Fehérjék másodlagos szerkezetének prediktálása Temporal Convolutional Network segítségével

Fischer Kornél

2021 december

1 Az alapfeladat

Ebben a félévben a Lewis Moffat és David T. Jones [1] cikkéből indultunk el. A cikkben ismertetik az S4PRED nevű programot, melynek célja fehérjék másodlagos szerkezetének megjóslása abban az esetben, amikor nem ismert a fehérjének homológja.

A legjobb módszer a másodlagos fehérje szerkezet megjóslására homológiára épül, ezekkel elérhető akár 84%-os pontosság is. Enélkül csak $71 - 72\%$ körüli eredményt szoktak elérni. A pontosságot Q_3 értékben mérik, ahol minden aminosavnak három lehetséges szerkezet van: alfa-hélix, béta-redő illetve kanyarok. Létezik 8 osztályú klasszifikáció is, de gépi tanulási szempontból a két feladat nem sokban különbözik. A Q_3 érték számításakor azt veszik figyelembe, hogy a 3 osztályú klasszifikáció esetén hány aminosav másodlagos szerkezete lett helyesen eltalálva.

Célunk a félév első felében a cikk eredményeinek reprodukálása volt, később pedig a háló architektúrájának kicserélése, és ezzel esetleg javítani az eredményükön.

Az [1] cikkből felhasználtuk a kétszeres tanítás technikáját is. Az olyan fehérjéknek a száma, aminek a másodlagos szerkezete helyesen van annotálva, nagyságrendekkel kisebb, mint a még nem feltérképezett fehérjék száma. Felhasználva, hogy a homológiára épült módszerek pontossága már elég nagynak tekinthető, a szerzők kiválogattak 1,08 millió olyan fehérjét a UniProtKB adatbázisból, amire igaz, hogy nem homológja egyik olyan fehérjének sem, ami a CB513 eleme. A CB513 egy 513 fehérjét tartalmazó adatbázis, a másodlagos szerkezet prediktálásakor egy sztenderd teszt-adatbázis. A kiválogatott fehérjéknek prediktálták a másodlagos szerkezetét a PSIPRED V4 [2] programmal, ami nagy pontosságú becsléseket ad. Ezzel a módszerrel sikerült nagy mennyiségű annotált fehérjét előállítani, amikkel könnyebb lett tanítani a modellünket. Az így szerzett adaokat hívjuk pszeudo-címkézett fehérjéknek. A szerzők ezt kiegészítették 10.000 olyan fehérjével, amit a Protein Database adatbázisből szedtek a PISCES szerver segítségével, és ismert volt a valódi másodlagos szerkezetük. Továbbá

félretettek 500 további fehérjét az utóbbi halmazból, validációs célokra. A modell az első halmazból tanult, és használták hozzá a validációs halmazt. A futásidő a végső modellek esetén 48-72 óra alatt futottak. A cikk szerzői használták az úgynevezett fine-tuning eljárást [3], amikor a már betanult a modellt még egy epoch-on keresztül tanítják, immáron a valósan címkézett fehérjéken. Ez az eljárás az ő esetükben majdnem 5 százalékot javított a modell pontosságán, így érték el a 75%-ot.

Mi nem használtuk a második adatbázist, csak az elsővel tanítottuk a modelljeinket. Későbbi javítási lehetőség lesz a fine-tuninghoz használt adatbázis megszerzése és használata.

A félév első felében reprodukáltam a cikk néhány eredményét. Megszereztem a teszteléshez használt CB513 nevű adatbázist. A cikk szerzői úgy írták meg a programot, hogy egyszerre csak egy fehérjét tudott feldolgozni, és csak egy speciális fájlformátumban tudta beolvasni az inputot, és az outputot is abban adta meg. Ezeket átírtam, hogy képes legyen a program CSV (Comma Separated Values) formátumban tárolt adatbázist is kezelni. Így megismételtük a méréseket, majd megmértük a Q_3 pontszámokat. A végső eredményünk 74.2% volt, ez megközelítette az ő eredményüket.

Miután végeztünk a reprodukálással, a cikket alapul használva, elkezdtük egy saját modell kipróbálását. Megszereztük a cikkben használt pszeudo-címkézett fehérjéket az aminosav szekvenciájukkal együtt. A cikkben használt háló az LSTM egy változatát, az AWD-LSTM-et használta. A tavalyi önálló projekt tapasztalata is megmutatta, hogy ez a modell nagy pontosságot tud elérni hasonló feladaton, de azt is tudjuk, hogy az Temporal Convolutional Networks alkalmanként jobban teljesít hasonló feladatokra, mint az LSTM, ezért megéri kipróbálni itt is.

2 A Temporal Convolutional Network

A temporal convolutional network (TCN) [4] architektúrák egy családja, melyeknek a megkülönböztető jegyei közé tartozik, hogy a prediktáláshoz csak az eddig látott elemeket használja, továbbá egy RNN-hez hasonlóan bármilyen hosszú sorozatot képes beolvasni, és ugyanolyan hosszú outputot visszaadni. Sajátossága továbbá, hogy akár nagyon messze képes legyen visszanézni a múltba prediktáláskor. Ehhez reziduális blokkokat tartalmazó mély hálókat fogunk használni, és dilated convolution-t.

2.1 Causal Convolutions

A feladat formalizáltan, adott egy $X_0, X_1, \dots X_T$ bemenet, és szükséges egy $Y_0 \dots Y_T$ kimenet, melyeket egyesével adjuk meg. Amikor valamilyen t-re Y_t-t megadjuk, akkor csak a $X_0 \dots X_t$ inputokat használhatjuk. Ez a causal megszorítás. Szóval egy modell az egy olyan f függvény, amire $f : X^{T+1} \to Y^{T+1}$, és

$f(X_0, \ldots, X_T) = y'_0, \ldots, y'_T$ az említett függéseket betartva. Célunk megtanulni egy olyan f függvényt, ami egy L veszteségfüggvényt minimalizál az elvárt kimenet és az általunk adott kimenet között:

$$minL(Y_0 \ldots Y_T, f(x_0, \ldots, x_T)) \tag{1}$$

Ahhoz, hogy a TCN elérje az első elvárt követelményt, ahhoz egydimenziós konvolúciót használ (fully connected network, FCN). Minden rejtett réteg ugyanolyan hosszú, mint a bemenet, továbbá használunk egy filterhossznál eggyel rövidebb hosszú zero padelést a szekvenciák elején. A második feltétel teljesítéséhez a TCN causal convolutions-t használ, vagyis olyan konvolúciót, ami betartja a causal megszorítást. A módszer hátránya, hogy ahhoz hogy a tényleg elég sok előismerete legyen a modellnek a becsléshez, nagyon mély hálót kell készíteni. Ezért kellett sokféle technikát használni konvolúciós hálókból, hogy jól legyen implementálva.

2.2 Dilated Convolutions

Egy egyszerű casual convolution csak a hálózat méretében lineárisan tud visszanézni az eddigi inputon. Ez probléma, ha tényleg nagyon sok előismeretre van szükségünk a prediktáláshoz. Ezért használjuk a dilated convolutions-t, aminek lényege hogy a receptív mező exponenciálisan nagy lesz. Formálisan, ha egy egy-dimenziós szekvencia inputunk van, $x \in R^n$ és egy f filterünk, $f : 0, \ldots k-1 \rightarrow R$, egy dilated konvolúció operátor F egy s szekvencia-elemen így definiálható:

$$F(s) = (x * df)(s) = \sum_{t=0}^{k-1} f(i) x_{s-d*i} \tag{2}$$

ahol d a dilation faktor, k a filter mérete, $s - d * i$ pedig arra vonatkozik, hogy a múltba tekintünk. A dilation lényege tehát hogy fix lépéseket teszünk minden szomszédos beolvasandó input közé. Ha $d = 1$, akkor a dilation convolutionből egy egyszerű konvolúció lesz. Ha d nagyobb, akkor tetején lévő outputunk egy nagyobb méretű bemenetből származik, ezáltal megnövelte a convolution háló receptív mezejét.

Így két módon tudjuk növelni a TCN receptív mezejét, vagy növeljük a k filter méretet, vagy a d dilation faktor méretét, ahol az effektív előzménye egy egy rétegnek $(k-1)d$. A háló méretében d exponenciálisan nő, vagyis a háló i. szintjén $d = 2^i$. Emiatt biztos lesz, hogy lesznek filterek, amik minden inputot elérnek, és egyszerre lesz nagyon nagy az effektív előzményük.

2.3 Reziduális kapcsolatok

Egy reziduális blokkban két ág van, egyiken transzformációk egy sorozatát hajtjuk végre $(F(x))$, majd az eredményt összeadjuk az eredeti bementi x-szel, és a végén van egy aktivációs függvényünk.

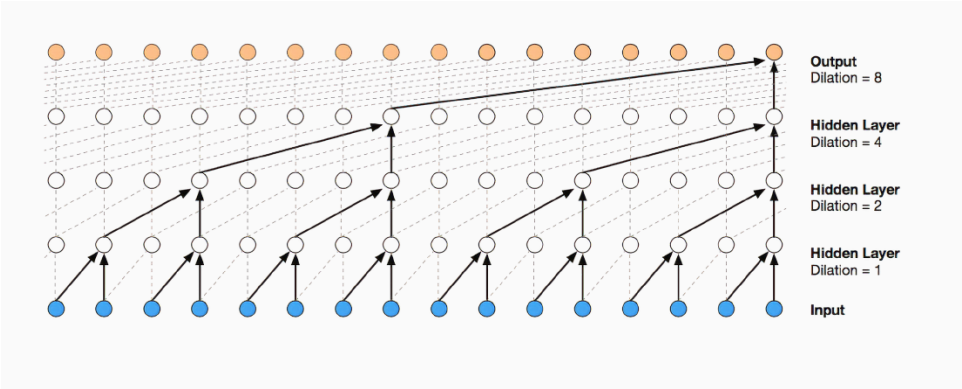

Figure 1: A dilation működése

Vagyis a rétegben azt kell megtanulnia a modellnek, hogy módosítja az identitás függvényt. Mély hálókban ez a változtatás hasznosnak bizonyul.

A TCN receptív mezejének mérete függ a háló mélységétől (n), a filter méretétől (k), a dilation faktorától (d), ezért a nagyméretű TCN-ek stabilizálása komplex feladat. Például, ha a jóslatunk függhet akár egy 2^{10} méretű inputtól, aminek a dimenziószáma is magas, akkor akár 10 rétegre is szükségünk lehet. Minden rétegben szerepelhet több filter is. Az eredeti cikkben a szerzők egy reziduális blokkban kétrétegnyi dilated causal convolution-t használtak, valamint ReLu aktivációs függvényt. Emellet weight normalizationt, valamint spatial dropoutot, aminek lényege, hogy minden tanító lépésnél egy teljes csatornát kinullázunk. Azért, hogy a bemenet és a kimenet hossza ugyanakkora legyen, használnunk kell egy 1x1-es konvolúciót is, hogy az elemenkénti összeadás ugyanakkora méretű tenzorokkal történjen.

A TCN modellnek előnyei közé tartozik a párhuzamosság, a konvolúciók egyszerre történhetnek, mert ugyanazokat a filtereket használják a rétegek. A tanítás és az értékelés közben is, egy hosszú bemeneti szekvencia egyszerre feldolgozható, nem kell részletekben csinálni, mint egy RNN-nél. A receptív mező mérete könnyen változtatható több paraméter segítségével is, például lehet növelni a dilated konvolúciós rétegek számát, nagyobb dilation faktort használhatunk, vagy a filtert is növelhetjük. Ezért könnyebben tudjuk kontrollálni a modell memória igényét.

A tanítás során a gradiensek számolása sem okoz akkora nehézséget, mint egy RNN esetén. Különösen hosszú szekvenciák esetén a TCN kevesebb memóriát használ a tanítás során, mint például az LSTM. Ennek oka, hogy az LSTM például sok részeredményt tárol, amik a későbbi cellákban kellhetnek. A TCN-nél viszont egy filter elég egy rétegre, ezért kevesebb memóriát használ. Fontos továbbá, hogy az RNN-hez hasonlóan különböző hosszú szekvenciákat is képes feldolgozni.

A modell hátránya, hogy kiértékeléskor egy sima RNN-nek elég megkapnia X_t-t és egy rejtett h_t állapotot tárolnia, hogy jósoljon. Ekkor az egész eddigi előzmény egy fix hosszú h_t-ben van tárolva, de a TCN esetén az egész eddigi szekvenciára szüksége lehet, emiatt ilyenkor nőhet a memória igénye.

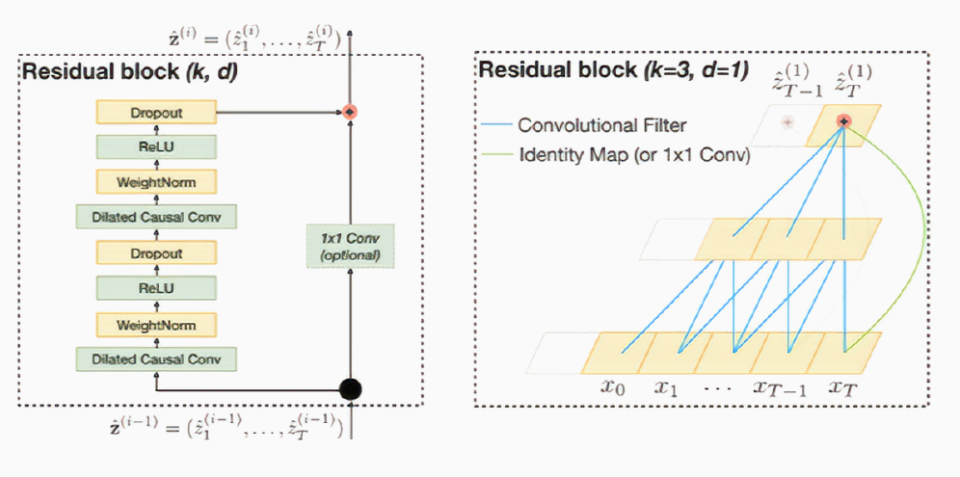

Figure 2: Egy reziduális blokk felépítése

2.4 Acausal Convolutional Neural Network (A-TCN)

Abban az esetben, ha nem csak a múltbeli információt használhatjuk a prediktáláskor, hanem a jövőbelit is, akkor használhatunk acasual TCN-t [5], ami a TCN egy változata. A mi esetünkben ez utóbbi modellt használtuk.

3 A feladat megoldása

A fehérje szekvenciák elkódolása hasonlóan ment a tavalyi LSTM-es feladathoz. Az aminosavakat kódoltuk egy 1 és 24 közti számmal, meghagyva a 0-t egy különleges karakter számára a padeléshez. A másodlagos szerkezet osztályokat is kódoltuk, 0-tól 3-ig hasonlóan. Tavaly kiderült, hogy a szótár betűinek permutálása nem befolyásolta az LSTM teljesítményét, így most is azt feltételeztük, hogy nem lesz érdemi különbség. Hasonlóan ismét figyeltünk arra, hogy a fehérjék hossz szerinti sorrendben legyenek, így elérve, hogy a lehető legkevesebb padelés kelljen használni egy-egy batchben. A tavalyi adatbázisban adtunk egy felső korlátot, eldobtunk minden 1500-nál hosszabb szekvenciát. Idén ezt kihagytuk, ugyanis ebben az adatbázisban gyakoriak voltak az akár 5-6000 hosszú protein szekvenciák is.

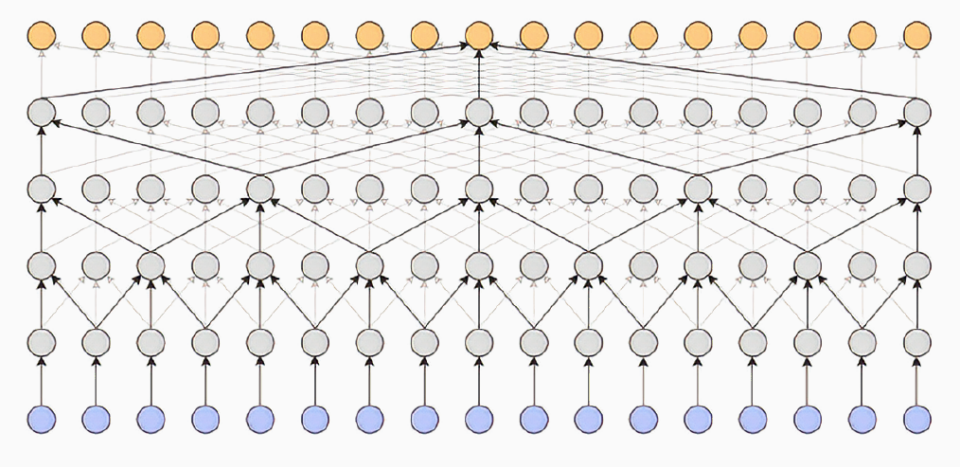

Figure 3: Acasual TCN felépítése

Ebben a feladatban a klasszifikálást nem egyszer kell megejteni fehérjénként, hanem minden egyes aminosavat külön kell prediktálni, a kimenet ugyanolyan hosszú, mint a bemenet. Emiatt figyelni kellett, hogy a pontosság számolásakor ne nézzük a padelt részeket, csak a fehérjén menjünk végig. Az értékeléskor a korábban említett Q_3 score-t néztük, vagyis az eltalált aminosavak számát.

3.1 Hiperparaméter optimalizálás

Az alapvető paraméterek, amiket állítanunk kellett: beágyazási dimenzió, a blokkok mérete, illetve azok száma, az ablakméret vagy filter, a feature-size, valamint a learning-rate és a batch-size.

A beágyazási dimenziót 16-ra állítottuk. Mivel a szótár mérete 25 volt, érdemes volt ennél kisebb számot kipróbálni, és ezen jól működtek a modellek. A dilation base adta meg, hogy mi legyen a hatvány alapjában. Ezt próbáltuk magasabbra állítani, de nem okozott különösebb javulást a végeredményben, így maradt végül kettő. Hasonló okból lett a feature size 64. Olyan szám kellett, amit oszt a beágyazási dimenzió. Az optimalizálás során sok számot kipróbáltunk 32 és 512 között, de ezek során úgy tűnt, érdemi javulás nem történik, ha túl magasra állítjuk ezt az értéket. A tanulási idő ellenben drasztikusan megnőtt, akár tízszeresére is. A learning rate-t és a batch-size-t egyszerre álíltottuk, klasszikusan fordítottan arányosan változtatva őket. A tapasztalataink miatt állítottuk végül be a learning rate-t 10^{-4}-re, és a batch méretet 256-ra. Optimizernek Adamot használtunk.

Sokféle konfigurációt kipróbáltunk, a tapasztalatunk az volt, hogy a modellek hatékonysága sokszor csak kis mértékben változott, drasztikus eltérés ritka volt,

ha megfelelően sok tanító adatot használtunk. A legtöbb modell train adathalmazon mért pontossága 82 és 84 százalék közé esett.

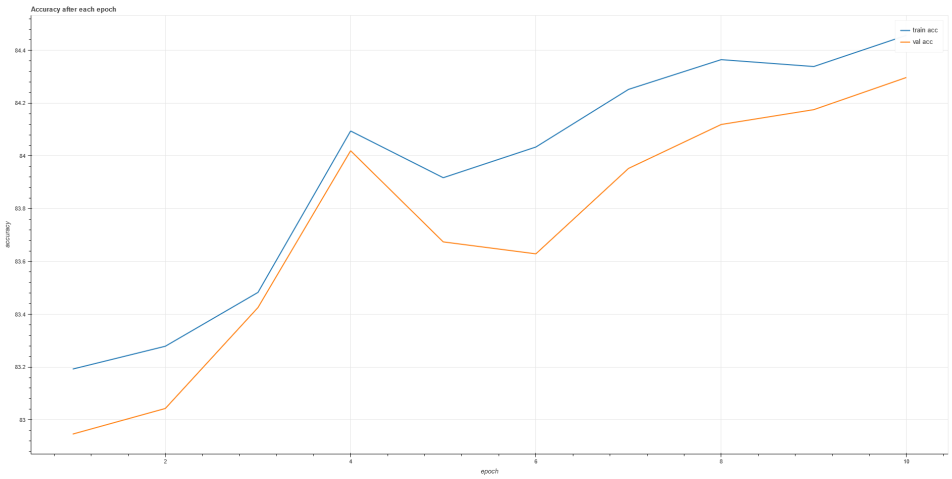

Figure 4: A pontosság alakulása a tanítás első 10 epochja során a legjobb modell esetén

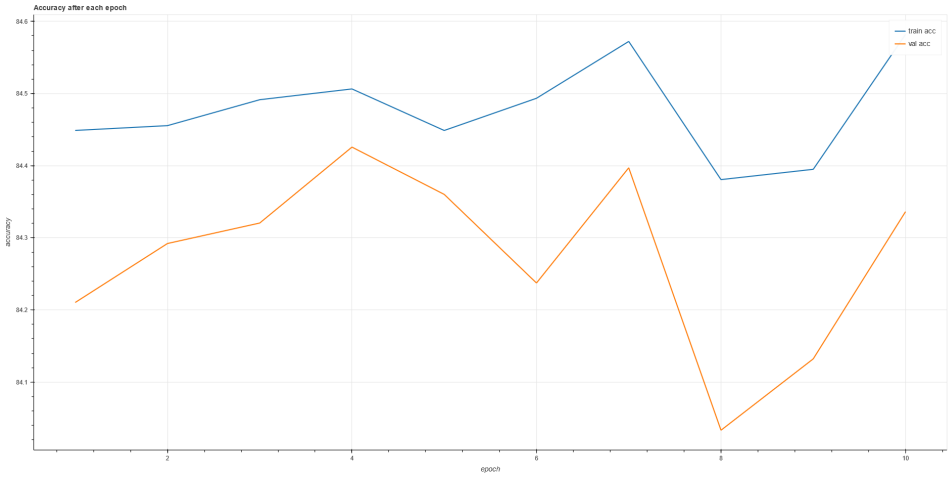

Figure 5: A pontosság alakulása a tanítás következő 10 epochja során a legjobb modell esetén

A tanítások után minden modellt kipróbáltunk a CB513 adathalmazon is. Az itteni protein szekvenciák jelentősen eltérnek mind a tanító, mind a validációs

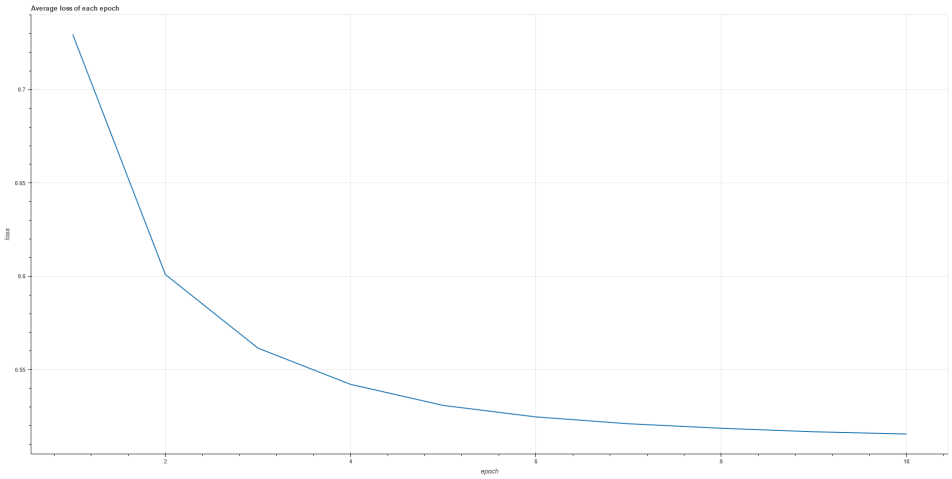

Figure 6: A veszteség alakulása a tanítás során

adathalmazban lévő szekvenciáktól. Ezen a pontosság 69 és 71 százalék között volt. Érdekes volt, hogy az itteni legjobb pontosságot egy olyan modellel értük el, ami még nincs optimalizálva. Túl nagyra volt hagyva a beágyazási dimenziója. Viszont mikor azt csökkentettük, a teljesítmény is romlott. Vagyis még további optimalizálásoknak helye van. Egy olyan modellel, amin a paraméterek jobban be voltak állítva, csak megközelíteni sikerült ezt az eredményt, túlszárnyalni nem.

A legjobb modell esetén az első 10 epoch során gyorsan tanult a modell, és úgy tűnt még érdemes lesz tovább tanítani, de a következő 10 epoch során a pontosság jelentősen nem nőtt, csak ingadozott. A két kép technikai okok miatt nem került egy ábrára, különböző alkalmakkor folyt a tanítás.

A tanítások során mi csak a pszeudo-címkézett adathalmazt használtuk. Ezen a cikk szerzői végül az LSTM-mel 71 százalékot értek el a CB513-on, de egy sokkal lassabb modellel. Ők utána egy fine-tuning fázissal elérték a 75 százalékot, mi viszont nem szereztük be azt az adathalmazt, amit ők ott használtak, így eddig csak ezt a részeredményt tudjuk összemérni. Itt pedig sikerült az 1 dimenziós konvolúciós hálóval ugyanolyan jó eredményt elérni, mint nekik.

3.2 Továbbfejlesztési lehetőségek

Észrevettük néhány mérésnél, hogy a modell nagyobb pontosságot ért el a validációs halmazon, mint a tanító halmazon. Ennek egy lehetséges oka, hogy nem független egymástól ez a két adathalmaz. Ezt korábban nem vettük figyelembe, mert az adathalmaz előállítása során a [1] cikk szerzői rendkívül odafigyeltek arra, hogy az adathalmazban benne levő fehérje szekvenciák egyáltalán

ne hasonlítsanak a CB513-ban található fehérjékre. Továbbá az adatok alapvetően az UniClust30 nevű adatbázisból származnak, ahol azok a fehérjék, amik legalább 30 százalékben megegyeznek, egy klaszterbe vannak sorolva, a klaszterekből pedig csak egy fehérjét választunk. Úgy tűnt, ez elég lesz arra, hogy ne legyen túl hasonló a két adathalmaz, de lehetséges, hogy nem így alakult. Emiatt egy javítás lehetne, ha mi magunk klasztereznénk ismét a meglévő egy millió fehérjét, 10 osztályba, és ezek közül kettőt választanánk validációs halmaznak, a többit pedig tanítónak hagynánk.

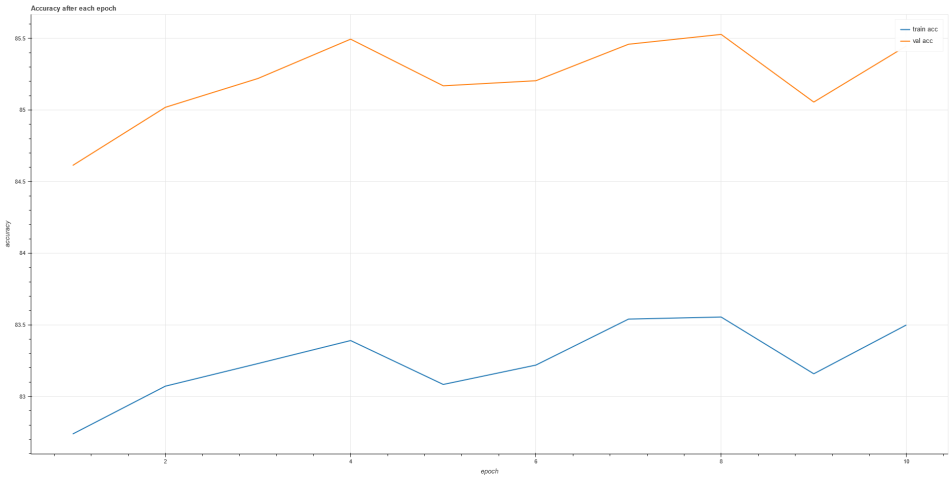

Figure 7: Néhány esetben validációs halmazon nagyobb pontosságot értünk el, mint a tanítón

Érdemes lesz megszerezni a valódi címkékkel ellátott adathalmazt, és azzal tovább javítani a modellt. A tapasztalatok szerint egy ilyen fine-tuning tud javítani a model teljesítményén, várhatóan itt is fog segíteni.

További lehetőség, hogy ha a modellben használt Adam optimizert kicseréljük például gradient boostingra. Lehet, hogy ettől nő a modell pontossága.

4 Irodalom

[1] Lewis Moffat, David T Jones: Increasing the accuracy of single sequence prediction methods using a deep semi-supervised learning framework, Bioinformatics, Volume 37, Issue 21, 1 November 2021, Pages 3744–3751 (2021)

[2] Buchan,D.W. and Jones,D.T. (2019) The psipred protein analysis workbench: 20 years on. Nucleic Acids Res., 47, W402–W407.

[3] Devlin,J. et al. (2019) BERT: pre-training of deep bidirectional transformers for language understanding. Proceedings of the 2019 Conference of the

North American Chapter of the Association for Computational Linguistics, pp. 4171–4186.

[4] Bai, S., Kolter, J.Z., Koltun, V.: An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv preprint arXiv:1803-.01271 (2018)

[5] Alqahtani, S., Mishra, A., Diab, M.: Efficient convolutional neural networks for diacritic restoration. In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). pp. 1442–1448 (2019)

[6] Ábra: Wavenet: A Generative Model for Raw Audio Synthesis `https://medium.com/@prantosh.das97/wavenet-a-generating-model-for-raw-audio-synthesis-610242071c12`