EÖTVÖS LORÁND TUDOMÁNYEGYETEM

TERMÉSZETTUDOMÁNYI KAR

Orvosi képek elemzése gépi tanulási módszerekkel

SZAKDOLGOZAT

Borbély Bernárd

Matematika BSc

Alkalmazott matematika szakirány

Témavezető:

Lukács András

Számítógéptudományi Tanszék

Budapest

2023

Köszönetnyilvánítás

Szeretném megköszönni a témavezetőmnek Lukács Andrásnak, aki szakértelmével, iránymutatásával és türelmével végig támogatta a szakdolgozatom megírását. Köszönöm Vas Bernadettnek, aki bevezetett a témakör technikai hogyanjába és segített eligazodni abban. Köszönettel tartozom a páromnak és barátaimnak, akik támogattak és biztattak a legnehezebb időkben.
Külön köszönettel tartozom a családomnak, akik támogattak és egyengették az utamat az egyetemi (és nem egyetemi) évek alatt.

Tartalomjegyzék

1. fejezet

Bevezetés

A gépi tanulás során az a célunk, hogy olyan modelleket építsünk, amelyek korábbi ismeretekből képesek tanulni, és ezután egy adott feladat megoldását előállítani. A modell építésében úgynevezett tanítóadatokat használunk, amelyeket megtanulva, a módszer képes jósolni, döntést hozni vagy akár akciót végrehajtani. A tanítás azon feltételezés mentén alapul, hogy a stratégiák, módszerek amik korábban eredményre vezettek, továbbra is sikeresek lesznek. A mély tanulás a gépi tanulás egy fajtája, ahol neurális hálókat használunk reprezentáció tanulásra. A neurális hálók alapja a percetron, egy a mesterséges neuron, amely az agyi neuron működésének egyfajta egyszerűsített modellje. Mély tanulás során a cél, hogy a neurális háló automatikusan megtanulja a lényeges tulajdonságokat felismerni, és ezt felhasználva tudja megoldani a kitűzött feladatot. Egy ilyen kitűzött feladat lehet a klasszifikáció. Klasszifikálás során az a célunk, hogy a bemenő adatot, előre meghatározott csoportokba osztályozzuk. Az egyszerű írott számjegy felismerésétől kezdve, egy képről eldönteni, hogy milyen objektum szerepel rajta, a csalárd tranzakciók felismerésén keresztül, az orvosi képek elemzéséig számos hasznos felhasználása létezik. Orvosi területről is szsámos példát hozahatunk: retináról alkotott képek alapján cukorbetegség diagnosztizálás, vagy mint ebben a dolgozatban, röntgen képek alapján Covid fertőzés észlelés. A mély tanulás egy másik fő kutatási területe a generatív modellezés. Ennek során a modell célja megismerni az adathalmazt (kép, hang, mozgás) olyan részletességgel, amely után képes új adatokat alkotni. Fontos hogy a modell ne az adathalmazt újraalkotni tanulja meg, hanem általánosítva értse és így generáljon. Alkalmazásaik az egyszerű kép generálástól, a szövegből kép alkotáson keresztül, a kép- és videó kitöltésen át a képből 3d-s objektum generálásig terjednek. Az első nagy előrelépést a Variational

Autoencoderek hozták (VAE), majd pedig őket a Generative Adversarial Network (GAN) architektúrák váltottak le.
A dolgozatom témája az, hogy egy Covid klasszifikálási feladatban, ahol a Covid osztály alulreprezentált, hogyan lehet generatív módszereket alkalmazni a klasszifikálás pontosságának a javítására. Célunk a kiegyensúlyozatlan adathalmazok *GAN*-nal történő augmentálásának jobb megértése. A generálás feladatának kivitelezésére ismertetjük a *Lipizzaner* működését [12], amely egy *GAN* tanító keretrendszer. A dolgozatom 2. fejezetében bemutatujuk a fontosabb felhasznált mély tanulásos eszközöket. Majd a 3. fejezetben *GAN*-ok, illetve a Lipizzaner működését vázoljuk fel. Ezután 4. fejezetben ismertetjük az általunk használt, *GAN*-okkal való augmentálás technikáját. Majd a mérési eredmények kiértékelésével folytatjuk az 5. fejezetben. A 6. fejezetben összefoglaljuk a dolgozat eredményeit és további javítási, ill. kutatási lehetőségeket vázolunk fel.

2. fejezet

Mély tanulásos eszközök

2.1. Konvolúciós hálók

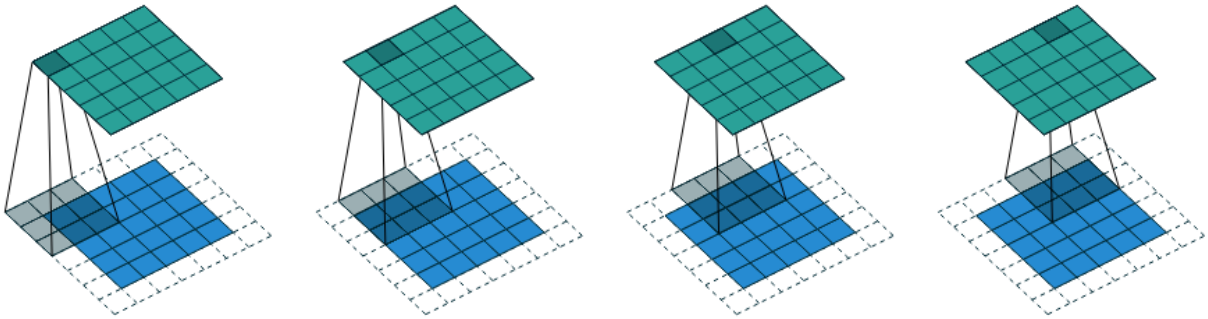
Ebben a fejezetben bemutatjuk a dolgozatban használt konvolúciós hálók felépítésének alapjait. Ez a fejezet [2] alapján készült. A konvolúciós hálók motivációja, hogy kihasználjuk, hogy a bemeneten az egymáshoz közeli pontok szorosabban kapcsolódnak egymáshoz, így például a képeknél felhasználható, hogy ez egy kétdimenziós struktúra. A konvolúciós hálókhoz általában 3 fajta réteget alkalmazunk: konvolúciós réteg, pooling-réteg és sűrűn kapcsolt réteg (fully-connected).
Konvolúciós réteg: Ez a réteg 3-dimenziós tenzorokat, úgynevezett filtereket használ, amik szélessége és magassága jellemzően kicsi, általában 3 vagy 5, és a mélysége megegyezik az előző réteg kimenetének a rétegeinek számával. RGB képnél a mélység 3 lenne, egy szürkeárnyalatos képnél 1, és mivel egy konvolúciós réteghez több filtert is definiálhatunk, a későbbi rétegekben az előző konvolúciós réteg filterszáma lesz az egyes filterek mélysége. A réteg ráilleszti a bemenet bal felső sarkára a filtert, és pontonként elvégezve a szorzást, összegzi azt. Majd sor-, illetve oszlopfolytonosan végig megy a képen, és így alkot meg egy 2-dimenziós kimeneti tenzort.
A réteg viselkedését a paramétereivel szabályozhatjuk. A kernel $K = (H, W)$ határozza meg a filter szélességét, és magasságát, ezeknek a súlyai nem változnak, ahogyan a bemeneti képen végigsiklik. Szintén szabályozható a filterek száma (F), ez a kimeneti kép mélységét fogja meghatározni. A stride-dal a filter csúszásának a lépésnagygságát tudjuk szabályozni, mindkét dimenzióban $S = (S_H, S_W)$.
Előfordulhat, hogy a kernel-méret és a stride miatt a kép szélén már nem marad elég pi-

xel, hogy ráilleszzük a filtert. Alapesetben ilyenkor a kép szélén lévő pixeleket kihagyjuk a kimenet alkotásakor, de alternatív megoldást szolgáltathat a padding $P = (P_H, P_W)$. Ez a paraméter a bemeneti kép széleit tölti fel vagy 0-kal (zero-padding), vagy a legközelebbi pixel értékével, így a kép széléről sem veszítünk információt.
A filter elemei közötti távolságot is tudjuk szabályozni, ezt dilation-nek $D = (D_H, D_W)$ nevezik. Gyakorlatban megnöveli a filter méretét (H, W)-ről $((H-1)*D_H), (W-1)*D_W)$ méretűre és egyenletesen eloszlatja az eredeti filter elemeit az új filteren, a maradékot pedig feltölti 0-kal. Ez olyan mintha a kernel két egymés melletti eleme között kihagynánk D_W helyet, illetve az egymás felettiek között D_H-t, így szétszorva hogy honnét szerezzük az információt.
A kimeneti kép méretére általánosan is felírható szabály:

$$
\begin{aligned}
H_{ki} &= \left\lfloor \frac{H_{be} + 2 * P_H - (H-1) * D_H - 1}{S_H} + 1 \right\rfloor \\
W_{ki} &= \left\lfloor \frac{W_{be} + 2 * P_W - (W-1) * D_W - 1}{S_W} + 1 \right\rfloor \\
\textit{Mélység} &= F
\end{aligned}
$$

Ahol H_{be} és W_{be} a kép bemenetének szélessége és magassága, és ugyanígy H_{ki} és W_{ki} pedig a kimeneté. Fontos hogy a bemeneti kép mélysége nem befolyásolja a kimenet méretetit.
A réteg tanulható paraméterei a filterek súlyai és egy eltolás vektor. A filterek képesek megtanulni különböző struktúrákat, például él-, sarokdektektálást és a későbbi rétegekben komplex objektumok, esetünkben akár egy tüdő detektálását is.
A pooling réteget általában dimenzió csökkentésre használjuk. A konvolúciós réteghez hasonlóan van kernel-e és stride-je. Nincsenek tanulható paraméterei, helyette előre meghatározott szabállyal dolgozik. Gyakran használatos az average-pooling, $2 * 2$-es kernellel és 2-es stride-dal, így egymást nem fedő klaszterekre bontja a képet, és minden $2 * 2$-es klaszterben veszi az elemek átlagát, így a kép mindkét dimenziójában felére csökken. Másik gyakran használt szabály a max-pooling, amely az elemek átlaga helyett a klaszter legnagyobb elemével dolgozik tovább.
A sűrűn kapcsolt rétegben az előző réteg összes neuronja össze van kapcsolva a következő réteg összes neuronjával. A réteg egy mátrixszorzással írható le amihez egy eltolás vektor is hozzáadunk. A szerepe általában a háló végén a klasszifikálás megvalósítása.
A konvolúciós háló előnyei, hogy kevés tanulható paraméterrel megvalósítható (hiszen a

konvolúciós rétegekben csak a filterek és eltolások súlyai tanulhatók, és csak egy fully-connected réteg szerepel), és jól párhuzamosítható.

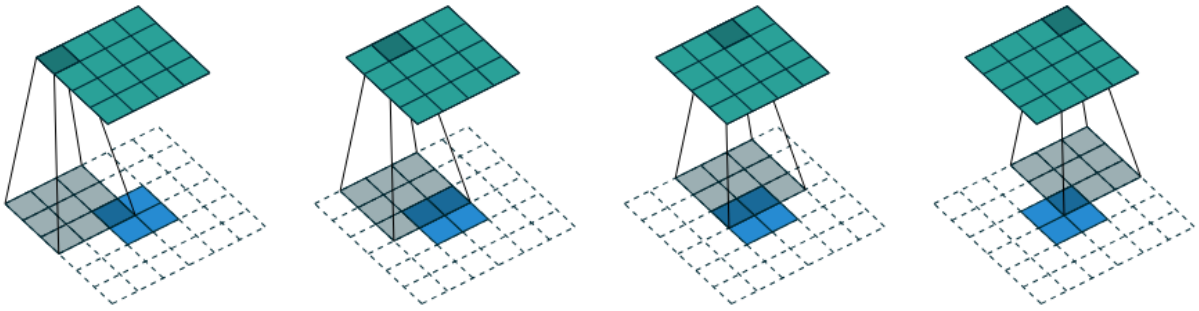


2.1. ábra. A képen egy egyszerű példa látható a konvolúció menetére. A kék rács a bemenet, a zöld a hozzátartozó kimenet, az árnyékolt rész pedig a kernel adott helyzetét jelenti. A *bemenet* $= (5, 5)$, a kernel $K = (3, 3)$, a stride $S = (1, 1)$, a padding $P = (1, 1)$. [2]

2.1.1. Transzponált konvolúció

Természetesen adódhat az igény, hogy olyan transzformációt szeretnénk használni, amely a konvolúciókkal ellentétes irányban halad. Például, hogy a transzformáció bemenetének a dimenziói megegyezenek valamilyen konvolúció *kimenetének* a méretével, míg a *kimenet* mérete a konvolúció bemeneti *dimenzióival* egyezik meg. Ezt a műveletet nevezzük *transzponált konvolúció*-nak (*transposed convolution*). Ennek a módszernek a célja, hogy egy konvolúciónak a bemeneti dimenzióit visszanyerjük, magát a bemeneti tenzort nem kapjuk vissza garantáltan. Például, ha van egy $4 * 4$-es bemenet, a kernel $3 * 3$-as, a stride $= 1$ és nincs padding, akkor a konvolúció egy $2 * 2$-es kimenetet fog adni. A *transzponált konvolúció*-nak ezesetben a kimenete lesz $4 * 4$-es, miután a $2 * 2$-es bemenetre alkalmazzuk.
Hasonlóan a konvolúcióhoz, a transzponált verziónak is vannak K' = kernel, S' =stride, P' =padding és D' = dilation paraméterei, amelyeket a konvolúcióhoz hasnolóan használ, azonban ezeket nem közvetlenül adjuk meg, hanem kikövetkeztetjük a beadott paraméterekből. Lényegében a megadott paraméterekre, mint konvolúciós paraméterekként gondolunk: ha adottak K =kernel, S = stride, P =padding paraméterek, és ezt a "*kimenetet*"

kaptuk, akkor milyen lehetett a konvolúciós réteg "*bemenete*"? Megjegyzendő, hogy $S' = 1$ mindig és $K' = K$ és a transzponált konvolúció a P' padding-gel és D' dilation-nel oldja meg, hogy a réteg kimenete megfelelő méretű legyen.



2.2. ábra. A képen egy egyszerű példa látható a *tanszponált konvolúció* menetére. A kék rács a bemenet, a zöld a hozzátartozó kimenet, az árnyékolt rész pedig a kernel adott helyzetét jelenti. Egy $4 * 4$-es *bemeneten*, $K = (3, 3)$-as kernellel, $S = (1, 1)$-es *stride*-dal és $p = (0, 0)$ *padding*-gel végzett *konvolúció* tanszponáltja. Ez megegyezik a $2 * 2$-es *bemeneten*, $K' = (3, 3)$-as kernellel, $P' = (2, 2)$-es *padding*-gel, egységnyi *stride*-dal (röviden: $input = 2 * 2$, $K' = K = (3, 3), P' = (2, 2), S' = (1, 1)$). [2]

2.2. Batch normalizálás

Mély hálók tanításánál megfigyelhető, az eltűnő gradiensek problémája. Ezt részben a $softmax$ féle aktivációs függvények is okozzák, mivel ahogy $|x|$ egyre nagyobb, a $softmax(x)$ deriváltja 0-hoz tart, így nagyon lecsökkenti a gradiens értékeket, ezzel lassítva a tanulást, emiatt nem is nagyon használják. Megoldási lehetőség más akivációs függvény használata, vagy ha sikerülne biztosítani, hogy $|x|$ értékei olyan helyre essenek ahol a $softmax$ deriváltja nem kicsi. Megfigyelhető az is, hogy egy háló mélyebb rétegeinek a bemenete nem csak az adott képtől függ, hanem a korábbi rétegek paramétereitől is, így a gradiens lépések után, megváltozik az egyes rétegek bemeneteinek az eloszlásai és ezáltal a későbbi rétegeknek folyamatosan alkalmazkodniuk kell az új eloszláshoz. Ezt a problémát nevezik Internal Covariate Shiftnek. Ezekre a problémákra ad megoldást Ioffe és Szegedy, 2015-ös cikkükben [5]. A módszer lényege, hogy ahogyan az SGD folyamán a képekkel batch-ekben dolgozunk, az egyes rétegek bemenetén a batch képei szerint normaizálunk egyet, hogy kevésbé változzon az eloszlás. Legyenek egy réteg bemenetei $x = (x^{(1)}, x^{(2)}, \ldots, x^{(d)})$ és

végezzünk el dimneziónként a normalizálást:

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}}.$$

Azonban szeretnénk, hogy ne veszítsünk a reprezentációs képességből, valamint hogy a hálóba illesztett transzformáció az identitás transzformációt is megtanulhassa. Ennek érdekében bevezetünk további két tanulható paramétert minden dimenzióhoz: $\gamma^{(k)}, \beta^{(k)}$. Ezek segítségével tudjuk majd nyújtani és eltolni a normalizált értéket:

$$y^{(k)} = \gamma^{(k)}\hat{x}^{(k)} + \beta^{(k)}.$$

Látható, hogy ha $\gamma^{(k)} = \sqrt{Var[(x)^{(k)}]}$ és $\beta^{(k)} = E[(x)^{(k)}]$, akkor az identitás transzformációt kapjuk.

---

1. Algorithm A Batch-normalizálás pszeudókódja lépésekbe szedve. A $\mathcal{B}$ egy mini-batch-et reprezentál. Első lépésben kiszámoljuk a mini-batch átlagot. A másodikban a mini-batch tapasztalati szórását. Majd az átlaggal és tapasztalati szórással normalizáljuk a bemenetet (az ε numerikus stabilitás miatt szerepel). És a végső lépésben alkalmazzuk a normalizált értékre a két tanulható paramétert. [5]

Bemenet: Legyenek az x értékei a mini-batch-en: $\mathcal{B} = \{x_{1\ldots m}\}$

A megtanulható paraméterek: γ, β

Kimenet: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch tapsztalati átlag}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \quad \text{// mini-batch szórás}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \varepsilon}} \qquad \text{// normalizálás}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \quad \text{// nyújtás és eltolás}$$

2.3. Augmentáció

Klasszifikáló hálók tanításánál bevett szokás, hogy az adathalmazt augmentálják vagy adathiány okokból vagy szimplán azért, hogy a klasszifikátor robosztusabb legyen. A szokásos *affin* augmentációk képeknél jellemzően valamilyen fokú forgatása a képnek, egy

részlet kivágása, függőleges tengelyre tükrözés. Emellett több próbálkozás is irányult neurális hálókkal való augmentálásra [9], illetve *GAN*-okkal történő stílus átvitelre [7].
Egy másik lehetőség GAN-okat használni új adatpontok generálására. Az eredeti GAN architektúrát használva azonban, amikor több osztályból is kikerülhet a generált elem, nem tudhatjuk előre melyik osztályba fog generálni. Ennek az orvoslására jöttek létre a Conditonal GAN-ok, amik a z zaj mellett, egy c feltételt is hozzátesznek a bemenethez, és így a diszkriminátor feladata is megváltozik: a valós-hamis döntés mellett, a kép osztályát is meg kell mondania (így valójában egy klasszifikáló háló a diszkriminátor). Erre építve készítettek Waheed és tsai. [16] egy *kiegészítő feltételes* GAN-t (Auxiliary Conditional Generative Adversarial Network, ACGAN), amellyel *Covid*-19 klasszifikáló háló tanítását segítették elő képek generálásával.
A GANok működését a 3.-ik fejezetben részletezzük.

2.4. Átviteli tanítás

A mindennapi életben előfordul, hogy ha megtanultunk elvégezni egy feladatot, akkor egy új és hasonló feladat megtanulásához fel tudjuk használni az első feladat során elsajátított képességeket. Például ha már tudunk gitározni, akkor könyebben vagy gyorsabban megtanulhatunk zongorázni, vagy ha megtanulunk főzni egy ételt, más receptek elkészítése is kevesebb gondot jelent. A feladat megtanulása után képesek vagyunk intelligensen, a releváns tapasztalatokat átemelni egy új feladat megoldására. Az átviteli tanulás (transfer learning) motivációja innen ered. Pan és Yang [8] 2010-ben írtak erről egy összefogó felmérést, de a fogalom már korábban is létezett. A fomális meghatározáshoz, előbb szükségünk lesz pár jelölésre és fogalomra.
Egy $\mathcal{D}$ *tartomány* két komponensből áll: egy $\mathcal{X}$ tulajdonság térből és egy $P(X)$ peremeloszlásból, ahol $X = \{x_1, \ldots, x_n\}$, és minden $x_i \in \mathcal{X}$. Esetünkben x_i egy röntgen kép tulajdonságainak valamilyen elkódolása, X egy minta halmaz és $\mathcal{X}$ pedig az összes kép halmaza.
Ha van egy adott $\mathcal{D} = \{\mathcal{X}, P(X)\}$ *tartomány*, definiálunk hozzá egy $\mathcal{T}$ feladatot, melyet szintén két komponenssel határozunk meg: egy $\mathcal{Y}$ címke térrel és egy objektív prediktív $f(\cdot)$ függvénnyel, amelyet nem ismerünk, de a tanuló adatból megtanulhatjuk. Röviden jelölve: $\mathcal{T} = \{\mathcal{Y}, f(\cdot)\}$. A tanuló adat $\{x_i, y_i\}$ párokból áll, ahol $x_i \in \mathcal{X}$ és $y_i \in \mathcal{Y}$. Valószínűségszámítási szempontból $f(x)$-et $P(Y|X)$-nek is írhatjuk, így $\mathcal{T} = \{\mathcal{Y}, P(Y|X)\}$. Az

$\mathcal{Y}$ címketér esetünkben: {*normális, nem-kovidos vírusos, kovidos*}
Az átviteli tanulásban szerepel egy forrás *tartomány* $\mathcal{D}_F$ és egy cél *tartomány* $\mathcal{D}_C$. Pontosabban $\mathcal{D}_F = \{(x_{F_1}, y_{F_1}), \ldots, (x_{F_n}, y_{F_n})\}$ a *forrás domain minta*, ahol $x_{F_i} \in \mathcal{X}_F$ mintakép és $y_{F_i} \in \mathcal{Y}_F$ a hozzátartozó címke. Hasonlóan $\mathcal{D}_C = \{(x_{C_1}, y_{C_1}), \ldots, (x_{C_n}, y_{C_n})\}$ a *cél tartomány minta*, ahol $x_{C_i} \in \mathcal{X}_C$ mintaelem és $y_{C_i} \in \mathcal{Y}_C$ a hozzátartozó címke. Ezen jelölések segítségével már tudjuk definiálni az átviteteli tanulást.

2.1. Definíció (Átviteli tanulás). *Adott $\mathcal{D}_F$ forrás domain és a hozzátartozó $\mathcal{T}_F$ feladat; $\mathcal{D}_C$ cél domain és a hozzátartozó $\mathcal{T}_C$ feladat. Az átviteli tanulás célja, hogy $f_C(\cdot)$ prediktív függvény teljesítményét javítsa $\mathcal{D}_C$-n, a $\mathcal{D}_F$ és $\mathcal{T}_F$-ből szerzett tudás segítségével, ahol $\mathcal{D}_F \neq \mathcal{D}_C$ vagy $\mathcal{T}_F \neq \mathcal{T}_C$.*

A $\mathcal{D}_F \neq \mathcal{D}_C$ kétféleképpen fordulhat elő. Vagy $\mathcal{X}_F \neq \mathcal{X}_C$ vagy $P(X_F) \neq P(X_C)$. Az elsőre példa mondjuk amikor különböző módszerekkel vannak elkódolva a képek, a másodikra pedig amikor más célből (mondjuk törés, zúzódás meghatározására) készült a röntgen kép. Ha adott konkrét $\mathcal{D}_F$ és $\mathcal{D}_C$, és $\mathcal{T}_F \neq \mathcal{T}_C$, az kétféleképpen fordulhat elő: $\mathcal{Y}_F \neq \mathcal{Y}_C$ vagy $P(Y_F|X_F) \neq P(Y_C|X_C)$. Az első esetre példa amikor különböző számú címke van a két feladatban, a másodikra pedig ha nagyon mások a mintaelemek eloszlása a címke halmazon.
Az átviteli tanulást ezen felül több kategóriába lehet sorolni, az alapján hogy a forrás és cél feladat megegyezik-e, illetve a két tartományhoz elérhetőek-e címkék. Esetünkben a cél feladat röntgen képek klasszifikálása lesz 3 kategóriába, a forrás feladat képek klasszifikálása volt 1000 kategóriába. Mindkét esetben elérhetőek a címkék a feladathoz, így az átviteli tanulásnak az *induktív* átviteli tanulás kategóiája, azon belül pedig a több feladatos tanulás (multitask-learning) esete érvényes. A forrás feladatról még írunk a 5. fejezetben.
Pan és Yang [8] három részfeladatot határozott meg az átviteli tanulás megvalósításához: *mit* vigyünk át, *hogyan* vigyünk át és *mikor* vigyünk át. A *mit* kérdés azt vizsgálja, hogy a tudás mely része vihető át a két *tartomány* vagy *feladat* között. Esetünkben egy korábban betanított modell reprezentációs képességét szeretnénk átvinni a saját modellünkbe. Miután lerögzítettük, hogy a tudás mely részét szeretnénk átvinni, a *hogyan* kérdés az átvitel módszertanával foglalkozik, a kifejlesztenddő tanuló algoritmusokkal amelyek megoldják a tudás átvitelét. Mi egy korábban betanított sikeres modell, első pár rétegének a súlyait fogjuk átemelni. Ez természetesen azt is jelenti, hogy a modellünk első pár rétegének a

struktúrájának meg kell felelnie a forrás modell első pár rétegével, hogy a súlytenzorok értelmezhetőek legyenek.
Végül pedig a *mikor* kérdés azt vizsgálja, hogy milyen esetekben érdemes használni az átviteli tanulást. Még specifikusabban, abban érdekelt, hogy mikor lehet az átviteli tanulás káros, azaz milyen esetben lehet eredménytelen az átvitel vagy akár káros is, ez utóbbit *negatív átviteli tanulásnak* is nevezik.

3. fejezet

Generative Adversarial Network modellek

A Generative Adversarial Network modellekt, avagy GAN-okat Ian Goodfellow és tsai. mutatta be 2014-ben [3]. A módszerben két neurális háló egymással versenyzik: a generátor és a diszkriminátor, és a tanulás az egymás elleni kiértékelés alapján történik. A generátor feladata, hogy minél hitelesebb hamisítványokat gyártson, a diszkriminátor feladata pedig, hogy felismerje a hamisító munkáját. Így nem egy címke alapján tanul a generáló, hanem az alapján, hogy sikerül-e megtévesztenie a diszkriminátort. Ez amiatt is fontos, mert rávilágít, hogy a generátor és diszkriminátor csak egyszerre tanulhat. Ugyanebben rejlik a GAN-ok tanításának a nehézsége is, hiszen ha túl jól teljesít az egyik, a másik nem kap igazán jó tanító jelet.

Formáisan a generáláshoz a generátornak egy p_z látens térből vett véletlen z vektorra van szüksége. A kép generálásához a generátor a zaj mellett egy G neurális hálót használ, a diszkriminátor hálóját pedig D-vel jelöljük. D a $[0, 1]$-be képez, és annak a valószínűségét adja meg hogy x valódi kép-e, ezáltal D valójában egy kétosztályú klasszifikáló háló. A két háló tanulása egy kétszemélyes játékként értelmezhető, ahol G hasznossági függvénye:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data(x)}}[logD(x)] + \mathbb{E}_{z \sim p_z(z)}[log(1 - D(G(z)))].$$

A diszkriminátor feladata, hogy maximalizálja $V(D, G)$-t, és a generátoré pedig, hogy minimalizálja $log(1 - D(G(z)))$-t. Gyakorlatban a tanítás elején gyenge a diszkriminátor, így $D(G(z))$ kicsi lesz, és emiatt gyenge a generátor tanító jele. Jobban működik, ha a generátor feladatát $logD(G(z))$ maximalizálására cseréljük, így erősebb gradienseket

kapunk a tanulás elején. A GAN-okat bemutatásuk óta, rengeteg helyen alkalmazzák, többek között képfelbontás javításra, arcok generálására, meglévő képek kiegészítésére és szövegből kép generálásra.

3.1. Evolúciós algoritmusok és a Lipizzaner modell

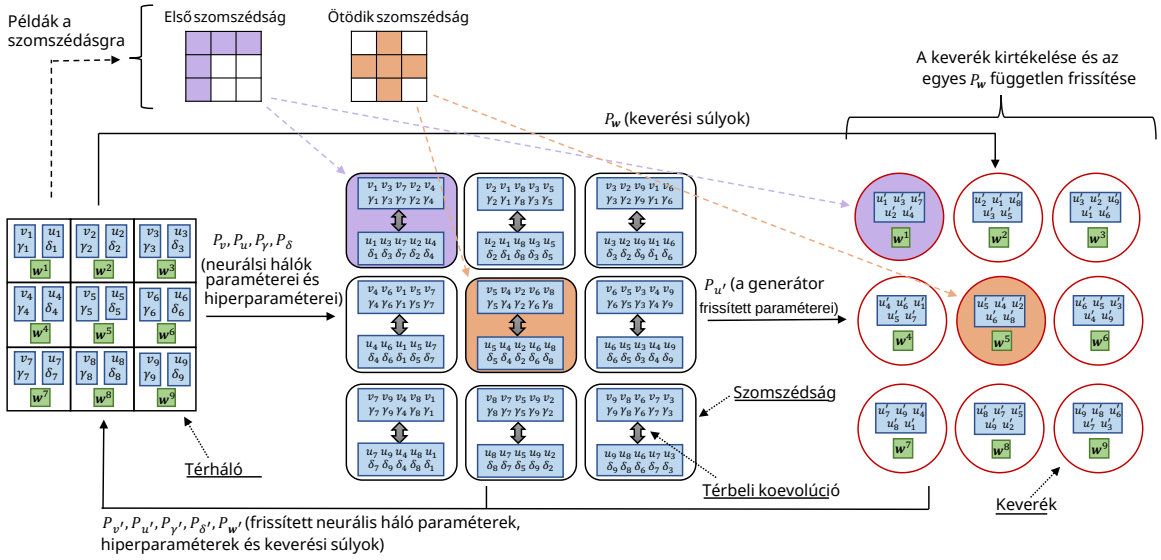
Az evolúciót megfigyelve jutottunk el az evolúciós algoritmusok ötletéig . Egyszerre több megoldó módszert versenyeztetünk egy feladat megoldására, és a legjobban teljesítő egyedekről származtatjuk, modellezük a következő generáció egyedeit. Így azok az egyedek amelyek valamilyen hasznos tulajdonságot kifejlesztettek, lesznek sikeresek és örökítik tovább a stratégiájukat. Ezen gondolat mentén már korábban létrejöttek olyan GAN tanító módszerek, amelyekben van egy generátor populáció és egy diszkriminátor populáció, amik egymással párhuzamosan fejlődnek. Így nem elég ha egy diszkriminátort meg tud téveszteni a generátor, és emiatt általánosabban kell hogy generáljon. A módszerben minden lehetséges generátor-diszkriminátor párt kiértékelünk egymás ellen és végrehajtjuk a tanítást. Azonban ez számításkpacitásilag nagyon költésgesnek bizonyult, mivel n darab generátor és diszkrimintor esetén $\mathcal{O}(n^2)$ a kiértékelendő párok száma. A szükséges számítások csökkentésének egy effektív módja a térbeli (toroid alapú) koevolúció, amely a versenyző populációk keveredését kontrollálja. A populációkat szétosztjuk egy térhálón és mindegyik cellának meghatározunk egy szomszédságot, így ha n_{cella} darab szomszédja van egy adott cellának, akkor az algoritmus $\mathcal{O}(n_{cella} * n)$ komplexitású lesz, amely egyre nagyobb és nagyobb populációk esetében $\mathcal{O}(n)$-nek tekinthető. A Lipizzaner ezt az ötletet valósítja meg.
A Lipizzaner egy gradiens alapú GAN tanító keretrendszer egy koevolúciós környezetben. A rendszer definiál egy rácsot, ahol a mezők szomszédságban állnak egymással (a 3.1.ábrán a lilával és narancssárgával kijelölt mezők egy-egy szomszédságot definiálnak). A rács minden mezőjbe egy GAN van (generátor-diszkriminátor pár), amelynek mindkét tagjára *egyed*-ként is tekintünk. A legjobb pár tanul egy iterációt (epochot) együtt, majd a mező elkéri a szomszédjainak a legjobb generátorát és diszkriminátorát. Így lesz egy mezőnek a szomszédjaitól egy-egy generátor-diszkriminátor párja, plusz egy saját, így alkotva egy generátor és egy diszkriminátor szubpopulációt a saját cellájában. Ezután a cella a szubpopulációból minden generátort kiértékel minden diszkriminátor ellen egy minta batch-en. A kiértékelés közben számontartja minden egyedhez a legkisebb loss-összeget,

ezzel rendelve egy mérőszámot az egyedhez ami alapján dönthetünk legjobb generátor és diszkriminátorról. A mező kiválasztja a legjobb generátort és legjobb diszkriminátort, őket tanítva tovább a következő iterációban. Fontos hogy tanítás csak egy párral történik, a többi egyedet csak kiértékeli egymás ellen ami jóval kevesebb számítási kapacitásba kerül. Minden iterációban a tanítás elvégzése előtt valamilyen valószínűséggel mutálja az egyes egyedekhez tartozó hiperparamétereket, majd az új paraméterrel végzi a tanítást. Mind a mutálás valószínűsége, mind a mutálás mértéke változtatható paraméter.

A Lipizzaner emellett minden cellában számon tart egy $\mathbf{w}$ *valószínűségi* vektort, melynek az egyes koordinátái az egyes szomszédoktól kapott generátorokhoz tartoznak, illetve egy koordináta a saját generátorhoz. Mivel valószínűségi vektorról van szó koordináták összege 1. Ez a vektor azt fogja meghatározni, hogy az adott szomszédtól származó generátor milyen valószínűséggel generál elemet a *példa* adathalmazba.

Az algoritmus során időnként a cellák kiértékelésére is sor kerül. Ilyenkor a cella nem egyszerűen a legjobb generátorát használja, hanem a $\mathbf{w}$ *valószínűségi* vektor által meghatározott arányok alapján készít egy *minta* adathalmazt. A cella kiértékelésének a menete úgy néz ki, hogy a cella készít egy k elemű mintahalmazt. A mintahalmaz készítésekor mindegyik kép esetében új sorsolás dönti el, hogy melyik generátor készíti az adott képet. Így ha w_i volt az i-edik generátorhoz tartozó *valőszínűségi* érték, akkor a k generált képből nagyjából $k * w_i$ darab-ot generált. A *minta* adathalmaz mellé vesz a cella k darab valódi képet, *valódi* adathalmaznak, majd kiszámolja a két adathalmaz *Fréchet-Inception*-távolságát, ez az érték jellemzi majd a cellának a $\mathbf{w}$-hez tartozó *kép generáló* minőségét. Ezt követően jön egy újabb evolúciós algoritmus lépés, ahol a $\mathbf{w}$-t mutálja. A mutáláshoz minden w_i koordinátához vesz egy $z_i \in \mathcal{N}(0, \sigma)$ értéket, és ebből $w'_i = \max(w_i + z_i, 0)$ módon származtatja, ahol σ az algoritmus megadható paramétere (alapértelmezetten 0.1). $\mathbf{w}'$-t normálva megkapja az új *valószínűségi* vektort. Az új vektorral szintén generál egy új *minta* adathalmazt, kiszámolja az új adathalmaznak a *valódi* adathalmazzal vett *Fréchet-Inception*-távolságát, és ez fogja jellemezni a cella $\mathbf{w}'$-hez tartotó *kép generáló* minőségét. *Fréchet-Inception*-távolságbnál a kisebb a jobb ezért ha a $\mathbf{w}'$-hez tartozó érték kisebb, akkor az új vektort tartja meg; ha nem, akkor az eredetit.

A rendszerben minden mező külön-külön és egyszerre végzi a tanítást. A Lipizzaner elosztott módon tanít, egy mezőn a tanítását egy kliens végzi és a feladat-kiosztási folyamatot és az eredmények összegzését egy *vezénylő* bonyolítja le.



3.1. ábra. A képen a Lipizzaner magas szintű architektúrája látható egy 3×3-as térhálón. $P_v = \{v_1, \ldots, v_9\}$ és $P_u = \{u_1, \ldots, u_9\}$ jelöli a neurális hálók paramétereit a diszkriminátor és generátor populációnak. $P_\gamma = \{\gamma_1, \ldots, \gamma_9\}$ és $P_\delta = \{\delta, \ldots, \delta_9\}$ a hiperparamétereit (pl. tanulási ráta), a diszkriminátor és és generátor populációknak. $P_w = \{\mathbf{w_1}, \ldots, \mathbf{w_9}\}$ pedig az keverési súlyokat. A $(\cdot)'$ jelölés a $(\cdot)$ értékét jelöli egy iterációnyi *koevolúció* után. [12]

4. fejezet

Augmentálás GAN-okkal

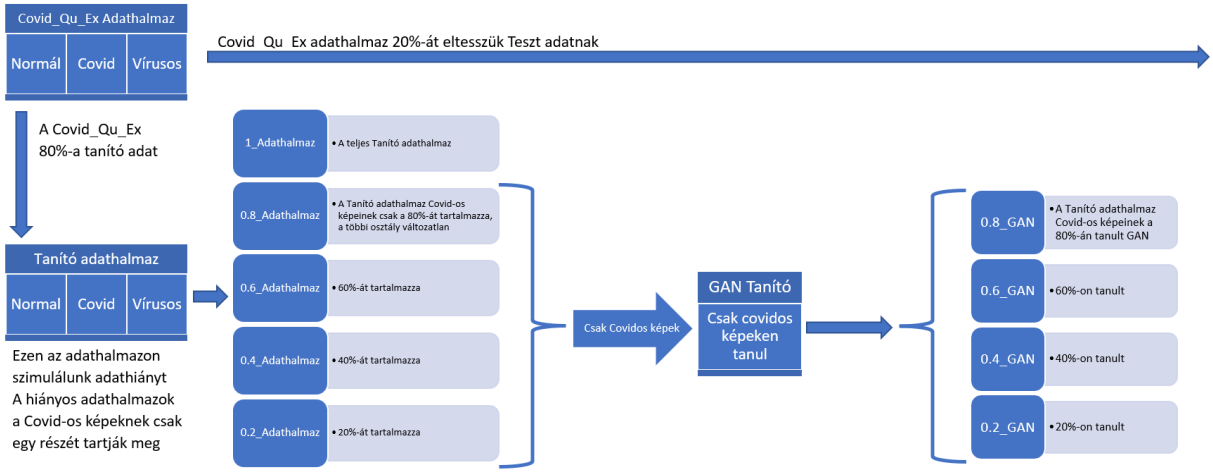
Sokféle módszer kialakult az évek alatt GAN-ok teljesítményének mérésére. A szakdolgozatomban adatagumentáció szempontjából vizsgáljuk a GAN-ok teljesítményeit. A központi kérdés, hogy milyen esetekben képes a GAN javítani a klasszifikátor teljesítményén. Tovább specifikálva, azt az esetet vizsgáltuk amikor klasszifikálás során van egy alulreprezentált osztályunk, és ebbe az osztályba szeretnénk GAN segítségével képeket pótolni, így kiegyensúlyozva az adathalmazt. Milyen feltételek mellett segít ez a módszer a klasszifikálás javításában?

A kérdés megválaszolásához tekintsünk egy kiegyensúlyozatlan adathalmazt, ahol csak egy osztályban van kevesebb kép mint a többiben. A hiányos osztály képeivel betanítunk egy GAN-t, majd kipótoljuk a hiányos osztályt az előbb említett GAN által generált képekkel. Így valójában két adathalmazunk lesz, egy hiányos és egy kipótolt. Betanítunk egy-egy klasszifikátort a két adathalmazon és összemérjük a teljesítményeiket.

A kísérletekhez a Covid-Qu-Ex adathalmazt [14] használtuk. Az adathalmaz tüdőröntgen képekből áll, amelyek 3 osztályba sorolhatók: normal, Covid és non-Covid. Az adathalmaz kiegyensúlyozott, minden osztályban nagyjából 10000 kép van.

Az elvégzett kísérleteknek két főbb fázisuk van: (1) Előkészítés és GAN tanítás, (2) Klasszifikátor tanítás.

Az előkészítés és GAN tanítás fázisát 4.1. ábrán foglaljuk össze. Először 20%-át félretesszük a képeknek, a klasszifikátorok tesztelésére. A maradék 80% kép lesz a tanító alaphalmaz. Az alaphalmazból fogunk elhagyni a Covid osztályból képeket így készítve hiányos adathalmazokat. Felmerül ilyenkor a kérdés, hogy mennyit hagyjunk el hogy a maradék képek továbbra is megfelelő minőségben reprezentálják a hiányos osztályt, és a

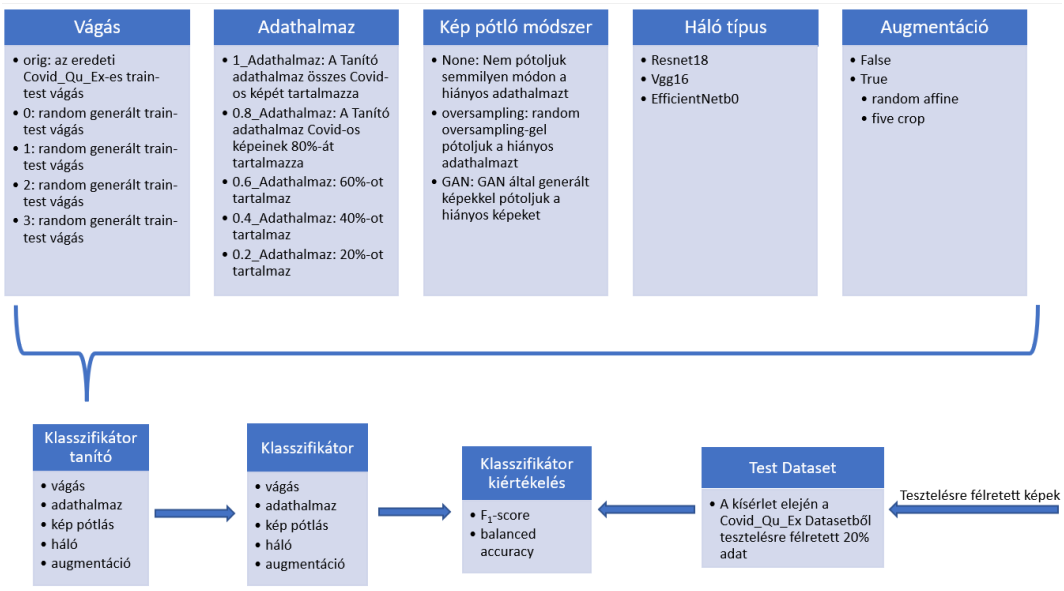


4.1. ábra. A GAN-ok betanításának összefoglalása

GAN is érdemben tudjon tanulni. Ezért több hiányos adathalmazt is készítünk, amikben különböző mértékben lesz alulreprezentálva a Covid osztály. Lesz egy adathalmaz amiben a tanító adathalmaz Covid osztálybeli képeinek a 80%-t tartjuk meg (lásd 4.1. ábrán *0.8_ Adathalmaz*), lesz egy ahol a 60%-át (*0.6_ Adathalmaz*), egy ahol a 40%-át (*0.4_ Adathalmaz*) és egy, ahol csak 20%-át (*0.2_ Adathalmaz*). Ezek az adathalmazok egymásba ágyazottak, vagyis a 60% megtartott Covid-os kép, a 80%-nak valódi részhalmaza. Lényegében ha az adathalmazra mint mintavételre gondolunk a tüdőröntgenképek eloszlásából, akkor egy mintavételből csak Covid-os adatpontokat hagyunk el, és azt vizsgáljuk, hogy a maradék Covid-os mintapont elég mértékben reprezentatív-e ahhoz, hogy a *GAN* rá tudjon tanulni. A hiányos adathalmazok elkészítése után, minden új adathalmazhoz készítünk egy hozzátartozó GAN-t, amely az adathalmaz Covid-os képein tanul (az ábrán *arány_ GAN*-ként jelenik meg, például a 80%-os adathalmazhoz 0.8_*GAN* tartozik). A GAN tanítást a *Lipizzaner* [12] segítségével végezzük (az ábrán *GAN Tanító*). A kísérlet második fázisában (lásd 4.2. ábra) már nem csak a Covid-os képek részhalmazával, hanem a másik két osztállyal is dolgoztunk. *Baseline*-ként minden *hiányos* adathalmazon betanítottunk egy klasszifikátort. Majd a hiányos adathalamzokat kipótoltuk a hozzájuk tartozó GAN által generált képekkel. A *kipótolt* adathalmazban szeretnénk, hogy minden osztályban nagyjából ugyanannyi kép legyen. Ezért azt tűztük ki célnak, hogy a Covid osztályban annyi kép legyen, mint a másik két osztály képei számának az átlaga. Így

$$\text{generált képek darabszáma} = \text{az átlag} - \text{meglévő Covid-os képek száma}. \tag{4.1}$$

Minden kipótolt adathalmazon betanítunk egy-egy klasszifikátort és összehasonlítjuk a teljesítményeiket a *Baseline*-val, a korábban félrerakott teszt adathalmazon.
Kontroll csoportnak készítünk olyan teszteket is, ahol a hiányos osztályt klasszikusabb módszerrel egészítjük ki. A mi esetünkben ez *random oversampling* volt. A meglévő Covid-os képek közül vettünk bele újra képeket az adathalmazba egy véletlen permutáció segítségével, amíg nem lett elég kép. Ha továbbra is hiányoztak képek, akkor újra vettük egy permutációját az eredetileg meglévő képeknek. Az utolsó körben csak annyi képet vettünk bele az adathalmazba, amennyi hiányzott ahhoz hogy teljes legyen, azaz, hogy a Covid osztályban annyi kép legyen, mint a másik két osztály átlaga.
A három vizsgált esetet összefoglaló néven *Kép pótlási módszernek* neveztük és az ábrán is ilyen néven jelenik meg. A három lehetőség: $None$, ez a hiányos adathalmaz, $oversampling$, a klasszikus augmentációs eset és GAN, a generálós módszer.
Továbbá minden kísérletet elvégeztünk úgy is, hogy használtunk *affin augmentációt*, és úgy is hogy nem (az ábrán röviden *Augmentáció*). Kétféle *affin augmentációt* használtunk: *random affin* elforgatás (3-4 fok), illetve five crop. Ezekről a Függelékben írunk bővebben.
A kísérleteket 3 különböző hálón végeztük el, ezeket a 4.2. ábrán *Háló*-ként jelölve. Ezek a hálók $ImageNet$-en előre betanított hálók és átviteli tanulást használva szabtuk át a 3 osztályos klasszifikálásra. A három háló: $ResNet18$, $VGG6$ és $EfficientNet\_b0$.
A kísérleteket elvégeztük 5 különböző tanító-teszt vágáson, hogy megbízhatóbb eredményeket kapjunk és szórást is tudjunk számolni (az ábrán *Vágás*-ként jelölve). Fontos megemlíteni, hogy nem generáltunk minden egyes GAN-t használó teszt esethez új képeket, hanem miután elkészült egy adott *vágás*-hoz és adott *arány*-hoz a GAN, legeneráltuk vele a hiányzó darabszámú képet (ezt az arány és 4.1 . egyenlet alapján pontosan ki lehetett számolni), majd a továbbiakban ezeket a képeket használtuk amikor GAN-nal történő képpótlást emlegetünk. (Például az *első* vágásnál, a 0.8-as adathalmazzal elkészített GAN-nal legeneráltuk a képeket, és minden háló esetében, amikor a 0.8-as *gan* eset méréseit végeztük, ezeket a legenerált képeket használtuk.)



4.2. ábra. Az klasszifikátorok tanításának magas szintű összefoglalása.

5. fejezet

Mérések

A méréseket az ELTE Matematikai Intézet Mesterséges Intelligencia Kutatócsoport GPU szerverein végezetük, NVIDIA A100-SXM4 80GB videókártyákkal. Minden tanítást (mind a GAN-ok, mind klasszifikátorokét), a *Neptune.ai* felhő szolgátatására *logoltuk* és onnan követtük nyomon. A Lipizzaner kódjai nagyrészt Python-ban íródtak, de mivel *docker swarm*-ot is támogat, és saját kezelőfelülettel is rendelkezik, így *C#* és *docker* fájlok is megtalálhatóak a fájlok között [1]. A klasszifikátorok és egyéb segédfájlok *Python*-ban íródtak. A tanításhoz a *Pytorch* nevű, mély tanulást támogató könytárát használtuk. Mindegyik felhasznált modell implementálva volt *Pytorchban*, így az átviteli tanulás egyszerűen kivitelezhető volt. A modelleket az *ImageNet*-es súlyukkal inicializáltuk, az utolsó sűrű réteg kimenetét lecsökkentettük 3-ra, hogy a feladatunkkal kompatibilis legyen a háló, és innen kezdtük a tanításokat. A mérések kódjai megtalálhatók a https://github.com/aielte-research/CovidGAN GitHub repositoryban.

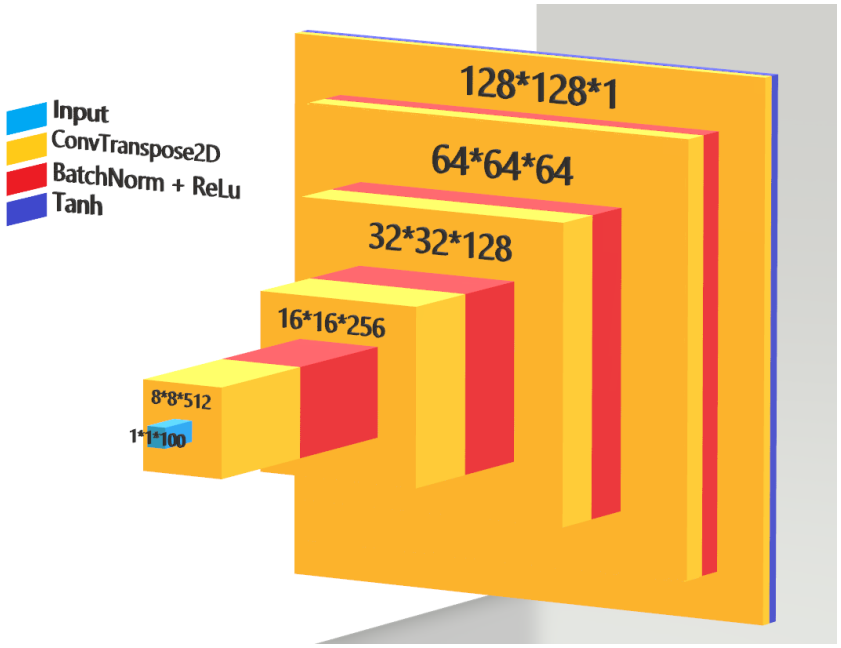
5.1. GAN paraméterek

A képek pótlására használt GAN-okat, Mathias Esteban *lipizzaner-covidgan* nevű projektjének [6] folytatásával készítettük, mely a Lipizzaner-t specializálta Covid-19-es képek generálására.
Mivel 5 *tanító-teszt* vágás volt, és mindegyiken elkészítettük a 4 darab hiányos adathalmazt $(0.8, 0.6, 0.4$ és$0.2)$, ezért ez összesen 20 GAN betanítását jelentette, melyek egyenként 8 órát vettek igénybe. Emiatt egy fixált paraméter halmazzal dolgoztunk. Minden

GAN-t egy $ConvolutionalGrayscale128 \times 128$ nevű hálón tanítottunk (lásd 5.2. ábra). A generáló háló bemenete $1 \times 1 \times 100$-as normális eloszlású *zaj* vektor, amit $1 \times 128 \times 128$ méretű, *szürkeárnyalatos* képet generál. A diszkriminátor halója $128 * 128$-as bemenettel rendelkezett, melyet *konvolúciós*, *batch-normalizálós* rétegek, majd *ReLu* aktivációs függvény követett, több rétegben egymás után, és a kimenete egy 1 dimenzós vektor volt. A tanítás paramtétereit a 5.1. ábrán foglaljuk össze.

Hyperparameter	Value
grid size	2×2
Network name	$convolutional\_grayscale128x128$
Default adam learning rate	2×10^{-4}
Number epochs	1000
Minibatch size	128
Mutation probability for learning rate	0.5
Mutation alpha for learning rate	10^{-4}
Score sample size	300
Score type	FID
Mixture mutation (σ)	0.01

5.1. ábra. A GAN tanítások hiperparaméterei.



5.2. ábra. A GAN háló vizuális reprezentációja (*convolutional_grayscale*128x128). A kék téglatest $1 \times 1 \times 100$ dimenziós zaj vektor, melyet $\mathcal{N}(0, 1)$-ből sorsolunk. A sárga rétegek *tanszponált konvolúciót* jelölnek, melyeknek a kimeneti mérete a blokk tetején jelzett. Majd ezek átmennek egy Batch-Normalizáló rétegen, amelyet egy ReLu aktivációs réteg követ. Az utolsó rétegben, a *tanszponált koncolúciót* egy *tangenshiperbolikusz* aktivációs függvény követ és ennek a kimenete lesz a generált kép. [6]

5.2. Klasszifikálás

A klasszifikálást egy *vágás* esetén végeztük el 4 hiányos adathalmazon ($0.8, 0.6, 0.4$ *és* 0.2), amelyek képeinek pótlására 2 lehetőség van (*oversampling* és *gan*), továbbá az 5 *baseline* modell (a 4 hiányos adathalmazhoz, plusz a teljes adathalmaz) elkészítése, majd minden eddigi esetet *affin augmentációval* és nélkül elkészíteni, amely egy vágásra, egy háló esetében $(4 * 2 + 5) * 2 = 26$ modell hiperparaméter optimalizálása. A hálókkal is számolva $3*26 = 78$ klasszifikátor hiperparaméter optimalizálását végeztük el. Az új *vágás*-ok esetében, azt feltételeztük, hogy a hiperparaméter nem változik. Egy klasszifikátor betanítása, esettől függően 30 perc és 2 óra között mozgott. A végleges hiperparaméterek a A. fejezetben szerepelnek (ResNet18: a A.1. táblázat, VGG16: A.2. táblázat és EfficientNet_b0: A.3 táblázat).

5.3. ResNet18

A mély reziduális hálókat (Deep Residual Networks, röviden ResNet) 2015-ben mutatták be He és tsai. [4], amivel meg is nyerték az azévi $ImageNet$ klasszifikáló versenyét és azóta is népszerű ez a technika. Ezért is választottunk egy ResNet alapú modellt, amellyel a kísérlet klasszifikálás részét elvégezzük. A cikkben több modellt is bemutatnak, a választásunk egy kisebb modellre esett: a Resnet18-ra, mely csupán $11,5$ millió paraméterrel rendelkezik.
A hálón elért eredményeket két mérőszámmal jellemeztük: F_1-score és balanced accuracy. Az eredmények 5.1. (F_1-score) és 5.2. (balanced accuracy) táblázatokban láthatók. A táblázat az átlagot, és az átlagtól vett abszolútértékes maximum szórást tartalmazza minden cellában. A sorok a különböző mértékű hiányos adathalmazokat jelentik, például a 0.8-as sor esetében a *Tanító Adathalmaz* Covid-os képeinek a 80%-át tartalmazza a hiányos adathalmaz. Az oszlopok 3 nagy kategóriára és 2 kicsire esnek szét. A nagy kategóriák a képpótlás módját jelentik (így például a *gan* azt jelenti, hogy a hiányos képeket GAN-ok segítségével pótoltuk), míg a két kicsi kategória az *affin augmentáció* alakalmazását jelenti ($True$ és $False$).
Az *affin augmentáció* használata jelentősen segít, ezért a $False$ és $True$ eseteket külön fogjuk összehasonlítani. Az összehasonlítások fő eszköze a boxplotok ábrái lesznek és a mérések átlagainak táblázata.
Először vizsgáljuk a ResNet18 hálón az *affin augmentáció* nélküli esetet. A hozzátartozó ábra 5.3 ábra, (a) részábrája, illetve a 5.1 táblázat. Minden összehasonlításnál az *alapvonal* (baseline) a képpótlás nélküli eset.
Az ábrán megfigyelhető, hogy a 0.8 és 0.6-os adathalmazokon az $oversampling$-es módszer bizonyult a legsikeresebbnek a kétféle valódi képpótlási módszer közül, azonban a *gan*-os módszer is javított a teljesítményen, a baseline-hoz képest. Az átlagokból is hasonló következtetést vonhatunk le. A 0.8-as adathalmazon amíg az *gan* nagyjából 1%-kal teljesített jobban átlagban, mint a *baseline*; addig az $oversampling$ 1.3%-os átlag-növekedést ért el (a táblázat 0.8-as sora, és $False$ oszlopai). A 0.6-as adadthalmaznál már a *gan* módszer ér el jobb átlagot, 2.8%-al jobban teljesít, mint a képpótlás nélküli eset, míg az $oversampling$ sem marad el sokkal ettől a teljesítmnytől.
A 0.4-es adathalmaznál szintén a *gan*-os képpótlási módszer hozza a legjobb eremények et. Látványosan jobban teljesít, mint a $None$ vagy $overasamling$ eset. Az átlagokat megnézve az is látszik, hogy az $oversampling$ ezesetben roszabb átlagot ért el, de a *gan* módszer

ismét 1.2% körüli javítást ér el átlagban. A 0.2-es esetben csak kis különbség figyelhető meg az ábrán az egyes esetek között. Mindkét módszer kicsit jobban teljesít, mint a *baseline*, azonban nem teljesen egyértelmű, melyik teljesít jobban a két módszer közül. A táblázatra tekintve a *gan* módszer éri el a legjobb átlagokat a 0.2-es sor False oszlopai közül, azonban nincs fél százalék különbség az eredmények között. Megjegyzendő, hogy az egyes boxok 5 adatpontból készültek el, és előfordul hogy ebből 2 kívülálló (például 0.4 *oversampling* vagy 0.2 *gan* eset), amely rámutat hogy vágástól függően akár $3-4$%-os eltérések is lehetnek az eredményekben, illetve, hogy több mérésre lenne szükség ahhoz, hogy szignifikánsat állíthassunk.

Az *affin augmentációt* használó eset következik. A felhasznált ábra a 5.4 ábra (a) részábrája, illetve a 5.1 táblázat.

A 0.8-as adathalmaz esetében az ábrán az látszik az árán, hogy mind a *gan*-os, mind az *oversampling*-es módszer egy kicsivel roszabbul teljesít, és ugyanezt a következtetést tudjuk levonni a táblázatból is (0.8-as sor, $True$ oszlopok): több mint 0.5%-kal magasabb az átlaga a képpótlás nélküli esetnek, mint a *gan*-osnak. Ez amiatt fordulhat elő, mert nem hiányzik túl sok kép, így nincs nagy hátrányban a képpótlás nélküli eset. A 0.6-os adathalmaznál, szintén az látható az ábrán, hogy kissé roszabb eredményeket ér el, mint a pótlás nélküli vagy az *oversampling*, azonban ez kevésbé tűnik súlyosnak, mint a 0.8-as esetben. Az átlagokra tekintve ez látszik beigazolódni: habár a pótlás nélküli eset átlaga a legnagyobb, az *oversampling* nem marad le sokkal, és a *gan*-hoz képest is csak 0.2%-kal teljesít jobban.

A 0.4-es adathalmazon mindkét módszer egyértelműen segít az ábra alapján, azonban csupán boxplot alapján nem lehet megállapítani, hogy egyik vagy másik többet javítana, illetve mennyit. Az átlagokat megnézve (0.4-es sor, $True$ oszlopok) az látható, hogy az *oversampling* módszer 0.4%-ot, a *gan* pedig majdnem 0.6%-ot javít az átlagokon.

A 0.2-es adathalmaznál az ábrán szintén ez a trend figyelhető meg. Ezesetben is mind a *gan*, mind az *ovesampling* ltványosan jobban teljesít mint a képpótlás nélküli alapeset. Az átlagokból (táblázat 0.2-es sora, $True$ oszlopok) az is kiderül, hogy az *oversampling* 1.3%-kal, a *gan* 1.5%-kal teljesít jobban átlagban, mint az alapeset.

Összességében az vonható le következtetésnek, hogy a $ResNet$ típusú hálón, minél kevesebb a hiányos adathalmazban a kép, annál jobban tud segíteni a *gan*-nal történő augmentáció. Mindeközben nem körözi le az *oversampling*-es módszert, és valószíínűleg a két módszer valamilyen keveréke tudná hozni a legjobb eredményeket. Erről kicsit bővebben a 6.1. fejezetben írunk még.

	None		oversampling		GAN	
	False	True	False	True	False	True
1	93.31±1.27	**93.86**±1.45				
0.8	91.94±2.76	**94.03**±2.67	93.28±1.98	93.63±2.07	92.92±2.07	93.47±2.23
0.6	91.02±2.90	**93.52**±2.53	92.84±2.18	93.50±1.99	92.88±2.38	93.30±2.49
0.4	91.48±2.04	92.64±2.44	91.43±2.26	93.02±2.18	92.61±2.20	**93.21**±2.37
0.2	90.78±1.85	90.83±2.10	90.94±1.66	92.16±2.44	91.06±2.17	**92.3**1±2.21

5.1. táblázat. Ez a táblázat a $ResNet18$ hálón végzett mérések eredményeinek az F_1-score átlagait és abszolutértékes maximum szórását tartalmazza. Minden sorban a legnagyobb átlag érték van kiemelve félkövérrel.

	None		oversampling		GAN	
	False	True	False	True	False	True
1	93.19±1.25	**93.76**±1.43				
0.8	91.81±2.74	**93.93**±2.64	93.17±1.93	93.54±2.02	92.81±2.01	93.37±2.19
0.6	91.02±2.66	**93.44**±2.48	92.76±2.09	93.41±1.94	92.78±2.34	93.22±2.44
0.4	91.37±2.03	92.57±2.41	91.37±2.22	92.95±2.13	92.55±2.16	**93.14**±2.34
0.2	90.77±1.85	90.85±2.11	90.97±1.67	92.15±2.43	91.06±2.18	**92.30**±2.24

5.2. táblázat. Ez a táblázat a $ResNet18$ hálón végzett mérések eredményeinek a *balanced accuracy − score* átlagait és abszolutértékes maximum szórását tartalmazza.

5.4. VGG16

A VGG modell-családot a Visual Geometry Group mutatta be 2014-ben [13] és egyben meg is nyerték a 2014-es ImageNet versenyt klasszifikálás és lokalizálás feladatban. A cikkben azt vizsgálták, hogy milyen hatással van a *konvolúciós* hálók teljesítményére, a hálók mélysége. A $VGG16$ 138 millió paraméterrel rendelkezik, ez a legnagyobb modellünk.
A háló F_1-score átlagai és maximum szórása a 5.3. táblázatban, a *balanced accuracy* átlaga és maximum szórása a 5.4. táblázatban találhatók. Az *affin augmentció* nélküli eredmények elemzését a 5.3. (b) ábra alapján készítjük.
A 0.8-as adathalmaznál, a kép pótlás nélküli módszer látszik a legsikeresebbnek, míg az

oversampling és a *gan* látszólag nem sokat változtat a sikerességen. Az általagokat megnézve a 5.3. táblázatban, ugyanez figyelhető meg. A 0.8-as sor, False oszlopai közül a pótlás nélküli (ábrán *None*) és *gan* pótlásos (ábrán *gan*) esetekben ugyanannyi az átlag, és az *oversampling* sem marad el sokkal . Ez magyarázható azzal, hogy a 0.8-as esetben, nem hiányzik túl sok kép a Covid-os képek közül. A táblázatban szintén megfigyelhető, hogy ugyan a *None* és *gan* eseteknek ugyanannyi az átlaga, de az abszolótértékes legnagyobb szórása kisebb a *gan*-nak.
A 0.6-os adathalmaz esetében, a boxploton a *gan* egy kicsit jobban teljesít, mint a másik kettő. Ennél az adathalmaznál az átlagok között is megfigyelhető (5.3. táblázat, 0.6-os sor, False oszlopok), hogy a *gan* éri el a legnagyobb átlagot, habár csupán 0.3% körüli javítással.
A 0.4-es adathalmazon, mind az *oversampling*, mind a *gan* kicsit rosszabbul teljesít a boxploton, mint a képpótlás nélküli eset, és a táblázatban látható, hogy a *None*-os esetnek nagyobb az átlaga, mint a másik kettőnek. Azonban itt sincs nagy különbség, csupán 0.2%-al jobb a pótlás nélküli eset átlaga, mint a *gan* módszeré.
A 0.2-es adathalmaznál az ábrán a *gan* és az *oversampling* is jobban teljesített, mint a képpótlás nélküli eset, és a kettő közül látszólag az *oversampling* volt hatásosabb. Ezt a megfigyelést támasztja alá az is, hogy a táblázatban a 0.2-es sorban, a *False* oszlopok közül az *oversampling*-hez tartozik a legnagyobb átlag, de a *gan* sem marad el sokkal, ám csak 0.1%-al teljesítenek jobban, mint az alapvonal.
Továbbra is megfigyelhetők, hogy egy-két boxplot esetében, akár 2 kívül álló pont is van, amely a kis számú mintaelemnek tudható be. Ennél a hálónál elmondható, hogy az egyes képpótlási módszerek eredményei nem különböznek annyira egymástól, amelyre a $VGG16$ háló mérete lehet egy indok.
Említésre méltó, hogy a 0.4-es adathalmaz esetében, a legjobb mért átlag egy *augmentáció nélküli* esethez tartozik, ahol nem használtunk képpótlási módszert. Szintén megjegyzendő, hogy a teljes adathalmazos mérések (a képen az 1.0-hoz tartozó boxplot) roszabb eredménnyekel rendelkeznek, mint a 0.8-as képpótlás nélküli mérések (a 5.3. (*b*) ábra első két sárga doboza). Ez magyarázható egy rosszabbul sikerült hiperparaméter-optimalizálással.
Az *affin augmentációt* használó tesztesetek elemzését a 5.4. (*b*) ábra és a 5.3. táblázat alapján végezzük.
A 0.8-as adathalmazon mind az *oversampling*, mind a *gan*-os módszer enyhe javulást hoz, a kegkisebb mért értékek magasabban vannak, mint a képpótlás nélküli esetben. Az átlagokat tekintve is hasonló következtetést lehet levonni: ugyan a *gan*-os eset átlaga a

legnagyobb, azonban csupán alig 0.1%-el nagyobb a másik kettőnél (5.3. ábra, 0.8-as sor, *True* oszlopok). Magyarázatként szolgálhat a háló mérete és az a tény, hogy nem sok kép hiányzik még ebből az adathalmazból.
A 0.6 adathalmaz esetében a boxplotok alapján az *oversampling* egy kicsit jobb, a *gan* pedig kicsit rosszabb eredményeket ért el, mint a képpótlás nélküli módszer. Ezt megerősíti a táblázat 0.6-os sora: a *True* oszlopok közül, a *None*-hoz tartozik a legnagyobb átlag, az *oversamplingé* egy kicsit kisebb, és a *gan* ugyan elmarad, de csak 0.2%-al. A 0.4-es adathalmazon a boxplot ábrán, a *gan*-os módszer nagyjából ugyanúgy teljesít mint a *baseline*, míg az oversampling egy kicsit roszabbul. A táblázat is ugyanezt erősíti meg: míg a 0.4-es sor, *True* oszlopai közül, a *None* és *gan* oszlopbeli átlagok megegyeznek, addig az *ovesampling* átlaga kicsit kisebb. Nagy különbségek továbbra sem figyelhetők meg.
A 0.2-es adathalmazon a boxplokon látható, hogy a *gan*-os módszer jobb eredményeket ér el, mint a másik kettő: a medián vonala magasabban van, mint a másik kettőé és az egész eloszlás nagyjából 1%-al magasabban van, ami jelentős különbség. A táblázat 0.2-es sorát, és *True* oszlopait megvizsgálva, szintén erre a következtetésre juthatunk: a *gan*-os módszer átlaga, majdnem 1%-kal nagyobb, mint a képpótlás nélküli eseté. A legtöbb esetben az látszik, hogy a *VGG* modell elég nagy és nem számít sokat a képpótlás, azonban amikor már csak 20%-a maradt meg a *Tanító* adathalmaz *Covid*-os képeinek, sikerült javítania a *gan*-os módszernek. Így az eredmények még ilyen nagy modell esetében sem cáfolják a *GAN*-nal történő adatpótlás lehetőségének validitását.

	None		oversampling		GAN	
	False	True	False	True	False	True
1	92.64±1.35	**93.65**±1.32				
0.8	93.36±2.37	93.54±1.95	93.31±1.48	93.49±1.77	93.36±1.55	**93.57**±1.89
0.6	92.90±1.68	**93.30**±2.07	92.85±1.90	93.26±1.94	93.16±1.81	93.07±2.48
0.4	**93.21**±1.67	92.95±1.72	92.48±2.07	92.87±1.89	93.07±2.12	92.95±2.18
0.2	91.39±1.33	91.22±1.21	91.54±1.57	91.62±1.94	91.5±1.31	**92.17**±1.14

5.3. táblázat. Ez a táblázat a $VGG16$ hálón végzett mérések eredményeinek A az F_1-score átlagait és abszolutértékes maximum szórását tartalmazza.

	None		oversampling		GAN	
	False	True	False	True	False	True
1	92.54±1.36	**93.56**±1.34				
0.8	93.26±2.33	93.44±1.91	93.20±1.41	93.39±1.72	93.24±1.49	**93.47**±1.84
0.6	92.80±1.63	**93.21**±2.02	92.75±1.87	93.18±1.90	93.06±1.77	92.98±2.45
0.4	**93.14**±1.65	92.88±1.72	92.39±2.07	92.81±1.86	93.00±2.12	92.89±2.19
0.2	91.39±1.38	91.22±1.25	91.54±1.6	91.62±1.96	91.5±1.33	**92.15**±1.22

5.4. táblázat. Ez a táblázat a $VGG16$ hálón végzett mérések eredményeinek a *balanced accuracy − score* átlagait és abszolutértékes maximum szórását tartalmazza.

5.5. EfficientNet

Az EfficientNet-modellcsaládot 2019-ben mutatta be Tan és V.Le [15]. A modellcsalád egy skálázási módszert alkalmaz, ahol a modell mélységét, a rétegek számát és a réteg bemeneti felbontását növelve, minél nagyobb pontosság elérése a cél, minél kevesebb, a modell tanításához szükséges floating point műveletek számával (FLOPs). A módszer meghatározza azt a 3 konstanst, amelyek segítségével a legeffektívebben lehet (pontosságban és FLOPs-ban) a modellek méretén növelni. Mi az $EfficientNet\_b0$ alapmodellüket választottuk harmadik hálónak, amelyen klasszfikálást tesztelünk. Ez a modell csupán 5.2 millió paraméterrel rendelkezik, így még a $ResNet18$-as modellnél is kisebb. A korábban kialakított szokás szerint, *affin augmentáció* használat alapján külön elemezzük ki az eredményeket. Az *affin augmentációt* nem használó eset kiértékelése a 5.3 ábra (c) alábrája alapján és 5.5 táblázat alapján végezzük. A 0.8-as adathalmaz esetében a boxploton, látszólag nem sokban különbözik egymástól a 3 képpótlási módszer eredményei. Annyi tisztán látszik, hogy mindkét pótlási módszer feljebb hozta a legkisebb mért értékeket. A táblázat 0.8-as sorának $False$ oszlopaira tekintve is ennek megerősítését kapjuk, a 3 módszer közül az *oversampling* kicsit jobban teljesít mint a másik kettő, de csupán 0.2% a különbség az átlagok között. Magyarázat lehet, hogy ennél az adathalmaznál nem hiányzik túl sok Covid-os kép, így az egyes képpótlási próbálkozások, még inkább vágáshoz hasonló módon viselkednek.
A 0.6-os adathalmaznál, a boxplotokat tekintve látható, hogy a *gan*-os módszer felül múlja, mind a képpótlás nélküli esetet, mind az *oversampling*-et. Az átlagokból erre megerő-

sítést nyerünk: a *gan* mérés átlaga 0.5%-kal nagyobb, mint a képpótlás nélkülié, és még az *oversampling*-et is felülmúlja 0.4%-kal.
A 0.4-es adathalmaznál szintén jól láthatóan jobb eredményeket ér el a *gan*, mint a pótlás nélküli eset; és még az *oversampling*-hez képest is egy kevéssel jobb a teljesítménye. A táblázat alapján számszerűsítve az eredményeket (0.2-es sor, $False$ oszlopok) azt kapjuk, hogy ezen adathalmaz esetében is 0.5%-kal teljesít jobban a *gan*, mint az alapvonal és az *oversampling* módszer.
A 0.2-es adathalmaznál, ugyan eléggé nagy szórása van a méréseknek, de a *gan* módszer itt is látható módon javít a teljesítményen. Ugyanakkor az *oversampling* jobb eredményeket ér el mint a *gan*. A táblázat alapján azonban látható, hogy a *gan* és az *oversampling* módszer átlaga gyakorlatilag ugyanannyi, és mindketten 0.6% feletti értéket javítanak az alapvonal átlagán.
Az *affin augmentációt* használó mérések kiértékelésére a 5.4. ábrát és 5.5. táblázatot használjuk.
A 0.8-as adathalmazon megfigyelhetően jobb eredményeket ér el a *gan* módszer, mint a pótlás nélküli módszer vagy az *oversampling*. A táblázatban (0.8-as sor, $True$ oszlopok) az is látható, hogy a *gan* módszer átlaga, mindkét módszernél 0.4%-al nagyobb. Kiemelendő, hogy a *gan*-os képpótlás 94.45%-os átlagot ért el, amely majdnem 0.5%-kal jobb, mint bármely másik hálón elért eredmény.
A 0.6-os adathalmazon az figyelhető meg, hogy habár a *gan* nem hoz kiemelkedően magas eredményeket, azonban konzisztensen 94% közeli eredményeket ér el, míg az *oversampling*-nél előfordul, hogy 93% alá esik; a pótlás nélküli esetben pedig 92% körüli mérés is akad. Az átlagokra tekintve (0.6-os sor, $True$ oszlopok) ez mégjobban kiemelődik, mert a *gan* átlaga 0.5%-kal nagyobb, mint a képpótlás nélkülié. Emellett az átlagokból az is kiderül, hogy a *gan* eset, 94% feletti átlagon teljesített, amely a második legnagyobb elért átlag.
A 0.4-es adathalmazon is az figyelhető meg, hogy mind a *gan*-os, mind az *oversampling*-es módszer jobban teljesít, mint a képpótlás nélküli. Azonban első ránézésre nem tiszta, hogy melyik volna jobb a kettő közül. A táblázat 0.4-es sorát megnézve az látható, hogy ugyan nem sokkal (0.1% alatti értékkel), de az *oversampling*-es módszer jobb átlaggal rendelkezik, mint a *gan*-os. Emellett mindkettő 0.4% körüli értékkel jobban teljesít, mint az alapvonal.
A 0.2-es adathalmaznál, az ábrán jól látható módon, a *gan* jobban teljesít, mint a pótlás nélküli eset; és az *oversampling* mindkettőnél sikeresebb. Az átlagokat megnézve az *oversampling* 1.1%-al teljesít jobban, mint az alapvonal, de a *gan* is 0.7%-al jobb ered-

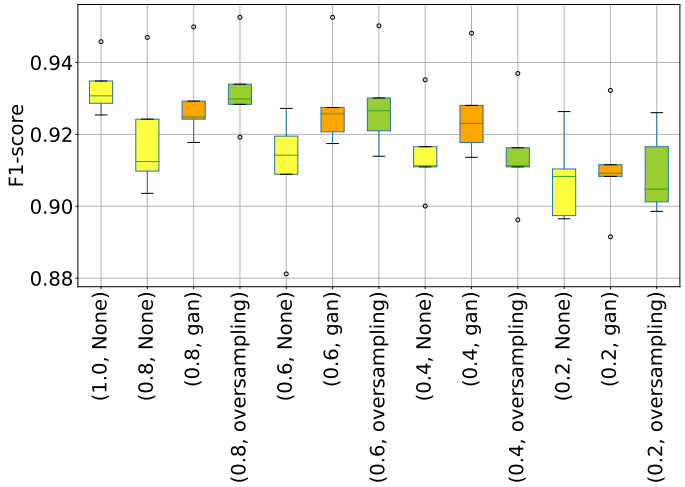
ményeket ért el. Összességében az vonható le következtetésnek, hogy az $EfficientNet$ típusú hálónál a *gan*-nal történő képpótlás inkább segített a hálónak tanulni. Minél nagyobb volt az adathiány, annál többet tudott segíteni, bár nem mindig többet, mint az *oversampling*. Valószínűleg a két módszer valamilyen vegyes használata hozná a legjobb eredményeket.

	None		oversampling		GAN	
	False	True	False	True	False	True
1	93.13±1.59	**93.73**±1.48				
0.8	93.16±1.78	94.04±1.66	93.31±2.56	94.03±2.11	93.13±1.57	**94.45**±1.92
0.6	92.97±2.08	93.54±2.16	93.03±1.85	93.81±2.10	93.49±1.48	**94.09**±1.78
0.4	92.70±2.12	93.42±1.87	92.68±1.84	**93.87**±1.68	93.23±2.32	93.75±1.92
0.2	91.79±1.67	92.34±1.98	92.39±1.92	**93.49**±1.92	92.38±1.58	93.07±1.95

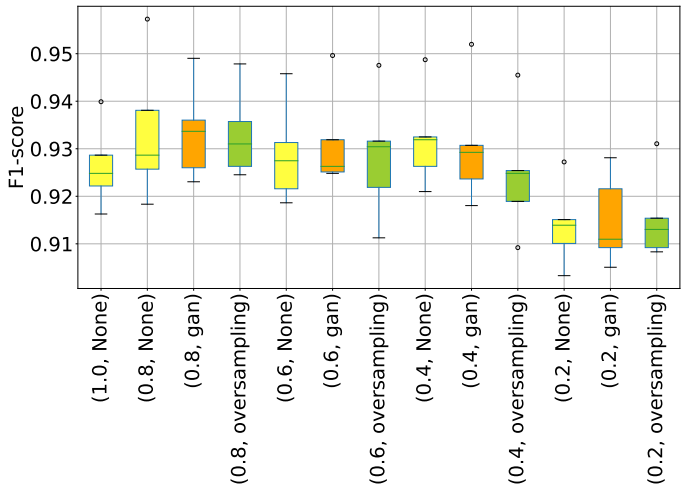
5.5. táblázat. Ez a táblázat a $EfficientNet\_b0$ hálón végzett mérések eredményeinek F_1-score átlagait és abszolutértékes maximum szórását tartalmazza.

	None		oversampling		GAN	
	False	True	False	True	False	True
1	93.03±1.54	**93.64**±1.51				
0.8	93.03±1.72	93.95±1.61	93.22±2.51	93.95±2.07	93.01±1.49	**94.37**±1.88
0.6	92.85±2.03	93.45±2.11	92.92±1.78	93.74±2.06	93.39±1.41	**94.03**±1.74
0.4	92.61±2.07	93.36±1.82	92.59±1.79	**93.79**±1.63	93.14±2.31	93.68±1.88
0.2	91.73±1.67	92.31±1.97	92.36±1.9	**93.45**±1.92	92.35±1.58	93.06±1.93

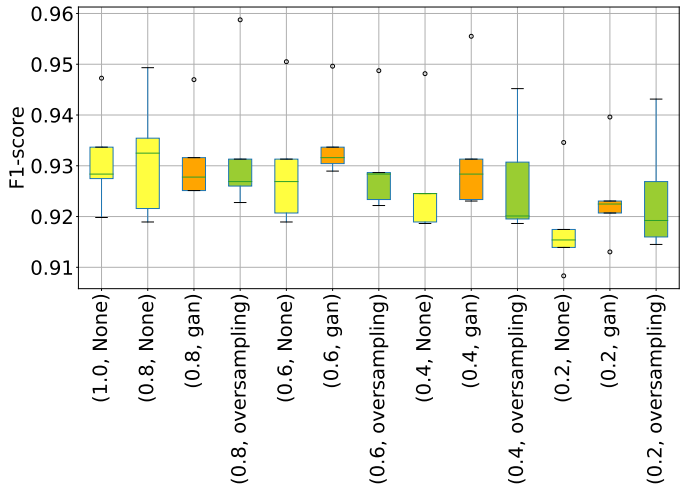
5.6. táblázat. Ez a táblázat a $EfficientNet\_b0$ hálón végzett mérések eredményeinek a *balanced accuracy* érték átlagait és abszolutértékes maximum szórását tartalmazza.



(a) ResNet18

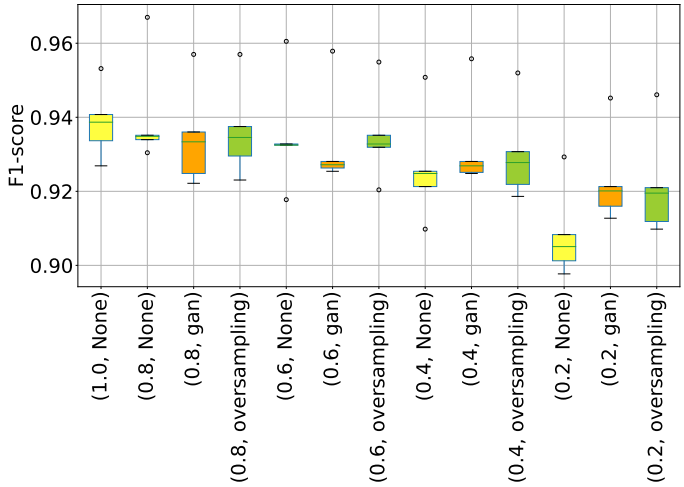


(b) VGG16

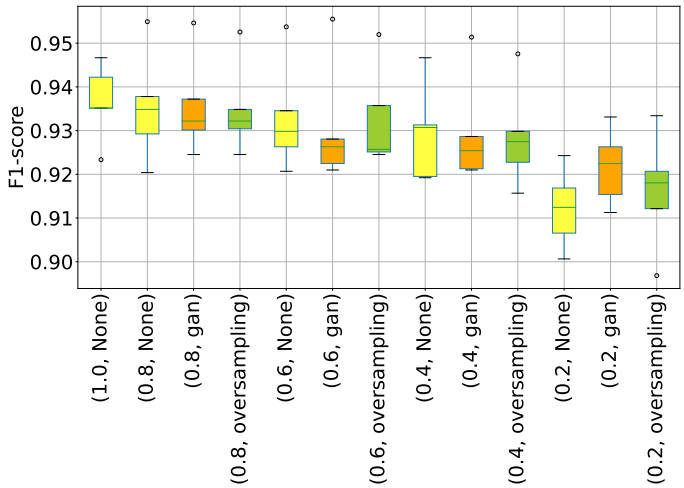


(c) EfficientNet_b0

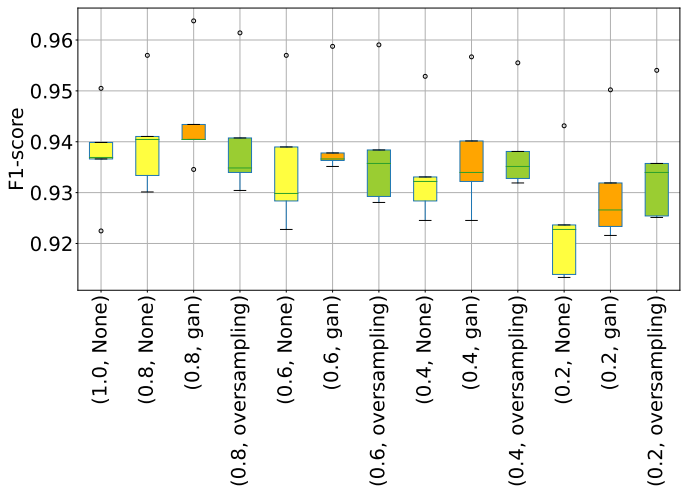
5.3. ábra. A különböző hálókon (ResNet18, VNN16 és EfficientNet_b0) mért F_1-score értékek boxploton, affin augmentáció nélküli esetben. A dobozok színei az egyes képpótlási módszerekhez tartoznak, (sárga: *None*; narancs: *gan*; zöld: *oversampling*).



(a) ResNet18



(b) VGG16



(c) EfficientNet_b0

5.4. ábra. A különböző hálókon (ResNet18, VGG16 és EfficientNet_b0) mért F_1-score értékek boxploton, affin augmentációt használó eset. A dobozok színei az egyes képpótlási módszerekhez tartoznak, (sárga: *None*; narancs: *gan*; zöld: *oversampling*).

6. fejezet

Összefoglalás

Az eredmények alapján az a következtetés vonható le, hogy mindhárom háló esetében képes volt a *GAN*-nal történő módszer segíteni a klasszifikálásban. Minél kevesebb volt az eredeti kép a *hiányos* adathalmazban, láthatóan annál többet javított a képek generálása. Habár nem mindig teljesített jobban, mint az *oversampling*, de jellemzően tudta tartani annak teljesítményét. Emiatt feltehetőleg a két módszer valamilyen keverése lehetne az amellyel a legjobb eredményeket el lehetne érni: a hiányzó képek x%-át *gan*-nal pótoljuk, a $(1 - x)$%-át pedig *oversampling*-gel. Az is megfigyelhető volt, hogy a kisebb hálóknak gyakrabban, és többet javított a teljesítményén a módszer.
Mivel alapvetően 90% feletti pontosság volt erlehető ezeken az adathalmazokon, ezért 1%-os javítás, a maximum elérhető javítások legalább 10%-át jelenti. Az egyes adathalmazokon 0.2% és 2.8% közötti javulásokat sikerült elérni, ezért ezek hozzávetőlegesen 2% és 28% közötti javításnak felelnek meg, a maximum elérhető javításokhoz viszonyítva.

6.1. További kutatási lehetőségek

A GAN által generált képek minőségén még tovább lehetne javítani, ha a generált kép előbb átmenne egy sikeres diszkriminátoron, és csak akkor használnánk a generált képet, ha a diszkriminátort sikerült megtévesztenie. Ezzel maguknak a generált képeknek a minőségét javíthatnánk.
Az összefoglalásban említett *gan-oversampling* vegyes képpótlási módszert is érdemes lehetne letesztelni. Többféle aránnyal is ki lehetne próbálni és így összehasonlítani.

Szintén érdekes lenne, hogy ha minden Covid-os képet lecserélnénk GAN által generált képekre és leellenőrizni az ilyen típusú adathalmazon tanult klasszifikátorok teljesítményét.
Pár klasszifikátor tanítási esetben a hiperparaméter-optimalizálás alatt, a tanító alaphallmazon a pontosság (*train accuracy*) elérte a 100%-ot a 15. iteráció környékén, emiatt nagyon lecsökkent a tanító jel. Ilyen esetekből több alkalommal is volt olyan adathalmaz amelynél *gan* által generált módszerrel pótoltuk a képeket. Emiatt felmerült, hogy lehet hogy túlságosan is *Covidos* jellegűek lettek a generált képek, így könnyen rátanult a háló. Ha ezeket a generált képeket egy kicsit elhomályosítanánk, mielőtt belerakjuk az adathalmazba és ha a homályosítás segítene a tanításon akkor a túl erős *Covdios* jelleg volt a probléma. Ez azért is volna baj, mert a tüdőröntgenkép *Covidos* jellege nem bináris dolog, hanem sokkal inkább egy spektrum és a spektrum minál nagyobb részéről szeretnénk generálni.
A generátorokhoz tartozó diszkriminátorokat megpróbálhatjuk kiértékelni mint *Covid - Nem Covid* klasszifikátorokat. Mivel a diszkriminátorok feladata a *Covid*-os képekről eldönteni, hogy valódi-e vagy generált, így feltehetőleg rátanul arra, hogy milyen tulajdonságokkal rendelkezik egy valódi *Covidos* kép. Ezáltal azt is el tudja dönteni, hogy egy adott kép rendelkezik a *Covidos* tulajdonságokkal, amely pedig már egy *Covid - nemCovid* klasszifikálásnak felel meg. Ha ez sikeres volna, akkor akár a diszkriminátort a klassszifikálás menetébe is be lehetne építeni, egyfajta bónusz klasszifikátorként.
Nagyobb térhálón is ki lehetne próbálni a GAN tanítást, mert nem használtuk ki a szerver teljes kapacitását, amikor csupán $2 * 2$-es térhálón tanítottunk, azonban ez még talán növelhető $3 * 3$-ra vagy $4 * 4$-re, amely a *Lipizzaner* modell előnyeit mégjobban kihozhatná.
A Lipizzanerbe lehetne implementálni a DCGANokat [10], amelyek segítségével alkalmas lenne *affin augmentált* képek generálására is. Emellett a hiperparaméter mutálás elég gyorsnak tűnik. Egy megoldásként ritkítani lehetne a hiperparaméterek mutálását, hogy ne vesszen el a tanító jel.
A *GAN* és klasszifikátor tanítására lehet tekinteni egy nagy rendszerként is, és előfordulhat, hogy a *GAN* tanítási paraméterei nem függetlenek a klasszifikátorétól. Vagyis lehet hogy egy fajta hiperparaméter beállítása a *GAN* modellnek az egyik vagy a másik klasszifikáló hálónak jobban segít.

Irodalomjegyzék

[1] Erik Hemberg Abdullah Al-Dujaili Tom Schmiedlechner és Una-May O'Reilly. *ALFA-group/lipizzaner-gan*. `https://github.com/ALFA-group/lipizzaner-gan.git`. 2018.

[2] Vincent Dumoulin és Francesco Visin. „A guide to convolution arithmetic for deep learning". *arXiv preprint arXiv:1603.07285* (2016).

[3] Ian Goodfellow és tsai. „Generative adversarial networks". *Communications of the ACM* 63.11 (2020), 139–144. old.

[4] Kaiming He és tsai. „Deep residual learning for image recognition". *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, 770–778. old.

[5] Sergey Ioffe és Christian Szegedy. „Batch normalization: Accelerating deep network training by reducing internal covariate shift". *International conference on machine learning*. PMLR. 2015, 448–456. old.

[6] Esteban Mathias. *lipizzaner-covidgan*. `https://github.com/mathiasesteban/lipizzaner-covidgan`. 2020.

[7] Agnieszka Mikołajczyk és Michał Grochowski. „Data augmentation for improving deep learning in image classification problem". *2018 international interdisciplinary PhD workshop (IIPhDW)*. IEEE. 2018, 117–122. old.

[8] Sinno Jialin Pan és Qiang Yang. „A Survey on Transfer Learning". *IEEE Transactions on Knowledge and Data Engineering* 22.10 (2010), 1345–1359. old. DOI: `10.1109/TKDE.2009.191`.

[9] Luis Perez és Jason Wang. „The effectiveness of data augmentation in image classification using deep learning". *arXiv preprint arXiv:1712.04621* (2017).

[10] Alec Radford, Luke Metz és Soumith Chintala. „Unsupervised representation learning with deep convolutional generative adversarial networks". *arXiv preprint arXiv:1511.06434* (2015).

[11] Sebastian Ruder. „An overview of gradient descent optimization algorithms". *arXiv preprint arXiv:1609.04747* (2016).

[12] Tom Schmiedlechner és tsai. „Lipizzaner: a system that scales robust generative adversarial network training". *arXiv preprint arXiv:1811.12843* (2018).

[13] Karen Simonyan és Andrew Zisserman. „Very deep convolutional networks for large-scale image recognition". *arXiv preprint arXiv:1409.1556* (2014).

[14] Anas M. Tahir és tsai. *COVID-QU-Ex Dataset*. 2022. DOI: 10.34740/KAGGLE/DSV/3122958. URL: https://www.kaggle.com/dsv/3122958.

[15] Mingxing Tan és Quoc V. Le. „EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". *CoRR* abs/1905.11946 (2019). arXiv: 1905.11946. URL: http://arxiv.org/abs/1905.11946.

[16] Abdul Waheed és tsai. „Covidgan: data augmentation using auxiliary classifier gan for improved covid-19 detection". *IEEE Access* 8 (2020), 91916–91923. old.

A. függelék

Mély tanulásos fogalmak

A.1. Optimalizáló algoritmusok

A neurális hálók tanítása közben a paraméterek frissítésének elvégézésre többféle módszer is kialakult. ezek közül ismertetünk néhányat [11].

Sztohasztikus gradiens ereszkedés (stochastic gardient descent), röviden SGD: Az egyik legegyszerűbb algoritmus a paraméterek frissítésére, egy η hiperparaméterrel rendelkezik. Legyenek a háló paraméterei θ, legyen J a veszteségfüggvény. Ekkor az SGD minden $x^{(i)}$ tanító példára, és hozzátartózó $y^{(i)}$ címkére frissíti a háló paramétereit.
$\theta = \theta - \eta \cdot \nabla_\theta J(\theta, x^{(i)}, y^{(i)})$

Adam optimalizáló: Adaptive Momentum Estimation, egy olyan optimalizáló algoritmus, amely minden paraméterhez adaptív tanulási rátát tart számon. Az egyik vezérlő gondolat mögötte, az hogy regularizáljuk a paraméterek gradiensének nagyságát, hogy ne ugráljon a paraméter értéke, hanem a tanulás folyamán valamilyen értékhez konvergáljon. Ennek érdekében a paraméterekhez számon tart egyfajta összegzett súlyvektort, az alkalmazott gradiensek alapján.
A másik ötlet amit alkalmaz, hogy tartsunk számon egy lendületvektort a paraméterekhez, amelyet valamilyen exponenciálisan lecsengő módszerrel firssítgetjük az új gradienssel. Hasonlóan egy dombról leguruló nehéz labdához, az a cél, hogy a gradiens iránya ne változzon nagyokat. Emiatt az Adam úgy viselkedi, mint egy dombról leguruló nehéz labda, amelyre hat a súrlódás. Az algoritmus a η mellett rendelkezik még β_1, β_2 hiperparaméterekkel, amelyek 0.9 és 0.999 körül szoktak lenni. (1 közeli értékek körül):
$g_t = \nabla_\theta J(\theta_{t-1})$

$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$ //lendületvektor
$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$ //súlyvektor
Az algoritmus m_t és v_t-t nullvektoroknak inicializálja a módszer, amely azt eredményezte, hogy a tanítás elején a 0 felé torzulnak, ezért még egy korrekciós taggal megszorozza a két értéket.
$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$
$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$
Ezen jelölések segítségével kifejezve a paraméterek frissítése:
$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \varepsilon} \hat{m}_t$
AdamW: Az AdamW, az Adam-nek a súly lecsengős (weight decay) változata. Mindössze g_t kiszámolásában történik változás, egy w wight decay hiperparaméter szerint.
$g_t = \nabla_\theta J(\theta_{t-1}) + w\theta_{t-1}$

A.2. Aktivációs függvények

Az aktivációs függvények a nemlinearitás megvalósítására vannak a neurális háló rétegei között (különben az egész neurális háló összeesne egy mátrixszorzássá). Két gyakori függvényt említünk meg, a $ReLu$-t és a $Tanh$-t.
$ReLu(x) = \max(x, 0)$. Ennek a függvénynek a 0 fölött a deriváltja 1, kevésbé lép fel az eltűnő gradiensek problémája.
$Tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, amely -1 és 1 közé normálja a kimenetet, emiatt alkalmazzák néha a kimenet előtt.

A.3. Klasszikus augmentációk

Kétféle klasszikus augmentációt mutatunk, az affin elforgatás és a five crop.
Az affin elforgatáshoz tartozik egy d fokszám, és a kimenetet véletlenszerűen $\pm d$ fokkal elforgatja.
A five crop augmentációhoz tartozik egy (H, W) magasság és szélesség vektor. A beadott kép négy sarkából és a közepéből kivág (H, W) nagyságú képeket, így egy példa képből 5-öt csinál. Itt természetesen elengedhetetlen, hogy a beadott kép mérete a kivágandó méreteknél nagyobb legyen.

name	**epoch**	**adam lr**	**batch**	**optim.**	**degree**	**w. decay**
resnet_0.2_gan_F	30	1.20E-04	32	adam	4	0.1
resnet_0.2_gan_T	30	1.00E-04	16	adam	4	0.1
resnet_0.2_overs_F	30	9.00E-05	32	adam	4	0.1
resnet_0.2_overs_T	30	1.00E-04	32	adam	4	0.1
resnet_0.2_None_F	30	2.40E-04	32	adam	4	
resnet_0.2_None_T	30	9.00E-05	32	adam	4	
resnet_0.4_gan_F	30	9.00E-05	32	adam	4	0.1
resnet_0.4_gan_T	30	9.00E-05	16	adam	4	0.1
resnet_0.4_overs_F	30	9.00E-05	32	adam	4	
resnet_0.4_overs_T	30	9.00E-05	32	adam	4	0.2
resnet_0.4_None_F	30	1.20E-04	32	adam	4	
resnet_0.4_None_T	30	9.00E-05	32	adam	4	0.1
resnet_0.6_gan_F	30	1.80E-04	32	adam	4	0.1
resnet_0.6_gan_T	30	9.00E-05	32	adam	4	0.1
resnet_0.6_overs_F	30	9.00E-05	32	adam	4	0.1
resnet_0.6_overs_T	30	9.00E-05	16	adam	4	0.2
resnet_0.6_None_F	30	2.40E-04	32	adam	4	
resnet_0.6_None_T	30	1.20E-04	32	adam	4	0.1
resnet_0.8_gan_F	30	9.00E-05	32	adam	4	0.1
resnet_0.8_gan_T	30	6.00E-05	16	adam	4	0.1
resnet_0.8_overs_F	30	1.20E-04	16	adamW	3	0.1
resnet_0.8_overs_T	30	9.00E-05	16	adam	3	0.1
resnet_0.8_None_F	30	9.00E-05	32	adam	4	
resnet_0.8_None_T	30	9.00E-05	16	adam	3	0.1
resnet_1_None_F	30	1.60E-04	16	adamW	3	0.1
resnet_1_None_T	30	1.60E-04	16	adamW	3	0.1

A.1. táblázat. A ResNet18 típusú hálókon használt hiperparaméterek.
A név összetétele $<$ *háló* $>$ _ $<$ *arány* $>$ _ $<$ *kép pótlás* $>$ _ $<$ affin *augmentáció* $>$.
Amikor az optimalizáló agloritmus *AdamW*, abban az esetben *adam weight decay* $= 0.1$
rövidítések: w.decay = weight decay, overs. = oversampling, F = False, T = True.

name	**epochs**	**adam lr**	**batch**	**optim.**	**degree**	**w. decay**
vgg_0.2_gan_F	30	1.00E-04	32	adam	3	0.1
vgg_0.2_gan_T	30	9.00E-05	16	adam	3	0.1
vgg_0.2_overs._F	30	9.00E-05	16	adam	3	0.1
vgg_0.2_overs._T	30	1.00E-04	32	adam	3	0.1
vgg_0.2_None_F	30	9.00E-05	32	adam	3	0.1
vgg_0.2_None_T	30	9.00E-05	32	adam	3	0.1
vgg_0.4_gan_F	30	9.00E-05	32	adam	3	0.1
vgg_0.4_gan_T	30	1.20E-04	32	adam	3	0.1
vgg_0.4_overs._F	30	9.00E-05	32	adam	3	0.1
vgg_0.4_overs._T	30	1.20E-04	32	adam	3	0.1
vgg_0.4_None_F	30	9.00E-05	32	adam	3	0.1
vgg_0.4_None_T	30	9.00E-05	16	adam	3	0.1
vgg_0.6_gan_F	30	1.20E-04	32	adam	3	0.1
vgg_0.6_gan_T	30	9.00E-05	64	adam	3	0.1
vgg_0.6_overs._F	30	9.00E-05	32	adam	3	0.1
vgg_0.6_overs._T	30	9.00E-05	32	adam	3	0.1
vgg_0.6_None_F	30	9.00E-05	32	adam	3	0.1
vgg_0.6_None_T	30	9.00E-05	32	adam	3	0.1
vgg_0.8_gan_F	30	9.00E-05	16	adam	3	0.1
vgg_0.8_gan_T	30	9.00E-05	16	adam	3	0.1
vgg_0.8_overs._F	30	1.20E-04	32	adam	3	0.1
vgg_0.8_overs._T	30	9.00E-05	32	adam	3	0.1
vgg_0.8_None_F	30	9.00E-05	32	adam	3	0.1
vgg_0.8_None_T	30	9.00E-05	16	adam	3	0.1
vgg_1_None_F	30	3.00E-05	32	adam	3	0.1
vgg_1_None_T	30	1.20E-04	16	adam	3	0.2

A.2. táblázat. A VGG16 típusú hálókon használt hiperparaméterek.
A név összetétele: $<$ *háló* $>$ _ $<$ *arány* $>$ _ $<$ *kép pótlás* $>$ _ $<$ affin *augmentáció* $>$.
Amikor az optimalizáló algoritmus *AdamW*, abban az esetben *adam weight decay* $= 0.1$
rövidítések:w.decay = weight decay, overs. = oversampling, F = False, T = True.

name	**epochs**	**adam lr**	**batch**	**optim.**	**degree**	**w. decay**
efficient_0.2_gan_F	30	1.20E-04	16	adamW	3	0.05
efficient_0.2_gan_T	30	1.20E-04	16	adam	3	0.05
efficient_0.2_overs._F	30	1.20E-04	16	adamW	3	0.05
efficient_0.2_overs._T	30	1.60E-04	16	adam	3	0.01
efficient_0.2_None_F	30	9.00E-05	16	adam	3	0.1
efficient_0.2_None_T	30	9.00E-05	16	adam	3	0.1
efficient_0.4_gan_F	30	1.60E-04	16	adam	3	0.05
efficient_0.4_gan_T	30	1.20E-04	16	adamW	3	0.05
efficient_0.4_overs._F	30	1.20E-04	16	adamW	3	0.1
efficient_0.4_overs._T	30	1.20E-04	16	adamW	3	0.05
efficient_0.4_None_F	30	9.00E-05	16	adam	3	0.1
efficient_0.4_None_T	30	9.00E-05	16	adam	3	0.1
efficient_0.6_gan_F	30	1.20E-04	16	adamW	3	0.05
efficient_0.6_gan_T	30	1.60E-04	16	adam	3	0.05
efficient_0.6_overs._F	30	1.20E-04	16	adam	3	0.1
efficient_0.6_overs._T	30	1.20E-04	16	adam	3	0.05
efficient_0.6_None_F	30	9.00E-05	16	adam	3	0.1
efficient_0.6_None_T	30	9.00E-05	16	adam	3	0.1
efficient_0.8_gan_F	30	9.00E-05	16	adamW	3	0.05
efficient_0.8_gan_T	30	1.60E-04	16	adam	3	0.05
efficient_0.8_overs._F	30	9.00E-05	16	adam	3	0.1
efficient_0.8_overs._T	30	1.20E-04	16	adam	3	0.01
efficient_0.8_None_F	30	9.00E-05	16	adamW	3	
efficient_0.8_None_T	30	9.00E-05	16	adam	3	0.1
efficient_1_None_F	30	9.00E-05	16	adam	3	0.1
efficient_1_None_T	30	9.00E-05	32	adam	3	0.1

A.3. táblázat. Az $EfficientNet\_b0$ típusú hálókon használt hiperparaméterek.
A név összetétele: < *háló* > _ < *arány* > _ < *kép pótlás* > _ < affin *augmentáció* >.
Amikor az optimalizáló algoritmus *AdamW*, abban az esetben *adam weight decay* = 0.1
rövidítések:w.decay = weight decay, overs. = oversampling, F = False, T = True.

NYILATKOZAT

Név: Borbély Bernárd

ELTE Természettudományi Kar, szak: Matematika

NEPTUN azonosító: J203EN

Szakdolgozat címe:
Orvosi képek elemzése gépi tanulási módszerekkel

A **szakdolgozat** szerzőjeként fegyelmi felelősségem tudatában kijelentem, hogy a dolgozatom önálló szellemi alkotásom, abban a hivatkozások és idézések standard szabályait következetesen alkalmaztam, mások által írt részeket a megfelelő idézés nélkül nem használtam fel.

Budapest, 2023.05.31

B. Bernárd

a hallgató aláírása