

TDK-dolgozat

Tempfli Levente

HierNet: Képfelismerés hierarchiába rendezett konvolúciós hálózatokkal

Szerzők:

Tempfli Levente

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

Témavezetők:

Sándor Csanád, Tanársegéd

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

Lezárás dátuma: 2022.04

Kivonat

A konvolúciós neurális hálózatok (CNN) hatékony módszernek bizonyultak a képfelismerés kihívást jelentő feladatának megoldására a képek jellemzőinek a kinyerésével és a bemeneti adatok belső reprezentációjának megtanulásával, de hagyományosan figyelmen kívül hagyják a bemeneti adatok hierarchiáját. Szeretnénk bevezetni egy új módszert a CNN-nek egy kvázi döntési fába szervezésére, ahol az élek a CNN-rétegeket és azok jellemző kivonási operációit, a csomópontok pedig az utválasztókat vagy osztályozókat jelentik. Így a bemeneti kép kategóriájának eldöntése úgy történik, hogy a nem levél csomópontnál követjük a besorolások útvonalát, a széleken alkalmazzuk a konvolúciós rétegeket, és a végső döntést egy levél csomóponton hozzuk meg. A döntési fa az adatok hierarchiáját képviseli, például a „járművek” csomópontjában az „autó” és a „teherautó” gyermekcsomópontok. Kis számú kategória esetén ez a fa manuálisan is felépíthető, de bevezetünk egy egyszerű, felügyelet nélküli módszert, amellyel több kategória esetén is létrehozhatunk hasonló fát. A teszteket a CIFAR10, CIFAR100 és a Tiny ImageNet adathalmazokon végezzük, amelyek hierarchiáját automatikusan építjük fel. A CNN gerinceként hagyományosan jól teljesítő modelleket használunk, mint például a ResNet és a VGG16, és összehasonlítjuk megközelítésünket az eredetiek teljesítményével.

Tartalomjegyzék

1. Bevezető	1
2. Elméleti áttekintő	3
2.1. Gépi tanulás	3
2.2. Deep Learning és neurális hálózatok	5
2.3. Konvolúciós neurális hálózatok	7
2.4. Konvolúciós hálózati architektúrák	9
3. Modell leírás	11
3.1. Alap architektúra	12
3.2. A tanítás folyamata	17
3.3. Hozzáadott rétegek a kategorizálás előtt	20
3.4. Előtanítás	22
3.5. Transfer Learning	22
4. Osztályhierarchia felépítés	23
4.1. Konfúziós valószínűségi mátrix	23
4.2. Csoportosítási algoritmus	24
5. Felhasznált adathalmazok	26
5.1. CIFAR10	26
5.2. CIFAR100	27
5.3. Tiny ImageNet	28
6. Alap- vagy gerincmodellek	29
6.1. ResNet a CIFAR10-en	29
6.2. ResNet a CIFAR100-on	31
6.3. ResNet a Tiny ImageNet-en	31
6.4. VGG16 a CIFAR100-on	33
7. Kísérletek	34
7.1. Dummy vagy Proof of Concept	34
7.2. HierNet a CIFAR10-en ResNet gerinccel	36

7.2.1.	Az alap HierNet	36
7.2.2.	Az osztályozó elé hozzáadott rétegek hatása	38
7.2.3.	Az előtanítás hatása	39
7.2.4.	A Transfer Learning hatása	41
7.3.	Automatizált csoportosítás a CIFAR100-on	41
7.4.	HierNet a CIFAR100-on ResNet gerinccel	43
7.4.1.	A hozzáadott rétegek és a felosztási pont hatása	44
7.4.2.	A különböző csoportosítások teljesítménye	44
7.4.3.	Az előtanítás esete	45
7.4.4.	Egyéb ResNet méretekre	45
7.4.5.	Tesztelési pontosságok	46
7.5.	HierNet a CIFAR100-on VGG16 gerinccel	46
7.6.	HierNet a Tiny ImageNet-en ResNet gerinccel	47
8.	Konklúzió	48

1. Bevezető

A Deep Learning területén a képek osztályozásának problémája az egyik legalapvetőbb és leginkább kutatott probléma. A konvolúciós neurális hálózatok (CNN-ek) bevezetése jelentős áttörést jelentett a területen, és az azóta az egyre kifinomultabb architektúrák és a konvolúciós hálózatokra épülő algoritmusok megjelenése évről évre növelik a modelljeink teljesítményét.

A hagyományos CNN-ek szekvenciális, egymás után lévő rétegekből épülnek fel, kezdve egy bemeneti réteggel, és befejezve egy simító és néhány teljesen kapcsolt (fully connected) réteggel. E hálózatok mögött az az elképzelés áll, hogy a konvolúciós művelet a bemenet kis területére hat, így kis részleteket is képes észlelni, szemben a hagyományos NN-vel, ahol egy kimeneti neuron minden bemeneti neurontól függ. Néhány ilyen konvolúció a hamarabbi rétegekben általában alacsonyabb szintű, míg a későbbiekben magasabb szintű jellemzőket von ki. Így több egymás után lévő konvolúciós réteg elég jól megtanulhatja egy objektum általános vizuális reprezentációját. Ezek a szekvenciális modellek azonban figyelmen kívül hagyják az adatok hierarchiáját, és minden osztályt egyformán megkülönböztethetőként kezelnek, miközben a valóságban vannak nagyon hasonló megjelenésű osztálycsoportok (pl. a kutya eléggé hasonlít a macskára, míg a tulipán sokkal jobban hasonlít a rózsára és orchideára, mint bármely állatra).

Szeretnénk bemutatni egy új CNN architektúrát, amely kihasználja az osztályok közötti hierarchiát, ez neveznék HierNet-nek. Az alapötlet az, hogy először szeretnénk néhány konvolúciós réteggel kinyerni néhány jellemzőt, majd ezek kimeneteiből megjósolni egy szuperosztályt (másképp: osztályok csoportját), majd egy másik halom konvolúciós rétegtől megkapni a képhez tartozó osztályt. Az egyik trükk az, hogy a konvolúciós rétegek második halmaza az első réteghalmaz kimenetére van ráköthetve, mivel az első halmaz által megtanult tulajdonságok a szuperosztályok megkülönböztetésére felhasználhatók a második halmaznál további jellemzők kinyerésére és majd az adott kategória meghatározására. A második trükk az, hogy míg az első rétegek közösek a szuperosztályok közt, addig minden szuperosztálynak megvannak a maga további rétegei. Így módon a szuperosztályban egy adott osztályt előrejelző konvolúciós réteghalmaz az adott szuperosztály osztályaira jellemző tulajdonságok kinyerésére összpontosíthat.

A HierNet egy faszerű architektúrába szerveződik, döntési faként is felfogható, azzal a különbséggel, hogy a fa élei konvolúciós, jellemzőkinyerési műveleteket tartalmaznak, a csomópontokban pedig megtörténik a besorolás, hogy melyik gyerek felé kell elindulni következőleg a csomópont előtt lévő élen történő jellemző kinyerések alapján. A levél csomópontjainak osztályozója adja meg a modell által prediktált osztályt. Bár a tanítás során a teljes fát kell tanítanunk, az osztályozás során elég csak egy útvonalat kiértékelnünk a gyökércsomóponttól egy levélcsomópontig a csomópontban található osztályozások kimenetei alapján, ami jelentősen csökkentheti a kiértékeléshez szükséges időt.

Modellünk felépítéséhez szükségünk van az osztályok hierarchiájára faként ábrázolva. Egy ilyen fában az osztályok a levelek, a belső csomópontok pedig a gyermek csomópontjainak a szuperosztályai. Kis számú osztály esetén egy ilyen fa akár manuálisan is megszerkeszthető, de bemutatunk egy módszert egy ilyen hierarchikus fa automatizált felépítésére.

Hasonló megközelítés az "Adaptive Neural Trees" [16]. Ők is fa alakú architektúrával rendelkeznek, hasonlóan az éleken konvolúciós rétegek (ún. "transformations"), a csomópontokban pedig osztályozók (a belső csomópontokban ún. "routers", a levél csomópontjaiban "solvers") és képbesoroláskor ők is csak egy útvonalat követnek. A különbség az, hogy a levélcsomópontjaik az összes osztályt valószínűségi eloszlását kiadják, és a fa hierarchiája nem az osztályok közötti logikai hierarchián alapul, hanem a tanítás során dinamikusan épül fel olyan módon, hogy véletlenszerűen hozzáad levél csomópontokat és éleket a fához, és megtartja őket, ha javítják a modell pontosságát. és elveti a változtatást, ha nem.

További hasonló modellek az "Attention Convolutional Binary Trees" [7]. Ők is egy fa alakú architektúra segítségével konvolúciós műveleteket rendeznek el az élek mentén, majd a csomópontokban útválasztókat. Ennek a hálózatnak az alapgondolata hasonló a miénkhez: hogy a levelek felé a konvolúciós rétegek egyre specifikusabb sajátosságokat tanuljanak meg. Van azonban néhány eltérés, például: bináris fát használnak, a széleken végzett műveletek aszimmetrikusak, és az eredmények a levelek eredményinek összegzésével kaphatók meg.

Az utolsó hasonló megközelítés, amelyet szeretnénk megemlíteni, az a "Branch Convolutional Neural Network" [18] vagy B-CNN. Ők már egy előre meghatározott osztályfát

használnak, és bár szekvenciális a modelljük, a CNN különböző mélységein vannak osztályozók a szuperosztályoknak, majd az osztályoknak. Tekintettel arra, hogy a szuperosztályok osztályozói korábbiak, mint a gyermekosztályoké, ez a cikk valóban megmutatta, hogy a szuperosztályok osztályozására kivont jellemzők újra hasznos alapot biztosítanak az utána lévő convolúciós rétegeknek, majd osztályozóknak.

Egy rövid elméleti áttekintés után részletes leírást adunk modellarchitektúránkról, majd a leírjuk a csoportosítási vagy hierarchia felépítési algoritmust. Ezután megtárgyaljuk azokat az adathalmazokat, amelyeken tesztelünk, és azokat a gerincmodelleket, amelyeket a HierNet éleihez használunk. Végül pedig néhány gerinc-adathalmaz páron végzett kísérletekkel megmutatjuk a modellünk tényleges teljesítményét, és összehasonlítjuk azokat a gerinchálózatok teljesítményével.

2. Elméleti áttekintő

Ebben a részben szeretnénk egy tömör leírást adni a HierNet megértéséhez szükséges elméleti háttérrel. Először a gépi tanulásról, majd a Deep Learning-ről, majd a konvolúciós hálózatokról, végül pedig a gerincként használt két architektúráról fogunk tárgyalni. Az itt közölt információk főleg a [2] könyvből származnak.

2.1. Gépi tanulás

A Deep Learning megértéséhez először meg kell ismerkednünk a gépi tanulással.

A gépi tanulás alapjai A [12] a gépi tanulás definícióját a következőképpen adja meg: "Egy számítógépes programról azt mondják, hogy tanul az E tapasztalatból bizonyos T feladatok osztályán, a P teljesítménymérő alapján, ha a teljesítménye a T feladatnál, P-vel mérve, az E tapasztalattal javul." Ahol a T feladat lehet például osztályozás (c osztályok egyikének hozzárendelése egy bemenethez) vagy regresszió ($x \in \mathbb{R}$ szám előállítása egy adott bemenethez). A P mérőszám lehet pontosság, hibaarány, score... . A tapasztalat lehet felügyelt (a kívánt eredmények meg vannak adva), felügyelet nélküli (annotáció nélkül) vagy mások, például megerősítő tanulás (reinforcement learning).

Valószínűségi felügyelt tanulás Feladatunk a valószínűségi felügyelt tanulás (Probabilistic Supervised Learning) lesz, nevezetesen egy olyan modell megalkotása, amely megbecsül egy $p(y|x)$ valószínűségi eloszlást egy x bemenetre. Esetünkben a x a bemeneti kép, az y pedig az egyik lehetséges osztály a c osztály közül. Ha θ a modell paramétervektora, akkor a valószínűségi eloszlás $p(y|x, \theta)$ formában írható fel, és a célunk az, hogy megtaláljuk azt a θ -t, amelyre a valószínűségi eloszlás illeszkedik az $X = \{x^{(i)}, y^{(i)}\}_{i \in \llbracket 1, N \rrbracket}$ adathalmazunkra.

Tanítási, validációs és tesztelési halmazok Általában három adathalmazt (vagy felosztást) különböztetünk meg: tanítási, validációs és tesztelési. A modellt betanítjuk (keresünk egy θ -t) a tanítóhalmaz mintái alapján. A cél azonban az, hogy egy modell jól teljesítsen a nem látott adatokon, és ennek a becsléséhez a tesztelési halmazon mért hibát használjuk (ez a tesztelési hiba vagy általánosítási hiba). A tesztelési halmaz independens a tanítási adatoktól, de ugyanabból az adatgeneráló eloszlásból származik. A hiperparaméterek olyan paraméterek, amelyeket a tanítás során nem adaptál az algoritmusunk, tehát ezeket mi állítjuk be a tanítás előtt. A validációs halmazon mért teljesítmény a hiperparaméterek hangolására szolgál.

Túlilleszkedés, alulilleszkedés és regularizáció Ha egy modell tökéletesen illeszkedik a tanítási adatokhoz (a tanítási hiba alacsony), de a tesztelési adatokon rosszul teljesít (nagy a tesztelési hiba), akkor azt szokták mondani, hogy a modell túlilleszkedésben (overfitting) szenved. Másrészt, ha egy modell gyengén teljesít a tanítási adatokon, akkor az alulilleszkedéstől (underfitting) szenved. Minden olyan gépi tanulási technikát, amelyet azzal a szándékkal vezettek be, hogy csökkentsék az általánosítási hibát, regularizációnak nevezzük.

Költség- és veszteségfüggvény Egyszerűen megfogalmazva, a költségfüggvény leírja a távolságot az előrejelzett értékek és a képzési halmaz valós értékei között. Általában az adathalmazban szereplő egyes példák veszteségfüggvényeinek összege. A következőképpen írható fel: $J(\theta) = \frac{1}{m} \sum_{i=1}^m L(f(x_i; \theta), y_i)$, ahol $f(x; \theta)$ a modellünk x bemenettel és θ paraméterekkel, kimenetként pedig a predikcióval. A veszteségfüggvény kiszámítja a mintánkénti távolságot, két példa:

- Négyzetes hiba: $L(f(x; \theta), y) = (f(x; \theta) - y)^2$
- Kategorikus keresztentrópia: $L(f(x; \theta), y) = - \sum_{i=1}^c y_i \log(p(y_i|x))$. Ezt gyakran használják kategorizálási feladatokhoz.

Gradiens ereszkedés A gradiens ereszkedés egy tanító algoritmus, amelyet a $J(\theta)$ költségfüggvény minimalizálására használnak. A cél az $\operatorname{argmin}_{\theta} J(\theta)$ megkeresése. A gradiens ereszkedés olyan módon működik, hogy követi a lefelé mutató gradienseket több iteráción keresztül, így minimalizálva a költségfüggvényt. Ezt olyan módon teszi, hogy a J függvény θ szerinti parciális deriváltját megszorozza egy kis értékkel és kivonja θ -ból. A következőképpen írható fel: $\theta \leftarrow \theta - \alpha \nabla_{\theta} J(\theta)$. Az α -t nevezzük a tanulási rátának vagy tanulási sebességnek. Ha túl magasra állítjuk, a tanítás nem konvergál, ha túl alacsonyra, akkor túl lassú lesz vagy beakad egy lokális minimumba. Léteznek tanulási ráta ütemezések, amelyek a tanítás előrehaladtával módosítják (általában csökkentik) a tanulási rátát. Továbbá általában az adatokat batch-ekre (kötegekre, részcsoportokra) osztják úgy, hogy minden iterációnál csak 1 köteggel alkalmazzák az ereszkedést, és amikor az összes batch egyszer végigiterált, akkor az egy epoch.

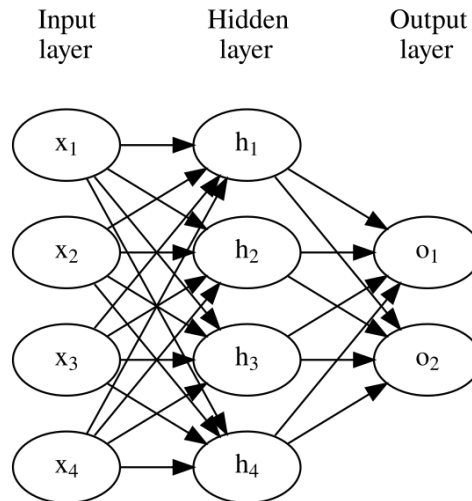
2.2. Deep Learning és neurális hálózatok

Ebben a részben röviden bemutatjuk a mesterséges neurális hálózatokat.

A neurális hálózatok alapjai Egy közönséges egyszerű mesterséges neurális hálózat egy bemeneti rétegből, több rejtett rétegből és egy kimeneti rétegből áll. Egy példa látható az 1. ábrán. Minden réteg számos neuront tartalmaz, a bemeneti rétegben lévő neuronok száma megegyezik a x bemeneti adatok méretével, ugyanez igaz a kimenetekre és az y -ra. A rejtett rétegek neuronjainak száma konfigurálható (ez például egy hiperparaméter). Egy réteg neuronjainak értékeit az előző réteg értékéből számítjuk ki: a rétegben lévő egy neuron értékét úgy számítjuk ki, hogy az előző rétegben lévő összes neuron értékét megszorozzuk más-más súlyokkal, majd ezeket összeadjuk, majd hozzáaduk egy eltolást (bias), és végül egy a aktivációs függvényt alkalmazunk rajta. Ezért hívjuk az ilyen típusú rétegeket teljesen kapcsolt rétegeknek. Legyen h egy réteg, g pedig a h előtti réteg, ebben az esetben $h_i = a(\sum_j w_{i,j} + b_i)$, ahol w a súlymátrix és b az eltolás vektor. Ha a rétegeket vektoroknak tekintjük, akkor a következőt írhatjuk: $h = a(Wg + b)$. A θ tartalmazza az összes súlyt és eltolást, így azokat kell optimalizálnunk, hogy a modell illeszkedjen az adathalmazhoz.

Aktivációs függvények Az aktivációs függvények szükségesek ahhoz, hogy modellünk a nemlinearitást biztosítsa. A népszerű aktivációs függvények a következők:

- szigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$



1. ábra. Egy egyszerű mesterséges neurális háló (Forrás: https://ufal.mff.cuni.cz/~straka/courses/npfl114/2122/slides/01/neural_network.svg)

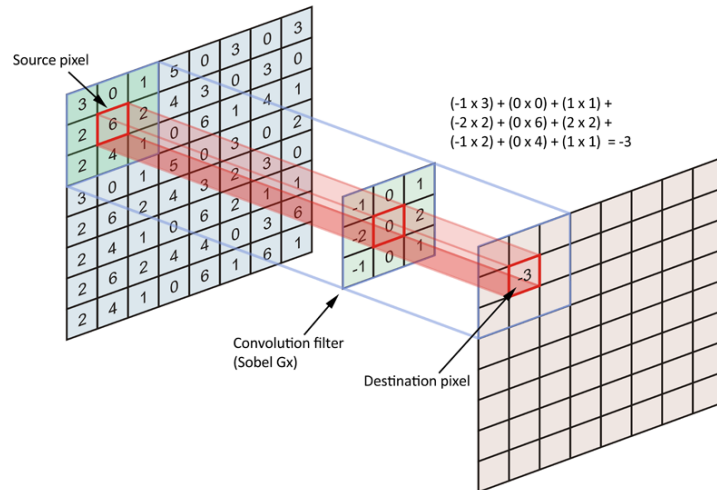
- Tanh: $\tanh(x) = 2\sigma(2x) - 1$
- ReLU: $\text{relu}(x) = \max(0, x)$
- Softmax: $\text{softmax}(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$

A kimeneti réteghez a szigmoidot általában logisztikus regresszióhoz, a softmax-ot pedig kategorikus predikcióhoz használják (valószínűségi eloszlást hoz létre, mert $\sum_i \text{softmax}(x)_i = 1$). A rejtett rétegeknél régebben a szigmoidot vagy a tanht-t használták, de a ReLU az utóbbi időben elterjedtebb.

Tanítás A neurális hálózatokat gradiens ereszkedéssel tanítják úgy, hogy minden súlyból kivonják a költségfüggvénynek a súly szerinti deriváltját, szorozva a tanulási rátával: $\theta_i = \theta_i - \alpha \frac{\partial L}{\partial \theta_i}$. A deriváltakat a backpropagation nevű technikával számítják ki, amely dióhéjban a veszteségfüggvény irányából az első réteg súlyai irányába halad, és a deriválási láncszabályt alkalmazza mindig a következő gradiens kiszámításához.

Neurális hálók regularizációja A neurális hálózatokhoz különféle regularizációs technikákat használnak, például:

- Korai leállás: Egyes modelleket ha túl sokáig tanítunk, hajlamosak túlilleszkedni.
- L2 regularizáció: Az összes súly négyzetes összege hozzáadódik a költségfüggvényhez, hogy büntesse a túl nagy súlyokat ($\tilde{J}(\theta; X) = J(\theta; X) + \lambda \|\theta\|_2^2$). “Weight Decay”-nek is nevezik.



2. ábra. Konvolúció (Source <https://i.stack.imgur.com/YDusp.png>)

- Adat augmentáció: Egyes adathalmazok esetében lehetőség van új adatok létrehozására a meglévőkből.
- Dropout: Ha “dropout”-t alkalmazunk egy rétegre, akkor minden neuronját p valószínűséggel deaktiváljuk a tanítás során. Segít egy neuron jelentőségének a csökkentésében.

2.3. Konvolúciós neurális hálózatok

A konvolúciós neurális hálózatok (CNN) a neurális hálózatok egy speciális típusa, amelyeket "rácsosan" struktúrált adatokhoz használnak. Léteznek olyan hálózatok, amelyek 1D-konvolúciót használnak az időbeli adatokhoz (pl. beszéd), de a 2D-konvolúciót használó hálózatokat fogjuk megtárgyalni, mert ezeket használják képekhez nagy sikerrel.

Konvolúciós rétegek A kétdimenziós konvolúciós műveletre találhatunk egy példát a 2. ábrán. Ez a művelet matematikailag felírva: $(K * I)_{i,j} = \sum_{m,n} I_{i+m,j+n} K_{m,n}$, ahol a K -t nevezzük kernelnek. A CNN-ekben nem csak egy, hanem több értékünk is van egyetlen pixelhez, ezeket csatornáknak nevezzük, így a kernelünk 3 dimenzióssá válik. Továbbá azt szeretnénk, hogy a konvolúciós réteg ne csak egy 2D-s kimenetet adjon, hanem egy olyat, ahol minden pixelnek van egy csatorna vektora. Ezt úgy érjük el, hogy több különböző kernelt alkalmazunk a bemeneten. Végül egy konvolúciós réteg a következőképpen írható fel:

$$(K * I)_{i,j,o} = \sum_{m,n,c} I_{i+m,j+n,c} K_{m,n,c,o}$$

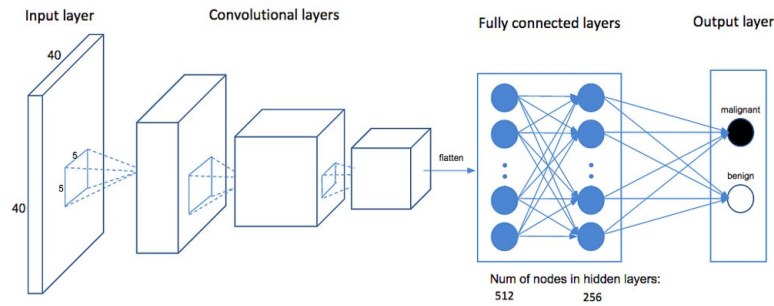
Motiváció A konvolúciós neurális hálózatok mögött három motivációs gondolat van:

- **Helyi interakció:** A hagyományos neurális hálókban egy réteg minden neuronja hatással van a következő réteg minden neuronjára, azonban a CNN konvolúciós műveleteket használ, amelyek csak kis területen hatnak, ami azt eredményezi, hogy egy neuront nem befolyásolja az összes előző neuron, hanem csak egy kis helyi részhalmaza. Ezáltal a CNN képes felismerni az apró, de jelentős részleteket. Egy másik ebből származó előny a paraméterek számának csökkentése.
- **Paraméterek megosztása:** A hagyományos NN-ben minden súlyt csak egyszer, egy neuronhoz használjuk, de egy CNN-ben minden pixelre ugyanazt a kernelt alkalmazzuk, így nem kell megtanulni minden pixel súlyozását, hanem csak általános kernelsúlyokat.
- **Shift invariance:** Ez azt jelenti, hogy ha a bemeneti objektum pozíciója meg is változik, a kimeneti reprezentációja változatlan marad. Ez igaz, mert a pixeleknek nincs külön súlya, van egy kernel súlyokkal, ami minden pixelre vonatkozik. Tehát egyes jellemzők helye nem számít, hiszen a kimeneti ábrázolás helytől függetlenül ugyanaz lesz.

Padding és stride A konvolúciók középpontjai nem lehetnek a bemenetek szélein (mert akkor egyes részek kiesnének a bemenet határain túlra), emiatt a kimenet felbontása csökken. A "same" padding úgy oldja meg ezt a problémát, hogy a bemenet oldalaihoz nulla értékeket fűz hozzá, hogy a felbontás ne változzon a konvolúció után. A "valid" padding az az eset, amikor egyáltalán nem fűzünk hozzá semmit. Ha csökkenteni akarjuk a kimenet méreteit, használhatunk stride-ot. Például egy 2-es stride azt jelenti, hogy csak minden második pixelre alkalmazzuk a konvolúciót, ami egy olyan kimenetet eredményez, amelynek a mérete 2-szer kisebb.

Max Pool réteg A pooling művelet a bemenet méretének csökkentésére szolgál. Ez bizonyos értelemben hasonlít a konvolúcióhoz, mert az értékek lokális régiójára hatunk, de azzal a különbséggel, hogy ezeknek az értékeknek a maximumát vagy átlagát számítjuk ki. Ez minden csatornán külön történik a csatornák számának megőrzése érdekében.

Batch Normalization A Batch Normalization [6] egy olyan technika, amellyel felgyorsítja a tanulást azáltal, hogy normalizálja egy konvolúciós réteg által kiadott értékeket egy batch-ben az aktivációs függvény előtt. Ha $x = (x_1, x_2, \dots, x_d)$ a bemenet, akkor a Batch Normalization



3. ábra. Szokványos CNN architektúra (Forrás: https://cdn-images-1.medium.com/max/1200/0*QyXSpqpm1wc_Dt6V.)

réteg kimenete $y_i = \gamma_i \times \hat{x}_i + \beta_i$, ahol $\hat{x}_i = \frac{x_i - \text{Exp}[x_i]}{\sqrt{\text{Var}(x_i)}}$ (az x_i normalizálva) és a β, γ tanulható paraméterek. A CNN-ek esetében azonban nem az egyes neuronokat normalizáljuk külön-külön, hanem egy csatorna értékeit normalizáljuk a batch és a térbeli dimenziók mentén.

CNN architektúra A 3. ábra bemutatja, hogy hogyan néz ki általában egy konvolúciós neurális hálózat architektúrája. Van egy bemeneti réteg, amelyet számos konvolúciós réteg követ, mindegyiket után aktiválás és opcionálisan Max Pool. Ezután az utolsó konvolúciós réteg kimenetét 1D-re lapítjuk (flattening), és néhány teljesen kapcsolt réteghez kapcsoljuk, amelynek az utolsó rétege a hálózat kimenete. Az AlexNet [10] volt az egyik első hálózat, amely átütő teljesítményt ért el az ImageNet [13] adatkészletén. A hálózatuknak 5 konvolúciós rétege volt ReLU aktiválással, 11×11 , 7×7 és 3×3 kernelméretekkel és néhány Max Pool réteggel köztük. Lapítás után 3 teljesen kapcsolt réteget használtak. Bár architektúrájuk a mai szabványokhoz képest elavult, valóban megmutatták a konvolúciós neurális hálózatok hatékonyságát.

2.4. Konvolúciós hálózati architektúrák

Szeretnénk bemutatni két népszerű CNN architektúrát, amelyeket a HierNetünk gerinceként is használunk.

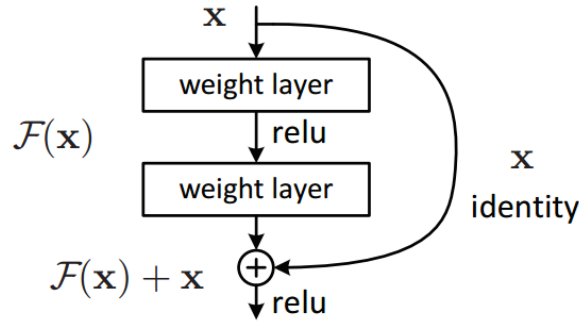
VGG16 A VGG16 [14] az ImageNet [13] képek osztályozására készült, ezért a bemeneti mérete 224×224 . A bemeneteket 5 csoportnyi konvolúciós réteg követi, mindegyik után egy Max Pool réteg, hogy a “feature map” méretét minden csoport között 2-szeresen csökkentsék. Az utolsó Max Pool után két teljesen kapcsolt réteg van, mindegyik 4096 neuronnal, ezt követi egy másik teljesen kapcsolt réteg 1000 neuronnal (mivel az ImageNetnek 1000 osztálya van), majd egy softmax aktiváció. Mindenhol máshol aktivációs függvényként a ReLU-t használják.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

4. ábra. VGG16 Konfigurációk(Forrás: [10])

A konvolúciós rétegek száma minden csoportban konfigurálható, és a szerzők 6 különböző konfigurációt biztosítanak. A 4. ábra mutatja a konfigurációkat és a teljes architektúrát. Figyelembe véve, hogy a cikk egyik célja az volt, hogy bemutassa a 3×3 konvolúciók felsőbbrendűségét a 7×7 és 11×11 konvolúciókkal szemben (amiket az AlexNet[10] használ például), az összes konvolúciós réteg (kivéve a "C" konfigurációban) 3×3 kernelt használ. A konvolúciós rétegek között a felbontás megőrzése érdekében 1-es stride-ot és *same* padding-et használnak. A VGG16 akkoriban szokatlanul mély architektúrája kiválóan bizonyult a kép jellemzőinek kinyerésében, és 2014-ben az ILSVRC (ImageNet Large Scale Visual Recognition Competition) második helyezettje lehetett.

ResNet A ResNet[3] hálózatban egy innovatív építőelem található: a Residual Block (a 5. ábra ábrázolja). Egy ilyen "residual" blokk néhány konvolúciós rétegből áll, majd egy "shortcut connection"-on keresztül a blokk előtti bemenet hozzáadódik a blokk kimenetéhez. Az aktivációk minden konvolúciós réteg után jelen vannak, kivéve a blokkban az utolsót, mert abban az esetben csak a "shortcut" kapcsolat után van. A "residual" blokkok létrehozását a degradáció problémája motiválja. Ha van egy mély hálózatunk, amikor újabb és újabb rétegeket adunk hozzá, a pontosság egy pont után csökkenni kezd. Egyszerűen fogalmazva,



5. ábra. Residual Block(Forrás: [10])

ezek a “shortcut” kapcsolatok lehetővé teszik az optimalizáló számára, hogy a tanítás során átugorja a szükségtelen rétegeket (különösen a korábbi szakaszokban), a gradiensek könnyű, a “shortcut”-on keresztül való visszapropagálásával. Ezt a hálózatot is az ImageNet[13] osztályozására tervezték, így a bemeneti méret 224×224 . A bemenetek után van egy konvolúciós réteg 7×7 konvolúcióval és egy 2-es "stride"-al, ezt követi a 3×3 max pool 2-es "stride"-al. Ezt követően 4 “residual” blokk csoport jön, 1 csoportban azonos típusú “residual” blokkok, tehát ugyanazt a konvolúciós réteget tartalmazzák. A hálózatot egy Global Average Pooling (minden csatornára kiszámítjuk az átlagot), egy 1000 egységből álló teljesen kapcsolt réteg és egy softmax aktiváció zárja. A “feature map”-ek felbontása minden blokkcsoport között lecsökken, olyan módon, hogy az elsőt kivéve mindegyik csoport első blokkjának első konvolúciós rétegénél 2-es stride-ot használunk. A Batch Normalization [6] közvetlenül minden konvolúció után és aktiválás előtt használatos. A szerzők többféle konfigurációt javasolnak, különböző n méretekben. A 6. ábra bemutatja a konfigurációkat és az általános architektúrát. A ResNet által bevezetett “residual” blokkok nagyon hatékonynak bizonyultak, és a modell a 2015-ös ILSVRC (ImageNet Large Scale Visual Recognition Competition) győztese lett.

3. Modell leírás

A HierNet feladata a bemeneti képek c kategóriákba sorolásának megtanulása, formálisan a $p(y|x)$ feltételes eloszlás megtanulása a $\{x^{(i)}, y^{(i)}\}_{i \in [1, N]}$ adathalbazon, ahol $x^{(i)} \in X$ egy kép, $y^{(i)} \in Y$ a megfelelő osztály és N a tanítási minták száma; Az X és Y a tanítási képek halmaza, illetve a c az osztályok halmaza. Először bemutatjuk a HierNet alapvető architektúráját, a tanítás módját, majd 3 technikát a modell pontosságának növelésére.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

6. ábra. ResNet Konfigurációk(Forrás: [3])

3.1. Alap architektúra

Ezennel formálisan meghatározzuk a HierNet architektúráját. Az architektúra egy döntési fa köré szerveződik, ezért az architektúrát egy $H = (T, F, C)$ triáddal definiáljuk, ahol T a fa topológiáját jelöli, F a fa élein végrehajtott jellemzőkivonási műveleteket, C pedig a csomópontokban végzett osztályozó műveleteket.

Topológia Tekintve, hogy ez egy fa, a T topológia egy V csomóponthalmazból és egy E élhalmazból áll, ahol $V = \{v_1, v_2, \dots, v_n\}$ és $E = \{e_1, e_2, \dots, e_m\} = \{(a, b) : a, b \in V, a \neq b\}$, ahol az (a, b) azt jelenti, hogy b az a gyereke. Egy és csak egy gyökércsomópont van: $root \in V$, tehát nincsen születe ($|\{(a, b) : b = root, a \in V\} \cap E| = 0$) és minden más $v \in V \setminus \{root\}$ csomópontnak csak egy szülője van ($|\{(a, b) : b = v, a \in V\} \cap E| = 1$). Ez a topológia a c osztály közötti hierarchiából épül fel, amelyet vagy manuálisan építünk fel valmami logikai hierarchia alapján, vagy egy algoritmussal az gerincmodell összetévesztései alapján (a 4. szakaszban leírtak szerint).

Műveletek Minden élhez nulla vagy több jellemző kinyerési művelet tartozik $f \in F$ (kvázi konvolúciók). Minden f függvény egy e_i élen helyezkedik el a j -ik pozícióban, tehát a bemenete az előtte levő $j - 1$ -edik művelet kimenete (hacsak nem az első), és a kimenete a bemenete a $j + 1$ -edik műveletnek (hacsak nem az utolsó). Minden élnek k számú művelete van, ezért a i -edik él j -edik jellemző kivonási műveletét $f_{i,j}$ -ként jelöljük, ahol $i \in \llbracket 1, m \rrbracket$ és $j \in \llbracket 1, k \rrbracket$. A műveletek iránya a szülő csomóponttól a gyerek csomópont irányába van, ami azt jelenti, hogy a kapcsolódó szülőcsomópont irányából jön egy bemenet, amin az élen lévő k számú művelet

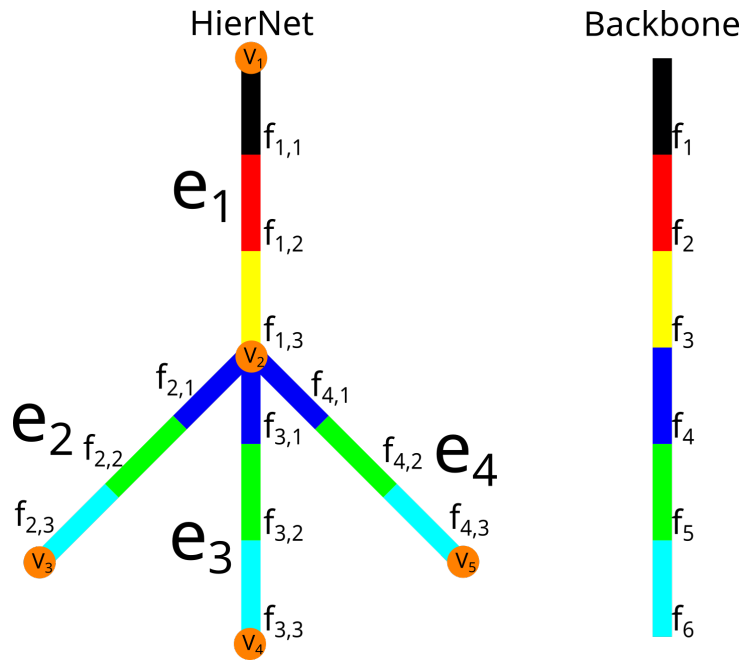
végrehajtódik sorrendben egymás után, tehát a e_i él a $f_{i,k} \circ f_{i,k-1} \circ \dots \circ f_{i,2} \circ f_{i,1}$ függvénnyel tekinthető egyenértékűnek. Az élen végzett első művelet esetén nincs megelőző művelet, amelyből a bemenetet vegyük, az utolsó művelet kimenetét vesszük az előző élről, vagyis arról az élről, amelynek a gyerekcsomópontja a jelenlegi él szülőcsomópontja. Hasonlóképpen, az utolsó művelethez nincs következő művelet, így a kimenet átadásra kerül egyrészt a következő élek (a jelenlegi él gyerekcsomópontjának a gyerekelei) első műveletének, másrészt az osztályozónak a csomópontban. A gyökércsomópont speciális eset, mert nincs előző él, ahonnan a következő élek első műveletei inputot kaphatnak. Tehát a gyökércsomópont esetében a gyerek élek bemenete a modell bemenete, a tényleges kép. Technikailag a jellemzőkivonási művelet lehet bármilyen folytonos, levezethető függvény tetszőleges bemeneti és kimeneti dimenzióval, de tekintettel arra, hogy feladatunk a képosztályozás, a konvolúciós neurális hálózatokban szokásos műveleteket használjuk, mint pl.: Konvolúciós rétegek (meghatározott számú szűrővel, stride-al és padding-el), aktiválások (általában ReLU), Batch Normalization [6], Max Pooling, Residual Blocks [3] stb. Fontos megjegyezni, hogy a művelet kimeneti dimenziójának meg kell egyeznie a a következő művelet bemeneti dimenzió követelményeivel.

Osztályozók Minden v_i csomópont tartalmaz egy c_i osztályozási függvényt. A csomópontok szerepe megegyezik a döntési fa csomópontjaival: eldönteni, hogy kiértékelés során az aktuális csomópontból melyik gyerek élen küldje tovább a mintákat, így a csomópontban található osztályozási funkció ezt a döntéshozatalt valósítja meg. A c_i függvény 3 függvény/művelet összetétele, $c_i(x) = c_{i,3}(c_{i,2}(c_{i,1}(x)))$. A c_i bemenete az előző él (a szülőcsomópont felé eső él) utolsó művelete által kiadott jellemző mátrix (feature map), amely 3 dimenziós (van szélessége, magassága és csatornáinak száma), tehát a $c_{i,1}$ szerepe ennek a dimenzióknak a csökkentése 1-re, amit egy egyszerű kisítmással (flattening) vagy egy Global Average Poolingal lehet elérni. A $c_{i,2}$ egy teljesen kapcsolt réteg, $v_{i_{out}}$ számú kimeneti neuronnal, ahol $v_{i_{out}}$ a v_i gyerekeleinek (vagy gyerekcsomópontjainak) száma. Formálisan: $c_{i,2}(x) = W * x + b$, ahol W a súlymátrix és b eltoló vektor (bias). Ha x $k * 1$ méretű, akkor a W mérete $v_{i_{out}} * k$, b mérete pedig $v_{i_{out}} * 1$. A $c_{i,3}$ egy Softmax aktiválási függvény $(c_{i,3}(z))_j = \frac{e^{z_j}}{\sum_k e^{z_k}}$, amely egy valószínűségi eloszlást ad ki a $v_{i_{out}}$ opciók között, és ezt használjuk a következő útvonal kiválasztásához a legnagyobb valószínűségű útvonal megkeresésével. Legyen $c_i^{e_j}$ az e_j él (vagy v_k csomópont) felé eső irány valószínűsége, ahol $e_{j_{parent}} = v_i$ és $e_{j_{child}} = v_k$.

Topológiai módosítások 2 topológiai módosítást vezettünk be minden modellhez:

1. Egy új csomópontot szúrunk be az adott osztályhierarchia gyökérének szülőjeként, így ez lesz az új gyökér. Ennek a döntésnek az a motivációja, hogy a gyökércsomópont bemenete maga a nyers kép, és valószínűleg nincs értelme a nyers képet kisimítani, majd "softmaxolni" minden előzetes tulajdonság kivonás nélkül. Ezzel a módosítással a fa első tényleges elosztó csomópontja előtt lesznek jellemező kinyerő műveletek, így az adott csomópontban lévő osztályozónak lesz mi alapján eldönteni a szuperosztályt.
2. Mivel a megadott hierarchiafa levelei mind osztályok, ezért nem kell az utolsó szinten lévő élekre műveleteket és a levélcsomópontokban osztályozásokat építeni, mert csak egy lehetséges osztály van. Ezért kivágjuk a fából az összes levélcsomópontot és hozzá vezető éleket, így a végső osztályt az ezelőtti utolsó előtti szinten lévő csomópontok adják. Így ahhoz, hogy a c osztályok továbbra is tudjuk osztályozni, az utolsó előtti szinten lévő csomópontok osztályozóinak kimeneti osztályainak számát meghagyjuk, így az új levelekben a lehetséges kimenetek száma megmarad c -nek.

Modellek mint gerinc Az éleken végzett műveletekhez ugyanazokat a műveleteket használjuk, ugyanabban a sorrendben, amelyek megtalálhatók a szabványos konvolúciós modellekben, mint például a ResNet [3] vagy VGG16 [14]. Ezeket hívjuk a HierNet „gerinc” modelljének. A gerincmodellben jelenlévő minden művelet (vagy réteg) a gyökércsomópont utáni éltől a levelek előtti élekig terjed. Mivel egy él kimenete minden következő él bemenetéhez kapcsolódik, a gyökértől a levélcsomópontig vezető útvonal követésével kapott műveletek megegyeznek a gerincmodell bemenetétől a kimenetig tartó műveletekkel. Ez alól kivételt képeznek a csomópontokban lévő osztályozások, mert a gerincmodellben csak a modell végén van osztályozási funkció, míg a HierNetnél minden csomóponton van osztályozás, és a levélcsomóponton a lehetséges kimeneti osztályok száma sokkal kisebb. Konfigurálható, hogy a gerincmodell műveleteit hogyan szeretnénk felosztani az élek között egy gyökértől levélig tartó útvonalon, de tettünk egy megkötést, hogy minden k szinten lévő élnek ugyanazokkal a műveletekkel kell rendelkeznie. Mindez a következőképpen képzelhető el: ha l számú levelünk van fában, az azt jelenti, hogy l számú gerincmodellünk van, de megosztják az első pár műveleteiknek a súlyait, majd csak csoportok osztják meg a következő pár réteg súlyait, majd csak az alcsoportok, míg végül az utolsó pár rétegen már nincs osztás, mind az l -nek individuális súlyai vannak. A 7. ábra szemlélteti a modellünk és a gerincmodell közötti kapcsolatot.



7. ábra. A HierNet és a gerinchálózat közötti kapcsolat. A különböző színek a különböző műveleteket/rétegeket jelzik. A műveletek ugyanazt az architektúrát és sorrendet követik, mint a gerincmodellben. Az azonos szinten lévő élek ugyanazt a műveletsort tartalmazzák.

Predikció folyamata Ha van egy bemeneti képünk, akkor a modellünkkel az osztályozást a következőképpen hajtjuk végre: alkalmazzuk a gyökér után lévő él műveleteit a bemeneti képre; az eredményt a gyökér utáni első csomópontban lévő osztályozási függvénybe tápláljuk; választunk egy alágat az osztályozási függvény kimenete alapján; alkalmazzunk az alág összes műveletét a korábban kapott feature map-en; megadjuk az eredményt az alág osztályozójának, ami visszatér a következő irányt; folytassuk a ciklust, amíg el nem érünk egy levelet; a levélben lévő osztályozó fogja visszaadni az adott bevitt képhez megfelelő osztályt. Egy csomópontban lévő kimeneti osztály száma könnyen átalakítható globális osztályszámmá, mivel ismerjük a levélcsomópontok sorrendjét. Tekintettel arra, hogy csak a gyökértől a levélig tartó műveletek kerülnek kiértékelésre, és a gyökértől a levélig tartó útvonalon végzett műveletek pontosan megegyeznek a gerincmodellével, a kiértékelési időnek hasonlóknak kell lennie a gerincmodellhez, a csomópontoknál hozzáadott osztályozók enyhe kivételével. Az előrejelzés során használt algoritmust az 1. Algoritmus mutatja be.

Az egészet összerakva A 8. ábra a HierNet architektúráját szemlélteti, de lejjebb található egy rövid összefoglalót is a HierNet működéséről:

1. A HierNet felépítése

1. Algorithm HierNet Osztályozó Algoritmus

Require: az *input* egy RGB kép az elvárt bemeneti felbontással

Ensure: *class* fogja tartalmazni a jóslt osztályt

```
current ← root
edge ← currentchildedges[0]
features ← edge(input)                                ▷ edge(x) = fk(fk-1(... f2(f1(x))...)), ahol
f ∈ {fi,j : f ∈ F, ei = edge}
current ← currentchildnodes[0]
class = current(features)                             ▷ current(x) = c(x), ahol c = ci és vi = current
while current ∉ Vleaves do
    edge ← currentchildedges[class]
    features ← edge(input)
    current ← currentchildnodes[class]
    class = current(features)
end while
class ← getGlobalClassPrediction(class, current)        ▷ Ez a függvény visszaadja a
lokálisan megjósolt szám globális osztályszámát, a jósló levél csomópont és a jóslt osztály
```

- 1.1. A *c* számú osztály közötti hierarchikus relációk faként vannak megadva (pl. a *root* csomópontnak gyerekként *ember*, *állat* és *növény* csomópontja van, míg *állat* csomópontban van *kutya* és *macska*).
- 1.2. Egy új csomópontot hozunk be, szülőként hozzáfűzve az eredeti gyökércsomóponthoz. (Ez azért szükséges, hogy az első felosztás előtt legyenek jellemző kinyerési műveletek)
- 1.3. Az összes levélet levágjuk a fából. (Redundáns)
- 1.4. A HierNet topológiája a kapott fa lesz.
- 1.5. A gerincmodell mellet meg van adva egy konfiguráció, amely leírja a gerinc rétegeinek elosztását a fa különböző szintjei között.
- 1.6. Az éleket feltöltjük műveletekkel (pl. Konvolúció, Batch Normalization, Aktiválás), úgy, hogy minden élszint a gerincmodellől kapja a műveletrit azonos sorrendben a konfiguráció szerint elosztva. Így a sekélyebb szintek megkapják a korábbi műveleteket (közelebb a bemenethez) a gerincből, a mélyebbek pedig a későbbieket.
- 1.7. Az osztályozási függvények a csomópontokban vannak elhelyezve, és a gyerekek számával megegyező számú Softmax kimenettel rendelkeznek. A levél csomópontjaiban lévő kimenetek száma megegyezik az adott csomópont kivágott gyerekcsoomópontjainak számával.
- 1.8. Minden él kimenete rá van kötve az adott élhez kapcsolódó gyerekcsoomópont

gyerekéleinek a bemeneteire, meg a gyerekcsomópontban található osztályozó függvény bemenetére. Az utolsó szinten lévő élek kimenetei csak az osztályozó függvényhez használatosak. A gyökércsomóponttól induló él bemenete magának a modellnek a bemenetét képezi..

2. Tanítás: A modell arra lesz tanítva, hogy a csomópontokban lévő bemeneti képet a fa minden szintjén a megfelelő szuperosztályokba sorolja, végül pedig a megfelelő osztályba egy levélben . A tanítási folyamat leírása a 3.2. szakaszban található.

3. Kategorizálás (vagy jóslás)

3.1. Van egy kategorizálandó kép

3.2. A gyökércsomópont utáni él műveleteit alkalmazzuk rajta, így kapunk egy tulajdonság reprezentációt (feature map)

3.3. A "feature map" a gyökér gyerekcsomópontjának osztályozó függvényébe tápláljuk

3.4. Az osztályozó kimenetétől függően gyerekcsomópontot választunk

3.5. Az új gyerekcsomópont felé tartó élen végzett műveletek alkalmazzuk a korábban kapott feature map-en

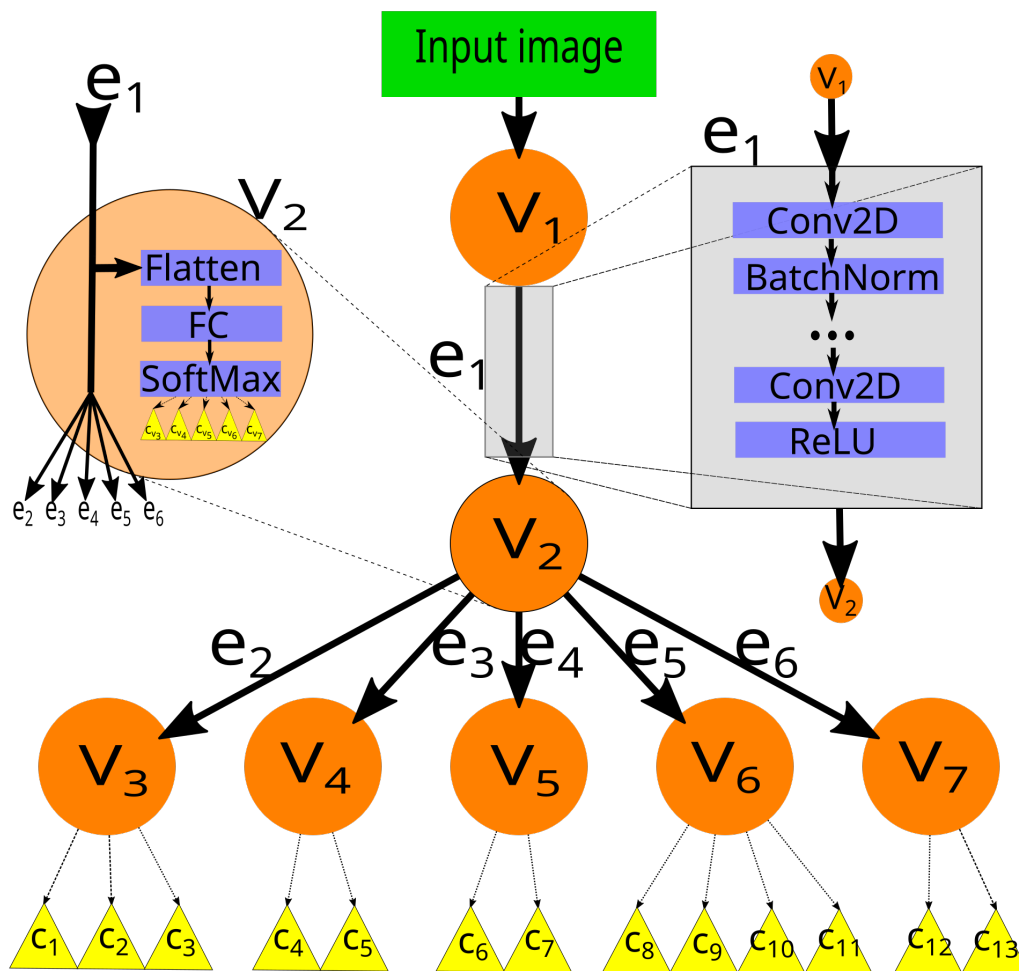
3.6. A ciklus addig ismétlődik, amíg el nem érünk egy levélcsomópontot

3.7. A levél csomópontjában lévő osztályozó kimenete adja a modell végső jóslatát.

3.2. A tanítás folyamata

A modell betanításával kapcsolatban a célunk egy olyan egyszerű tanítási módszer létrehozása volt, amely "end-to-end" tanítható, tehát a HierNet-et olyan módon tudjuk tanítani, ahogy a gerincmodellt. Vagyis ha van egy bemenetünk egy képpel, egy kimenetünk c osztállyal, akkor egy veszteségfüggvény (loss function) alkalmazható legyen erre a kimenetre, amely szerint egyszerűen "backpropagation"-el és gradiensereszkedéssel tudjuk a modellt optimalizálni.

Ennek az „end-to-end” jellegnek az eléréséhez a következőket tesszük: A levél csomópontok mindegyik kimeneti valószínűségét megszorozzuk az adott levélhez vezető őscsomópontok az odavezető kimeneti valószínűségeivel. Formálisan legyen R_{v_i} a *root* és v_i közötti útvonal éleinek halmaza, és o_i a v_i osztályozójának új megszorozott kimenete .Ebben az esetben



8. ábra. A HierNet alap architektúrája. Az élek ugyanazokat a műveleteket tartalmazzák, mint a gerinc. A csomópontokban az előző él kimenete kapcsolódik a következő élekhez, és az osztályozó függvényhez, amely a predikció során a következő csomópontot adja meg.

$o_i = c_i * \prod_{j \in R, k \in \{k: v_k = e_{j_{\text{parent}}}\}} c_k^{e_j}$. Ezután a levélcsonópontok új osztályozási kimeneteit (o_i) összefűzzük balról jobbra, és mivel csak egy útvonal felel meg egy kimeneti osztálynak, összesen c számú osztályt kapunk az összefűzésből, és így megkaptuk a kimenetünket. Szeretnénk felhívni a figyelmet arra, hogy az összes kiírt valószínűség összege 1. Így amikor a modellt gradiensereskedéssel tanítjuk, a levelekben lévő osztályozókhoz vannak gradiensek, mivel ezek összefűzve kimenet, de a belső csomópontok mindegyik osztályozójához is, mivel a kimenetet az ezek által kiadott valószínűségekből is épül fel. Magától érthetődő, hogy ha a csomópontok osztályozóihoz vannak gradiensek, akkor az élekben végzett műveletekhez is vannak, mivel az osztályozó a szülő éltől kapja a bemenetét.

Ezek a különböző szintekről összeszorozott valószínűségek felfoghatók feltételes valószínűségekként. Vegyünk egy 2 szintű topológiát, A gyökércsomóponttal, B_i gyerekcsomópontokkal és $C_{i,j}$ osztályokkal, amelyeket a B_i csomópont osztályozója ad ki. Legyen A^{B_i} az az esemény, hogy az A csomópontban lévő osztályozó a B_i felé irányít, a $B_i^{C_{i,j}}$ az az esemény, hogy a B_i a $C_{i,j}$ felé irányít és $O^{C_{i,j}}$ az az esemény, hogy a modell a $C_{i,j}$ osztályt adja ki. Bayes tétele szerint: $P(A|B) = \frac{P(B|A)*P(A)}{P(B)}$. Ha behelyettesítjük eseményeinket $P(O^{C_{i,j}}|A^{B_i}) = \frac{P(A^{B_i}|O^{C_{i,j}})*P(O^{C_{i,j}})}{P(A^{B_i})}$. Az egyenletet átrendezve a következőt kapjuk: $P(O^{C_{i,j}}) = \frac{P(O^{C_{i,j}}|A^{B_i})*P(A^{B_i})}{P(A^{B_i}|O^{C_{i,j}})}$. $P(A^{B_i}|O^{C_{i,j}}) = 1$, mert ha a $C_{i,j}$ ki lesz választva, akkor biztosan B_i is ki lett kiválasztva az A -ban. Továbbá $P(O^{C_{i,j}}|A^{B_i}) = P(B_i^{C_{i,j}})$, mert ha a B_i már ki van választva az A -ban, akkor a valószínűsége, hogy $C_{i,j}$ is ki legyen, csak a B_i osztályozótól függ. Tehát $P(O^{C_{i,j}}) = P(A^{B_i}) * P(B_i^{C_{i,j}})$, ami azt jelenti, hogy egy osztály kiválasztásának valószínűsége a prediktor csomópontban való kiválasztódás valószínűsége szorozva annak valószínűségével, hogy a prediktor csomópontot kiválasztja a szülője. Ez általánosítható több szintre is.

Ezzel a megközelítéssel minden modellnél két pontosságot tudunk közölni: egy "feltételes pontosságot" és egy "útválasztási pontosságot". Előbbit a tanításhoz használt kimenetből számítjuk ki (az összefűzött o_j -s), míg az utóbbit a csomópontok osztályozó kimenetei alapján a gyökértől egy levélig vezető útvonal követésével (a 3.1. szekcióban vagy az 1. algortimussal bemutatott módszer).

Fontos megjegyezni, hogy a kiadott osztályok sorrendje a megadott hierarchikus fától függ, ezért minden modellnél szükséges a címkék átrendezése az adathalmazban, hogy egy címkének

a “one-hot” reprezentációjával megadott osztálya illeszkedjen a kívánt helyhez a hierarchiában.

A kategorikus keresztentrópia veszteségfüggvényt használunk a HierNet betanításához, ami a következőképpen definiálható: $L = \sum_{i=1}^c y_i * \log(p(y_i|x))$, ahol y_i a valódi valószínűsége egy adott c_i osztálynak, míg a $p(y_i|x)$ a modellünk által prediktált valószínűség. Ennek a veszteségfüggvénynek az a nagy előnye, hogy egy i valós osztály a modellben csak a hozzá megfelelő kimenetéhez hoz létre gradienst, így csak az i -ik osztályhoz vezető éleket és az osztályozókat módosítja a tanítás során, a hozzá nem tartozó ágak módosítása nélkül.

Bizonyítás. Legyen o_j a modell kimenete a j -edik osztályhoz, ahol az osztályok teljes száma c .

$$L = \sum_{i=1}^c y_i * \log(p(y_i|x)) = \sum_{i=1}^c y_i * \log(o_j) \text{ a veszteségfüggvény}$$

$$\frac{\partial L}{\partial o_j} = \frac{y_j}{o_j}, \text{ mert amikor } i \neq j, \text{ akkor a kifejezések mindegyike állandónak számít}$$

Tekintettel arra, hogy y one-hot kódolásban van, $y_j = 0$, ha a j nem az igazi osztály ($j \neq y_{true}$), így a gradiensek a következők lesznek:

$$\frac{\partial L}{\partial o_j} = 0, \text{ amikor } j \neq y_{true}$$

$$\frac{\partial L}{\partial o_j} = \frac{y_j}{o_j}, \text{ ha } j = y_{true}$$

Ez bizonyítja, hogy nincs gradiens o_j -ra, ha a j nem az igazi osztály.

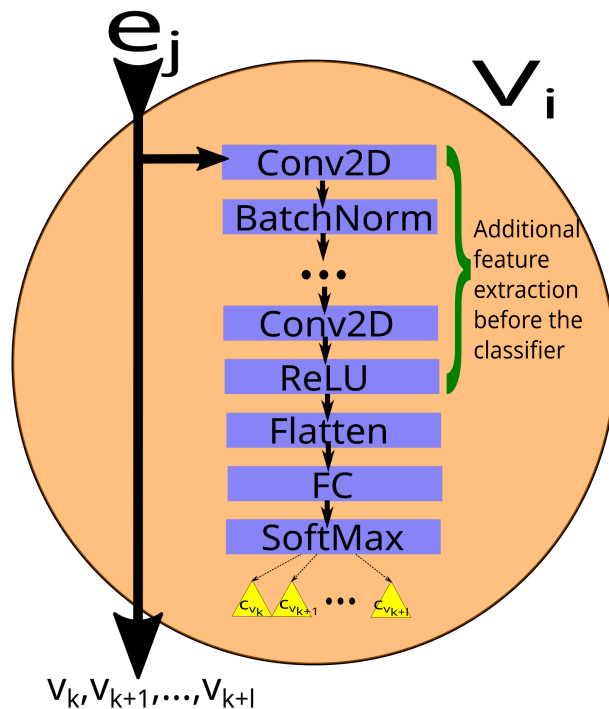
□

Ami a tanítási algoritmust, a tanulási rátát vagy más hasonló hiperparamétereket illeti, általában ugyanazt a konfigurációt használjuk, mint az alapmodell.

3.3. Hozzáadott rétegek a kategorizálás előtt

Most bemutatunk egy feljavítást a csomópontok architektúrájában a pontosságuk növelése érdekében: Hozzáadhatunk néhány további réteget az élről érkező bemenet és az osztályozási függvény közé. Ennek a módosításnak az illusztrációja a 9. ábrán látható.

Ennek az a motivációja, hogy a szuperosztályokba való kategorizáláshoz további különböző jellemzők kinyerésére lehet szükség. Korábban az osztályozó által a csomópontokban használt “feature map” ugyanaz volt, mint a következő rétegnek átadott. Ezzel a módosítással azonban további konvolúciók is vannak a végrehajtva, amelyek eredményét csak az osztályozáshoz használjuk, így ez kinyerhet az adott csomópont által kategorizált szuperosztályokra jellemző tulajdonságokat.



9. ábra. A csomópont enyhe módosítása: További jellemző kivonó rétegek az osztályozók előtt a csomópont pontosságának növelése érdekében

Az élről érkező bemenet után hozzáadott további rétegek is követik a gerincmodell architektúráját. Tekintettel arra, hogy az előző él utolsó művelete a gerincmodell egy rétegének felel meg, a csomópontban lévő néhány k hozzáadott réteg megfelel a gerincmodell következő k rétegeinek.

A levél csomópontok kivételt képeznek, mert nem tudunk hozzájuk további rétegeket hozzárendelni. Ennek az az oka, hogy a gyökértől a levélig tartó útvonal architektúrája megegyezik a gerinchálózat architektúrájával, így a levél csomópontok osztályozóinak bemenete a gerincmodell utolsó rétegének kimenete, ezért nincs fennmaradó réteg az utolsó után, amit hozzá tudnánk fűzni.

Ennek a megközelítésnek a hátránya a kissé megnövelt modellméret. Korábban a predikció során a gyökértől a levélig tartó útvonal kiértékelési komplexitása hasonló volt a gerincmodelléhez, a csomópontokban lévő osztályozók értékelésének egy kis különbségével. Ezzel a módosítással ez a különbség növekszik a hozzáadott műveletek kiértékelésével az érintett csomópontokban, ezért kulcsfontosságú, hogy ne adjunk az osztályozókhoz hozzá túl sok réteget.

3.4. Előtanítás

Egy másik fejlesztés a modellen a gyökér utáni első csomópont előtanítása (pre-training). Ez az a csomópont, ahol az első tényleges osztályozás megtörténik, a megadott hierarchiafa gyökere (lásd a 3.1. szakaszt). Ezért ennek a csomópontnak az előzetes betanítása már elég nagy pontosságot fog eredményezni annak az első osztályozónak, amikor a fő tanítás elkezdődik, és a csomópont előtti minden réteg már megtanulja akkorra néhány jelentősebb jellemző kivonását, amely hasznos lehet a következő rétegek tanításához

Az előtanítás során a kategorikus keresztentropia veszteség függvényt alkalmazzuk az osztályozó kimenetére abban az első csomópontban, és a tanulási adathalmaz egyedeit ellátjuk a szuperosztályaik címkéivel (a tanított pontban a gyerek élek közül melyik vezet ahhoz a levélhez, amely az adott osztályt megjósolja.). Bár az egyszerűség kedvéért ugyanazt a tanulási rátát és epoch számot használjuk, mint a fő tanításnál, elismerjük, hogy potenciálisan az előtanítás ideje lecsökkenthető más tanítási beállításokkal hasonló teljesítményt mellet.

A kísérleteinkben bemutatjuk a különbséget aközött, ha a hálózat előre tantított részének a súlyait a főtanítás idejére lefagyaszttjuk, meg ha nem fagyaszttjuk le, tehát hagyjuk tovább hangolni.

3.5. Transfer Learning

Modellünk utolsó feljavítása a “Transfer Learning” bevezetése. Mivel már rendelkezünk a gerincmodell egy betanított példányával, a súlyai újra felhasználhatók a HierNet számára. A motiváció az, hogy ugyanaz a feladat, ugyanazokkal az osztályokkal, ugyanazon az adathalmazon és a gerinc által tanult információk hasznosak lehetnek azáltal, hogy jobb kiindulópontot biztosítanak a modellünk tanításunkhoz, mint a nulláról való tanítás véletlenszerűen inicializált súlyokkal.

Tekintettel arra, hogy a modellünk pontosan ugyanazokat a rétegeket használja, mint a gerincmodell, a súlyok átvitele könnyen megvalósítható, ha minden réteg súlyait minden élen (vagy csomóponton, ha hozzáadott rétegeket is használunk) a gerincen megfelelő rétegnek a már betanított súlyaira állítjuk. Például a 7. ábrán látható HierNet esetében a gerinchálózatban lévő rétegek súlyait a modellünkben a minden azonos színnel rendelkező réteghez másoljuk.

4. Osztályhierarchia felépítés

Modellünk topológiája (a 3.1. szakaszban leírtak szerint) egy fából épül fel, amely az adathalmaz c osztálya közötti hierarchiát képviseli. A fa levelei a c osztályok, a belső csomópontok pedig a szuperosztályokat vagy csoportokat képviselik. Az elképzelés az, hogy egy ideális hierarchikus fában minél közelebb van 2 osztály legalacsonyabb közös őse, annál rokonabbak az osztályok, vagyis annál kisebbek a vizuális különbségek az ehhez a 2 osztályhoz tartozó képeknél.

Egy ilyen fa manuálisan felállítható kevés számú osztály esetében (pl. 10), ezért a CIFAR10 (részletek a 5.1. szakaszban) adathalmazhoz egy manuálisan felépítettet használunk. Ennek az adathalmaznak az osztályainak egy példahierarchiája a 3. táblázatban található, és ennek a csoportosításnak az okai a 7.1. szakaszban vannak részletezve.

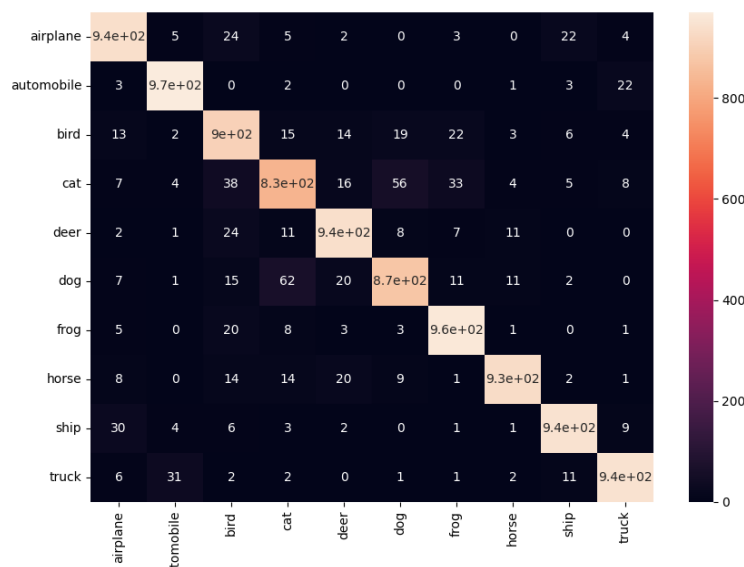
Sokkal több osztállyal rendelkező adathalmazok esetében viszont a manuális felépítés nehézkessé és időigényessé válik, ezért bemutatunk egy algoritmust, amely egy konfúziós valószínűségi mátrixot használ a c osztályok g csoportokba való csoportosítására, és ezekből a csoportosításokból szerkeszthető egy hierarchikus fa.

4.1. Konfúziós valószínűségi mátrix

Ahhoz, hogy csoportosíthassuk őket, rendelkezünk kell valamilyen módon az osztályok közötti vizuális hasonlósági kapcsolatokkal.

A konfúziós mátrix egy c sort és c oszlopot tartalmazó mátrix, ahol az oszlopok a modell által prediktált értékeket, a sorok pedig a valódi értékeket reprezentálják. Tehát a $CM[i][j] = k$ azt jelenti, hogy a modell k -szor prediktálta a képnek a j osztályt, ahol a i a kép valódi osztálya. Ezeket a mátrixokat általában a validációs halmaz kiértékelése alapján építik fel úgy, így megmutatják, hogy az osztályok közül melyiket sorolják gyakran másik osztályba. Egy példa a konfúziós mátrixra a 10. ábrán látható.

A konfúziós valószínűségi mátrix hasonló az konfúziós mátrixhoz, azzal a különbséggel, hogy az igaz címkének megfelelő sorban nem a megjósolt egy osztályok halmozódnak fel, hanem az adott sor osztályának képeire prediktált valószínűségek. Ekkor minden sort



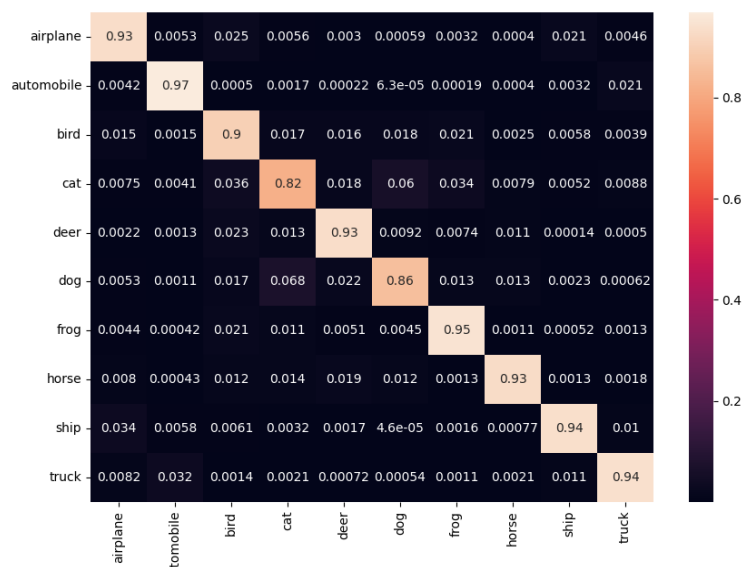
10. ábra. A ResNet ($n = 5$) gerincmodellen és a CIFAR10 validációshalmazán kiértékelve az adathalmaz 10 osztályának konfúziós matrixa

elosztunk az adott igaz címkéhez tartozó példák számával, így kapunk egy sort az átlagos valószínűségekből, amelyek azt az esélyt jelentik, hogy a sorhoz tartozó osztály valamilyen más osztálynak lesz besorolva. Tehát: $CPM[i][j] = p$ annak valószínűségét jelenti, hogy egy i osztályú képet j osztálynak prediktál a modell. Egy példa látható a 11. ábrán.

Vizuális hasonlóságok méréseként a betanított gerincmodell validálási halmazának kiértékeléséből összeállított konfúziós valószínűségi mátrixot használjuk, azzal a motivációval, hogy ha a i és j osztályok összetévesztésének valószínűsége nagy, akkor ezek vizuálisan jobban hasonlíthatnak, mint abban az esetben, amikor alacsony. Úgy gondoljuk, hogy a konfúziós valószínűségi mátrix jobb, mint az egyszerű konfúziós mátrix, mert az utóbbihoz egy precíziós eredményének hozzáadásakor csak a legnagyobb valószínűségű osztályt vesszük figyelembe, és valószínűleg értékes információkat veszítünk a többi osztályok közötti valószínűség-eloszlásról.

4.2. Csoportosítási algoritmus

Most bemutatunk egy algoritmust c számú osztály g csoportokba való csoportosítására a 4.1. szakaszban bemutatott konfúziós valószínűségi mátrix alapján. Bár ez az algoritmus csak osztálycsoportokat hoz létre és így 2-es mélységű hierarchikus fát eredményez (0. szint: gyökér; 1. szint: osztálycsoportokat reprezentáló csomópontok, 2. szint: osztályokat



11. ábra. A ResNet ($n = 5$) gerincmodellen és a CIFAR10 validációshalmazán kiértékelve az adathalmaz 10 osztályának konfúziós valószínűség matrixa

reprezentáló csomópontok), alkalmazva az algoritmust a létrehozott csoportokon rekurzívan létrehozhatunk mélyebb fákat is.

Két c_1 , c_2 osztály közelségét a következőképpen határozzuk meg: $prox(c_1, c_2) = CPM[c_1][c_2] + CPM[c_2][c_1]$ (az oda- és visszairányú összetévesztés valószínűsége), két csoport közelsége pedig: $prox(G_1, G_2) = \frac{1}{|T|} \sum_{(c_1, c_2) \in T} (prox(c_1, c_2))$, ahol $T = G_1 \times G_2 = \{(c_1, c_2) : c_1 \in G_1, c_2 \in G_2\}$ (A két csoport összes osztálykombinációjának a közelségének az összege)

A csoportosítás úgy működik, hogy először minden osztályt külön csoportnak tekintünk, majd minden iterációnál összevonjuk a legnagyobb közelségben lévő 2 csoportot, amíg nem lesz több összevonható csoportpár. Tekintettel arra, hogy ez az algoritmus önmagában minden osztályt egyetlen nagy csoportba vonna össze, két megkötést alkalmazunk két csoport egyesítésére:

- A két csoport együttes mérete nem haladhat meg egy bizonyos értéket.
- A két csoport közelségének meg kell haladnia egy bizonyos értéket.

Azt feltételezzük, hogy az első feltételre azért van szükség, hogy elkerüljük a túl nagy csoportok létrejöttét, míg a többi csoport kardinalitása alacsony, míg a második feltétel azért

szükséges, hogy csak olyan osztályokat csoportosítsunk össze, amelyek elég közel vannak egymáshoz. Ezzel az algoritmussal és a két beállításának a változtatásával egy konfúziós valószínűségi mátrixból sok különböző csoportosítást generálhatunk azzal a tulajdonsággal, hogy egy csoport osztályai általában közelebb lesznek egymáshoz, mint a többi csoport osztályaihoz. Átlagosan a csoportok száma egyenesen arányos a közelségi határértékkel, és fordítottan arányos a csoportlétszám határértékével.

A CIFAR100 adathalmaz különböző csoportosítási beállításai a kísérleteknél, a 7.3. szakaszban találhatók.

5. Felhasznált adathalmazok

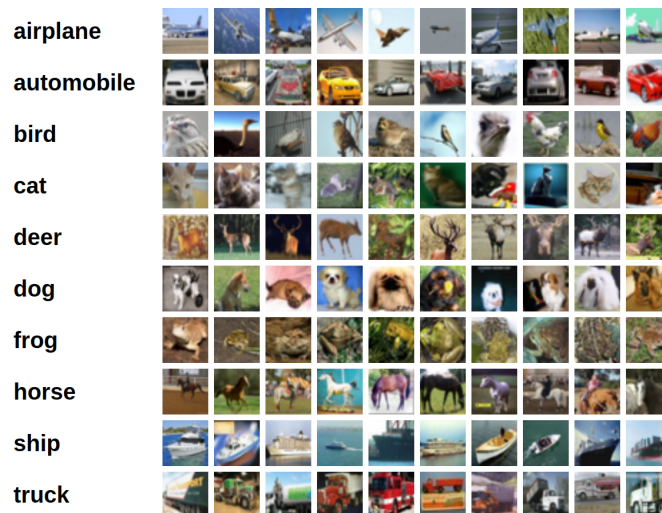
Ahhoz, hogy modelljeink teljesítményét összehasonlítsuk az alapmodellek teljesítményével, 3 különböző adathalmazt használunk: CIFAR10, CIFAR100 és Tiny ImageNet.

5.1. CIFAR10

A CIFAR10 [9] adathalmaz a Tiny Images adathalmaz egy részhalmaza, amely csaknem 80 millió 32×32 -es felbontású RGB képet tartalmaz, melyeket az internetről gyűjtöttek össze 75062 kulcsszó használatával olyan népszerű keresőmotorokban, mint a Google, Flickr és mások. A CIFAR10 a fent említett szülőadathalmaz képeinek manuális felcímkézésével készült, 4 kritériumot követve:

1. Minden képnek fotórealisztikusnak kell lennie.
2. A kép címkéjének tökéletesen meg kell egyeznie a kép tartalmával.
3. A címkével leírt objektumnak csak egy példánya jelenhet meg a képen.
4. Az objektumnak elég láthatónak kell lennie ahhoz, hogy az azonosítása egyértelmű legyen.

Összesen 60000 kép van kiválasztva, majd 10 különböző osztályba osztva, olyan módon, hogy mindegyik osztálynak pontosan 6000 megfelelő minta legyen. A 10 osztály a következő: repülőgép, autó, madár, macska, szarvas, kutya, béka, ló, hajó és teherautó. A 10 osztály mindegyikére találhatóak példák a 12. ábrán. A CIFAR10 képei megtartották az eredeti adathalmazból származó 32×32 -es felbontásukat és RGB színmodelljüket.



12. ábra. Példák a CIFAR10 adathalmazból. (Forrás: <https://www.cs.toronto.edu/~kriz/cifar.html>)

Az eredeti szerzők a 60000 képet felosztották 50000 tanítási és 10000 tesztképre, ami 5000 képzési képet és 1000 tesztképet jelent osztályonként. Mivel nincs konkrét validációs halmaz, ugyanazt a gyakorlatot követjük, mint a ResNet [3] szerzői, azaz modelljeinket a tanítási halmaz alapján tanítjuk, és a teszhalmaz segítségével értékeljük ki a kísérletek pontosságát.

Az adathalmaz forrása: <https://www.cs.toronto.edu/~kriz/cifar-10-binary.tar.gz>

5.2. CIFAR100

A CIFAR100 [9] adathalmaznak ugyanazok a szerzői, mint a CIFAR10 adathalmaznak, és néhány alapvető szempontból hasonló: a Tiny Images adathalmaz egy részhalmaza és ugyanazon kritériumok alapján készült; 60000 32×32 méretű RGB képet tartalmaz, 50000 és 10000 képre osztva tanítás és tesztelés céljából. A különbség az, hogy 100 osztálya van és mindegyik osztályban 600 kép található.

A CIFAR10 adathalmaznál alkalmazott gyakorlattal szemben ennél az adathalmaznál nem csak a teszhalmazt használjuk az értékeléshez. Ehelyett továbbosztjuk az 10000 példát tartalmazó eredeti teszhalmazt egy validációsra és egy tesztelésre, amelyek mindegyike 5000 képet tartalmaz. A felosztás véletlenszerűen történik, de azon szigorú szabály szerint, hogy mindkét részhalmaznak azonos mennyiségű képet kell tartalmaznia a 100 osztály mindegyikéből. Ennek a döntésnek a háttérében az áll, hogy a CIFAR100 adathalmaz

osztályozására használt HierNet modellek architektúrája automatikusan, a validációs adathalmaz az alapmodellen való kiértékeléséből származó konfúziós valószínűség mátrix alapján épül fel. Ha csak egy teszhalmazt használtunk volna, a teszhalmazon számolt konfúziós valószínűség mátrix alapján felépített hierarchia kvázi információt szivárogtatott volna ki a teszhalmazból a tanítás számára, így az eredmények kevésbé lennének érvényesek. Ez a probléma kiküszöbölhető, ha külön validációs halmazt használunk: bár az validációs halmazból lehet információ kiszivárog, a teszhalmaz independens információhalmaz marad, így nem befolyásolja a képzést.

Az adathalmaz forrása: <https://www.cs.toronto.edu/~kriz/cifar-100-binary.tar.gz>

5.3. Tiny ImageNet

Az ImageNet [13] egy népszerű adathalmaz, több mint 14 millió, az internetről gyűjtött, kézzel címkézett képet tartalmaz. Ennek az adathalmaznak egy részhalmazát az ImageNet Large Scale Visual Recognition Challenge (ILSVRC) használja. Ez egy verseny volt, mely 2010 és 2017 között évente került megrendezésre azzal a szándékkal, hogy támogassák az objektumészlelési és képosztályozási algoritmusok kutatását. Ebben az adathalmazban körülbelül 1,2 millió tanítási, 50 ezer validációs és 100 ezer tesztelési kép található 1000 osztályra osztva. A képek felbontása adathalmazenként változó, de sok modell gyakran 256×256 -ra vagy 224×224 -re méretezi át őket.

A Tiny ImageNet az ILSVRC adathalmaz egy részhalmaza, amely három módon is csökkenti az eredeti halmaz méretét:

1. 1000 helyett 200 különböző osztályt tartalmaz.
2. A képek $64 \times 64 \times 3$ méretűre lettek átméretezve.
3. Az minták száma lecsökken: a tanítási adathalmaz 100000 képet tartalmaz, a validációs és a teszhalmaz meg egyenként 10000 mintát tartalmaznak. A képek egyenletesen oszlanak el a 200 osztály között.

Az ILSRC adathalmaz helyett a Tiny ImageNet-et használjuk, főleg a számítási erőforrások hiánya miatt. Tekintettel arra, hogy az eredetiben 12-szer több tanítási kép van és minden kép sokkal nagyobb felbontású, ez sokkal hosszabb betanítási időt igényelne, ami meghaladja a korlátainkat. A Tiny ImageNet egy másik kisebb előnye, hogy osztályonként egyenlő mennyiségű kép található, ami kiküszöböli az osztálykiegyensúlyozatlanság problémáját.

Bár az adathalmaz külön felosztást biztosít validációs és teszhalmazra, a tesztképek nincsenek címkékkel ellátva, ezért ugyanazt a gyakorlatot követjük, mint a CIFAR100 esetén: az eredeti teszhalmazt 5000 validációs és 5000 tesztképre bontjuk, tiszteletben tartva, hogy egyenlő mennyiségű képek kerüljenek osztályonként a 2 felosztásba.

Az adathalmaz forrása: <http://cs231n.stanford.edu/tiny-imagenet-200.zip>

6. Alap- vagy gerincmodellek

Annak érdekében, hogy modelljeink teljesítményét összehasonlíthassuk a mögöttes gerincmodellek teljesítményével, a gerincként használt modelleket újra implementáltuk a CIFAR10, CIFAR100 és a Tiny ImageNet adathalmazokra.

6.1. ResNet a CIFAR10-en

A ResNet implementációnk a CIFAR10 adathalmazra nagyjából megegyezik a ResNet cikkben [3] leírtakkal.

A bemeneti (32x32) réteget egy konvolúciós réteg követi 16 szűrővel, 3×3 konvolúciókkal, *same* padding-el és 1-es stride-al. Ezt követi a Batch Normalization [6], majd a ReLU aktiváció. Ezt követően $3 * n$ Residual Block következik egymás után. Az első n blokk 16 szűrőt tartalmaz, a következő n blokk 32-t és az utolsó n blokk 64-t. Az első, a második és a harmadik n blokkban a “feature map” mérete rendre 32×32 , 16×16 , 8×8 . A hálózat utolsó néhány rétege: “global average pooling”, majd egy teljesen kapcsolt réteg 10 neuronnal és végül egy softmax aktiváció. Minden Residual Block 2 konvolúciós réteget tartalmaz 3×3 konvolúcióval, *same* padding-el és 1 stride-al. Minden konvolúciós réteg után van egy Batch Normalization[6], és vannak ReLU aktiválások az első konvolúció után, meg a “shortcut connection” után. A dimenziócsökkentéshez 2-es stride-ot használunk minden

(kivéve az első) halmaz Residual Block első konvolúciós rétegében. Ha a blokk kimenetének mérete és a “shortcut connection” mérete nem egyezik, 1×1 konvolúciót használunk a “shortuct connection”-on. Az $n = \{3, 5, 7, 9, 18\}$ méretű modelleket teszteltük.

A kategorikus keresztentrópiát használjuk veszteségfüggvényként, és az optimalizálást a Sztochasztikus Gradiensereszkedéssel (SGD) hajtjuk végre 0.9-es momentummal, 200 epoch-ig, 128-as batch mérettel. A tanulási ráta 0,1-ről kezdődik, majd elosztjuk 10-el a 32 000-ik iterációnál (~ 82 -es korszak) és a 48 000-ik iterációnál (~ 123 -as korszak). 0,0001-es weight decay-t használunk és ezt is a tanulási rátával arányosan csökkentjük. Az $n = 18$ kivétel, mert ott 2000 iterációnyit (~ 5 epoch) tanítunt az elején 0.01-es tanulási rátával, bemelegítésként.

A bemeneti képek előfeldolgozási lépései:

1. A képeket 255-tel osztjuk, hogy a pixelértékeket $(0, 255)$ -ről $(0, 1)$ -re képezzük le.
2. Kiszámolunk egy átlagot a tanítási képek minden pixelére és ezeket az átlagokat minden képből kivonjuk.
3. A következő 2 adat augmentációt alkalmazzuk a tanítási képeken:
 - Véletlenszerű vízszintes tükrözés (random flip)
 - Véletlenszerű eltolás 0,125-ös tényezővel mind a vízszintes, mind a függőleges tengelyen, a képen kívüli pixeleket szürkével kitöltve.
4. A címkéket one-hot enkódolásra alakítjuk

Ahogy az 5.1. szakaszban már említettük, ugyanazt a gyakorlatot követjük, mint az eredeti szerzők: a teljes 50 000-es képhalmazt használjuk a tanításra, és a fennmaradó 10 000 képet használjuk a teljesítmény (vagy pontosság) megméréséhez.

A 1. táblázat felsorolja ennek az alapmodellnek a tanítási és validációs pontosságát a különböző n méretekre. A mi implementációink pontossága közel áll az eredeti szerzők [3] által jelentett pontosságokhoz.

n	#rétegek	Tanítási pontosság	Validációs pontosság	Eredeti validációs pontosság [3]
3	20	99.06	91.57	91.25
5	32	99.73	92.22	92.49
7	44	99.83	92.57	92.83
9	56	99.88	92.99	93.03
18	110	99.97	93.38	93.57

1. táblázat. Az általunk leimplementált ResNet különböző n méretekkel elért pontossága a CIFAR10 adathalmazon. Az utolsó oszlop az eredeti szerzők által elért pontosságot mutatja.

n	#rétegek	Tanítási pontosság	Validációs pontosság	Tesztelési pontosság
3	20	86.53	67.07	65.96
5	32	94.23	67.75	67.08
7	44	97.04	68.69	68.12
9	56	98.37	68.67	68.38
18	110	99.69	71.58	71.33

2. táblázat. A tanítási, validációs és tesztelési pontossága a ResNet modellünknek a CIFAR100 adathalmazon, különböző n méretekkel

6.2. ResNet a CIFAR100-on

Mivel a CIFAR100 adathalmazhoz nem létezik hivatalos ResNet implementáció, pontosan ugyanazt az architektúrát, tanítási beállításokat és kép-előfeldolgozási lépéseket használjuk, mint a CIFAR10 adathalmaznál. Az egyetlen különbség az architektúrát tekintve természetesen a kimeneti neuronok száma az utolsó teljesen kapcsolt rétegben, ami ebben az esetben 10 helyett 100.

A CIFAR100 esetében eltérő validációs- és tesztadathalmazunk van (a 5.2. szekcióban felvázolt okok miatt), így tudjuk jelteni az alapmodell pontosságát a tesztadathalmazon, és összehasonlíthatjuk a végső HierNet modell tesztpontosságaival (végső modell = a legmagasabb validációs pontossággal rendelkező).

A 2. táblázat a 3 különböző adathalmazon való pontosságot mutatja különböző modell nagyságra.

6.3. ResNet a Tiny ImageNet-en

A ResNet[3] felépítésénél a Tiny ImageNet adathalmaz számára többnyire a [15] cikket követtük.

Ami a modell architektúráját illeti, ugyanazokat a konfigurációkat használjuk, mint a 6. ábrán, csak néhány kisebb eltéréssel. Az első különbség az, hogy az első 7×7 konvolúció és a max pooling helyett egy 3×3 konvolúciót használunk 32 szűrővel, *same* padding-el és 1 stride-al, majd ezt követi egy Batch Normalization és egy ReLU aktiválás. A második különbség természetesen az utolsó teljesen kapcsolt rétegben a 200-ra csökkentett kimenetek száma. A harmadik különbség a “residual” blokkok megváltozott architektúrája: A műveletek sorrendjét átszerveztük a [4]-ben található "full pre-activation" sorrend szerint. Tulajdonképpen az új kiosztásban egy Residual Block egy Batch Normalization-el kezdődik, majd jön egy ReLU aktiválás és csak utána a konvolúciós réteg és ez a minta ismétlődik a "shortcut connection"-ig.

Ezenkívül a [5]-ben bevezetett sztochasztikus mélységet (Stochastic Depth) használjuk a “residual” blokkokhoz, ahol az utolsó blokknál a kiesési arányt 0,5-re állítottuk.

A használt veszteségfüggvény a kategorikus keresztentrópia, és SGD-t használunk ennek az optimalizáláshoz 0,9-es momentummal 100 epochig. Egy batch méretét 64-re csökkentjük, hogy a modell beleférjen a GPU memóriájába. Egy 4000 iterációs, 0,01-es tanítási rátával végrehajtott bemelegítő tanítás után a tanítási ráta 0,1-ről indul, míg a “weight decay” 0,0001-ről és mindkettő 10-es faktoral csökken az 68000-es és 100000-es iterációknál.

Az adataugmentációval kapcsolatban a [15] szerzői nem pontosították, hogy milyen módszereket használtak, ezért ugyanazokat fogjuk használni, mint a többi modellenél: véletlenszerű vízszintes tükrözést és véletlenszerű eltolást 0,125-ös faktoral. Az adat-előfeldolgozás további lépései a következők: a pixelértékek leképezése (0,1) közé, a pixelenkénti átlag kivonása és az osztálycímkek átalakítása one-hot enkódolásra.

Gerincként a HierNet-hez azt a változatot használtuk, ahol $n = 18$. A pontosságok az alábbiakban láthatóak:

- Tanítási pontosság: 67.07%
- Validációs pontosság: **60.02%**
- Tesztelési pontosság: **58.21%**

6.4. VGG16 a CIFAR100-on

A CIFAR100 adathalmazhoz nincs hivatalos VGG16[14] implementáció, de nagyjából ugyanazt az architektúrát használjuk, mint amit az ImageNet [13] osztályozásához használnak, csak csökkentett méretekkel és néhány apró módosítással.

Gerincmodellünk architektúrájaként az eredeti szerzők által bevezetett "D" konfigurációt használtuk. Ennek a konfigurációnak 5 csoportnyi konvolúciós rétege van egymás után. Az elsőnek 2 konvolúciós rétege van, mindegyik 64 szűrővel, a másodiknak szintén 2, de 128 szűrővel, a harmadik 3 darab 256 szűrős konvolúcióval rendelkezik és negyedik meg az utolsó 3 konvolúciós réteggel, mindegyik 512 szűrővel. Minden konvolúciós réteg 3×3 kernelmérettel, *same* padding-el, 1 stride-al és ReLU aktiválással rendelkezik. Minden csoport után van egy Max Pool réteg, hogy csökkentse a "feature map" méreteit. Így a méret az 5 csoportban egyenként 32,16,8,4 és 2 lesz. A súlyok inicializálására a [1]-ben bemutatott "Xavier uniform" inicializálót használjuk. Az eltolásokat (biases) 0-val inicializáltuk.

Kicsit módosítottuk a hálózat néhány utolsó rétegét. A simító (flattening) réteg után csak egy hozzáadott, teljesen kapcsolt réteget használunk 512 neuronnal (szemben az eredetivel, amely 2 teljesen kapcsolt réteget tartalmazott, mindegyik 4096 neuronnal). Ezután következik egy 100 neuronból álló réteg, amelyre utána softmax aktivációt alkalmazunk. Használunk továbbá "dropout"-t az első teljesen kapcsolt rétegen 0,5-re beállított aránnyal.

Bár a VGG16 alkotói nem használnak Batch Normalization-t [6] (akkor még nem volt elterjedt módszer), a tanítás felgyorsítására céljából mi minden aktiválás után beszúrunk egy ilyen réteget.

A veszteségfüggvényünk a kategorikus keresztentrópia, és a sztochasztikus gradiensereszkedést használjuk az optimalizáláshoz 0,9-es momentummal 120 epoch-on keresztül. A tanulási ráta ütemezését illetően kísérletezéssel azt találtuk, hogy a [11]-ben bemutatott "cosine decay" ütemezés elég jól működik esetünkben. Nincsenek a cikkben bemutatott újraindítások, csak egy periódust használunk, 0,1-es kezdő (maximális) tanulási rátával és 0 minimális tanulási rátával. Így a tanulási ráta fokozatosan csökken 0,1-ről 0-ra a 120 epoch alatt. Regularizációs módszerként a "weight decay"-t használjuk, amely 0,0001-től indul, és a tanulási rátával arányosan csökkent.

Technikai okok miatt kisebb, csak 64-es batch méretet használunk (a VGG16 alapú HierNet variációinak a GPU memóriaigénye miatt nem használhatunk 256-ot, de még 128-at sem).

A bemenet előfeldolgozási lépései ugyanazok, mint a ResNet esetében a CIFAR10-en (lásd a 6.1. részt), nevezetesen: a képpontok a (0,255) tartományból a (0,1)-be vannak leképezve; a képpontonkénti átlagokat levonjuk a képekből; véletlenszerű vízszintes tükrözést és véletlenszerű eltolást (faktor = 0,125) alkalmazunk a tanítási képeken; az osztálycímkeket one-hot kódolásra konvertáljuk.

A tanítás eredménye a következő:

- Tanítási pontosság: 97,92 %
- Validációs pontosság: **72.02%**
- Tesztelési: **71.57%**

7. Kísérletek

Időhiány miatt minden kísérletünket 2 maximális mélységű hierarchikus fákkal rendelkező modelleken végeztük, ami azt jelenti, hogy az így létrejövő HierNetben az első él után van egy osztályozó csomópont, amit egy szint más osztályozó csomópont követ, amelyek visszaadják a prediktált osztályt (hasonlóan a 8. ábrán látható architektúrához). Ezért a gerincmodellből származó műveletek eloszlása az élek 2 szintjén egyetlen "felosztási ponttal" ábrázolható: az első szint élén (a gyökér után) lesznek azon rétegek, melyek a felosztási pont előtt vannak és minden második szinten lévő él rétegei a felosztási pont utáni rétegek lesznek.

7.1. Dummy vagy Proof of Concept

Első kísérletünk csak egy "proof of concept" volt, azzal az egyetlen céllal, hogy kiderítsük, hogy ezek a HierNet modellek a 3.2. részben leírt tanítási technikával egyáltalán tanulnak-e valamit. Ezen a ponton nem állt szándékunkban felülmúlni az gerincmodell pontosságát.

Ehhez a HierNethez használt hierarchia vagy csoportosítás a 3. táblázatban látható. Ezt a hierarchiát manuálisan építettük fel az osztályok vizuális hasonlóságai alapján:

Nr	Szuperosztály	Osztályok
1	Emlős	Macska, Kutya, Szarvas, Ló
2	Jármű	Autó, Teherautó, Hajó
3	Repülő dolog	Repülőgép, Madár
4	Béka	Béka

3. táblázat. A CIFAR10 adathalmaz osztályainak manuálisan felépített hierarchiája

- A macskáknak, kutyáknak, szarvasoknak és lovaknak is van 4 lába van, van fejük, szemek és egy hasonló törzs
- az autók, teherautók nagyon hasonlítanak egymásra, és alakjukat, íveiket tekintve mindkettő hasonló a hajókhoz
- a repülőgépek és a madarak hasonlóak abban a tekintetben, hogy mindkettőnek szárnya van, és megjelenhetnek az égen
- a béka egyedül van a szuperosztályában, tekintve, hogy nem hasonlít a fent említett osztályok egyikéhez sem.

Ez a hierarchia hierarchikus fává szervezhető úgy, hogy egyetlen gyökérrel rendelkezik, a felsorolt szuperosztályok belső csomópontok, az osztályok pedig levelek.

A gerincmodell itt a következőkből áll: a bemenet után 2 konvolúciós réteg 32 szűrővel, melyet Max Polling követ, majd egy “droupout” majd még 2 konvolúciós réteg, Max Polling-al és “droupout”-al utána, végül pedig egy kiegyensúlyozó (flattening) réteg, melyet követ egy 10 neuronnal rendelkező teljesen kapcsolt réteg softmax aktivációval. Minden konvolúciós réteg 3×3 -as kernelt használ, *same* padding-el, 1-es stride-al és ReLU aktivációval. A Max Pooling rétegek 2-es faktorral csökkentik a bemenetük felbontását, míg a “droupout”-okat 0,2-es valószínűsége állítottuk. Lehet ez nem optimális architektúra, de tekintve, hogy ez csak egy “dummy” modell, ez nem igazán számít. A modellt az Adam algoritmus[8] segítségével 60 epoch-ig tanítjuk, 0,01 kezdő tanulási rátával, és ezt osztva 10-zel a 7800-as (20 epoch) és az 15600-as (40 epoch) iterációnál. A bemeneti adatok előfeldolgozása ugyanaz, mint a ResNet modellnél a CIFAR10-en (lásd a 6.1. részt), augmentáció nélkül. Az elért tanítási pontosság 67.01%, míg a validációs **66.20%**.

A HierNet felépítését illetően a felosztási pontot 4-ra választottuk, ami azt jelenti, hogy a HierNet felépítésénél az első szint éle (a gyökértől az első osztályozó csomópontig)

ugyanazokat a műveleteket fogja végezni, mint a gerincmodell első 4 rétege (2 konvolúció+Max Pool+Dropout) és a második szint összes élén az összes maradék művelet lesz. Az osztályok 4 szuperosztályba vannak csoportosítva, ezért a második szinten összesen 4 él vezet a 4 levélcsomóponthoz. Egy osztályozó az első szinten lévő csomópontban 4, míg a 4 levélcsomópontban rendre 4, 3, 2 és 1 kimenettel rendelkezik. Modellünk ugyanazzal az algoritmussal van tanítva, ugyanannyi epoch-ig és ugyanolyan tanulási ráta ütemezéssel, mint a gerincháló. Az adatok előfeldolgozása is ugyanaz, mint a gerincmodellnél, azzal a különbséggel, hogy az osztálycímkéket át kellett rendezni, hogy megfeleljenek a HierNet kimeneti osztálysorrendjének (a levelek összefűzött o_i -jai). Így a képzési pontosság 60,66%, az validációs pontosság pedig **58,59%**.

Bár a HierNet ezen verziójának teljesítménye jóval alacsonyabb, mint a hozzá tartozó gerinchálózat teljesítménye, ez a kísérlet megmutatta, hogy ezek a modellek a mi képzési technikánkkal valóban képesek megtanulni a képek osztályozását, így folytathatjuk a kimerítőbb kísérletezést jóval hatékonyabb és elterjedt gerincmodellekkel.

7.2. HierNet a CIFAR10-en ResNet gerinccel

A (ResNet, CIFAR10) az a gerinc-adathalmaz pár, mely a legkimerítőbben tesztelt HierNet variációnk. Az alapötletet, majd az összes fejlesztést először ezen a változaton teszteltük több különböző konfigurációval együtt, hogy kiderüljön, működnek-e, és ha igen, milyen jól. Először egy alap HierNetet mutatunk meg bármiféle továbbfejlesztés nélkül 2 különböző felosztási ponttal, majd a csomópontokban hozzáadott rétegek hatása kerül kiértékelésre, ezt követi az előtanítás, majd az Transfer Learning hatásának felmérése.

7.2.1. Az alap HierNet

Első kísérletünkben egy ResNet alapú HierNetet hoztuk létre azzal a céllal, hogy a CIFAR10 képeket osztályozzuk. Az alapverziót használjuk minden továbbfejlesztés nélkül. Az ehhez használt osztályhierarchia megegyezik a „dummy” kísérletnél használt hierarchiával (lásd a 7.1. szakaszt vagy a 3. táblázatot).

Úgy döntöttünk, hogy gerincként a ResNet $n = 9$ -es verziójával kísérletezünk. Figyelembe véve, hogy amikor $n = 9$, a ResNet összesen 27 “residual” blokkot tartalmaz, első körben

Feloszt. pont	Gyökér pontos.	Felt. pontos.	Útvál. pontos.	Gerinc pontos.	Diff
9	91.18	86.96	86.88	92.99	-6.11
18	95.70	91.78	91.73	92.99	-1.26

4. táblázat. A teljesítménye 2 különböző felosztási pont beállításnak egy alap HierNetben, ResNet $n = 9$ gerinccel, a CIFAR10-re.

kísérletezünk 9-es és 18-as felosztási pontokkal. A felépítést itt nem részletezzük, az eddig leírtak alapján egyértelmű.

A bemenetek előfeldolgozása ugyanaz, mint az gerincnél (lásd a 6.1. részt), két különbséggel:

- az osztálycímkéket át kell rendezni, hogy megfeleljenek a levelek által kiadott osztályok sorrendjének.
- minden minta kell tartalmazza a szuperosztályt is, amibe tartozik, így egyrészt mérni tudjuk a hierarchia gyökercsomópontjának a pontosságát, másrészt szükség lesz rá az említett gyökér előtanításánál.

Az tanítás ugyanúgy történik, mint a gerinc tanítása, akkor amikor $n = 18$ (néhány epoch-nyi bemelegítő tanítással).

Az eredmények a 4. táblázatban láthatók.

Először is megfigyelhető, hogy a gyökér előtti több réteg és a levélcsomópontok felé eső éleken kevesebb réteg hatalmas különbséggel jobban teljesít. Ennek oka lehet az a jelenség, hogy amikor a felosztási pont alacsonyabb, akkor még nincs elég réteg ahhoz, hogy az első osztályozó előtt megfelelő tulajdonságkinyerést végezzen. A 18-as osztási pontot fogjuk használni a többi kísérlethez a (ResNet, CIFAR10) párral.

Másodszor, mindkét esetben az útválasztási pontosság (amelyet a predikció során használunk) alacsonyabb, mint a feltételes pontosság (amelyet a tanítás során használunk) (mindkettő definiálva a 3.2. részben). Ez érthető, tekintve, hogy az utóbbit optimalizáljuk. Emiatt a többi kísérletnél is az várható, hogy az útválasztási pontosság általában alacsonyabb lesz, mint a csak tanításra használté.

#hozzáad.	Gyökér pontos.	Felt. pontos.	Útvál. pontos.	Gerinc pontos.	Diff
0	95.70	91.78	91.73	92.99	-1.26
2	96.28	92.66	92.67	92.99	-0.32
3	96.44	92.75	92.71	92.99	-0.28
5	96.51	92.85	92.79	92.99	-0.20
9	96.72	93.12	93.11	92.99	+0.12

5. táblázat. A teljesítménye a különböző osztályozó elé hozzáadott rétegek beállításnak egy alap HierNetben, ResNet $n = 9$ gerinccel, a CIFAR10-re. A felosztási pont minden esetben 18

Harmadszor, egyik HierNet sem haladta meg a gerincmodell pontosságát, így folytatjuk a kísérletezést a különböző fejlesztésekkel.

7.2.2. Az osztályozó elé hozzáadott rétegek hatása

Ezután kísérletezzünk az első fejlesztésünkkel: a hozzáadott rétegek csomópontokban az élről érkező bemenetek és az osztályozó között.

Ezeket a kísérleteket azonos adat-előfeldolgozással, hierarchiával és tanítási beállításokkal végeztük, mint korábban.

A modell teljesítményét először $n=9$ -el és 18-as felosztási ponttal teszteltük 2, 3, 5 vagy 9 további réteg hozzáadásával. Ahogy a 3.3. szakaszban említettük, nem adunk (és nem is tudunk adni) további rétegeket a levélcsomópontok osztályozói elé, így csak az első csomópont (hierarchikus gyökér) kapja ilyen további rétegeket.

A 5. táblázatban felsorolt eredmények azt mutatják, hogy az osztályozó előtti további hozzáadott konvolúciós rétegek határozottan pozitívan befolyásolják a modellünk teljesítményét. Csak 2 réteg hozzáadásával a pontosság majdnem 1%-al javul, míg a 3, 5 vagy 9 réteg hozzáadásának teljesítménye között kisebb a különbség, mind a gyökér pontosság, mind az útválasztási pontosság tekintetében. Bár ez nagyot javított a modellünkön, még minding csak a 9 hozzáadott réteget tartalmazó HierNet múlja felül a gerincmodell pontosságát, viszont ennyi további réteg hozzáadása egy kissé túlzás, mivel ebben az esetben csak az első csomópontban lévő osztályozó kiértékelésének a komplexitása megegyezik a gerinc kiértékelésének komplexitásával.

n	Felosz. pont	Gyökér pontos.	Felt. pontos.	Útvál. pontos.	Gerinc pontos.	Diff
3	6	95.75	91.34	91.32	91.57	-0.25
5	10	96.16	92.26	92.22	92.22	0
7	14	96.32	92.70	92.65	92.57	+0.08

6. táblázat. A teljesítménye a 2 osztályozó elé hozzáadott rétegnek egy alap HierNetben, ResNet $n = \{3, 5, 7\}$ gerinccel, a CIFAR10-re. A felosztási pont minden esetben $\frac{2}{3}$ -a rétegek teljes számának

#hozzáad.	Gyökér pt. (előtanítás)	Gyökér pt.	Felt. pt.	Útvál. pt.	Gerinc pt.	Diff
2	95.78	95.66	93.16	93.17	92.99	+0.18
5	96.01	96.82	93.38	93.36	92.99	+0.37
9	96.39	96.92	93.55	93.52	92.99	+0.53

7. táblázat. Az előtanítás teljesítménye a különböző osztályozó elé hozzáadott rétegek számához viszonyítva egy HierNetben, ResNet $n = 9$ gerinccel, a CIFAR10-re. A felosztási pont minden esetben 18. Az első oszlop az előtanítás utáni gyökérpontosságot mutatja.

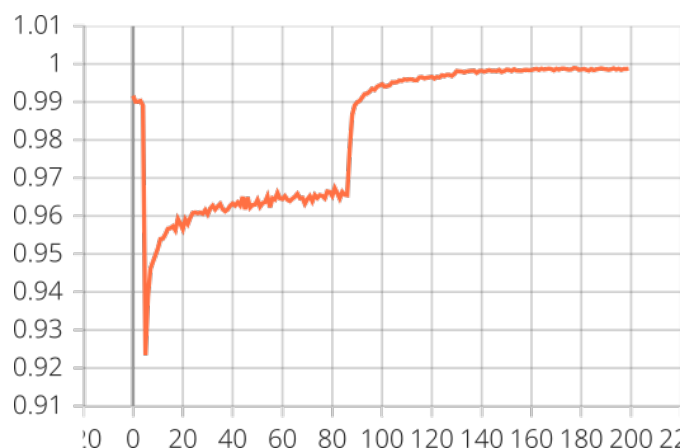
Ezután kiértékeljük a HierNeteket a ResNet összes többi verziójával gerincként, azaz amikor $n = \{3, 5, 7\}$. Mindegyikhez 2 réteget adunk hozzá és egy olyan felosztási pontot használunk, amely $\frac{2}{3}$ -a a rétegek számának (ugyanazt az arányt, amit a $n = 9$ -nél használtunk). Az eredményeket a 6. táblázat tartalmazza.

7.2.3. Az előtanítás hatása

Folytatjuk a kísérletezést a gyökércsomópont előzetes tanításával a 3.4. szakaszban leírtak szerint. A tanítási beállítások, a hierarchia és az adatok előfeldolgozása ugyanaz. Az előtanítás idejére minden konfiguráció marad, azzal a különbséggel, hogy az első csomópont softmax kimenetén alkalmazott kategorikus keresztentrópia veszteségfüggvény van optimalizálva.

Első körben a különböző számú hozzáadott réteggel teszteljük az előtanítás hatásait. $n = 9$, a felosztási pont 18, és az osztályozó előtti további rétegek száma 2, 3 és 9. A 7. táblázat felsorolja az eredményeket, és ha összehasonlítjuk a 5. táblázat eredményeivel, akkor egyértelmű, hogy ebben az esetben az előképzésnek rendkívül pozitív hatása volt, és minden variáció felülmúlta az alapul szolgáló modell pontosságát.

Érdekes jelenség, hogy az első csomópontban az osztályozó pontossága a fő betanítás során megnőtt, pedig mást optimalizáltunk. Ez azt jelentheti, hogy az előtanítás során használt optimalizáló nem elég jó, és beszorul egy lokális minimumba vagy nyeregpontra, majd a



13. ábra. Az első pontban (hierarchikus gyökér) lévő osztályozó tanítási pontosságának változása a fő tanítás során, amikor azt megelőzően előtanítottuk

n	Gyökér pontos.	Felt. pontos.	Útvál. pontos.	Gerinc pontos.	Diff
3	95.76	91.63	91.58	91.57	+0.01
5	96.36	92.61	92.57	92.22	+0.35
7	96.39	92.88	92.85	92.57	+0.28

8. táblázat. Az előtanítás teljesítménye egy HierNetben, ResNet $n = \{3, 5, 7\}$ gerinccel, a CIFAR10-re. A felosztási pont minden esetben $2n$ és 2 réteg van az osztályozó elé hozzáadva.

fő edzés első epoch-jainak magasabb tanulási rátája kiugrasztja ebből a pontból. A 13. ábra megerősíti ezt a feltételezést, mert az látható, hogy a fő tanítás során a pontosság néhány százalékkal csökken, majd visszakúszik, magasabbra, mint korábban.

A második körben az előre betanított súlyok befagyasztásával kísérleteztünk. Tekintettel arra, hogy a főtanítás során megnövekedett a gyökér pontossága, ez nem túl kecsegtető, de kipróbáltuk, hátha mutat valami újat. Egy modell 2 darab hozzáadott réteggel, $n = 9$ -el, 18-as felosztási ponttal, először előtanítva, majd az első csomópont előtti összes súlyt befagyasztva 90,61%-os útválasztási pontosságot eredményezett, ami 2,56%-al alacsonyabb, mint a nem befagyasztott megfelelője (ami a 7. táblázatban látható). De legalább megmutatta nekünk, hogy még akkor is, amikor az első néhány réteg be van fagyasztva a fő tanításban, azok már megtanultak néhány olyan jellemző kivonást, amelyek hasznosak a következő rétegek számára.

A harmadik körben futtatuk az előtanításos kísérleteket $n = \{3, 5, 7\}$ -re 2 hozzáadott réteggel és $2 \cdot n$ felosztási ponttal. Az eredményeket a 8. táblázat tartalmazza, és a 6. táblázatban található, nem előtanított társaikhoz képest jelentős javulás tapasztalható.

n	Előtan.	Gyökér pontos.	Felt. pontos.	Útvál. pontos.	Gerinc pontos.	Diff
3	nem	96.10	92.19	92.19	91.57	+0.62
3	igen	96.20	92.43	92.39	91.57	+0.82
5	nem	96.31	92.86	92.87	92.22	+0.65
5	igen	96.46	92.99	92.95	92.22	+0.73
7	nem	96.36	93.14	93.09	92.57	+0.52
7	igen	96.44	93.06	93.03	92.57	+0.46
9	nem	96.67	93.40	93.37	92.99	+0.38
9	igen	96.90	93.76	93.74	92.99	+0.75

9. táblázat. A Transfer Learning és az előtanítás kombinációjának a teljesítménye egy HierNetben, ResNet $n = \{3, 5, 7, 9\}$ gerinccel, a CIFAR10-re. A felosztási pont minden esetben $2n$ és 2 réteg van az osztályozó elé hozzáadva.

A kísérletek ezen pontján már minden modellünk felülmúlja a gerincként használt ResNet modell teljesítményét a CIFAR10-en.

7.2.4. A Transfer Learning hatása

Utolsó kísérletsorozatként a (ResNet, CIFAR10) páron a Transfer Learning-el kísérleteztünk. Az egyes HierNet-ek súlyait az ugyanazon n mérettel rendelkező már betanított ResNet modellekből másoljuk át.

A 9. táblázat eredményei azt mutatják, hogy a Transfer Learning önmagában óriási különbséget jelent a teljesítmény tekintetében minden n méreten, és a legtöbb esetben, ha a Transfer Learning-et az előtanítási módszerrel kombináljuk, a teljesítmény még jobb lesz. Ez bizonyítja, hogy a gerincmodell betanítása során elsajátított konvolúciók általi jellemzőkinyerések valóban hasznosak a HierNet számára.

Végül a 9. táblázatban szereplő útválasztási pontosságok azok a végső pontosságok, amelyeket a HierNet számára jelenteni szeretnénk, amikor a CIFAR10 képeket osztályozunk 3, 5, 7 vagy 9-es méretű ResNet-gerinccel.

7.3. Automatizált csoportosítás a CIFAR100-on

Mielőtt bemutatnánk a HierNet-es kísérleteket a CIFAR100 adathalmazon, szeretnénk egy kicsit írni a 4. szakaszban bemutatott csoportosító algoritmus által létrehozott csoportosításokról. Tekintettel arra, hogy a csoportosítási algoritmusunknak 2 paramétere

Max csoportméret	Minimum közelség	#csoportok
50	0.005	7
20	0.005	9
50	0.0075	10
50	0.01	12
10	0.005	14
10	0.01	15
50	0.015	15
20	0.015	16
10	0.02	23
50	0.025	28
50	0.05	50

10. táblázat. A csoportosítási algoritmusunk által létrehozott csoportok száma különböző maximális csoportméretekkel és minimális közelségi beállításokkal

van: a maximális csoportméret és a minimális közelség, először a csoportok teljes számát mutatjuk meg néhány különböző beállításpár esetén, majd csak egy csoportosításra helyezzük a hangsúlyt, azt bemutatva, hogy a benne lévő csoportok valójában mely osztályokat tartalmazzák. Az összes alábbi csoportosítás egy betanított ResNeten ($n=9$) végzett validációs halmaz kiértékelése által generált konfúziós valószínűségi mátrixból épül fel.

A 10. táblázat a különböző beállításoknak a kimeneti csoportosításra gyakorolt hatását mutatja. Megfigyelhető, hogy a csoportok száma egyenesen arányos a közelségi határértékkel (például ha a csoportlétszám 50-re van rögzítve, minél nagyobb a minimális közelségi határ, annál több csoportra van szükség), és fordítottan arányos a maximális csoportlétszám (például ha a minimális közelség 0,015 , akkor még 1 csoportra van szükség, ha 50-ről 20-ra csökkentjük a csoport méretkorlátját).

Koncentráljunk az 50-es csoporthatárral és 0,0075-ös minimális közelséggel rendelkező csoportosításra, amely 10 csoportot tartalmaz. Most felsoroljuk néhány csoport tartalmát a csoportosításban, hogy megmutassuk, a vizuális hasonlóságokkal rendelkező osztályok tényleg egy csoportba kerülnek:

- 1. CSOPORT: baba, lány, nő, fiú, férfi
- 2. CSOPORT: pálmafa, erdő, fenyőfa, fűzfa, juharfa, tölgy
- 3. CSOPORT: akvárium halak, pisztráng, lepényhal, rája, cápa, delfin, bálna

- 4. CSOPORT: szekrény, szék, televízió, ágy, kanapé, billentyűzet ...
- ...

Bár a csoportok többnyire teljesen értelmesek, van néhány apró kivétel, például:

- a "híd" és a "ház" a járművek kategóriába tartozik
- a "borz" maga egy csoport
- a "gomba" az állatok csoportjába tartozik

De még ezekhez is lehet esetleges magyarázatokat találni: a ház formái hasonlóak némely járműhöz, a borz színei összetéveszthetetlenek és a gombák hasonló kontextusban jelennek meg, mint a legtöbb állat, ráadásul nem a megszokott növénytípus megjelenésük van.

7.4. HierNet a CIFAR100-on ResNet gerinccel

A következő gerinc-adathalmaz pár, amelyet tesztelünk, a (ResNet, CIFAR100). A hálózat architektúrája automatizált csoportosításokból épül fel. Az összes kísérletnél a tanítási beállítások (például optimalizáló, tanulási ráta, weight decay, epoch-ok száma, batch méret) ugyanazok lesznek, mint a ResNet gerincénél, amikor $n = 18$. A bemeneti előfeldolgozás szintén ugyanaz lesz, kivéve az újrendezett címkéket és a hozzáadott szuperosztály címkéket (ugyanúgy, mint a CIFAR10 esetében a 7.2. szakaszban)

A CIFAR100 esetében van különböző validációs és tesztelési adathalmazunk, ezért a különböző kísérleteknél csak a validálási pontosságokat jelenítjük meg, és tesztelési pontosságot csak azoknak a modelleknek adjuk meg minden $n = \{3, 5, 7, 9\}$ -re, melyek az adott méretre a legnagyobbak a validálási pontosságuk alapján.

Az első körben különböző felosztási pontokkal és hozzáadott rétegek számával fogunk kísérletezni, amikor $n = 9$. A második körben a különböző csoportosítások hatását mutatjuk be, amikor $n = 5$. A harmadik körben az előtanítás hatásaival fogunk kísérletezni ebben az esetben. Ezután minden n méretre fogunk modelleket tanítani az előzőben letesztelték közül 2 ígéretes csoporton. Végül tudjuk a tesztelési pontosságokat jelenteni minden n -re és összehasonlítani a gerincmodell tesztelési pontosságával.

Felosz. pont	Hozzáad.	Gyökér pts.	Felt. pts.	Útvál. pts.	Gerinc pts.	Diff
14	6	82.27	70.69	69.99	68.67	+1.32
16	4	83.09	71.43	71.13	68.67	+2.46
18	2	82.27	70.03	69.57	68.67	+0.9
20	0	83.05	69.61	69.29	68.67	+0.62

11. táblázat. A teljesítménye a különböző osztályozó elé hozzáadott rétegek számának és a különböző felosztási pontoknak egy HierNetben, ResNet $n = 9$ gerinccel, a CIFAR100-ra. A csoportosítás paraméterei: max csoportméret 10 és minimum közelség 0,005. A validációs pontosság van megmutatva.

7.4.1. A hozzáadott rétegek és a felosztási pont hatása

Ezennel teszteltük az osztályozó előtt hozzáadott rétegek és a felosztási pont kiválasztásának a hatását, amikor $n = 9$. A tesztelt párok (6, 14), (4, 16), (2, 18) és (0, 20). Transfer Learning-t alkalmaztunk, de előtanítást nem, hogy megtakarítsuk elég sok tanítási időt. A használt csoportosítás a következő beállításokkal jön létre: maximális csoportméret = 10 és minimális közelség = 0,005. Az eredmények a 11. táblázatban láthatók. A következő kísérleteknél a legnagyobb pontosságúhoz hasonló arányokat használunk.

7.4.2. A különböző csoportosítások teljesítménye

A második körben a 10. táblázatban szereplőkkel megegyező paraméterű 11 csoportosítás teljesítményét nézzük meg. Alacsonyabb, 5-s n -t fogunk használni, hogy gyorsabban lehessen futtatni a kísérleteket. 9 lesz a felosztási pont és 3 a hozzáadott rétegek száma. Transfer Learning-t használunk, de nincs előtanítás.

Az eredmények a 12. táblázatban láthatók. Többé-kevésbé a kisebb csoportszámú modellek jobban teljesítenek. A első csomópont (hierarchikus gyökér) pontosságai közti szignifikáns eltérés érthető, mivel a különböző csoportosítások eltérő számú csoportot tartalmaznak, így az első csomópont osztályozójának különböző számú lehetőségei vannak különböző modellekben. A HierNet teszteléséhez az $n = \{3, 7, 9\}$ -re a (50, 0.0075) csoportosítást (mert ez az első a pontosság szerint) és a (50, 0.025) csoportosítást használjuk (mert ez átlagon felüli pontosságot kapott, annak ellenére, hogy a második legtöbb csoporttal rendelkezik)

Max csop.	Min. köz.	#csop.	Gyökér pts.	Felt. pts.	Útvál. pts.	Gerinc pts.	Diff
50	0.0075	8	90.73	71.65	71.31	67.75	+3.56
50	0.01	10	87.30	71.76	71.23	67.75	+3.48
10	0.01	14	83.23	70.73	70.51	67.75	+2.76
50	0.005	6	92.55	70.65	70.43	67.75	+2.68
50	0.025	26	81.21	70.53	70.35	67.75	+2.6
20	0.005	9	85.28	70.35	70.01	67.75	+2.26
20	0.015	16	83.77	70.55	69.89	67.75	+2.14
50	0.015	16	83.03	70.43	69.81	67.75	+2.06
10	0.005	13	82.41	70.41	69.77	67.75	+2.02
50	0.05	46	77.62	69.87	69.65	67.75	+1.9
10	0.02	23	80.31	69.59	69.09	67.75	+1.34

12. táblázat. A teljesítmények összehasonlítása különböző csoportosításokkal egy HierNetben, ResNet $n = 5$ gerinccel, a CIFAR100-ra. Transfer Learning-t használunk, de előtanítást nem. A felosztási pont 9, a hozzáadott rétegek száma pedig 3. A validációs pontosság van megmutatva.

7.4.3. Az előtanítás esete

Teszteltük az előtanítás és a Transfer Learning kombinációjának a hatásait is. Bár az előtanítás kiválóan működött a CIFAR10 adathalmazon végzett kísérletek során (7.2. szakasz), ebben az esetben nem javította a pontosságot. A tesztet az előző kísérlet legjobban teljesítő modelljével végezték el, amely 71.31%-os pontossággal rendelkezik. Az előtanított modell 70,11 % útválasztási pontosságot ért el a validációs halmazon, ami jelentősen, 1,2 %-al alacsonyabb, mint a nem előtanított megfelelője. Ha nem észleljük a határozott pozitív hatását az előtanításnak erre a modellváltozatra, akkor nem fogjuk használni, aminek nagy előnye a lecsökkent tanítási idő.

7.4.4. Egyéb ResNet méretekre

Most minden $n = \{3, 7, 9\}$ -re is kiértékeljük a modellünket, hasonló felosztási ponttal és hozzáadott rétegarányokkal, mint amit a 7.4.1. szakaszban kiválasztottunk, és azzal a 2 csoportosítási beállítással, amit kiválasztottunk a 7.4.2. szakaszban választottunk. Tehát a használt felosztási pontok rendre 5, 12, 16, amikor $n = 3, 7, 9$, míg a hozzáadott rétegek száma 2, 4, 5, és a minden n -re a tesztelt csoportosítások a (50, 0, 0075) és a (50, 0, 025).

A 13. táblázat tartalmazza az eredményeket. Megfigyelhető, hogy minden n -ra a (50, 0.0075) csoportosítás jobb, ezért ezt választjuk ki a végső modellekhez, amikre megmutatjuk a tesztelési pontosságot.

n	Max csop.	Min köz.	#csop.	Gyökér pts.	Felt. pts.	Útvál. pts.	Gerinc pts.	Diff
3	50	0.0075	8	86.76	69.05	68.31	67.07	+1.24
3	50	0.025	25	79.01	68.11	67.11	67.07	+0.04
7	50	0.0075	10	88.38	71.59	71.21	68.69	+2.52
7	50	0.025	32	80.71	71.05	70.73	68.69	+2.04
9	50	0.0075	10	90.54	71.81	71.47	68.67	+2.8
9	50	0.025	28	80.93	71.15	70.73	68.67	+2.06

13. táblázat. A teljesítmények összehasonlítása 2 különböző előzőleg kiválasztott csoportosítással egy HierNetben, ResNet $n = \{3, 7, 9\}$ gerinccel, a CIFAR100-ra. Transfer Learning-t használunk előtanítás nélkül A felosztási pont 9, a hozzáadott rétegek száma pedig 3. A felosztási pont és a hozzáadott rétegek száma rendre (5, 2), (12, 4) és (16, 5) $n=\{3, 7, 9\}$ -re. A validációs pontosság van megmutatva.

n	HierNet útválasztási pontosság	Gerincháló pontosság	Diff
3	68.89	65.96	+2.93
5	70.45	67.08	+3.37
7	71.25	68.12	+3.13
9	72.15	68.38	+3.77
18	N/A	71.33	N/A

14. táblázat. A tesztelési pontossága a HierNet-nek Resnet gerinccel a CIFAR100-ra a különböző n méretek esetén.

7.4.5. Tesztelési pontosságok

Most, hogy a paraméterek összes hangolása és tesztelése befejeződött, felfedjük a végső modellek tesztelési pontosságát minden $n = \{3, 5, 7, 9\}$ esetében. Az eredmények a 14. táblázatban érhetők el

Kijelenthetjük, hogy hálózatunk minden $n = \{3, 5, 7, 9\}$ esetében olyan jelentősen felülmúlta a ResNet gerinchálózatot, hogy a legkisebb HierNet ($n = 3$) pontossága nagyobb, mint a második legnagyobb ResNet ($n = 9$) pontossága és a második legnagyobb pedig még a legnagyobbat ($n = 18$) is felülmúlja 0,82%-al.

7.5. HierNet a CIFAR100-on VGG16 gerinccel

Következőleg bemutatjuk a HierNetet egy másik gerinccel: VGG16[14] (leírása a 6.4. részben). A cél adathalmaz a CIFAR100. Ezek az egyetlen kísérletek, amelyeket a VGG16, mint gerincen futtatunk, a háló nagyon sok paramétere és hosszú tanítási ideje miatt.

Max csop.	Min közel.	#csop.	Felosz.	Hozzá.	Utvál. pts.	Gerinc pts.	Diff
50	0.003	4	7	4	71.91	72.02	-0.11
50	0.005	6	7	4	72.56	72.02	+0.54
50	0.0065	10	7	4	72.53	72.02	+0.51
50	0.004	5	6	4	73.40	72.02	+1.38
50	0.005	6	6	4	73.18	72.02	+1.16
50	0.006	8	6	4	72.74	72.02	+0.72

15. táblázat. A VGG16 gerinchálózattal rendelkező HierNet különböző csoportosítási, felosztási pont és hozzáadott rétegbeállításainak összehasonlítása Transfer Learning-el, de előtanítás nélkül. A validációs pontosságok vannak feltüntetve.

A hálózat architektúrája a csoportosító algoritmusunk által előállított topológia szerint épül fel. Minden kísérletnél a betanítási beállítások és az adatok előfeldolgozása ugyanaz lesz, mint a gerincmodellnél, viszont a korábbiakhoz hasonlóan a címkék átrendeződnek, hogy megfeleljenek a HierNet kimeneti sorrendjének. Transfer Learning-t alkalmazunk, de nem előtanítást nem, időmegtakarítás céljából.

Futtattunk néhány kísérletet különböző csoportosításokkal, felosztási pontokkal és hozzáadott rétegek számával, majd kiválasztottuk a legmagasabb validációs pontossággal rendelkezőt, hogy annak a tesztelési pontosságát összehasonlítsuk a gerinchálózat pontosságával. Az validációs halmaz eredményei a 15. táblázatban láthatók.

Kiválasztottuk a konfigurációt az 50-es maximális csoportosítással, 0,004-es minimális közelséggel, 6-os felosztó ponttal és 4-es hozzáadott rétegekszámával. Ennek a HierNetnek a tesztelési pontossága **71.99%**, ami 0,42%-el nagyobb, mint a 71.57%-os gerinchálózati tesztelési pontosság.

7.6. HierNet a Tiny ImageNet-en ResNet gerinccel

Végül szeretnénk bemutatni, hogy hogyan teljesít egy ResNet gerinchálózattal rendelkező HierNet a Tiny ImageNet-en. Az itt használt ResNet egy kicsit eltér a CIFAR10/100-nál használttól, ennek leírása a 6.3. szakaszban található. A csoportosítással, tanítással és adat-előfeldolgozással kapcsolatos eljárásunk megegyezik a korábbiakkal.

Először is végzünk néhány kísérletet különböző HierNet konfigurációkkal, majd megmutatjuk a tesztpontosságát a legjobbnak (a validációs pontosság alapján) és

Max csop.	Min közel.	#csop	Felosz.	Hozzá.	Utvál. pts.	Gerincmodell pts.	Diff
50	0.0035	8	4	2	58.51	60.02	-1.51
50	0.0035	8	5	2	60.98	60.02	+0.96
50	0.0025	6	5	2	60.80	60.02	+0.78
50	0.01	22	5	2	60.66	60.02	+0.64

16. táblázat. A ResNet ($size = 18$) gerinchálózattal rendelkező HierNet különböző csoportosítási, felosztási pont és hozzáadott rétegbeállításainak összehasonlítása a Tiny ImageNeten, Transfer Learning-el, de előtanítás nélkül. A validációs pontosságok vannak feltüntetve.

összehasonlíttuk az alapmodellel.

A 16. táblázat a 4 teszt eredményét mutatja a validációs halmazon. Kiválasztottuk a legjobbat, mégpedig azt, amelyiknél 50 a maximális csoportlétszám, 0,0035 a minimális közelség, 5 az felosztási pont és 2 a hozzáadott rétegek száma. HierNetünk jól teljesített, a tesztelési pontossága **61.17%** lett, ami +2.96 %-al magasabb, mint a gerinc (58.21%) pontossága.

8. Konklúzió

Ebben a cikkben bemutattuk a HierNet konvolúciós neurális hálózati architektúrát, amely kihasználja az osztályok közötti vizuális hasonlóságokat és hierarchiát. Ezt úgy értük el, hogy a hierarchikus kapcsolatok alapján egy fát építettünk fel, ahol az élek a jellemzőket kivonó konvolúciókat, a csomópontok pedig az osztályozó vagy útválasztó függvényeket képviselik. Így az ugyanabban a csoportban lévő osztályok megoszthatnak bizonyos tulajdonság kivonó műveleteket,.

A kísérleteink eredményei megerősítik, hogy ez az architektúra a tervezett módon működik. És bár szinte minden esetben felülmúltuk a gerinchálózatokat, határozottan van még mit javítani.

Az egyik fejlesztés lehet egyszerűen több tesztelés és kísérletezés. Mivel nem egyértelmű, hogy pontosan mi a jó arányú felosztása a rétegeknek a szintek között, vagy milyen arányban kell hozzáadni rétegeket az osztályozó elé és ezek az értékek hogyan viszonyulnak a csoportosításokhoz vagy az osztályok számához, további tesztelés esetlegesen érdekes összefüggéseket tárhat fel.

Ami az architektúrát illeti, mi csak 2 mélységű hierarchia fákon teszteltük a modellünket, de ki lehetne próbálni mélyebb hierarchia fákkal is, ahol nem csak 2 szintnyi él van.

Egy másik fejlesztés lehet egy kifinomultabb csoportosítási algoritmus. Csoportosítási algoritmusunk gyakran olyan csoportfákat hoz létre, amiben például egy csoportnak 40 osztálya van, néhány másinak csak pár darab. Bár valószínűleg lehetetlen teljesen kiegyensúlyozott fát létrehozni, mert egyes osztályok valóban jobban megkülönböztethetők, míg egy másik nagy osztályhalmaz nagyon hasonló, de javíthatnánk az algoritmusunkat, azzal, hogy figyelembe vegye a fa egyensúlyozottságát.

Ami az tanítást illeti, mivel ugyanazt az optimalizálót, tanulási ráta ütemezést, “weight decay”-t használtuk, mint a gerincmodelleknél, ezért nagy a valószínűsége annak, hogy ami az alapmodelleken működik, az nem optimális a HierNet számára, következésképpen az tanítási beállításokat is jobban megvizsgálhatnánk.

Végül hasznos lehet megvizsgálni, hogy a megosztott élek valójában milyen jellemzőket vonnak ki, és ezzel szemben az egyes csoportok élei milyen jellemzőket vonnak ki. Ezt hasonló megközelítéssel vizualizálhatnánk, mint amit a [17]-ban bemutatnak.

Hivatkozások

- [1] Xavier Glorot és Yoshua Bengio. „Understanding the difficulty of training deep feedforward neural networks”. *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop és Conference Proceedings. 2010, 249–256. oldal.
- [2] Ian Goodfellow, Yoshua Bengio és Aaron Courville. *Deep Learning*. [http : / / www . deeplearningbook . org](http://www.deeplearningbook.org). MIT Press, 2016.
- [3] Kaiming He és mások. „Deep residual learning for image recognition”. *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, 770–778. oldal.
- [4] Kaiming He és mások. „Identity mappings in deep residual networks”. *European conference on computer vision*. Springer. 2016, 630–645. oldal.
- [5] Gao Huang és mások. „Deep networks with stochastic depth”. *European conference on computer vision*. Springer. 2016, 646–661. oldal.
- [6] Sergey Ioffe és Christian Szegedy. „Batch normalization: Accelerating deep network training by reducing internal covariate shift”. *International conference on machine learning*. PMLR. 2015, 448–456. oldal.
- [7] Ruyi Ji és mások. „Attention convolutional binary neural tree for fine-grained visual categorization”. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, 10468–10477. oldal.
- [8] Diederik P Kingma és Jimmy Ba. „Adam: A method for stochastic optimization”. *arXiv preprint arXiv:1412.6980* (2014).
- [9] Alex Krizhevsky, Geoffrey Hinton és mások. „Learning multiple layers of features from tiny images”. (2009).
- [10] Alex Krizhevsky, Ilya Sutskever és Geoffrey E Hinton. „Imagenet classification with deep convolutional neural networks”. *Advances in neural information processing systems* 25 (2012).
- [11] Ilya Loshchilov és Frank Hutter. „Sgdr: Stochastic gradient descent with warm restarts”. *arXiv preprint arXiv:1608.03983* (2016).
- [12] Tom M. Mitchell. *Machine learning, International Edition*. McGraw-Hill Series in Computer Science. McGraw-Hill, 1997. ISBN: 978-0-07-042807-2. URL: <https://www.worldcat.org/oclc/61321007>.

- [13] Olga Russakovsky és mások. „ImageNet Large Scale Visual Recognition Challenge”. *International Journal of Computer Vision (IJCV)* 115.3 (2015), 211–252. oldal. DOI: [10.1007/s11263-015-0816-y](https://doi.org/10.1007/s11263-015-0816-y).
- [14] Karen Simonyan és Andrew Zisserman. „Very deep convolutional networks for large-scale image recognition”. *arXiv preprint arXiv:1409.1556* (2014).
- [15] Lei Sun. „ResNet on Tiny ImageNet”. 2017.
- [16] Ryutaro Tanno és mások. „Adaptive neural trees”. *International Conference on Machine Learning*. PMLR. 2019, 6166–6175. oldal.
- [17] Matthew D Zeiler és Rob Fergus. „Visualizing and understanding convolutional networks”. *European conference on computer vision*. Springer. 2014, 818–833. oldal.
- [18] Xinqi Zhu és Michael Bain. „B-CNN: branch convolutional neural network for hierarchical classification”. *arXiv preprint arXiv:1709.09890* (2017).