normalize

def input_generator(dir_path, dir_path_y, subsets,
    batch_size=8):
   while True:
    for subset in subsets:
     subset_dir_path = dir_path + '/subset' + str(subset)
     subset_dir_path_y = dir_path_y + '/subset' + str(subset)

     onlyfiles = [f for f in listdir(subset_dir_path) if
         isfile(join(subset_dir_path, f))]
     i = 0
     xs = []
     ys = []
     for filename in onlyfiles:

         with gzip.open(join(subset_dir_path,filename), 'rb')
             as f:
             x = pickle.load(f)
```

```
        with gzip.open(join(subset_dir_path_y,filename), 'rb
            ') as f:
            y = pickle.load(f)
        x = (x-x.min())/(x.max()-x.min() +0.000001)
        xs.append(x)
        ys.append(y)
        i += 1
        if i >= batch_size:
            xs_np = np.expand_dims(np.stack(xs, axis=0),
                axis=3)
            ys_np = np.expand_dims(np.stack(ys, axis=0),
                axis=3)
            yield (xs_np, ys_np)
            i = 0
            xs = []
            ys = []


def dice_coef(y_true, y_pred):
    smooth = 1.
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = K.sum(y_true_f * y_pred_f)
    return (2. * intersection + smooth) / (K.sum(y_true_f) +
        K.sum(y_pred_f) + smooth)


def dice_coef_loss(y_true, y_pred):
    return -dice_coef(y_true, y_pred)

def step_decay(epoch):
    initial_lrate = 0.001
    drop = 0.1
    epochs_drop = 5.0
    lrate = initial_lrate * math.pow(drop,..
            math.floor((1+epoch)/epochs_drop))
    return lrate
```

```
lrate = LearningRateScheduler(step_decay)

wideness = 1

def unet_model():
    inputs = Input((512, 512, 1))
    conv1 = Conv2D(16*wideness, (3, 3), activation='relu',
        padding='same')(inputs)
    conv1 = Dropout(0.2)(conv1)
    conv1 = Conv2D(16*wideness, (3, 3), activation='relu',
        padding='same')(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

    conv2 = Conv2D(16*wideness*2, (3, 3), activation='relu',
         padding='same')(pool1)
    conv2 = Dropout(0.2)(conv2)
    conv2 = Conv2D(16*wideness*2, (3, 3), activation='relu',
         padding='same')(conv2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)

    conv3 = Conv2D(16*wideness*4, (3, 3), activation='relu',
         padding='same')(pool2)
    conv3 = Dropout(0.2)(conv3)
    conv3 = Conv2D(16*wideness*4, (3, 3), activation='relu',
         padding='same')(conv3)
    pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)

    conv4 = Conv2D(16*wideness*8, (3, 3), activation='relu',
         padding='same')(pool3)
    conv4 = Dropout(0.2)(conv4)
    conv4 = Conv2D(16*wideness*8, (3, 3), activation='relu',
         padding='same')(conv4)
    pool4 = MaxPooling2D(pool_size=(2, 2))(conv4)

    conv5 = Conv2D(16*wideness*16, (3, 3), activation='relu'
        , padding='same')(pool4)
    conv5 = Dropout(0.2)(conv5)
```

```
conv5 = Conv2D(16*wideness*16, (3, 3), activation='relu'
    , padding='same')(conv5)

up6 = merge([UpSampling2D(size=(2, 2))(conv5), conv4],
   mode='concat', concat_axis=3)
conv6 = Conv2D(16*wideness*8, (3, 3), activation='relu',
     padding='same')(up6)
conv6 = Dropout(0.2)(conv6)
conv6 = Conv2D(16*wideness*8, (3, 3), activation='relu',
     padding='same')(conv6)

     up7 = merge([UpSampling2D(size=(2, 2))(conv6), conv3
         ], mode='concat', concat_axis=3)
conv7 = Conv2D(16*wideness*4, (3, 3), activation='relu',
     padding='same')(up7)
conv7 = Dropout(0.2)(conv7)
conv7 = Conv2D(16*wideness*4, (3, 3), activation='relu',
     padding='same')(conv7)

up8 = merge([UpSampling2D(size=(2, 2))(conv7), conv2],
   mode='concat', concat_axis=3)
conv8 = Conv2D(16*wideness*2, (3, 3), activation='relu',
     padding='same')(up8)
conv8 = Dropout(0.2)(conv8)
conv8 = Conv2D(16*wideness*2, (3, 3), activation='relu',
     padding='same')(conv8)

up9 = merge([UpSampling2D(size=(2, 2))(conv8), conv1],
   mode='concat', concat_axis=3)
conv9 = Conv2D(16*wideness, (3, 3), activation='relu',
   padding='same')(up9)
conv9 = Dropout(0.2)(conv9)
conv9 = Conv2D(16*wideness, (3, 3), activation='relu',
   padding='same')(conv9)

conv10 = Conv2D(1, 1, 1, activation='sigmoid')(conv9)
```

```
    opt = Adam(lr=0.01)

    model = Model(inputs=inputs, outputs=conv10)
    model.summary()
    model.compile(optimizer=opt, loss=dice_coef_loss,
        metrics=[dice_coef])

    return model

model = unet_model()

gen = input_generator("/mnt/g2big/lung-luna/1
    _1_1mm_slices_lung_filtered", "/mnt/g2big/lung-luna/1
    _1_1mm_slices_nodule_filtered", [0,1,2,3,4,5,6,7],
    batch_size=32)
val_gen = input_generator("/mnt/g2big/lung-luna/1
    _1_1mm_slices_lung_filtered", "/mnt/g2big/lung-luna/1
    _1_1mm_slices_nodule_filtered", [8,9], batch_size=32)


for j in range(100):
  (x,y) = gen.next()
  for i in range(8):
    print(x[i])
    img = Image.fromarray(np.reshape(x[i], [512,512]))
    img = img.convert('RGB')
    img.save("x_{0}.png".format(i))
    img = Image.fromarray(np.reshape(y[i], [512,512]))
    img = img.convert('RGB')
    img.save("y_{0}.png".format(i))
quit()

model.fit_generator(gen, epochs=50, verbose=1,
    steps_per_epoch=10000, validation_data=val_gen,
    validation_steps=1000, callbacks=[lrate])
```

```
for j in range(100):
  (x,y) = gen.next()
  preds = model.predict(x)
  for i in range(8):
    print(preds[i].shape)
    img = Image.fromarray(np.reshape(preds[i], [512,512]).
        astype('uint8'), mode='1')
    img = img.convert('RGB')
    img.save("pred_{0}_{1}.png".format(i,j))
```