

CNN - Projektmunka

CeNN és CNN összehasonlítás

Vajna Levente

2025. május 6.

1. Bevezetés

Jelen dokumentáció egy kísérletsorozatot mutat be, amelyben differenciálegyenletekre épülő Cellular Neural Network (CeNN), egy módosított ResNet-alapú konvolúciós neurális háló (CNN) és ezek hibridje versenyez a CIFAR-10 képosztályozási feladaton. Kíváncsi voltam, hogy a CeNN dinamikus feldolgozása vagy a jól bevált mélyháló-architektúra nyújt-e jobb teljesítményt. A rendelkezésre álló számítási erőforrások és az időkorlátok miatt a gyors kísérletezéshez a viszonylag kis, ám széles körben alkalmazott CIFAR-10 adathalmazt választottuk. A projektet személyes érdeklődésem vezérelte a képfeldolgozás és a neurális hálózatok gyakorlati lehetőségei iránt, célom pedig a hálózati architektúra tanulási hatékonyságra gyakorolt hatásának feltárása.

Minden kód megtalálható a repositorymban.

Repository link: <https://dev.itk.ppke.hu/vajna.levente/cnn-project.git>

2. Háttér

2.1. CIFAR-10

A CIFAR-10¹ egy 60 000 darab 32×32 pixeles színes képgyűjtemény. A minták 50 000 tréning- és 10 000 teszt példányra vannak osztva. Az adathalmaz tíz, egyenlő arányban (6000 kép/osztály) képviselt kategóriát tartalmaz: repülőgép, gépjármű, madár, macska, őz, kutya, béka, ló, hajó és teherautó. Eredetileg 2009-ben Alex Krizhevsky és munkatársai által az 80 milliós Tiny Images adatbázisból származó képek címkézett alhalmazaként jelent meg, hogy egységes alapot adjon az osztályozó modellek összehasonlításához. Alacsony felbontása ellenére a CIFAR-10 a gyors kísérletezés és a különböző képfeldolgozó architektúrák benchmarkolásának ma is széles körben használt standardja.



1. ábra. CIFAR-10

2.2. ResNet-18

A ResNet-18² egy 18 rétegből álló konvolúciós hálózat, amely maradék- (skip) összekötésekkel könnyíti meg a mély rétegek tanulását és mérsékli az eltűnő gradiens problémát.

2.3. Cellular Neural Network

A Cellular Neural Network (CNN), más néven Cellular Nonlinear Network, egy rácsszerűen rendezett dinamikus cellákból álló háló, ahol minden cella csak a közvetlen szomszédjaival kommunikál, és állapotát nemlineáris, lokális szabályok frissítik. Ezt a párhuzamos, helyi feldolgozást valós idejű képfeldolgozó feladatokra – például éldetektálásra, zajcsökkentésre és szegmentálásra – optimalizálták.

¹<https://www.cs.toronto.edu/~kriz/cifar.html>

²<https://www.geeksforgeeks.org/resnet18-from-scratch-using-pytorch/>

3. Implementáció

3.1. Környezet

A kódot Google Colab jegyzetfüzetben (ipynb) fejlesztettem, ahol a CUDA-gyorsítású Tesla T4 GPU jelentősen felgyorsította a modellek tanítását. A cellákra bontott szerkezet lehetővé tette számomra, hogy az adatelőkészítést, a modelldefiníciót, a tanítási ciklust és a kiértékelést különálló, átlátható egységekben kezeljem. Az ipynb formátum szorosan ötvözi a kódot és a magyarázó szövegeket, támogatja a valós idejű eredményvizualizációt, és egyszerűsíti a notebook megosztását.

3.2. CeNN modell

A CeNN-ben három tanulandó paraméter szerepel: visszacsatoló kernel (A), előreccsatoló kernel (B), mindegyik $3 \times 3 \times 3$ méretű konvolúciós sablonként implementálva a három színsatornának megfelelően, valamint a bias (Z). A rendszer dinamikáját $h=0.1$ lépésközü Euler-integrációval és iterációnként 20 időlépéssel szimuláltam a differenciálegyenlet megoldása során, hasonlóképp ahhoz, ahogyan a laborgyakorlatokon néztük. Az aktivációs függvény egy módosított nemlineáris karakterisztikát követ ($\alpha=0.01$ paraméterrel), amely a kimeneteket a $[-1,1]$ tartományra korlátozza. A paramétereket Adam-optimalizálóval frissítettem 0.01-es tanulási rátával. A tanítást 64-es mini-batch mérettel és 30 epoch alatt végeztem, amelynek során a négyzetösszeg hibaértéket (MSE) minimalizáltam.

3.3. CNN modell

A CNN esetében egy előtrénelt ResNet-18 architektúrát használtam, amelyet a `resnet18()` hívással töltöttem be. A kompatibilitás érdekében a konvolúciós első réteget 3×3 kernelre ($\text{stride}=1$, $\text{padding}=1$) módosítottam, eltávolítottam a maxpool réteget internetes források javaslatára, és a végén egy $512 \rightarrow 10$ kimenetes lineáris (fully-connected) réteget alkalmaztam. Eredetileg ez a layer ennél sokkal több osztályra klasszifikál, de mivel itt csak 10-féle osztályunk van, így elegendő csupán ennyire.

Az adatelőkészítés során RandomHorizontalFlip és RandomCrop (32 , $\text{padding}=4$) augmentációt, amik biztosítják, hogy a tanulás során kicsit másképp is tanítva legyen a modellnek az adott kép, jelen esetben körbevágva, vagy megforgatva. Természetesen sok másféle augmentációt is lehetett volna alkalmazni, például színskálán változtatni, zajt hozzáadni, greyscale-lé tenni másképp forgatni vagy tükrözni, de most ehhez a projekthez ennyit elegendőnek tartottam. Ezek után ToTensor és Normalize transzformációkat végeztem el, hogy torch kompatibilis legyen, illetve az értékek normáltak legyenek. Ez elősegíti a gyorsabb és hatékonyabb tanulást, illetve a pontosabb predikciókat. A CIFAR-10-et 64-es mini-batch mérettel, azaz 64 képenként fog hibát számolni és backpropagation-t végrehajtani, illetve shuffle=True paraméterrel töltöttem be a DataLoader segítségével. Ez a paraméter általában a train fázisban szokott igaz értéket felvenni, hogy az is elősegítse a tanulást, hogy nem mindig ugyanabban a sorrendben látja a képeket. (Teszt fázisban ez hamis szokott lenni.) A tanításhoz Adam optimalizálót ($\text{learning rate} = 0.001$) és CrossEntropyLoss-t használtam. Ezek klasszifikációhoz kiválóan szoktak teljesíteni.

3.4. Hybrid modell

A hibrid megközelítés a CeNN és CNN modellek egymásra építésével valósul meg. A bemeneti képek először áthaladnak a CeNN előfeldolgozáson, amely differenciálegyenlet-alapú állapotfrissítéseket végez a képen, kiemelve bizonyos jellemzőket. Ez a feldolgozott kimenet aztán a ResNet-18 bemenetévé válik, amely a képosztályozást végzi. Az elkészített modell a paramétereit (A, B, Z) és a CNN komponenseket egyidejűleg tanulja az Adam optimalizálóval, 64-es mini-batch mérettel, 30 epoch során.

A teljesítményadatok alapján a hibrid modell (91.45%) pontossága a tiszta CeNN (90.02%) és a tiszta CNN (92.40%) között helyezkedik el. Ez arra utal, hogy bár a CeNN előfeldolgozás egyes képjellemzőket hatékonyan kiemel, a teljes ResNet architektúra önmagában jobb generalizációs képességet mutat a CIFAR-10 adatkészleten.

3.5. Train

A tanítási folyamatot PyTorch-ban implementáltam, kihasználva a keretrendszer teljesítménybeli képességeit. Az algoritmus a következő lépésekből állt:

1. **Epoch ciklus inicializálása:** A tanítást 30 teljes epochon keresztül végeztem, minden modell esetében azonos számú iterációt biztosítva:
`for epoch in range(30):`
2. **Tanulási mód beállítása:** A `model.train()` utasítással a modellt tanulási módba helyeztem, amely különösen fontos a BatchNormalization és Dropout rétegek megfelelő működéséhez.
3. **Mini-batch feldolgozás:** Az adatokat 64-es csoportokban dolgoztam fel a DataLoader segítségével, ezeken iterálok végig (nagyjából 780 batch):
`for batch_idx, (data, targets) in enumerate(trainloader):`
4. **Eszközre helyezés:** Az adatokat és címkéket a megfelelő számítási eszközre (GPU) mozgattam a gyorsabb feldolgozás érdekében:
`data, targets = data.to(device), targets.to(device)`
5. **Gradiensek nullázása:** Az `optimizer.zero_grad()` parancs minden batch-nél törölte az előző gradienseket, megelőzve a nem kívánt akkumulációt.
6. **Előrehaladás:** A `outputs = model(data)` művelettel számítottam ki a modell által predikált értékeket.
7. **Veszteség számítása:** A `loss = criterion(outputs, targets)` utasítással határoztam meg a predikció és a valós címkék közötti eltérést. Ez a metrika eltért modellenként, volt Cross Entropy, meg MSE.
8. **Visszaterjesztés:** A `loss.backward()` hívással számítottam ki a gradienseket minden tanítható paraméterre.
9. **Optimalizálás:** Az `optimizer.step()` paranccsal frissítettem a modell paramétereit a gradiens értékek alapján.
10. **Monitorozás:** A tanulási folyamat nyomon követéséhez minden 100. (vagy 50.) batch után kijeleztem az aktuális állapotot:
`if batch_idx % 100 == 0:`
`print(f'Epoch:{epoch+1},Batch:{batch_idx},Loss:{loss.item():.4f}')`

A különböző modellek esetében eltérő optimalizálási beállításokat alkalmaztam. A CeNN modellnél 0.01-es, míg a CNN és hibrid modelleknél 0.001-es tanulási rátát használtam az Adam optimalizálóval. A veszteségfüggvények is eltértek: a CeNN-nél négyzetes középhiba (MSE), míg a többinél keresztentropia (CrossEntropyLoss) került alkalmazásra.

A tanítási folyamat során megfigyelhető volt, hogy a modellek veszteségértéke folyamatosan csökkent, ami megfelelő konvergenciát jelzett. (Fig 2) A 30 epoch elegendőnek bizonyult a stabil eredmények eléréséhez.

A folyamatot mindhárom esetben figyelemmel végigkísértem, és a következő következtetést vontam le. Mivel a CNN esetén minden epoch után a test dataset-tel meg is néztem (Fig 2b), hogy mennyire hatékony, folyamatosan láttam korrelációt aközött, hogy a training dataset-en mennyit téved, és a test dataset-en mennyi a hiba.

```
Epoch: 29, Batch: 50, Loss: 0.3820, Accuracy: 0.8906
Epoch: 29, Batch: 100, Loss: 0.4393, Accuracy: 0.8750
Epoch: 29, Batch: 150, Loss: 0.5519, Accuracy: 0.8281
Epoch: 29, Batch: 200, Loss: 0.2818, Accuracy: 0.9219
Epoch: 29, Batch: 250, Loss: 0.4767, Accuracy: 0.8438
Epoch: 29, Batch: 300, Loss: 0.2885, Accuracy: 0.9219
Epoch: 29, Batch: 350, Loss: 0.4896, Accuracy: 0.8594
Epoch: 29, Batch: 400, Loss: 0.3687, Accuracy: 0.8906
Epoch: 29, Batch: 450, Loss: 0.2621, Accuracy: 0.9219
Epoch: 29, Batch: 500, Loss: 0.3818, Accuracy: 0.8906
Epoch: 29, Batch: 550, Loss: 0.3212, Accuracy: 0.9062
Epoch: 29, Batch: 600, Loss: 0.3866, Accuracy: 0.8906
Epoch: 29, Batch: 650, Loss: 0.3533, Accuracy: 0.9062
Epoch: 29, Batch: 700, Loss: 0.3396, Accuracy: 0.9062
Epoch: 29, Batch: 750, Loss: 0.3327, Accuracy: 0.9219
Epoch: 30, Batch: 0, Loss: 0.1774, Accuracy: 0.9531
Epoch: 30, Batch: 50, Loss: 0.1823, Accuracy: 0.9531
Epoch: 30, Batch: 100, Loss: 0.2740, Accuracy: 0.9219
Epoch: 30, Batch: 150, Loss: 0.2063, Accuracy: 0.9531
Epoch: 30, Batch: 200, Loss: 0.3956, Accuracy: 0.8906
Epoch: 30, Batch: 250, Loss: 0.2873, Accuracy: 0.9062
Epoch: 30, Batch: 300, Loss: 0.3908, Accuracy: 0.8906
Epoch: 30, Batch: 350, Loss: 0.2038, Accuracy: 0.9531
Epoch: 30, Batch: 400, Loss: 0.2114, Accuracy: 0.9531
Epoch: 30, Batch: 450, Loss: 0.1701, Accuracy: 0.9531
Epoch: 30, Batch: 500, Loss: 0.5614, Accuracy: 0.8281
Epoch: 30, Batch: 550, Loss: 0.3801, Accuracy: 0.8906
Epoch: 30, Batch: 600, Loss: 0.2480, Accuracy: 0.9375
Epoch: 30, Batch: 650, Loss: 0.2774, Accuracy: 0.9219
Epoch: 30, Batch: 700, Loss: 0.4436, Accuracy: 0.8594
Epoch: 30, Batch: 750, Loss: 0.2464, Accuracy: 0.9375
```

(a) CeNN tanulás vége

```
Epoch: 28, Batch: 100, Loss: 0.0683
Epoch: 28, Batch: 200, Loss: 0.0676
Epoch: 28, Batch: 300, Loss: 0.0878
Epoch: 28, Batch: 400, Loss: 0.0681
Epoch: 28, Batch: 500, Loss: 0.0660
Epoch: 28, Batch: 600, Loss: 0.0616
Epoch: 28, Batch: 700, Loss: 0.0702
Epoch 28 Tesztpontosság: 92.49%
Epoch: 29, Batch: 100, Loss: 0.0736
Epoch: 29, Batch: 200, Loss: 0.0594
Epoch: 29, Batch: 300, Loss: 0.0706
Epoch: 29, Batch: 400, Loss: 0.0659
Epoch: 29, Batch: 500, Loss: 0.0609
Epoch: 29, Batch: 600, Loss: 0.0652
Epoch: 29, Batch: 700, Loss: 0.0598
Epoch 29 Tesztpontosság: 91.98%
Epoch: 30, Batch: 100, Loss: 0.0463
Epoch: 30, Batch: 200, Loss: 0.0531
Epoch: 30, Batch: 300, Loss: 0.0564
Epoch: 30, Batch: 400, Loss: 0.0621
Epoch: 30, Batch: 500, Loss: 0.0658
Epoch: 30, Batch: 600, Loss: 0.0604
Epoch: 30, Batch: 700, Loss: 0.0585
Epoch 30 Tesztpontosság: 92.40%
```

(b) CNN tanulás vége

```
Epoch: 28, Batch: 0, Loss: 0.0769
Epoch: 28, Batch: 100, Loss: 0.1254
Epoch: 28, Batch: 200, Loss: 0.1225
Epoch: 28, Batch: 300, Loss: 0.0684
Epoch: 28, Batch: 400, Loss: 0.1082
Epoch: 28, Batch: 500, Loss: 0.0621
Epoch: 28, Batch: 600, Loss: 0.0434
Epoch: 28, Batch: 700, Loss: 0.0655
Epoch: 29, Batch: 0, Loss: 0.0395
Epoch: 29, Batch: 100, Loss: 0.0380
Epoch: 29, Batch: 200, Loss: 0.0291
Epoch: 29, Batch: 300, Loss: 0.0632
Epoch: 29, Batch: 400, Loss: 0.0638
Epoch: 29, Batch: 500, Loss: 0.1050
Epoch: 29, Batch: 600, Loss: 0.0911
Epoch: 29, Batch: 700, Loss: 0.0645
Epoch: 30, Batch: 0, Loss: 0.0548
Epoch: 30, Batch: 100, Loss: 0.0504
Epoch: 30, Batch: 200, Loss: 0.0496
Epoch: 30, Batch: 300, Loss: 0.0294
Epoch: 30, Batch: 400, Loss: 0.0203
Epoch: 30, Batch: 500, Loss: 0.0696
Epoch: 30, Batch: 600, Loss: 0.1455
Epoch: 30, Batch: 700, Loss: 0.0719
```

(c) Hybrid model tanulás vége

2. ábra. A három model tanulásának utolsó epoch-jai

3.6. Mentés

A modellek súlyainak mentése kulcsfontosságú a gépi tanulási folyamatban, mivel megőrzi a hosszú és erőforrás-igényes tanítás eredményeit. Enélkül a notebook/program bezárásakor elveszne minden betanított súly, és újra kellene kezdeni a teljes tanítási folyamatot, ami jelentős idő- és erőforrás-veszteséget jelentene.

A projektben nagyjából a következő módon mentettük a modelleket:

```
torch.save('model_state_dict': model.state_dict(), 'model.pth')
```

A ResNet-18 esetében a mentett fájl jelentősen nagyobb, mivel tartalmazza az előre betanított súlyokat is. Ezek a súlyok komplex képfelismerési feladatokra lettek optimalizálva, és a transfer learning révén jelentősen javítják a modell teljesítményét.

A PyTorch `state_dict()` mechanizmusa a modell összes tanulható paraméterét (súlyok, bias-ok) és a futási állapotot (pl. optimalizáló állapota) is elmenti, így később pontosan ugyanabban az állapotban folytatható a tanítás vagy használható inferenciára a modell.

A mentett modellek később egyszerűen betölthetők a `torch.load()` függvénnyel, majd a `load_state_dict()` metódussal alkalmazhatók az új modell példányra.

4. Kiértékelés

A kiértékelés maga az alábbi függvény segítségével történt:

```
1 def evaluate_model(model, model_name):
2     correct = 0
3     total = 0
4     with torch.no_grad():
5         for images, labels in testloader:
6             images, labels = images.to(device), labels.to(device)
7             outputs = model(images)
8
9             # CeNN specialis ertekeles
10            if isinstance(model, CeNN):
11                expected = torch.ones_like(outputs)
12                expected[labels == 0] = -1
13                correct_dist = torch.mean((outputs - expected)**2, [1,2,3])
14                incorrect_dist = torch.mean((outputs - (-expected))**2, [1,2,3])
15                correct += (correct_dist < incorrect_dist).sum().item()
16
17            # CNN/Hibrid standard ertekeles
18            else:
19                _, predicted = torch.max(outputs.data, 1)
20                correct += (predicted == labels).sum().item()
21
22            total += labels.size(0)
23
24    print(f'{model_name} pontossaga: {100 * correct / total:.2f}%')
```

Az `evaluate_model` függvény a modellek tesztalmazon való teljesítményét méri. A `torch.no_grad()` gradiens számítás nélkül végigiterál a tesztadatokon, és modell-specifikus kiértékelést végez.

- CeNN esetén távolság-alapú metrikát alkalmaz: a kimenet és a várt érték (1 vagy -1) közötti négyzetes távolságot hasonlítja az ellentétes érték távolságához. Helyes a predikció, ha közelebb van a várt értékhez.
- CNN és hibrid modelleknél standard osztályozási logikát használ: a legnagyobb valószínűségű osztályt (`torch.max`) hasonlítja a valódi címkéhez.

A függvény végül kiszámítja és kiírja a pontosságot: $(\frac{\text{helyes predikciók}}{\text{összes minta}} \times 100)\%$. A két különböző kiértékelési módszer a modellek eltérő kimeneti formátumából adódik: a CeNN távolságokat, míg a CNN és hibrid modellek osztályvalószínűségeket modelleznek.

Ezt lefuttattam mindhárom betöltött modellre, majd egy kicsit hosszadalmas számolás után a 3. ábrán látható értékeket kaptam.

```
[ ] evaluate_model(cenn_model, "CeNN")  
⇒ CeNN pontossága: 90.02%  
  
[ ] evaluate_model(cnn_model, "CNN")  
⇒ CNN pontossága: 92.40%  
  
▶ evaluate_model(hybrid_model, "Hibrid modell")  
⇒ Hibrid modell pontossága: 91.45%
```

3. ábra. Eredmények

Az eredmények azt mutatják, hogy a hagyományos CNN modell teljesített a legjobban, míg a differenciálegyenlet-alapú CeNN valamivel alacsonyabb pontosságot ért el. A hibrid megközelítés, amely ötvözi a két módszert, köztes eredményt produkált, ami arra utal, hogy a CeNN előfeldolgozás részben hasznos jellemzőket emel ki, de önmagában a ResNet architektúra hatékonyabb a CIFAR-10 osztályozási feladaton.

Az igazat megvallva azt vártam, hogy javítani fog az eredményen a celluláris hálózat által előfeldolgozott kép, de úgy tűnik, hogy jelen kísérletben nem ez történt. Persze elképzelhető, hogy hosszabb trainelés (több epoch) után legyőzné, és persze az is, hogy teljesen megfordulnak az eredmények.

Az elképzeléseim szerint a konvolúciós hálózat jobban teljesít, és ez jelen esetben be is igazolódott. Persze igaz, hogy a CeNN csak egyszintes volt, míg a CNN egy előre tanított, igen komoly modell újrahasznosítása, ráadásul augmentációval. Ezért sem akartam többet belőle, hogy ne legyen túl nagy előnye.

5. Összegzés

A projekt során három különböző neurális hálózati megközelítést hasonlítottam össze a CIFAR-10 képosztályozási feladaton. A kiértékelés alapján a hagyományos konvolúciós neurális háló (ResNet-18) teljesített a legjobban 92.40%-os pontossággal, míg a differenciálegyenlet-alapú Cellular Neural Network (CeNN) 90.02%-ot ért el. A két megközelítés hibrid kombinációja 91.45%-os eredményt hozott.

Bár kezdetben arra számítottam, hogy a ResNet architektúra felülmúlja a CeNN-t – ami be is igazolódott – azt hittem, a hibrid megközelítés mindkettőnél jobb teljesítményt fog nyújtani. Az eredmények mégis értékes betekintést adtak a különböző modelleszaládok erősségeibe és korlátaiba. Elégedett vagyok a kísérlet kimenetével, hiszen általa jelentősen közelebb került hozzám mind a konvolúciós, mind a celluláris neurális hálózatok világa. A projekt során szerzett tapasztalatok és a modellek viselkedésének mélyebb megértése alapja lehet későbbi tanulmányaimnak, különösen a neurális architektúrák és dinamikus rendszerek területén.