Választott feladat: SSL in the wild [1]

A feladatunk egy önfelügyelt tanulási módszernek (SSL) a kiválasztása, implementálása és tesztelése volt, kifejezetten olyan adatokon tanítva és tesztelve ami nem az ImageNet adatkészlethez tartozik. Mivel ezeket az algoritmusokat általában az ImageNet-en való előtanításra használják, így ezek a módszerek túl vannak illesztve specifikusan erre az adatkészletre nézve. A feladat során azt vizsgáltuk, hogy egy másik adathalmazon tanítva milyen pontosság érhető el. A kiértékeléshez több különböző tanítását végeztük ugyanannak a felépítésű hálónak, majd ezeket hasonlítottuk össze.

A csapat tagjai:

- Püspök-Kiss Balázs (BL6ADS)
- Czifrus Hanna (H2MJ2U)
- Farkas Réka (IXAGK7)

Adatkészlet:

Az adatkészletet a Kaggle-ről [2] választottuk ki. A cél egy olyan kép adatkészlet választása volt, ami kellően nagy számú képet tartalmaz és lehetséges vele tanítani, tesztelni és ellenőrizni egy SSL algoritmust.

Adatkészlet bemutatása:

A kiválasztott adatkészlet a BIRDS 525 SPECIES-IMAGE CLASSIFICATION [3] lett. Ez 525 madárfajról tartalmaz 89885 képet előre tisztítva és besorolva. Az adatkészlet 3 részre bontott (tanító, tesztelő és validáló halmazokra). Minden adatkészlet külön alkönyvtárakat tartalmaz a különböző fajtákhoz, változó számú fájllal. Minden kép 224 x 224 x 3 típusú JPG és csak egyetlen madár található rajta. Általában a madár a pixelek legalább 50 %-át foglalja el. A képeken kívül egy CSV fájlt is tartalmaz az adatkészlet, amiben az egyes képek csoportosítása, a képek elérési útvonala és címkézése szerepel.

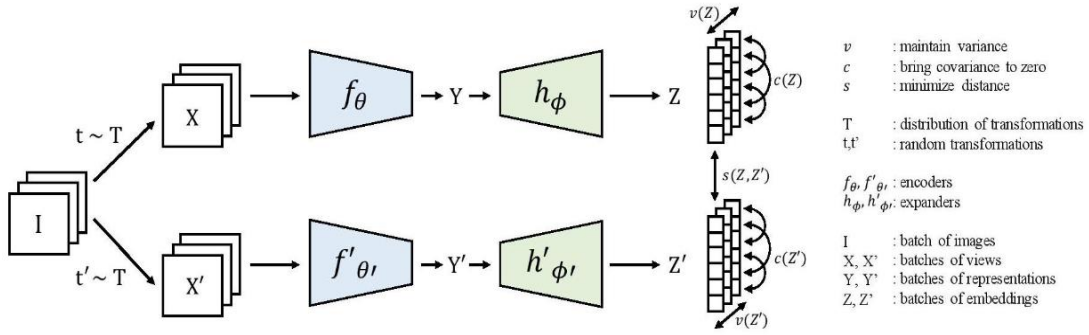
- Tanító halmaz mérete: 84635
- Teszt halmaz mérete: 2625
- Ellenőrző halmaz mérete: 262

Tanítási metódusok:

A választott SSL metódus:

A jelenleg meglévő önfelügyelt tanítási módszerek közül a VICReg (Variance-Invariance-Covariance Regularization) módszert választottuk. A módszer egy olyan veszteségfüggvény használatán alapul, amely három tagból áll (ahogy a módszer neve is mutatja):

- Invariancia: átlagos négyzetes távolság a beágyazási vektorok között.
- Variancia: a beágyazás változóinak szórását egy adott küszöbérték felett tartja.
- Kovariancia: a beágyazási változók közötti kovarianciákat igyekszik nullázni, ezzel megakadályozva az összeomlást.



1. A VICReg felépítése

A módszer pontos leírása, illetve a használt veszteség függvény képlete megtalálható a cikkben. [4]

Munka leírása:

A feladatot párhuzamosan implementáltuk Colab-ban futattható notebook-ok formájában illetve lokálisan és konténerizációban lefutattható Python fájlokban. A feladatunkban előforduló hasonló futattási módok miatt egy konfigurációs réteget használtunk, a Facebook Meta Hydra framework-jét. Az SSL és az osztályzóréteg tanítását, a különböző osztályzók futásának konfigurálást, a paraméter optimalizáció során beállított paraméter változtatásokat, optimalizációkat mind ennek segítségével végeztük.

Az adatkészlet előkészítése:

Elsőként az adatkészlet kiválasztása volt a feladat. A kereséshez több online elérhető adatbázist néztünk meg. Ezekben igyekeztünk olyan adatkészletet keresni, ami képkészletét tekintve hasonló az ImageNet, és subset-jeihez, de nem ahhoz tartozik. Nehézséget jelentettek a nagy és tetszetős minták, amik ellenben rosszul voltak dokumentálva. Előfordult, hogy egy tetszetős kép-adatkészletről csak mások kommentje alapján derült ki, hogy valójában egy részkészlete az ImageNetnek.

A kiválasztott “madaras” adatkészletet az első `notebooks/Data_preparation.ipynb` fájlban töltöttük le és vizsgáltuk meg. Kipróbáltuk, hogy hogyan kell egy saját Dataset osztályt

létrehozni, és bár ezt ebben a notebook-ban benne hagytuk, a valódi tanítás során a LightlyDataset segítségével töltöttük be a modellekbe.

A tényleges tanításnál, a `notebooks/Training_and_testing.ipynb` fájlban, az eredeti train-test-valid csoportosításokkal haladtunk. Az egyes Dataset-ek betöltésekor különböző transzformációkat használtunk. Mivel a feladat SSL tanítás volt egy osztályozó réteggel kiegészítve így 3 transzformációt definiáltunk a Dataset-ek betöltéséhez:

- `train_vicreg_transform`: A kiválasztott SSL előtanításhoz tartozó transzformációs halmaz.
- `train_classifier_transforms`: A Torchvision által biztosított transzformációs halmaz az osztályozó tanításához. A tanítás miatt random torzítás értékekkel dolgozunk.
- `test_transforms`: A Torchvision által biztosított transzformációs halmaz az osztályozó teszteléséhez és ellenőrzéséhez. A tesztelés és az ellenőrzés miatt állandó értékekkel dolgozunk

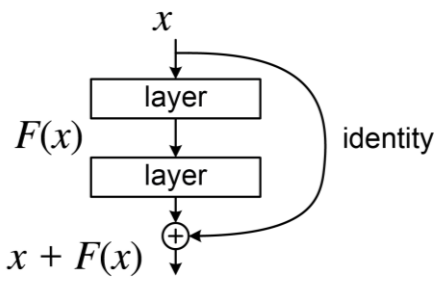
Ezekkel a transzformációkkal módosítottuk és állítottuk össze a 4 Dataset-et (a LightlyDataset osztályból):

- `dataset_train_vicreg`: A kiválasztott SSL tanításhoz szükséges adatkészlet. Csak train adatra van szükségünk ennél a hálónál.
- `dataset_train_classifier`: Az osztályozó tanításhoz szükséges adatkészlet.
- `dataset_test`: Az osztályozó teszteléséhez szükséges adatkészlet, a sorrendjük állandó, nem keverjük meg.
- `dataset_valid`: Az osztályozó ellenőrzéséhez szükséges adatkészlet, a sorrendjük állandó, nem keverjük meg.

A két megvalósított LightningModule osztály a kiválasztott SSL metódus, a VICReg (Variance-Invariance-Covariance Regularization) és az osztályozó metódus.

A tanítás folyamata:

A általunk betanított neurális háló egy ResNet-18 gerincből és egy osztályozóból épül fel.



2. Példa egy reziduális blokkra

A ResNet [5] olyan konvolúciós neuronhálók családja, ahol a hálók úgy nevezett reziduális blokkokból épülnek fel. A blokkok lényege, hogy a bennük lévő rétegek után számolt gradienshez a blokk végén hozzáadjuk a blokk bemenetét. Ez a módszer segít kezelni a

gradiensek eltűnésének vagy robbanásának problémáit, ezáltal mélyebb neuronhálók is hatékonyabban taníthatók. Az 2. ábrán látható egy egyszerű reziduális blokk felépítése.

Az általunk használt ResNet-18 architektúra a nevéből adódóan egy 18 rétegű ResNet típusú háló. A neuronháló első 17 rétege egy-egy konvolúciós réteg, majd ezeket követi egy teljesen összekapcsolt, vagy más néven lineáris réteg.

2-féle felépítése neuronhálóra végeztünk tanításokat:

Mindkét neurális háló első 17 rétege megegyezik a ResNet-18 architektúra első 17 rétegével, ami után kétféle osztályozót kötöttünk. Az egyik esetben ez az osztályozó egyetlen lineáris rétegből állt (a notebook-ban fc_net néven definiált), a másik esetben pedig egy több rétegből álló osztályozót definiáltunk (a notebook-ban deep_net néven definiált). Ez 3 rejtett lineáris rétegből (nn.Linear) áll, amelyeket a Batch Normalization rétegek (nn.BatchNorm1d) és az aktivációs függvények (nn.ReLU) követnek. Az osztályozó utolsó rétege ebben az esetben is egy, a kimeneti osztályokat meghatározó teljesen összekötött réteg.

5-féle különböző tanítást végeztünk a mindkét neuronhálóra:

- Elsőként egy ImageNet-en előtanított ResNet-18 háló első 17 rétegét továbbtanítottuk önfelügyelt módon a VICReg módszer használatával az általunk választott Birds adathalmazon. Ezután a betanított súlyokat befagyasztva hozzákötöttünk egy lineáris réteget a hálóhoz, majd a keletkezett hálót ismét tanítottuk az adathalmazunkon. A súlyok befagyasztása miatt ez teljesen megfelel annak a módszernek, ha a klasszifikációs réteget önmagában tanítják az adathalmazon, és a tanítás után kötik csak össze az előzőleg tanított konvolúciós rétegekkel.
- A fentebb leírt tanítást a súlyok befagyasztása nélkül is megtettük. Tehát az ImageNet-en előtanított gerincet VICReg-gel továbbtanítottuk, majd ehhez egy lineáris réteget kapcsolva mind a 18 réteget tanítottuk az adathalmazunkon.
- Az SSL metódus használata nélkül, az ImageNet-en előtanított gerincháló súlyait befagyasztottuk, és mögé kapcsolva a saját klasszifikációs rétegünket tanítottuk a hálót az adathalmazunkon.
- Az előbb említett tanítást most is elvégeztük a súlyok befagyasztása nélkül. Tehát az ImageNet-en előtanított gerincháló mögé kötöttük a klasszifikációs rétegünket, és ezt a 18-rétegű hálót tanítottunk a Birds adatain.
- Végül egy előtanítatlan gerinchálóból és a hozzákapcsolt lineáris rétegből álló hálót tanítottuk az adathalmazunkon.

Optimalizáció:

Az hiperparamétereket Optuna segítségével optimalizáltuk az osztályozó hálózathoz.

Az optimalizált értékek halmazába tartozott a learning rate, batch méret, valamint egyéb hálózati paraméterek, például a rejtett rétegek száma és dimenziói is. Az optimalizáció közben alkalmaztunk szűrést (pruning), korai megállással (early stopping), valamint modern mintavételezési algoritmusokat (TPE sampling), hogy minél gyorsabban megtaláljuk az optimális paramétereket a modellhez.

Az eredmény ismertetésében a fully-connected (FC) réteges tanítással ellentétben a mély neurális hálózat (DNN) több paraméterét is optimalizálással határoztuk meg és az alapján tanítottuk az osztályozó modellt.

Kiértékelés:

A kiértékelés során elsőként a pontosság és veszteség metrikákat mértük le mindegyik modell esetén a teszthalmazon. Mindkét metrikára kapott eredményeket megjelenítettük egy-egy oszlopdiagramon az összehasonlíthatóság végett. Egy modell annál jobb, minél jobban közelíti a pontosság érték az egyet (a 100%-os pontosságot), és a veszteség értéke minél jobban közelíti a nullát.

Továbbá minden modell esetén megjelenítettük a teszthalmazon számolt tévesztési mátrixát (confusion matrix). A mátrix egyik tengelye a modell által jósolt osztályt mutatja, a másik tengelye a tényleges osztályokat. Az egyes cellákban azon teszhalmazbeli minták száma szerepel, melyekre a valódi osztály és a prediktált osztály megegyezik a cellához tartozó értékekkel. Tehát a mátrix átlójában szerepelnek azok a cellák, ahol a predikciók helyesek voltak, így az adott modell annál jobb, a tévesztési mátrixának átlójában minél nagyobb értékek, és az átlón kívül minél kisebb értékek szerepelnek. [6]

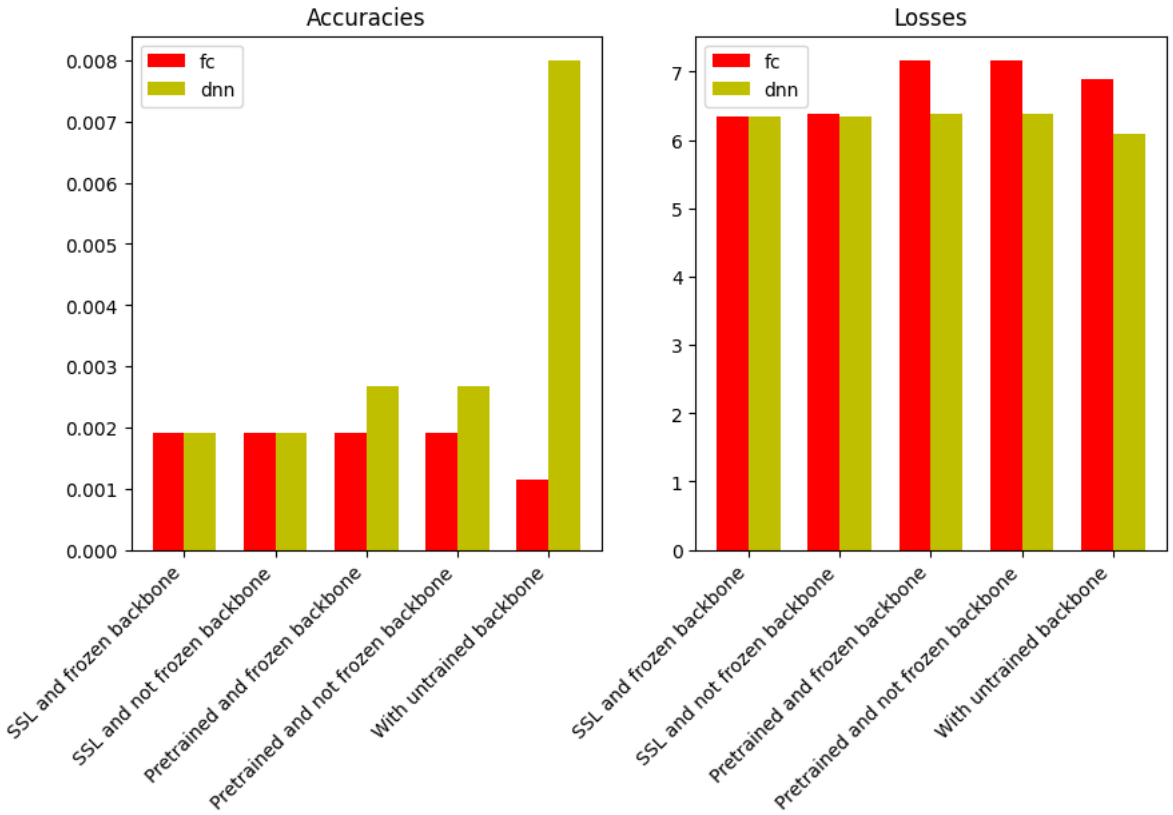
Eredmény:

Az feladathoz tartozó SSL metódust sikeresen megvalósítottuk és beépítettük a rendszerbe, majd tudtunk hozzákapcsolni különböző beállítású klasszifikációs rétegeket. A működést bemutató notebook fájlokban illetve magában a hydra környezetben is azonos működés érvényesül. Igyekeztünk ahhoz ragaszkodni, hogy konzisztensen tartsuk a változásokat annak ellenére, hogy más a rétegződése a két nézetnek.

A konténerizáció okozott nehézséget, mivel sokkal lassabb volt a tanítás, mint bármelyikünk eszközén. Beletettük a VICReg és az osztályozó tanítását és az egész kiértékelését, azonban túl jól nem használható.

Az adathalmaz nagysága, illetve az erőforrások hiánya jelentős gondot okozott számunkra a tanítások kikísérletezése során.

Az eredményekből sajnos azt kaptuk, hogy hiába igyekeztünk optimalizálni a tanítást, nem változott a kapott pontossága az osztályozásnak. Ahogy a 3. ábra is mutatja a modellek teszthalmazon vett pontosságát az optimalizálás során sem igazán tudtuk növelni, illetve a veszteséget sem sikerült csökkenteni.



3. Pontosság és veszteség metrikák

Jövőbeni változtatási lehetőségek:

Az optimalizációk pontosításán felül lehetséges lenne kipróbálni milyen eredményeket kapnák másfajta augmentációkra, az adatok másfajta előkészítésre. A tanítási halmazon végzett képtranszformációkból melyikek lennének a hatékonyak, mekkora mértékig befolyásolnák a tanítást. További összehasonlításokat lehetne végezni más SSL metódusokkal végzett előtanításokra.

Irodalomjegyzék

[1] „Self-supervised learning “in-the-wild",” 03 2023.. [Online]. Available: https://edu.vik.bme.hu/mod/forum/discuss.php?d=1570. [Hozzáférés dátuma: 06. 10. 2023].

[2] „Kaggle,” [Online]. Available: https://www.kaggle.com/. [Hozzáférés dátuma: 09. 12. 2023].

[3] „BIRDS 525 SPECIES- IMAGE CLASSIFICATION,” 09. 12. 2023. [Online]. Available: https://www.kaggle.com/datasets/gpiosenka/100-bird-species.

[4] „VICReg: Variance-Invariance-Covariance Regularization For Self-Supervised Learning,” 2022. [Online]. Available: https://arxiv.org/abs/2105.04906. [Hozzáférés dátuma: 07. 12. 2023.].

[5] X. Z. S. R. J. S. Kaiming He, „ResNet,” 10 12 2015. [Online]. Available: https://arxiv.org/abs/1512.03385. [Hozzáférés dátuma: 09 12 2023].

[6] „MulticlassConfusionMatrix,” [Online]. Available: https://lightning.ai/docs/torchmetrics/stable/classification/confusion_matrix.html#multiclassconfusionmatrix. [Hozzáférés dátuma: 10 12 2023].

[7] PyTorch, „resnet18,” [Online]. Available: https://pytorch.org/vision/main/models/generated/torchvision.models.resnet18.html. [Hozzáférés dátuma: 09 12 2023].