EÖTVÖS LORÁND TUDOMÁNYEGYETEM

TERMÉSZETTUDOMÁNYI KAR

Objektum felismerés gépi tanulási módszereinek vizsgálata

Szakdolgozat

Bottyán Martin

Matematika Bsc

Matematikai elemző szakirány

Témavezető

Lukács András

Számítógéptudományi Tanszék

Budapest

2019

Köszönetnyilvánítás

Elsősorban szeretnék köszönetet mondani konzulensemnek, Lukács Andrásnak a konzultációkon nyújtott segítségért, valamint az útmutatásaiért, amik nélkül nem jöhetett volna létre ez a szakdolgozat. Továbbá köszönettel tartozom családomnak és barátnőmnek a végtelen támogatásért és türelemért.

Tartalomjegyzék

1. Bevezetés **4**

2. Dimenziócsökkentés lineáris módszerekkel **6**

2.1. Szinguláris érték felbontás és számítása 6

2.2. Dimenziócsökkentés SVD-vel 7

3. Felhasznált prediktív modellek **8**

3.1. Benchmark modellek 8

3.1.1. Döntési fa 8

3.1.2. Véletlen erdő 10

3.2. Boosting 11

3.2.1. AdaBoost 11

3.2.2. XGBoost 13

3.3. Neurális háló 15

3.3.1. Sűrű háló 16

3.3.2. Konvolúciós neurális háló 20

4. Alkalmazás ruhatípusok felismerésére **24**

4.1. Adathalmaz és feladat 24

4.2. Benchmark modellek teljesítménye 25

4.3. Boosting modellek teljesítménye 28

4.4. Sűrű háló teljesítménye 30

4.5. Konvolúciós neurális háló teljesítménye 32

4.5.1. Adatbővítés transzformációkkal 34

4.6. Programkód 36

1. fejezet

Bevezetés

Köszönhetően az informatika rohamos fejlődésének az utóbbi években az elektronikus adatok generálási és tárolási költsége jelentősen csökkent. Ennek következménye, hogy a feldolgozható adatok mennyisége drámaian megnőtt. Így egyre komplikáltabb feladat lett a nagyobb adathalmazokból történő releváns információk kinyerése és ezekből következtetések levonása. Ennek hatására alakult ki az adattudomány és a hozzá rendkívül közel álló gépi tanulás.

A gépi tanulás a mesterséges intelligencia egyik legfontosabb technológiája, amely lehetővé teszi rendszerek számára, hogy automatikusan tanuljanak és javuljon teljesítményük a rendelkezésre álló adatokból anélkül, hogy explicit beleavatkoznánk. Meghatározó trenddé vált a mesterséges intelligencia utáni érdeklődés mind egyéni és vállalati szinten is. Ez az egyik motiváció, ami inspirálja a fejlesztőket az új és új gépi tanuló algoritmusok megalkotására. A gépi tanulás egyik fontos feladata az osztályozás, más néven klasszifikáció. Egy klasszifikációs modell ismert osztálybesorolású adatokon képes arra, hogy meghatározza, megjósolja további adatok osztálybesorolását, ahol az osztályok előre megadott diszkrét értékek.

A szakdolgozatom első részében ismertetem a feladatok előkészítésében felhasználható dimenziócsökkentésre egy lineáris algebrai módszert, a szinguláris érték felbontást vagy röviden SVD-t (singular value decomposition). Rendkívül széleskörű alkalmazhatósága miatt nagyon népszerű mátrix transzformáció. Használható adatok dimenziócsökkentésére, képek tömörítésére és zaj csökkentésére egyaránt.

Ezt követően bemutatom a dolgozatban használt klasszifikációs modelleket. Dönté-

si fát, és az ezekből felépülő véletlen erdőket, a hibrid modellek egy további osztályát, a boosting algoritmust, valamint a sűrű és konvolúciós neurális hálót.

Végül ezeket a módszereket fogom alkalmazni ruhákat ábrázoló képek osztályozására. Az elvárásoknak megfelelően, a legjobban teljesítő konvolúciós neurális háló esetében különböző adatbővítési technikákkal próbálok tovább javítani a modell pontosságán.

2. fejezet

Dimenziócsökkentés lineáris módszerekkel

Tipikus probléma gépi tanuló algoritmusoknál, hogy a bemeneti értékek dimenziója rendkívül magas. Képek esetén ez gyakran több milliós nagyságrendű, de tabuláris adatoknál sem ritka a több ezres attribútumszám, ami lényegesen megnöveli az algoritmusok futási idejét. A dimenzió csökkentése nem csak a számítási idő rövidítése miatt hasznos, hanem az adatok sűrítése miatt a zaj kiszűrése miatt is. Erre a problémára ad megoldást a szinguláris érték felbontás, vagy röviden SVD (Singular Value Decomposition). Az SVD bemutatásához a [8] forrást dolgoztam fel.

2.1. Szinguláris érték felbontás és számítása

2.1.1. Tétel. *(Szinguláris érték felbontás) Legyen $A \in \mathbb{R}^{n \times m}$ mátrix, ekkor felírható az alábbi alakban:*

$$A = U \Sigma V^T \tag{2.1}$$

ahol az U $n \times n$-es ortogonális mátrix, oszlopvektorai az AA^T sajátvektorai, a V $m \times m$-es ortogonális mátrix és oszlopvektorai az $A^T A$ mátrix sajátvektoraival egyeznek meg. A Σ egy $n \times m$-es mátrix, főátlójában az AA^T sajátértékeinek nem negatív négyzetgyökei - az úgynevezett szinguláris értékek - állnak. A többi eleme nulla.

A felbontás kiszámításának alapötlete, hogy az A mátrixot beszorozzuk transzponált-

jával, így a következőt kapjuk:

$$A^T A = V\Sigma^T U^T U\Sigma V^T. \tag{2.2}$$

Mivel U egy ortogonális mátrix, ezért $U^T = U^{-1}$, így

$$A^T A = V\Sigma^T \Sigma V^T \tag{2.3}$$

ami kiszámítható, hiszen ez $A^T A$ spektrálfelbontása, mert $A^T A$ egy szimmetrikus mátrix, V ortogonális mátrix $A^T A$ sajátvektorai illetve $\Sigma^T\Sigma = diag(\sigma_1^2, \sigma_2^2, ..., \sigma_m^2)$, amik $A^T A$ sajátértékei. Az U mátrix hasonlóan hiszámítható AA^T felírásával.

2.2. Dimenziócsökkentés SVD-vel

A (2.1) felbontásban szereplő Σ mátrix diagonális elemei csökkenő sorrendben helyezkednek el, azaz $\sigma_1 \geqslant \sigma_2 \geqslant ...\sigma_r > \sigma_{r+1} = ... = \sigma_m = 0$. Legyen $1 < t < m$ a csökkentett dimenzió száma, ekkor hagyjuk el az $m - t$ legkisebb szinguláris értéket és a szinguláris értékekhez tartozú U mátrix beli oszlopokat. Legyen

$$A_t = U_t \Sigma_t \tag{2.4}$$

$n \times t$-s, A t dimenzióra lecsökkentett mátrixa, ahol U_t $n \times t$-s U mátrix és Σ_t $t \times t$ diagonális mátrix, főátlókban a t legnagyobb szinguláris értékekkel.

Képek kisebb rangú approximációjához az Eckart-Young tételben szereplő becslést fogom alkalmazni.

2.2.1. Tétel. *(Eckart-Young) Legyen $A \in \mathbb{R}^{n\times m}$ egy r rangú mátrix, $1 \leqslant k \leqslant r$. Az A mátrix legjobb k rangú közelítése 2-es normában az*

$$A_k = \sum_{i=1}^{k} u_i \sigma_i v_i^T, \tag{2.5}$$

távolságuk pedig

$$||A - A_k||_2 = \sigma_{k+1}. \tag{2.6}$$

3. fejezet

Felhasznált prediktív modellek

Legyen adott egy T tanító adathalmaz n darab megfigyeléssel

$$T = (x_1, y_1), ..., (x_n, y_n), \tag{3.1}$$

ahol $x_i \in \mathbb{R}^m$ az input magyarázó változók vektora, y_i pedig a célváltozó ami egy diszkrét értékkészleten belül vesz fel értékeket, például $\{1, 2, ..., K\}$. K az osztályok számát jelöli. Általában feltételezzük, hogy a tanító adatok egymástól függetlenek, és egy ismeretlen valószínűségi eloszlásból származó azonos eloszlású minták. A cél az, hogy találjunk egy olyan $C(x)$ osztályozó függvényt a tanító adathalmaz alapján, ami egy új, eddig még nem látott bemeneti x értékhez hozzá rendeli a hozzá tartozó osztály címkét, y-t.

3.1. Benchmark modellek

Benchmark modelleknek nevezzük azokat a modelleket, amelyek egy viszonyítandó standard megoldást nyújtanak a szofisztikáltabb algoritmusok jóságának mérésére. Ilyen modellek például a k-legközelebbi szomszéd, döntési fa, illetve annak a hibrid modell változata, a véletlen erdő. Ezek közül a döntési fát és a véletlen erdőt fogom ismerteni a [1] és a [2] források alapján.

3.1.1. Döntési fa

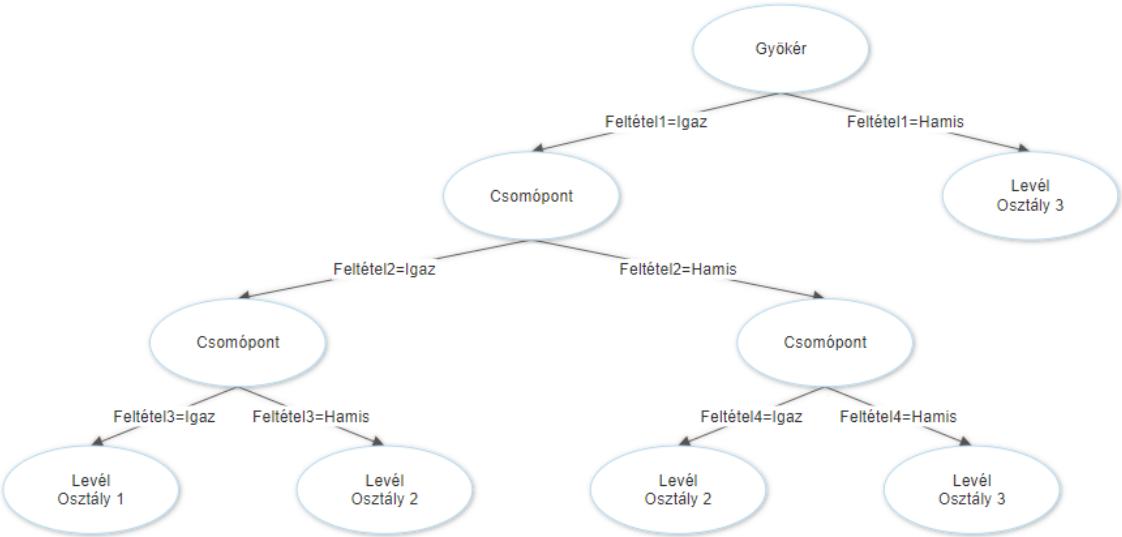
Egy döntési fa generálásakor kezdetben egyetlen csúcsunk van, a gyökér, amely tartalmazza a teljes tanító adathalmaz elemeit. Kiválasztjuk azt az attribútumot, amely a leg-

jobban szeparája az adathalmazt, ez lesz a gyökérben a vágó attribútum. Az attribútum összes értéke tartozik pontosan egy elágazáshoz. Az algoritmus a minták minden egyes partíciójánál rekurzívan ugyanezt az eljárást alkalmazza. Abban az esetben, ha csúcsban szereplő minták mind ugyanabból az osztályból származnak, akkor az a csúcs levél lesz.

A vágó attribútum kiválasztásához egy bizonyos kötségfüggvény optimális értékének meghatározása a feladat. A két leggyakrabban használt költségfüggvény az információnyereség és a Gini index.

CART algoritmus

A CART (*Classification and Regression Tree*) egy bináris fa építő algoritmus, azaz olyan fát generál, amely minden csúcspontjából maximum 2 részfa nyílhat. A vágáshoz szükséges attribútumok kiválasztásához osztályozó fa esetén a Gini indexet veszi alapul.



3.1. ábra. Példa bináris döntési fára

3.1.1.1. Definíció:. *Legyen T egy adathalmaz, amely n mintát tartalmaz. Az n minta legyen besorolva K különböző osztályba. Ekkor a Gini index a következő:*

$$G(T) = 1 - \sum_{j=1}^{K} \left(\frac{|T_j|}{|T|} \right)^2, \tag{3.2}$$

ahol $|T_j|$ a j osztályba sorolt minták száma.

Minden csúcspontban - a bináris szeparációk miatt - két diszjunkt halmazra bontjuk a benemő T adathalmazt, egy n_1 elemszámú T_1 halmazra és egy n_2 elemszámú T_2 halmazra. Ezt úgy akarjuk megtenni, hogy a legjobban törekedjünk arra, hogy az osztályok szerinti homogenitást elérjük. Ehhez szükséges minden attribútum esetén kiszámítani az adott csomópontban a hozzá tartozó súlyozott Gini indexek összegét, ami:

$$G(T) = \frac{n_1}{n}G(T_1) + \frac{n_2}{n}G(T_2). \tag{3.3}$$

Az összes attribútum közül nevezzük ki vágónak azt, amelynél a kapott $G(T)$ érték a legkisebb.

Az algoritmusnak több megállási kritériuma is létezik. Ha az alábbi feltételek valamelyike teljesül, akkor leáll az algoritmus:

- Az adott csomópont minden eleme ugyanabba az osztályba tartozik.
- Nem marad több attribútum, ami alapján tovább lehetne vágni az adathalmazt.
- A fa mélysége elérte a meghatározott limitet.
- A csomópontban az elemek száma nem éri el a megadott korlátot.

A kritériumok között szerepelt mélységi limit és csomópont elemszámának alsó korlátjának meghatározása azért fontos, mert erősen befolyásolják a modell teljesítményét, ugyanis a döntési fa legnagyobb hátránya a túltanulás. Ami annyit jelent, hogy a modell túlságosan jól illeszkedik a tanuló adathalmaz elemeire és megtanulja a bennük lévő zajt, ezért egy tesztelő adathalmazon gyengébben osztályoz.

3.1.2. Véletlen erdő

A véletlen erdő algoritmus több döntési fából felépülő osztályozó módszer. Alapötlete, hogy több száz vagy akár több ezer egyszerűsítetten generált döntési fák kimeneteleiből többségi szavazás alapján klasszifikál. Döntési fák építéséhez tipikusan a fentebb ismertetett CART algoritmust használja. Ahhoz, hogy ezekből akár több száz különbözőt generáljunk, ahhoz többszörös mintavételezést fogunk használni. Az eredeti T tanuló adathalmazunkból visszatevéses mintavételezéssel alkossunk m darab részhalmazt. Majd

ezeken a mintavételezett adathalmazokon alkalmazzuk a CART algoritmust a fa építéséhez az eredeti változathoz képest annyi különbséggel, hogy minden egyes csomópontban a vágó attribútum kiválasztásához nem az összes közül választunk, hanem az összes változó véletlenszerűen kiválasztott részhalmazából. Ezek után egy adatrekord osztályozása a következő képlettel történik:

$$\frac{1}{m}\sum_{i=1}^{m} D_i(x). \tag{3.4}$$

Ahol m a véletlen erdő fáinak száma, D_i pedig az i-edik mintavételezésből épített döntési fa általi predikció.

Számos ok miatt is jó benchmark modellnek számít több feladatnál is, mert jól skálázható, azaz könnyen alkalmazható nagy adathalmazokra is a mintavételezés és a változó szelekció miatt. A fa építéséhez szükséges paraméterek mellett csak a fák száma a fontos paramétere, ezért nem igényel sok paraméter keresést, ami miatt rövid idő alatt felépíthető modell.

3.2. Boosting

A boosting egy olyan hibrid klasszifikációs módszer, ami a véletlen erdőhöz hasonlóan több gyengén teljesítő modell kombinációjából egy pontosabb együttes modellt épít. Különbség azonban, hogy egy boosting modell szekvenciálisan építi a gyenge klasszifikátorokat, míg a véletlen erdő esetén egymással párhuzamosan épít. Mindegyik gyenge modellt, vagy más néven alap osztályozót az őt megelőző osztályozó modell hibái szerint módosított adathalmazon tanítja be. Ami történhet a tanuló adathalmazok súlyainak változtatásával az AdaBoost esetén illetve a reziduális hibákra történő illesztéssel az XGBoost esetén. Az AdaBoost és az XGBoost bemutatása során a [3] és [7] forrásokat használtam fel.

3.2.1. AdaBoost

Legyen T egy tanító adathalmaz (3.1) szerint. Mindegyik megfigyeléshez rendeljünk hozzá egyenlő súlyt,

$$w_i = \frac{1}{n}, \qquad i = 1, 2, ..., n.$$

Legyen M az alap osztályozók száma, ami maga az iterációk száma is egyben. Leggyakrabban az úgynevezett *decision stump*-ot használják alap modellként, ami egy egyetlen csomópontból álló döntési fa. Az első iterációnál a kezdeti súlyú tanuló adathalmazon tanítsunk klasszifikátort. Ezután teszteljük a modellt a tanító adatokon és kiszámoljuk a súlyozott átlagos hibát:

$$err^{(m)} = \frac{\sum_{i=1}^{n} w_i \chi(y_i \neq C^{(m)}(x_i))}{\sum_{i=1}^{n} w_i}. \tag{3.5}$$

Ezt a súlyozott átlagos hibát használjuk fel a következő paraméter kiszámításához:

$$\alpha^{(m)} = \log \frac{1 - err^{(m)}}{err^{(m)}} + \log(K-1) \tag{3.6}$$

ami a modell teljesítményét leírő paraméter. A következő lépésben módosítjuk a tanuló adathalmaz pontjaihoz tartozó súlyokat a alábbi képlet alapján:

$$w_i \leftarrow w_i \exp\Big(\alpha^m \chi\big(y_i \neq C^{(m)}(x_i)\big)\Big), \tag{3.7}$$

ami a következő iterációhoz tartozó tanuló adathalmaz súlyai lesznek. Egyértelműen látszódik, hogy amennyiben az aktuális modell rosszul klasszifikált egy adatpontot, akkor az iteráció ezen szakazában megnöveljük a hozzá tartozó súlyt. Az iteráció legvégén újra lenormáljuk a tanuló adatok súlyát, hogy az összegük - a kezdetben megadott súlyokhoz hasonlóan - 1 legyen:

$$w_i \leftarrow \frac{w_i}{\sum_{l=1}^{n} w_l}. \tag{3.8}$$

A ciklus végén az M modellből felépülő C adaboost klasszifikáció predikcióit meghatározó képlet a következő:

$$C(x) = \operatorname*{argmax}_k \sum_{m=1}^{M} \alpha^{(m)} \chi(C^{(m)}(x) = k). \tag{3.9}$$

Amennyiben a $C^{(m)}$ alap tanuló pontossága a tanuló adathalmazon $\frac{1}{K}$, azaz megegyezik véletlenszerű találgatással, akkor $\alpha^{(m)} = 0$, és az m modell nem befolyásolja a predikciós képlet kimenetelét. Értelemszerűen, ha $C^{(m)}$ nagyobb pontossággal klasszifikál, akkor $\alpha^{(m)}$ növekszik.

Az adaboost egyik nagy hátránya, hogy a rosszul klasszifikált adatrekordok súlyának iteratív növelése miatt kifejezetten érzékeny a mintában esetlegesen elõforduló kilógó (outlier) értékekre.

3.2.2. XGBoost

Legyen T egy tanító adathalmaz (3.1) szerint. Az XGBoost (eXtreme Gradient Boosting) az Adaboosthoz hasonlóan szintén egy iteratívan döntési fákat generáló együttes modell, amelynek predikciója a következő:

$$\hat{y}_i = \sum_{k=1}^{M} D_k(x_i), \tag{3.10}$$

ahol M a felépített döntési fák száma, $D_k(x_i)$ pedig a k-adik iterációban felépített döntési fa általi predikció. A t-edik iterációhoz tartozó predikció ezek alapján

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} D_k(x_i). \tag{3.11}$$

Ahogy minden osztályozó algotitmusnál, itt is szükség van egy veszteségfüggvényre, amit a tanulás során minimalizálunk. Az XGBoost esetén a veszteségfüggvény két függvény összege:

$$Obj = L + \Omega, \tag{3.12}$$

ahol L a tanító adathalmazon egy szokásos veszteségfüggvény, ami azt méri, hogy a tanuló adatokon mennyire prediktív a modell. A dolgozatban használt L a többosztályú klasszifikációs feladatoknál használt logisztikus veszteségfüggvény:

$$L = -\frac{1}{T} \sum_{i=1}^{T} \sum_{j=1}^{K} y_{ij} \log(p_{ij}). \tag{3.13}$$

Az Ω pedig egy szabályozó függvény, aminek a célja, hogy a modell jó általánosító erővel bírjon és elkerülje a túltanulást.

$$\Omega = \gamma S + \frac{1}{2}\lambda \sum_{j=1}^{S} w_j^2 \tag{3.14}$$

Ahol S a döntési fában lévő levelek száma, w_j a fa j-edig leveléhez rendelt predikciós érték.

Az $Obj(y, \hat{y})$ veszteségfüggvény optimalizálásához a gradiens lejtés (gradient descent) módszert alkalmazzuk, aminek a lényege, hogy kiszámoljuk $\partial_{\hat{y}} Obj(y, \hat{y})$ minden iterációban és $\hat{y}$-t módosítjuk a gradiens alapján a legnagyobb csökkenés irányába, hogy Obj-t minimalizáljuk. A veszteségfüggvényünk a t-edik iterációban legyen:

$$Obj^{(t)} = \sum_{i=1}^{T} L(y_i, \hat{y_i}^{(t)}) + \sum_{i=1}^{t} \Omega(D_i) = \sum_{i=1}^{T} L(y_i, \hat{y_i}^{(t-1)} + D_t(x_i)) + \sum_{i=1}^{t} \Omega(D_i) \quad (3.15)$$

Mivel a gradiens kiszámolása egy időigényes numerikus feladat, ezért alkalmazzuk a másodrendű Taylor sorfejtését:

$$Obj^{(t)} \simeq \sum_{i=1}^{T} \left[L(y_i, \hat{y}^{(t-1)}) + g_i D_t(x_i) + \frac{1}{2} h_i D_t^2(x_i) \right] + \sum_{i=1}^{t} \Omega(D_i) \quad (3.16)$$

ahol $g_i = \partial_{\hat{y}^{(t-1)}} L(y_i, \hat{y}^{(t-1)})$ és $h_i = \partial^2_{\hat{y}^{(t-1)}} L(y_i, \hat{y}^{(t-1)})$. A korábbi iterációkban már kiszámolt L és Ω függvényértékeket eltávolíthatjuk, azaz a konstansokat. Így az alábbit kapjuk:

$$Obj^{(t)} = \sum_{i=1}^{T} \left[g_i D_t(x_i) + \frac{1}{2} h_i D_t^2(x_i) \right] + \Omega(D_t), \quad (3.17)$$

ami veszteségfüggvényünk az t-edik iterációban. A feladat az, hogy olyan optimális D_t fát építeni minden iterációban, amivel minimalizáljuk $Obj^{(t)}$-t.

A döntési fa építése során két fontos feladat megoldása szükséges, az egyik a levelek predikciós értékeinek meghatározása, a másik pedig a fa struktúrájának megadása. Definiáljunk egy $w_{q(x)}$ függvényt, ahol $q(x)$ jelentse azt a levelet, amelyhez az adatot rendeltük és $w_{q(x)}$ pedig a $q(x)$ levélhez rendelt predikciós értéket jelölje. Továbbá legyen:

$$I_j = \{i | q(x_i) = j\} \quad (3.18)$$

egy indexhalmaz, ami tartalmazza azoknak a tanuló adatpontoknak az indexeit, amelyeket a j-edik levélhez rendeltünk. Írjuk át a (3.17) veszteségfüggvényt a következőre:

$$Obj^{(t)} = \sum_{j=1}^{S} \left[\Big(\sum_{i \in I_j} g_i \Big) w_j + \frac{1}{2} \Big(\sum_{i \in I_j} h_i + \lambda \Big) w_j^2 \right] + \gamma S, \quad (3.19)$$

ahol már a levelek szerint összegzünk, mivel az egy levélhez tartozó adatpontok predikciója megegyezik. Látható, hogy ez w_j-re nézve egy másodfokú egyenlet, ezért az optimális w_j meghatározása egy egyszerű feladat.

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda} \quad (3.20)$$

A levelekhez tartozó optimális predikciós értékeket visszahelyettesítve (3.19) veszteségfüggvénybe:

$$Obj^{(t)} = -\frac{1}{2}\sum_{j=1}^{S}\frac{\left(\sum_{i\in I_j} g_i\right)^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma S, \tag{3.21}$$

A döntési fa felépítéséhez minden csomópontban mohó módszerrel keresünk megfelelő vágást, ami minimalizálja $Obj^{(t)}$-t. Legyen I indexhalmaz azoknak az adatpontoknak az idexeivel, amelyek a csomóponthoz tartoznak, valamint legyen I_L és I_R a csomóponthoz tartozó bal és jobb oldali levélhez rendelt adatpontok indexeinek halmaza. Ekkor $Obj^{(t)}$ csökkenésének az értéke a vágás után:

$$gain = \frac{1}{2}\left[\frac{\left(\sum_{i\in I_L} g_i\right)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{\left(\sum_{i\in I_R} g_i\right)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{\left(\sum_{i\in I} g_i\right)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma \tag{3.22}$$

A legnagyobb csökkenéshez tartozó attribútumot válasszuk a csomópont vágó attribútumává. A döntési fa építését addig folytatjuk, amíg a fa el nem éri a maximális mélységet. Azokban a pontokban, ahol negatív a $gain$ értéke, azokat töröljük a döntési fából.

3.3. Neurális háló

A neurális hálózat egy olyan számítási feladatok megoldására létrejött párhuzamos feldolgozást végző modell, amely eredete a biológiai rendszerekből származtatható. Az idegrendszer tanulmányozása, az idegsejt felépítésének, illetve működésének megismerése indította el azt a gondolkodást, hogy kíséreljünk meg az élő szervezetekben létező, bonyolult rendszerek mintájára létrehozni számító rendszereket.

Egy neurális háló számos egyszerű, de erősen összekapcsolt elemből áll, amiket általában neuronnak, de gyakran csomópontnak vagy egységnek is hívnak. Minden neuron a bemeneti értékeit más, a hálózathoz tarozó neuronokból vagy külső adatforrásból kapja, és ezekből egy kimeneti értéket számol, ami további neuronok bemenetei lehetnek. Abban az esetben ha nem kapcsolódik további neuronokhoz, akkor a modell predikciója a kimeneti értéke. A neuronok rétegesen helyezkednek el.

Ebben a részben először bemutatom egyetlen neuron működését, majd a teljes neurális háló struktúráját. Végül a kimondottan képfeldolgozás és objektum felismerés feladatára összpontosító mély tanuló algoritmust ismertem, a konvolúciós neurális hálót.

3.3.1. Sűrű háló

Perceptron felépítése

A legegyszerűb neurális hálózat egyetlen neuronból áll, mely önmagában kevés számítást tud elvégezni, viszont egymással összekapcsolva azonban komplex struktúrát alkotnak, amivel komolyabb számítások elvégzésére alkalmas. A perceptron fogalmát 1958-ban Frank Rosenblatt [4] alkotta meg.

- Bemenet: Legyen T egy ismert tanuló adathalmaz (3.1) szerint. Mivel Rosenblatt két kimenetelűnek definiálta a perceptront ezért legyen $K = 2$. Az x vektorok dimenziója pedig legyen p.

- Súlyok: Legyen $w = (w_1, w_2, ..., w_p)$ súlyvektor, és mindegyik súly a vele azonos indexű attribútumhoz tartozik. A súlyok ismeretlen paraméterek és a feladat ezen szabad változók meghatározása.

- Csomópont: A perceptron magja, ami egy p dimenziós vektort kap bemenetként. Ez a vektor a fentebb említett p komponensű adatforrásból áll. Továbbá a csomóponthoz tartozik egy b szabad változó. Az b az úgynevezett eltolás vagy torzítás (bias). Az eltolásnak az a szerepe, hogy jobb illesztést biztosítson az adatokra azzal, hogy az aktivációs függvény értékét egy konstans értékre állítja. Az eltoláshoz is tartozik egy súly, amit jelöljünk w_b-nek. A csomópont szerepe összegezni a bemeneti értékeit a hozzájuk tartozó súlyokkal:

$$s(x_i) = \sum_{j=1}^{p} w_j x_{i,j} + w_b b, \tag{3.23}$$

  ahol $i \in (1, 2, ..., n)$ a T tanító adathalmaz i-edig adatrekordja.

- Aktivációs függvény: Egy olyan differenciálható függvény, ami a csomópont $s(i)$ értékét alakítja át olyan értékké, ami a feladatnak megfelelő intervallumba esik. Rosenblatt perceptronjában ez a függvény a lépésfüggvény.

- Kimenet: A perceptron kimenete x_i bemenetre:

$$\hat{y}(x_i) = f(s(x_i)), \tag{3.24}$$

  ahol f az aktivációs függvény.

Sűrű háló felépítése

A sűrű háló rétegekbe szervezett neuronokból áll. Három különböző funkciójú rétegből épül fel a hálózat, bemeneti, rejtett és kimeneti rétegből. A bemeneti réteg fogadja az adatforrást és a benne található neuronok száma megegyezik az adatok attribútumának számával. Semmilyen számítás nem történik az itt elrendezett neuronokban, csupán csak továbbítják az adatot a következő rétegnek. A rejtett rétegből, tetszőleges számú és méretű lehet. Végül egy kimeneti réteg, ahol a neuronok száma klasszifikációs feladat esetén az osztályok száma és mindegyik osztályhoz tartozik egy neuron. Az sűrű háló egy előrecsatolt (*feedforward*) hálózat, ami azt jelenti, hogy az információ áramlás egy irányba történik a bemenettől a rejtett rétegeken keresztül egészen a kimenetig. Egy réteg összes neuronja összeköttetésben áll - az áramlási irányt tekintve - az előző és a következő réteg összes neuronjával.

A sűrű háló alkalmazásának két fő lépése van, a hálózat struktúrájának definiálása, azaz a rétegek számának, a rétegekben használt neuronok számának, és azok aktiválási függvényeinek meghatározása. A másik a hálózat betanítása, ami a súlyok meghatározását jelenti a veszteségfüggvény minimalizálása érdekében.

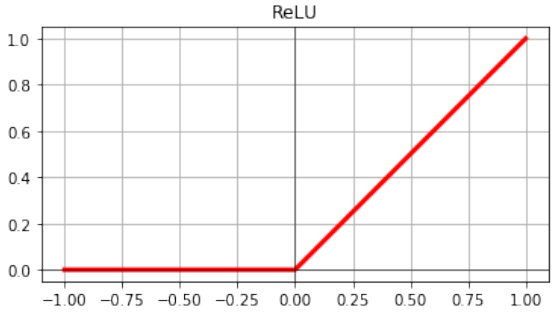
Aktivációs függvény

Mindegyik neuronnak megvan a saját aktivációs függvénye, ami rétegenként (bemeneti értéket leszámítva) változó típusú is lehet. Dolgozatomban két aktivációs függvényt alkalmaztam, az egyik a mély tanuló modelleknél leggyakrabban használt, a ReLU (*Rectified Linear Unit*). Negatív bemenetre 0-t továbbít, pozitív kimenetre pedig a bemeneti értékét, azaz

$$f(x) = max(0, x). \tag{3.25}$$

Észrevehető, hogy 0-ban nem differenciálható, mivel a jobb és bal oldali deriváltak nem egyeznek meg. Ez a gyakrolatban nem jelent gondot a deriváltak numerikus meghatározásánál.

A másik aktivációs függvény, amit használtam, az a *softmax*, amit a kimeneti réteg neuronjai használnak több osztályú klasszifikációs feladat megoldására. A K darab kimeneti rétegbeli neuron kimeneti értékeit lenormálja, ami alapján mindegyik osztályhoz egy



3.2. ábra. A ReLU aktivációs függvény.

valószínűséget rendel. A legnagyobb valószínűséghez tartozó osztály lesz a predikció. A j-edik osztályhoz tartozó valószínűség:

$$f(s)_j = \frac{e^{s_j}}{\sum_{k=1}^{K} e^{s_k}}, \tag{3.26}$$

ahol s_j a j-edig kimeneti összegző csomópont kimenete. Az exponenciális nagyításnak az a szerepe, hogy a kis bemeneti különbségek látványosak lesznek a kimeneti értékek között.

Veszteség függvény

A T tanító adathalmazunkból ismert y osztály attribútumhoz tartozó vektor minden egyes elemét kódoljuk át *one-hot* vagy más néven *dummy* formába. Az átkódolás az eredeti y_i 1 és K közötti számot (kategorikus osztály címkét) egy K dimenziójú vektorkba képezi. Ez a vektor legyen $t = (t_i, ..., t_K)$, ahol $t_{y_i} = 1$ és az összes többi eleme 0. A dolgozatomban használt veszteségfüggvény a kategorikus kereszt entrópia (*Categorical Cross-Entropy*) függvény:

$$L = -\sum_{i=1}^{K} t_i \log(f(s)_i), \tag{3.27}$$

ahol $f(s)_i$ a *softmax* aktivációs függvény (3.26). A cél, hogy L függvényt minimalizáljuk, ezt a súlyok értékeinek a módosításával fogjuk elérni. Ez a folyamat a hálózat tanítása.

Hálózat tanítása

A hálózat tanításának folyamata a következő lépésekből áll. Először szükséges a kezdeti súlyokat megadni. Két réteg közötti súlyok meghatározásához az

$$\mathcal{U}\left(-\sqrt{\frac{6}{|u_{be}|+|u_{ki}|}}, \sqrt{\frac{6}{|u_{be}|+|u_{ki}|}}\right) \tag{3.28}$$

egyenletes eloszlást használtam a szakdolgozatban, ahol $|u_{be}|$ és $|u_{ki}|$ jelenti bemenetként, valamint a kimenteként szolgáló réteg neuronjainak a számát.

Ezt követően a kezdeti súlyokon végigáramoltatjuk a tanító pontokat a hálózaton. A kimeneti $\hat{y}$ értékeket összevetjük a tényleges y kimenetelekkel és a súlyokat pedig megváltoztatjuk a hiba csökkentése érdekében.

Epoch és Batch

Az *epoch* és a *batch* két fontos paramétere a neurális háló tanításának. A *batch* mérete definiálja az egyszerre áramoltatott adatpontok számát mielőtt a hálózat belső paramétereit módosítanánk. A hálózaton szükségszerű többször betanítani a teljes tanító adathalmazt, hogy a súlyok optimálisak legyenek. Az *epoch* száma jelenti, hogy a teljes tanító adathalmazt hányszor áramoltatjuk a hálózaton keresztül.

Adam optimalizálás alkalmazása

A súlyok módosításához egy fejlett gradiens alapű módszer az Adam *(Adaptive Moment Estimation)* optimalizálás. Az Adam módszert a [6] alapján mutatom be.

Osszuk fel az adatpontjainkat $1, 2..., J$ részhalmazra (*batch-re*) és jelöljük a L_j-vel a j adatpontjain számolt veszteségfüggvényt. Továbbá $g_j = \dfrac{\partial L_j}{\partial w_{j-1}}$ legyen az L_j $j-1$ részhalmaz adatpontjaihoz rendelt súlyok szerinti parciális deriváltja. Számítsuk ki

$$m_j = \beta_1 m_{j-1} + (1-\beta_1) g_j, \tag{3.29}$$

$$v_j = \beta_2 v_{j-1} + (1-\beta_2) g_j^2 \tag{3.30}$$

a gradiens (m_j) és a gradiens négyzetének (v_j) exponenciális mozgó átlagát. Ezek a mozgó átlagok egyben a gradiens első és második momentumának becslései. A képletekben szereplő β_1, β_2 1-hez közeli szabad paraméterek, amelyek szabályozzák a csökkenés mértékét a mozgó átlagoknak, általánosan megválasztott értékük 0,9 és 0,999. Szükséges megadni $j = 1$ első iteráció esetén a kezdő m_0 és v_0 értékeket, amik legyenek 0-k,

ami torzulást eredményez 0 felé a momentum becsléseinkben. A torzítatlan becslések a következőek lesznek:

$$\hat{m}_j = \frac{m_j}{(1-\beta_1)}, \tag{3.31}$$

$$\hat{v}_j = \frac{v_j}{(1-\beta_2)} \tag{3.32}$$

Ezek alapján kiszámíthatóak a módosított súlyok,

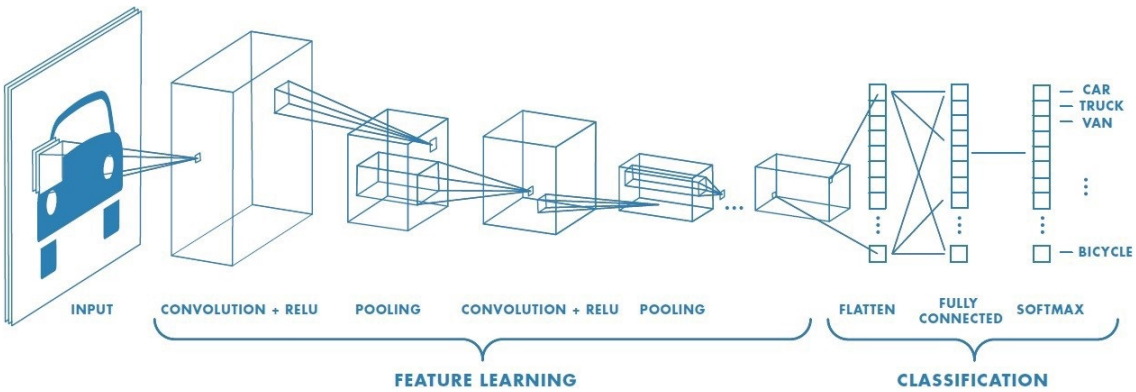
$$w_j = w_{j-1} - \alpha \frac{\hat{m}_j}{\sqrt{\hat{v}_j} - \varepsilon}, \tag{3.33}$$

ahol α (tanulási ráta) és ε szintén szabad paraméterek, általánosan megválaszott értékük 10^{-3} és 10^{-8}.

Az Adam optimalizáló eljárás minden egyes *batch* végigáramoltatása után hajtandó végre, amit szükséges több *epoch*-on keresztül alkalmazni, hogy a hálózat belső paraméterei optimálisak legyenek a veszteségfüggvény minimalizálásához.

3.3.2. Konvolúciós neurális háló

A konvolúciós neurális háló (*convolutional neural network, CNN*) egy speciális neurális háló, ami egy alapvető módszer, ha kíváló pontosságot kívánunk elérni objektum felismerés feladatkörben. Az eddig leírt neurális hálótól eltérően a bemeneti adatok nem vektorok, hanem mátrixként reprezentált képek, ahol a mátrix elemei a kép pixelei. További különbség, hogy a CNN egy vagy több konvolúciós rétegből áll, és gyakran tartalmaz mintavételi, úgynevezett *pooling* rétegeket.



3.3. ábra. A CNN felépítése konvolúciós rétegekből és a sűrű hálóból [9].

Konvolúciós réteg

Egy CNN-ben az első réteg mindig egy konvolúciós réteg. A konvolúciós művelet ráfókuszál bemenet különböző jellemzőre. Az első konvolúciós réteg azonosítja az alacsony szintű tulajdonságokat a bemeneti képen, mint például az éleket, a vonalakat és a sarkokat, a második réteg pedig a színekre és a textúrákra hagyatkozik. A további konvolúciós rétegek pedig azonosítják a magasabb szintű tulajdonságokat vagy mintákat a képen.

Legyen I bemenet egy $N \times N$ pixelből álló szürkeskálás kép. Definiáljunk egy $K \in \mathbb{R}^{t \times t}$ filter mátrixot (3.28) egyenletes eloszlásból származó kezdeti értékekkel. A K filter mátrix szerepe, hogy az I képet konvulálja egy új I' képpé. Minden pixele $I'(i,j)$ a kimeneti képnek elemenkénti szorzatok összege a filter mátrixnak és az eredeti képnek:

$$I'(i,j) = \sum_{m=1-\lceil \frac{t}{2} \rceil}^{\lfloor \frac{t}{2} \rfloor} \sum_{n=1-\lceil \frac{t}{2} \rceil}^{\lfloor \frac{t}{2} \rfloor} I(i+m, j+n) K(m,n) \tag{3.34}$$

Egy fontos részlet az I határon túli elemeinek kezelése, ugyanis előfordulhat, hogy például $t = 3$, $i = j = 0$ megválasztásoknál $I'(0,0)$ kiszámításához szükség van $I(-1,0)$ elemére. Ezen probléma kezelésére a két legismertebb módszer a *valid* és a *same padding*. A *valid* esetében nem számoljuk ki azokat a $I'(i,j)$ kimenti képbeli pixeleket, ahol I határain kívülre eső értékekkel kellene számolni, így a kimeneti kép egy kisebb dimenziójú mátrix lesz. Ennek a mátrixnak a mérete $(N-t+1) \times (N-t+1)$, ami problémát jelenthet azokban az esetekben, ha I dimenziója eleve kicsi és ha I határaira eső pixelek releváns információt tartalmaznak, mivel így ezek a pixelek kevesebb filterben szerepelnek, mint egy nem a kép szélére eső pixel, ami ronthat a modellünk teljesítményén. A *same* eljárással I határon túli elemeit 0-val helyettesítjük, így I' dimenziója megegyezik I dimenziójával és több információt tartalmaz a határ beli elemekről.

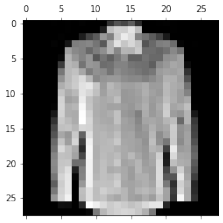
Egy konvolúciós réteg felépítéséhez szükséges paraméterek a filterek száma, mérete és a közöttük levő lépésköz, valamint a *padding* során használt módszer. Természetesen I' továbbítása előtt minden egyes kiszámított pixelén alkalmazunk egy megválasztott aktivációs függvényt. A sűrű hálóhoz hasonlóan a ReLU itt is egy megfelelő választás.

Konvolúciós rétegek egy példán keresztül

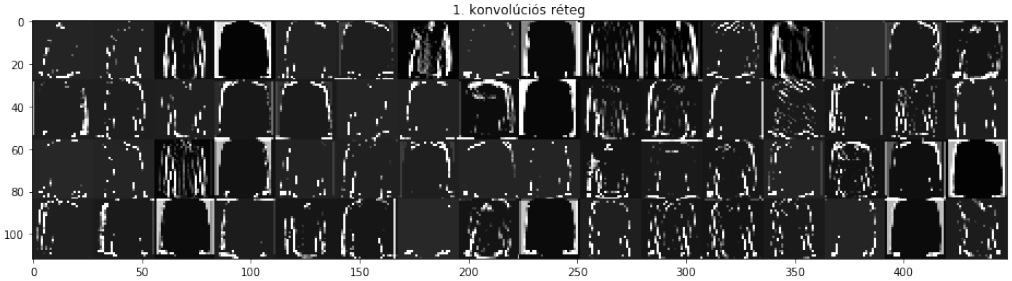
A 3.5. ábrán láthatunk egy feladatbeli képet, ami a CNN bemeneti rétegében a képen láthatóan néz ki.

A 3.6. ábrán az első konvolúciós rétegben használt K filter által készített konvulált képek láthatóak a ReLu aktivációs függvény alkalmazása után. Itt a feladatra alkalmazott CNN-en mutatom be az aktivációs térképet, ahol 64 filter által konvulált kép (*channel*) található.

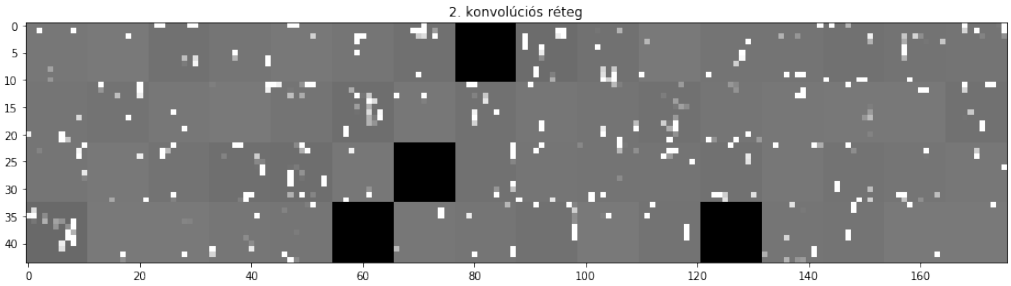
A 3.7. ábrán a második konvolúciós réteg aktivációs térképe látható. Az itt talalható képeken már alkalmaztam az első konvolúciós réteg után egy *pooling* réteget, ami miatt kisebb méretűek a *channel*-ek.



3.4. ábra. A CNN egyik bemeneti képe.



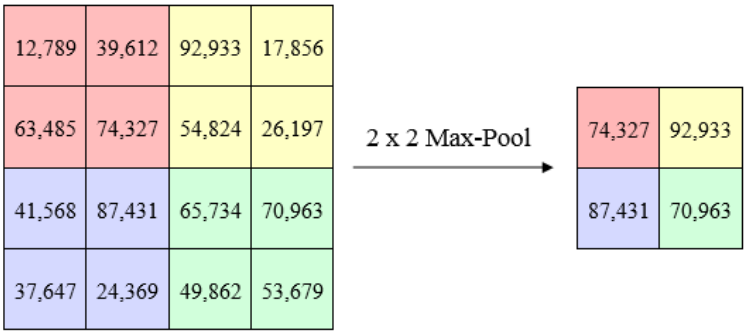
3.5. ábra. A 3.6. ábrán szereplő képhez tartozó aktivációs térkép az első konvolúciós rétegben.



3.6. ábra. A 3.6. ábrán szereplő képhez tartozó aktivációs térkép a második konvolúciós rétegben.

Pooling és sűrű hálóhoz kapcsolás

A *pooling* tipikusan a konvolúciós réteg után végrehajtott művelet. Végrehajtása során csökkenti a dimenzióját a képeknek azzal, hogy csökkenti a pixelszámát a megelőző konvilúciós réteg kimenetelének. A dolgozatomban használt eljárás a *Max-Pooling*.



3.7. ábra. A *Max-pooling* eljárás 2×2-es filterrel egy egyszerű példán.

A *Max-Pooling* a maximális értéket adja vissza a filter által lefedett képrészből, így hasznos a domináns jellemzők kiválasztásában és a zaj szűrésében.

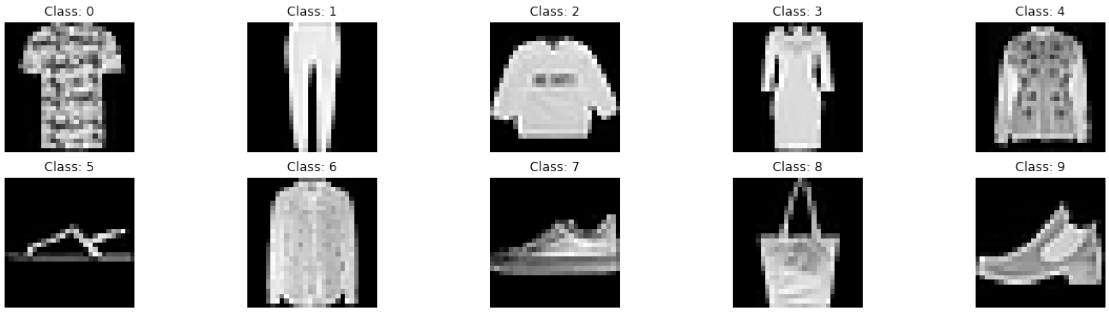
A konvolúciós rétegek után a hálózat a korábban ismertetett sűrű hálónak továbbítja a kimeneti értékeit. Mivel a konvolúciós rétegek kimenete egy mátrix és a sűrű háló bemenetként egy vektort vár, ezért a kimeneti mátrixot szükséges átindexelni egy oszlopvektorrá.

4. fejezet

Alkalmazás ruhatípusok felismerésére

4.1. Adathalmaz és feladat

Az objektum felismeréshez használt adathalmaz a Zalando Research által publikált Fashion-MNIST, ami 60000 tanító és 10000 tesztelő adatrekordból áll. Mindegyik adatpont egy 28×28-as szürkeskálás képből áll, illetve az első oszlopában egy címke, ami az adatpont 10 különböző osztályát jelöli. Mindegyik osztályhoz tartozik 6000 tanító és 1000 tesztelő adathalmazbeli rekord. A képek mindegyik pixelének értékkészlete a 0-tól 255-ig tartó egészek halmaza, ahol 0 a fekete és 255 a tiszta fehér szín. A feladat az, hogy



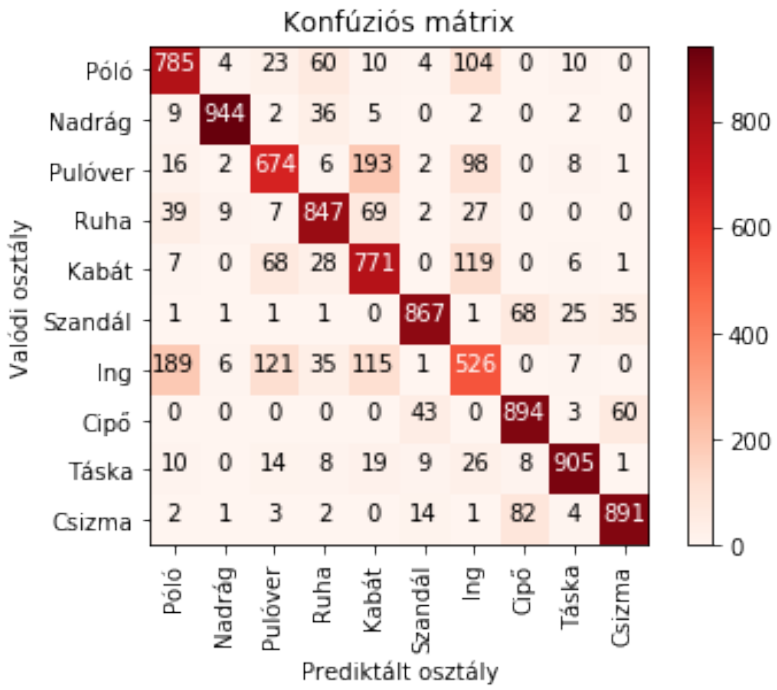
4.1. ábra. Példák az adathalmazból, az osztályok sorban: póló, nadrág, pulóver, ruha, kabát, szandál, ing, cipő, táska, csizma.

a 3. fejezetben bemutatott algoritmusok alapján olyan klasszifikációs modellt építeni, ami minél jobban osztályozza a modell által még nem ismert tesztadatokat a ruhák típusa szerint. Először érdemes a benchmark modelleket alkalmazni, hogy megismerjük a feladat

nehézségét és legyen viszonyítási alapunk.

4.2. Benchmark modellek teljesítménye

Több osztályú klasszifikációs modellek teljesítményénél a legtöbb információval bíró kiértékelési mutató a tévesztési vagy más néven konfúziós mátrix. Bemutatásához a CART algoritmussal épített döntési fa konfúziós mátrixát használom.



4.2. ábra. A döntési fa konfúziós mátrixa az eredeti tanító adathalmazon.

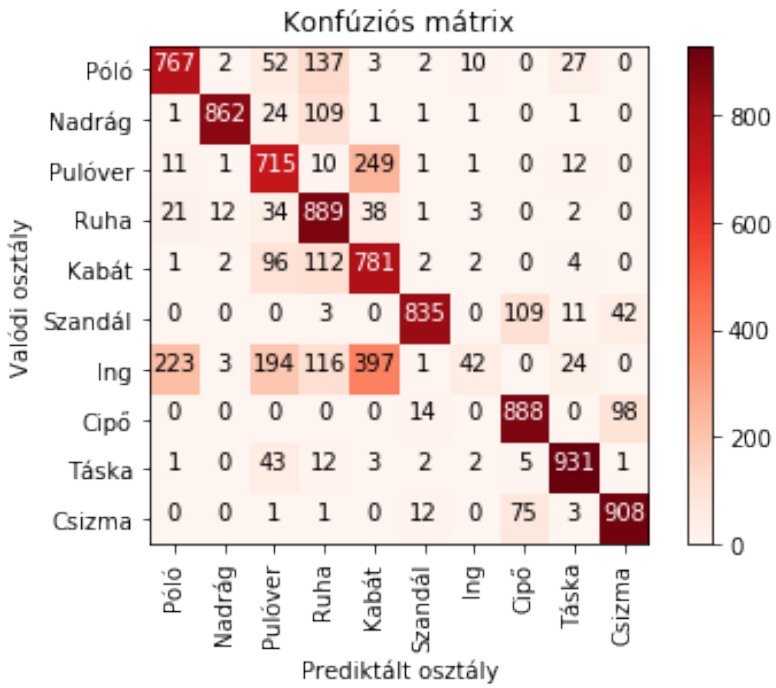
Legyen C a konfúziós mátrix, ekkor $C_{i,j}$ $i, j \in (1, 2, ..., 10)$ jelenti azt a számot, ahány i-edig osztályba tartozó adatot a modellünk a j-edik osztályba sorolt. A mátrix diagonális elemeiben lévő számok a megfelelő osztályhoz tartozó, helyesen klasszifikált képek száma.

Egy másik kiértékelési metrika a pontosság (accuracy), ami kiegyensúlyozott méretű és fontosságú osztályok esetén egy releváns mutató, aminek a számolása a konfúziós

mátrix értékeiből történik:

$$accuracy = \frac{\sum_{i=1}^{10} C_{i,i}}{\sum_{i=1}^{10}\sum_{j=1}^{10} C_{i,j}} \tag{4.1}$$

A másik alkalmazott benchmark modell a véletlen erdő, ami 100 darab szintén a CART algoritmussal épített döntési fát tartalmaz.

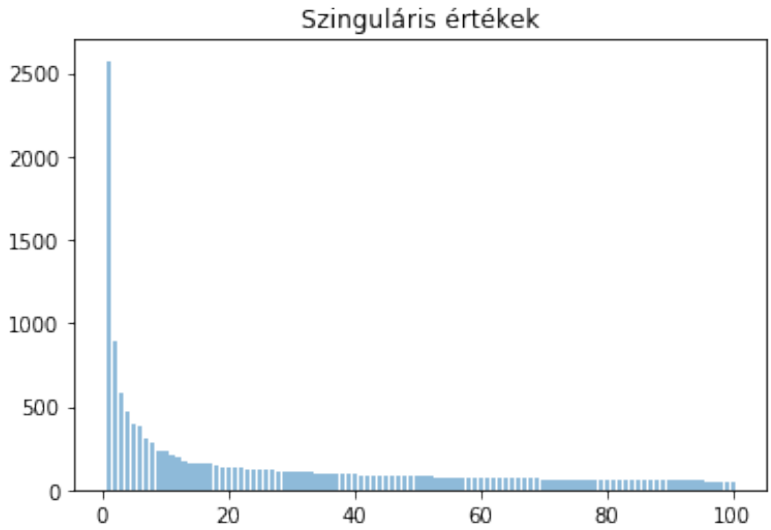


4.3. ábra. A véletlen erdő konfúziós mátrixa az eredeti taníto adathalmazon.

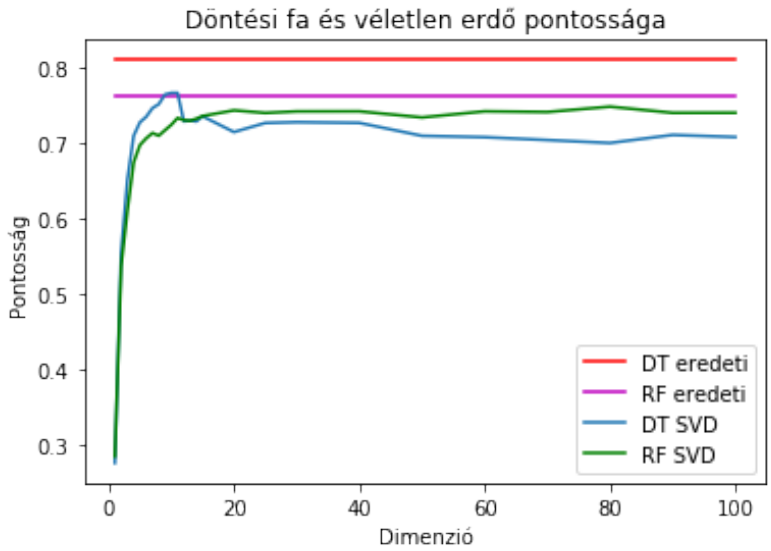
A tanító adathalmazunk dimenziója $28 \times 28 = 784$, ami nagyban növeli a számítási időt. Feltételezhetjük, hogy nincsen szükség az összes pixel információjára, hogy osztályokba tudjuk sorolni a képeket, ezért alkalmazzuk az adatokon az SVD-t, hogy kihangsúlyozzuk a képeket meghatározó faktorokat, csökkentsük az eredeti dimenziót és a képekben előforduló zajt a (2.4) alapján.

A 4.4. ábrán láthatóak a tanító adathalmazon alkalmazott SVD mátrix diagonális elemei, azaz a szinguláris értékei. Dimenziócsökkentés során az alacsony szinguláris értékek 0-ra állításával csökkenthetjuk a zajt.

A 4.5. ábrán látható a benchmark modellek teljesítménye az eredeti és a csökkentett dimenziós adathalmazon. Érdemes megjegyezni, hogy nagyon alacsony dimenzióra



4.4. ábra. A tanító adathalmaz 100 legnagyobb szinguláris értéke.



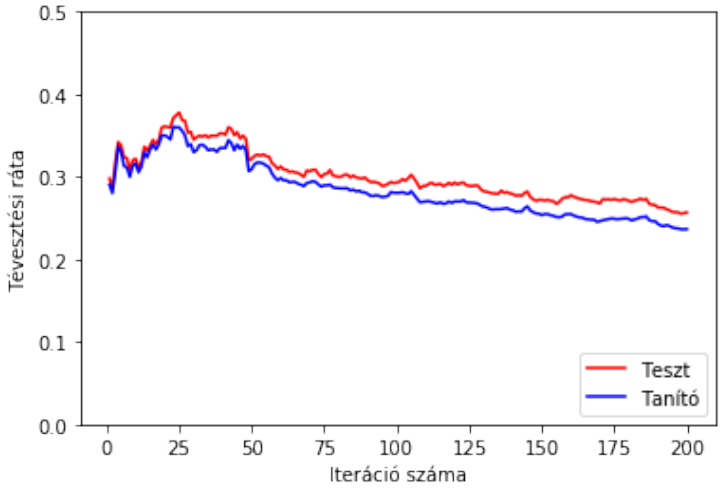
4.5. ábra. A dimenziószám és a pontosság viszonya.

(10-11) lecsökkentett adathalmazon is az eredeti pontossághoz közeli értéket érnek el a benchmark modellek, ami az eredeti adatok méretének alig több mint az 1%-a.

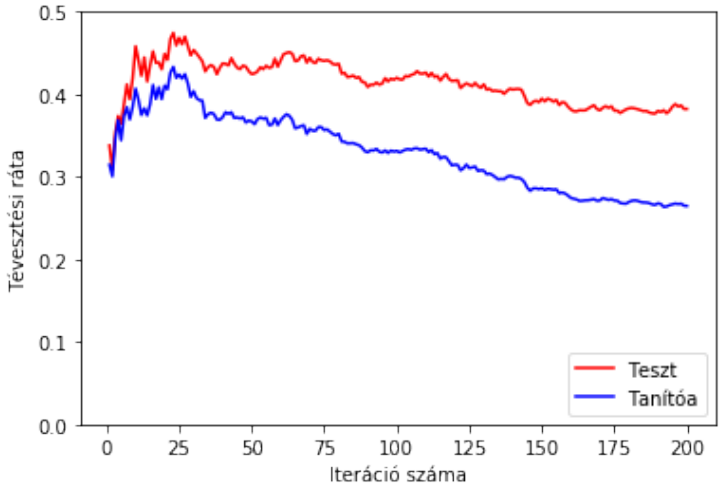
Továbbiakban a két benchmark modellünket fogjuk viszonyítási alapul venni, azaz a 0,7618 pontosságú véletlen erdőt és a 0,8104 pontosságú döntési fát.

4.3. Boosting modellek teljesítménye

A boosting alapú modellek közül először vizsgáljuk meg az Adaboost teljesítményét. Az Adaboost alap osztályozójaként 5 maximális mélységű döntési fát alkalmaztam 200 iteráció mellett. A modell kiértékeléséhez vizsgáljuk meg a tévesztési rátát, ami az $1 - accuracy$ alapján számolandó. A 4.6. és a 4.7. ábrák alapján látható, hogy a modell pontossága (0.7433) gyenge, mivel mindkét benchmark modellünk jobban teljesít ennél.

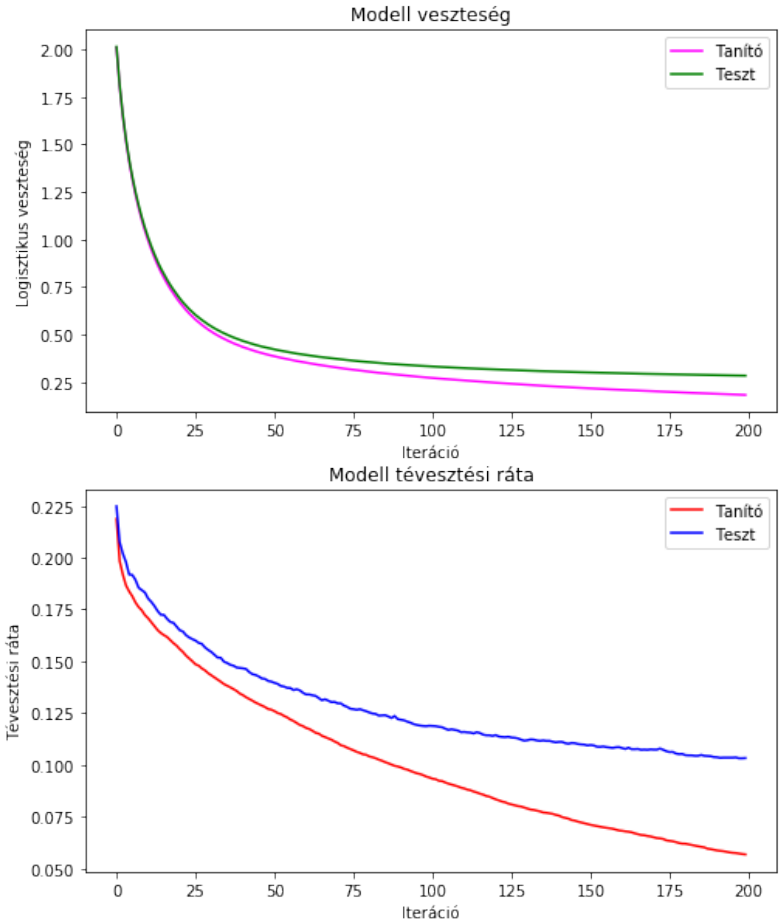


4.6. ábra. Adaboost tévesztési rátája az eredeti adathalmazon az egyes iterációknál.



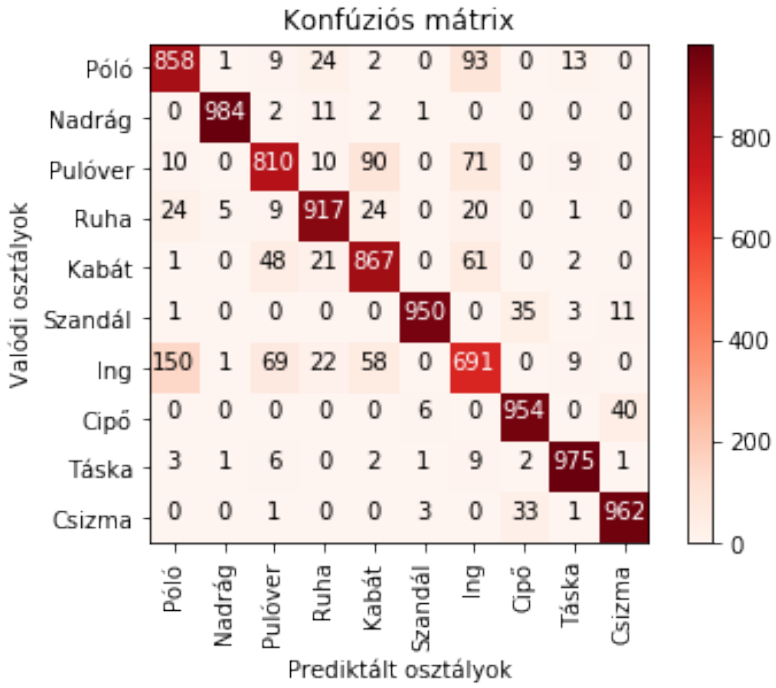
4.7. ábra. Adaboost tévesztési rátája a csökkentett dimenziójú adathalmazon az egyes iterációknál.

Továbbiakban térjünk át a jelenleg legfejlettebb boosting alapú algoritmusra, az XGBoostra. Az Adaboost modellhez hasonlóan most is 200 döntési fát generálunk a tanulás során.



4.8. ábra. Az XGBoost logaritmikus veszteségfüggvénye és tévesztési rátája a tanulás során.

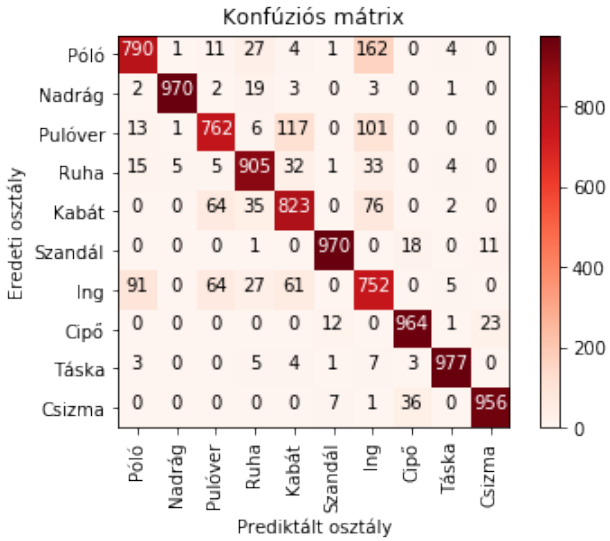
A 4.9. ábrán, a konfúziós mátrixon látható az XGBoost hatékonysága a teszt adatokon elért 0,8968 pontosságból. Az eddigi modellek konfúziós mátrixaiból leszűrhető, hogy a feladat legnagyobb nehézsége különbséget tenni a póló (0), pulóver (2), kabát (4) és az ing (6) osztályok között.



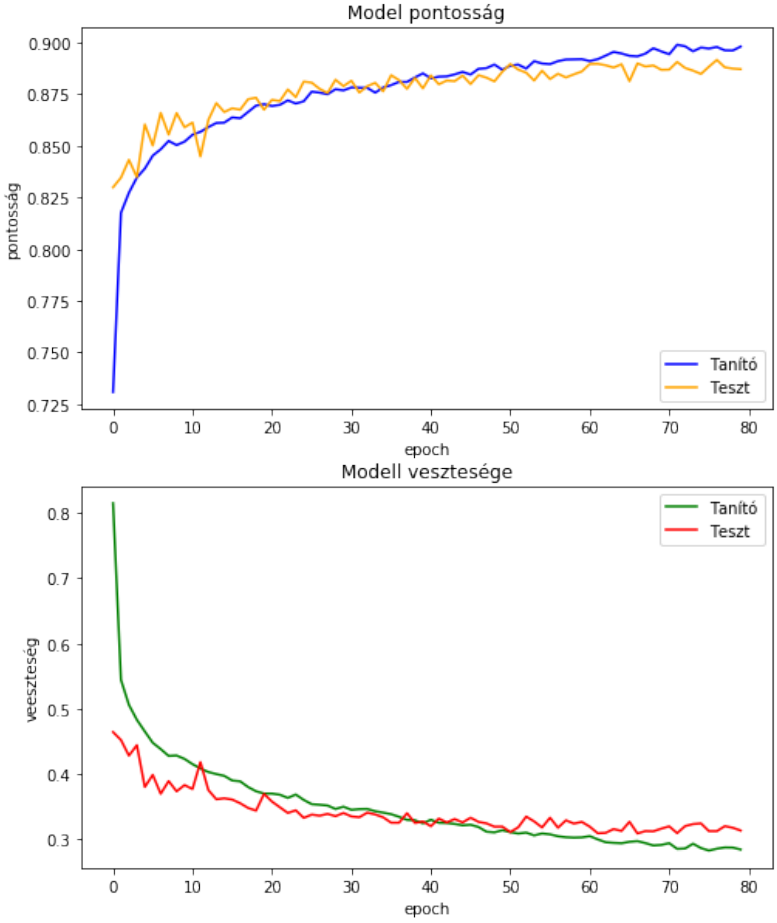
4.9. ábra. Az XGBoost konfúziós mátrixa.

4.4. Sűrű háló teljesítménye

A feladat során használt sűrű háló 4 rétegből áll. Egy 784 egységből álló bemenet rétegből, valamint egy 392 és egy 64 neuronokból álló rejtett rétegből. Végül egy 10 neuront tartalmazó kimeneti réteggel ér véget a hálózat.



4.10. ábra. A sűrű háló konfúziós mátrixa.



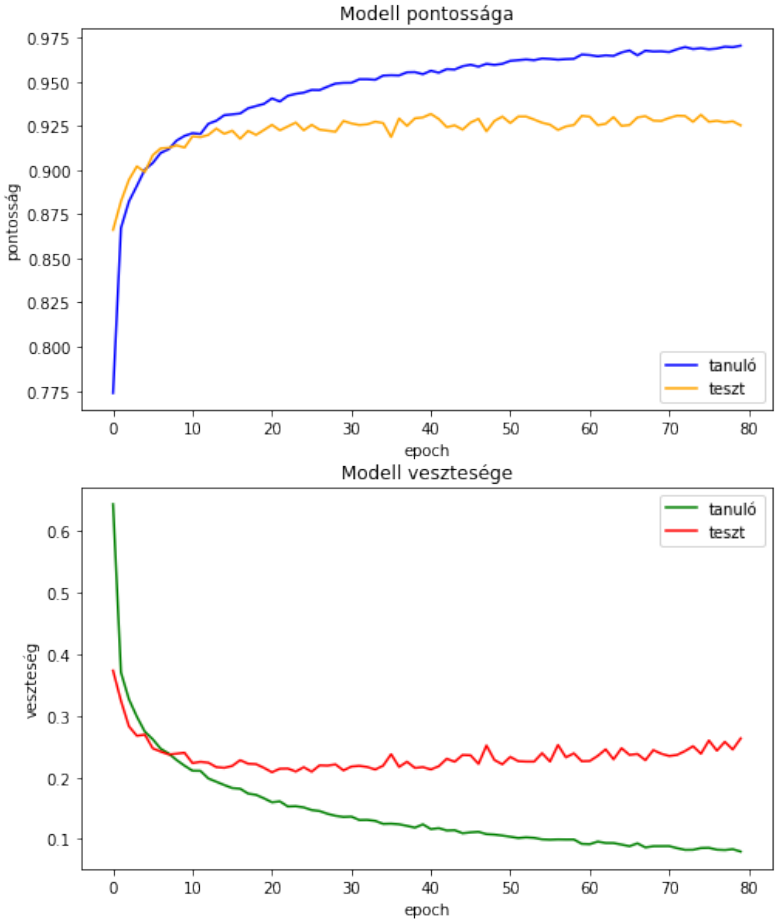
4.11. ábra. A sűrű háló tanulása során az *epoch*-ok és a pontosság, illetve veszteségfüggvény értékének viszonya.

A 4.10 ábrán figyelhetjük a sűrű háló tanulási folyamatát, látható, hogy szükséges több *epoch* szám az optimális súlyok meghatározásához, mert a tanítási fázis elején alacsonyabb a tanuló és teszt adatokon a pontosság. Az alkalmazott sűrű hálót 80 *epoch* szám mellett 250 méretű *batch*-eken tanítottam.

A tanítás után a modell által még nem ismert tesztelő adatokon a teljesítménye (0,8869) hasonló az XGBoost teljesítményéhez. Itt is elmondható, hogy ugyanaz, mint az XGBoostnál, hogy az ott említett osztályok felismerése itt is problémát jelent.

4.5. Konvolúciós neurális háló teljesítménye

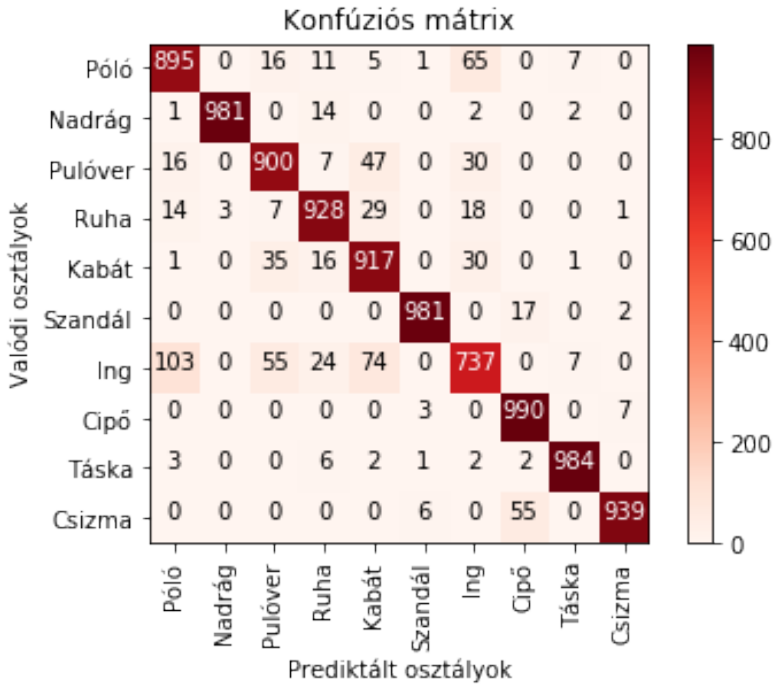
Az alkalmazott konvolúciós neurális háló 2 konvolúciós rétegből áll, mindkét esetben 4×4-es filtert és 64 *channel*-t alkalmaztam és mindkét réteghez tartozik egy-egy 2×2-es *Max-Pooling* mintavételező réteg. A második $pooling$ réteg utáni 64 5×5-ös képek egy 1600 hosszú vektorba töltődnek bele, amik egy két rejtett réteget tartalmazó sűrű hálóhoz csatlakoznak bemenetként.



4.12. ábra. A konvolúciós neurális háló tanulása során az *epoch*-ok és a pontosság, ill. veszteségfüggvény értékének viszonya.

A tanulási folyamton láthatjuk, hogy a CNN igen korán, 10-15 *epoch* környékén elérte a teljesítménye csúcsát. A további *epoch* tanítása feleslegesnek bizonyul, mivel már nem javul a tesztelő adatokon a modell pontossága, ilyen eseteknél érdemes megállítani a tanulási folyamatot a túltanulás elkerülése céljából.

Az osztályozási pontosság szempontjából a CNN bizonyult a legjobban teljesítőnek



4.13. ábra. A konvolúciós neurális háló konfúziós mátrixa.

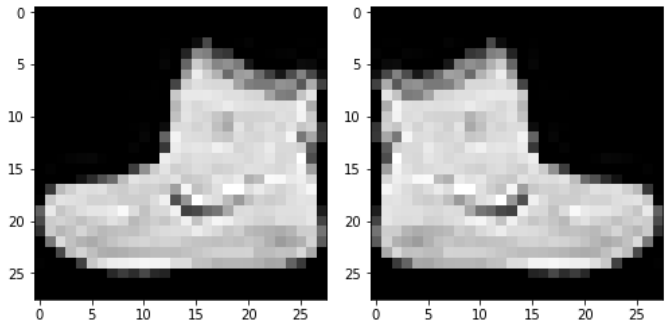
a maga 0,9252 értékével. A többi modellnél azért bizonyul jobbnak képfeldolgozásnál, mert míg a korábbi módszerek a pixeleket független értékekként kezelték, addig a CNN a pixelekből felépülő geometriai formákra fókuszál. A modell alkalmazhatóságát bizonyítja az is, hogy a korábbi modelleknél fellépő osztályozási nehézségek a CNN-nél kevésbé szembetűnőek.

Modell	Pontosság
Döntési fa	0,8104
Véletlen erdő	0,7618
Adaboost	0,7433
XGBoost	0,8968
Sűrű háló	0,8869
CNN	0,9252

4.14. ábra. A modellek pontossága az eredeti adathalmazon.

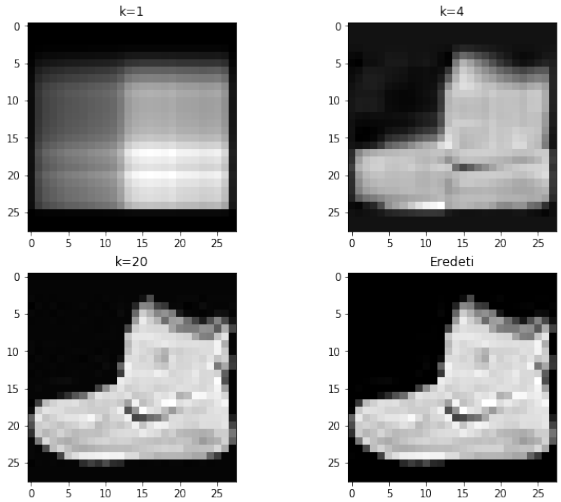
4.5.1. Adatbővítés transzformációkkal

A 3. fejezetben ismertetett prediktív modellek közül a konvolúciós neurális háló módszere adta a legjobb megoldást a feladatra. Ebben a fejezetben is az a célunk, hogy az elért pontosságot tovább javítsuk. Erre egy technika, hogy több adaton tanítjuk a modellünket, de teljesen új adatok hiányában az eredeti adathalmaz transzformált adataival bővítjuk az adathalmazt. Egy bevált módszer, hogy a tanító adathalmaz összes elemének a tükrözöttjével bővítjük az adatainkat.



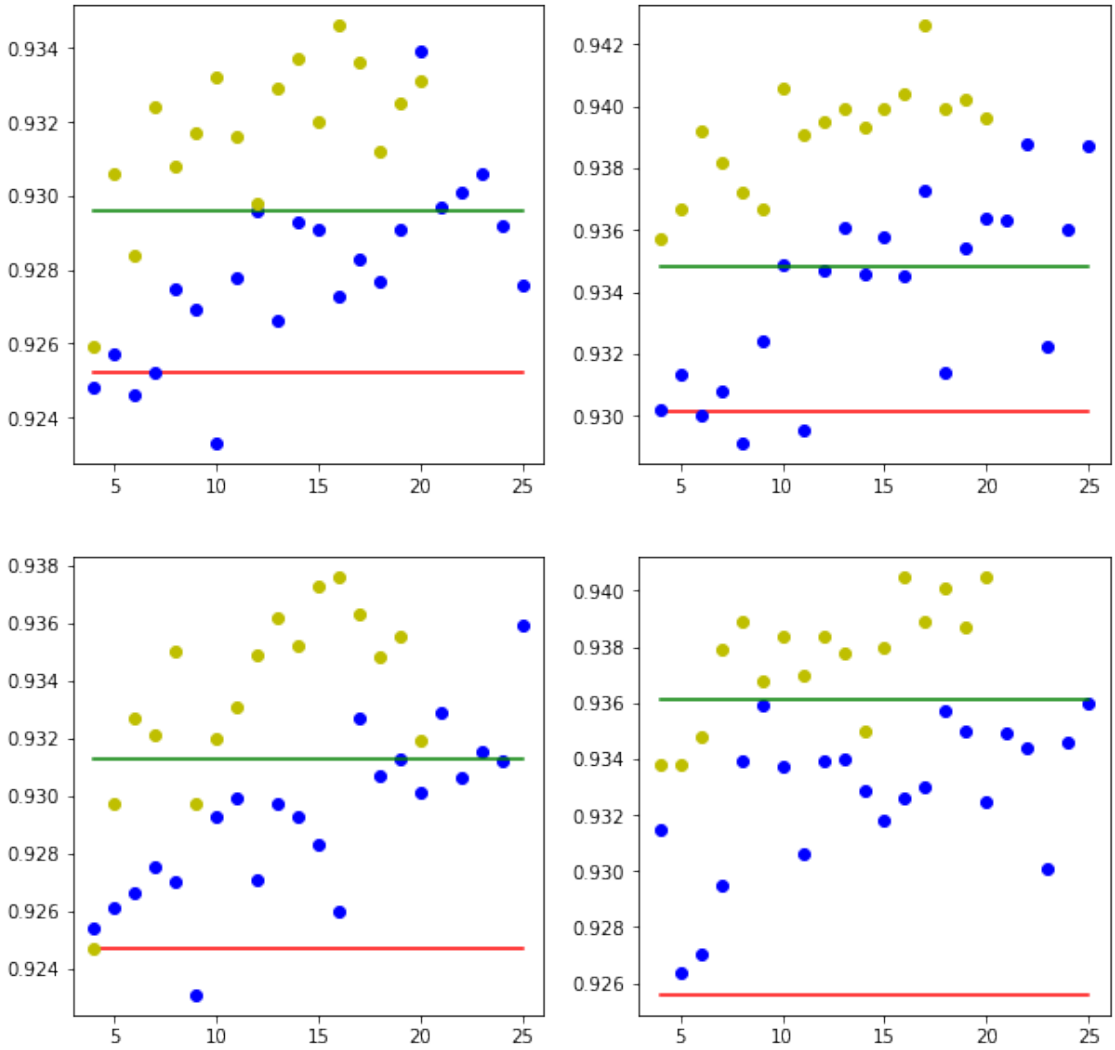
4.15. ábra. Egy tanító adathalmazbeli példán a tükrözés.

Egy másik stratégia amit alkalmaztam adatbővítéshez, hogy az Eckart-Young tételben (2.5) megfogalmazott kisebb rangú mátrix approximációkkal bővítettem az adathalmazt. Itt fontos paraméter a k közelítő mátrix rangja, ami miatt szükséges több mérést elvégezni, hogy a bővítéshez a legjobban javító k-t válasszuk.



4.16. ábra. Egy tanító adathalmazbeli képen a k-ad rangú mátrix approximációk.

A 4.17. ábrán a különböző adatbővítési módszerekről egy összehasonlítást végeztem a CNN modell pontosságának befolyásolása alapján. A piros egyenes jelenti a CNN pontosságát az eredeti tanító adathalmazon, a zöld egyenes a tükrözött adatokkal való bővítési pontosság. A kék pontok az eredeti adathalmaz és a megfelelő rangú közelítő mátrixokon elért pontosság. Végül a sárga pontok egy kombinált stratégia eredményei, ahol a bővített adathalmaz áll az eredeti, a tükrözött, valamint ezek kisebb rangú mátrix közelítéseiből, ami az eredeti tanító adathalmaz méretének négyszeresére bővített változatát jelenti. Tisztán látszik, hogy az eredeti teljesítményt mindegyik bővítési stratégiával javítani lehetett.



4.17. ábra. CNN modell esetén különböző adatbővítési technikák alkalmazásával elért pontosságok eltérő teszt adatokon. Az y tengelyen a modell pontossága a teszt adatokon, az x tengelyen pedig a bővítéshez használt közelítő mátrixok rangja.

és a kombinált módszer javított a legtöbbet a CNN pontosságán a kipróbáltak közül.

4.6. Programkód

A programkódok megírásához a Python 3.6-os verzióját használtam a Google Colaboratory fejlesztői környezetben. A python notebook-ok megtalálhatóak az alább github linken: `https://github.com/bottyanmartin/szakdolgozatprogramkod.git`

5. fejezet

Összefoglalás

Összefoglalva a tapasztalatok alapján elmondható, hogy a bemutatott modellek különböző hatékonysággal teljesítették az osztályozó feladatot, ami arra enged következtetni, hogy egyes modellek kevésbé alkalmasak a képfeldolgozásra, mint például az Adaboost vagy a véletlen erdő. Ennek oka lehet, hogy nagy dimenziós képadatokkal számolva sokkal lassabban ér el olyan magas iteráció számot, ami mellett megfelelő pontosságot érne el a modell.

A konvolúciós neurális háló és az XGBoost modelleknél rövidebb idő alatt és jobb teljesítmény érhető el képfeldolgozás és objektum felismerés feladatkörben. Ezen modellek esetében érdemes lenne kipróbálni más adatbővítési stratégiágat, például a képek egy vagy több pixellel történő eltoltjaival, vagy a képek elforgatottjaival történő bővítést. Továbbá kipróbálható több bővítési technika kombinációjával bővített adathalmazon a modell teljesítményét vizsgálni hosszabb tanulási folyamat mellett.

Irodalomjegyzék

[1] Rutkowski, L., Jaworski, M., Pietruczuk, L., Duda, P. (2014). The CART decision tree for mining data streams. *Information Sciences*, 266, 1-15.

[2] Breiman, L. (2001). Random forests. *Machine learning*, 45(1), 5-32.

[3] Hastie, T., Rosset, S., Zhu, J., Zou, H. (2009). Multi-class adaboost. *Statistics and its Interface*, 2(3), 349-360.

[4] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6), 386.

[5] Altrichter, M., Horváth, G., Pataki, B., Strausz, Gy., Takács, G., Valyon, J. (2006). Neurális Hálózatok. Budapest, *Panem Könyvkiadó Kft*.

[6] Kingma, D. P., Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

[7] He, Tong: XGBoost: eXtreme Gradient Boosting. `http://www.saedsayad.com/docs/xgboost.pdf` (2017.10.23)

[8] Golub, G. H., Van Loan, C. F. (2012). *Matrix computations* (Vol. 3). JHU press.

[9] Saha, Sumit: A Comprehensive Guide to Convolutional Neural Networks - The ELI5 way. `https://towardsdatascience.com/` (2018.12.15)

[10] Hijazi, S., Kumar, R., Rowen, C. (2015). Using convolutional neural networks for image recognition. *Cadence Design Systems Inc.: San Jose, CA, USA*.