

”MI” a művész

Informatika és szoftverfejlesztés

XXIII. Kari Tudományos Diákköri Konferencia

Marosvásárhelyi Kar

Sapientia Erdélyi Magyar Tudományegyetem

Hallgató

Babos Dávid

Számítástechnika III.

Témavezető

Drd. Csaholczi Szabolcs

Matematika-Informatika Tanszék

2024

Kivonat

A kutatásom lényege, hogy egy olyan telefonos applikációt készítssek, amely a mai ember számára sokkal megfoghatóbbá teszi a művészet vizuális ágát. A manapság népszerű Mesterséges Intelligencia lehetőséget nyújt az emberek számára, hogy egyedi/páratlan képeket alkothassanak. Viszont sok olyan személy van aki egy meglévő fotóját szeretné egyszerűen feldobni. Erre egy jó megoldást nyújthat a Photoshop használata, viszont ezen eszköz megtanulása, hosszú folyamattal és türelemmel jár.

Ebből kifolyólag született meg az ötlet egy olyan telefonos alkalmazásra, ahol könnyedén lehet már meglévő képekre artistikus stílust ruházni. Erre a kihívásra megoldást nyújt a Neural Style Transfer [1] módszer, ami két képet képes összevonni oly módon, hogy az első tartalmát valamint a második stílusát őrizze meg. Így válik lehetővé olyan tájképek alkotása amelyeket, mintha maga Van Gogh festett volna.

Kulcsszavak: Művészet, Mesterséges Intelligencia, Generative AI, Konvolúciós Neurális Hálózat, Képszintézis, Android alkalmazás, Közösségi média

Tartalomjegyzék

1. Bevezető	4
2. Elméleti áttekintés	5
3. Projekt célja	7
4. Használt Technológiák	10
5. Neural Style Transfer	11
5.1. Mély képreprezentációk	13
5.1.1. Tartalom reprezentáció	13
5.1.2. Stílus reprezentáció	15
5.1.3. Style Transfer	16
5.2. Eredmények	19
5.2.1. Kompromisszum a tartalom és a stílus megfeleltetése között	19
5.2.2. A kép inicializálása	20
5.2.3. Teljes variáció veszteség	21
6. Alkalmazás megvalósítása	22
6.1. Szerver oldal	22
6.1.1. Weights and Biases	23
6.1.2. Endpoint-ok	25
6.2. Adatbázis	26
6.3. Android alkalmazás	27
6.3.1. Bejelentkezés és Regisztráció	27
6.3.2. Főképernyő	29
6.3.3. Stílus hozzáadása	30
6.3.4. Kép generálása	31
6.3.5. Generált képek megtekintése	32
7. Összefoglaló	33
8. Továbbfejlesztési lehetőségek	34
8.1. Közösségi média	34
8.2. Üzleti logika	34
8.3. Neural Style Transfer	35
9. Hivatkozások	37

1. Bevezető

Már az idők kezdete óta az embereket körbeveszi a művészet fogalma. A művészet képi formában jelen van és volt a mindennapokban: rajzok, festmények, fényképek, stb. Viszont a 2020-as években nagy népszerűségnek örvendő mesterséges intelligencia a művészet megfoghatóságát egy teljesen új szintre emelte és olyan személyek számára is elérhetővé tette, akiknek nehezükre esik ecsetet ragadni.

A mesterséges intelligenciának sikerült nagyon sok féle szempontból megmozgatni az emberek kreativitását vagy egyes esetekben hozzájárult munkájuk megkönnyebbítésében. Elsődleges sorban rá lehet tekinteni a képfelismerésre, amely megalapozta az önvezető autókat, rákos sejtek hatékonyabb felismerését, vagy emberi arcok osztályba sorolását. Másodlagos sorban megjelentek a chatbot-ok amelyek megkönnyíthetik az esszék írását, vagy mesék/forgatókönyvek alkotására képesek. Erre egy nagyon jó példa lehet a ChatGPT [2], amelyet jelen pillanatban 180 millió személy használ fel. Következőekben nagy sikernek örvendhetnek a képalkotó MI-k, mint például a DALL-E [3], amelyek egy szöveg alapú promptból képesek teljesen egyedi képeket alkotni.

A generatív mesterséges intelligencia nem csak szövegből képes képek alkotására, hanem van lehetősége meglévő képek módosítására, például azokat artistikussá vagy fotórealisztikussá alakítva. Ezen módszerek egyike a Style Transfer, ami visszanyúlik az 1980-as évekig, ahol jelfeldolgozással és filterek használatával voltak képesek rajzfilmesíteni képeket. Viszont manapság már megjelent egy ennél sokkal fejlettebb módszer, a Neural Style Transfer, amely mesterséges neurális hálózatok felhasználásával képes a fent említett kihívások megoldására.

2. Elméleti áttekintés

Megfigyelhetjük, hogy manapság több módszer is létezik képek generálására vagy akár módosítására. Az MI megjelenése előtt a képfeldolgozási technikákat alkalmazták különböző filterek alkalmazására a képeken. Viszont a kezdeti mesterséges intelligencia új fokra tudta emelni ezt a mesterséges neuronális hálózatok alkalmazásával. Kezdetekben VAE [4] (Variational autoencoder) tette lehetővé képek generálását hatalmas adathalmazok betanítása után, viszont ezt hamar lekörözték a GAN [5] (Generative adversarial network) architektúrájú neuron hálók. Manapság pedig nagy sikernek örvendhetnek a Diffúziós modellek [6], amelyeknek áttörő sikere legyőzte a generatív versengő hálózatokat (GAN).

Az imént említett hálózatoknak viszont van egy nagy előnyük és egyben hátrányuk is, ami nem más mint, a képen látható kontextust promptok formájában kell megfogalmazni. Így igen érzékeny lehet a beadott szövegre, amennyiben nincs a megfelelő módon körbeírva, a kontextus, nem azt a képet fogja generálni ami a felhasználó fejében megszületett. Ilyen típusú alkalmazásokra jó példa lehet a DALL-E 2 [3], amely remek módon képes új képek létrehozásában, viszont kevésbé mondható sikeresnek abban, hogy egy meglévő képet művésziesebbé tegyen. Ez mind jellemző a diffúziós modellek többségére.

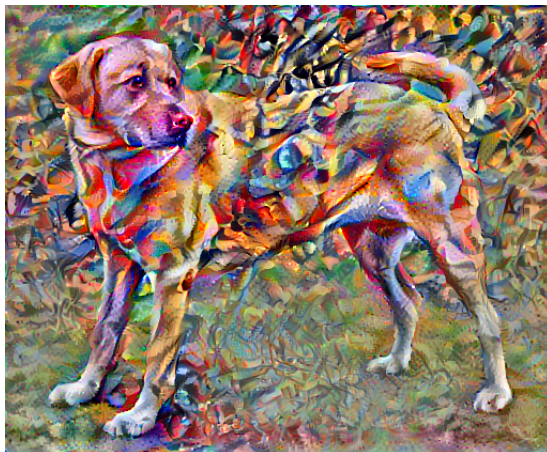
Annak ellenére, hogy nagy sikere van a generatív modelleknek és képesek lenyűgöző képeket alkotni a semmiből, nagyon sok felhasználó egy meglévő képet szeretne módosítani. Természetesen erre alkalmas lehet a telefonokba beépített képszerkesztő opció, vagy akár manapság használt közösségi felületeken elérhető filterezés. De ezek inkább csak egy meglévő képet szerkesztenek, mintsem valami kitörő újdonságot ültessenek bele.

Erre megoldást nyújthat a képek szintézise, amely a Mesterséges Intelligencia segítségével igen csak látványosan tudott fejlődni. Erre egy jó példa lehet a DeepDream [7], amely lehetővé tette olyan képek alkotását, melyek eufórikus hatást keltenek. Ezek az általában ismerős objektumokat, például állatokat vagy tárgyakat ábrázoló képek lehetnek, de torzított, elmosódott vagy megnagyobbított részleteket is tartalmaznak.



1. ábra. DeepDream segítségével megalkotott kép, amely egy Labradort ábrázol ”Inception” stílusban [8]

A képek artisztikusságának tételében a nagy áttörést a Neural Style Transfer nyújtotta. Ami azon alapszik, hogy bármely képre rá lehet öltetni egy másik kép stílusát. Ez lehetővé teszi egy kép átformálását úgy, hogy megőrzi annak általános tartalmát, de stílusa például egy híres festőművész stílusára emlékeztet. Ez a módszer igen nagy szabadsággal rendelkezik elvégre nem csak egy teli tartalommal rendelkező képet lehet megadni bemenetként, hanem a stílussal rendelkező képet is, ami lehet egy impresszionista festő alkotása, vagy egy éjszakai kép egy nagyvárosról, vagy akár egy olyan alkotás amit a felhasználó korábban hozott létre. Így a kreativitást egészen a csillagos égig lehet elvinni.



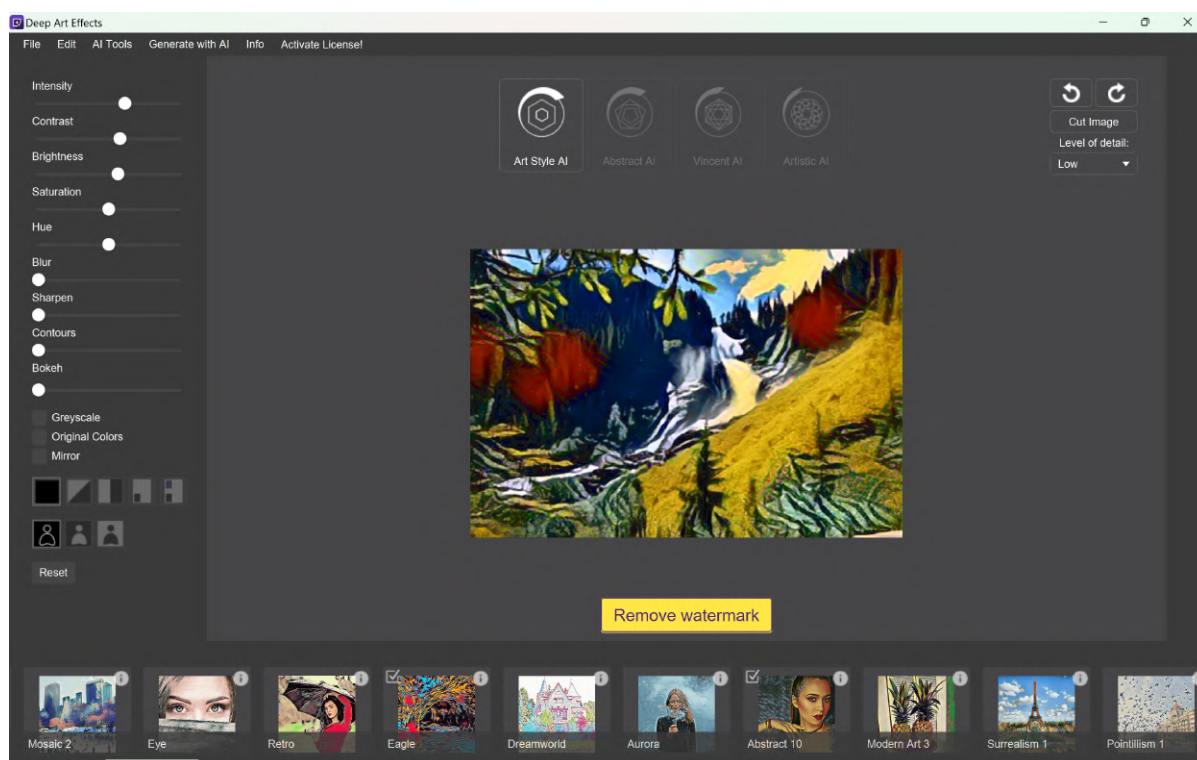
2. ábra. A Neural Style Transfer lehetővé teszi, hogy egy kutyát olyan formában tüntessünk fel, mintha azt Kandinsky jómaga festette volna [1]

3. Projekt célja

A projekt azon az ötleten alapszik, hogy egy olyan alkalmazás váljon elérhetővé a mindennapi személyek számára, amelyekkel artisztikus képeket tudnak alkotni, és ezeket megosztani egymás között. Így a művészet és kreativitás hatékony módon fog tudni terjeszkedni a felhasználók között. Ebből kifolyólag egy olyan alkalmazást hoztam létre amiben a fent említett Neural Style Transfer algoritmus felhasználásával képeket alkothasson a felhasználó és későbbiekben ezeket meg tudja osztani mások között.

Ehhez hasonló alkalmazások már léteznek:

- **Deep Art Effects** [9]: egy asztali/mobil applikáció, ahol NST-vel, lehet képeket stilizálni. Többnyire egy képszerkesztő alkalmazásra hasonlít, ahol lehet állítani a beadott kép, kontrasztját, fényerősségét, homályosítani és mellette még egy kiválasztott stílust ráruházni.



3. ábra. Egy valós tájképre Eagle effekt alkalmazása, amelytől úgy néz ki mintha festve lenne

- **Prisma** [10]: egy telefonos applikáció aminek alap kontextusa, hogy artistikus effekteket alkalmazzon a képekre. Ezek mellett még csomó más képszerkesztési opció is van, többek között: élesítés, szaturáció, kontraszt, árnyékolás. A megalkotott képet pedig a későbbiekben le lehet tölteni.



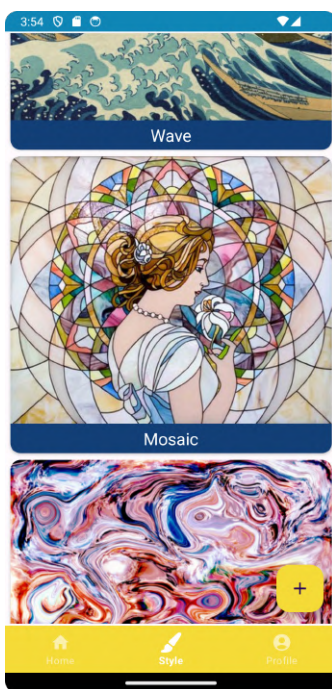
4. ábra. A Prisma segítségével Brad Pitt-ről való képet lehet különböző stílusokban ábrázolni

Az imént említett alkalmazások remek módon használják fel a Neural Style Transfer módszert és artistikus képek megalkotását teszik lehetővé a felhasználó számára, viszont egy nagy hátránnyal lehet szembesülni: a stílusok korlátozva vannak a fejlesztők által. Tehát, ha a felhasználó nem találja meg az általa kedvelt művész stílusát, akkor nincs lehetősége ahhoz hasonló képet alkotni. A stílusok bővítésének lehetősége teljes mértékben csak a fejlesztők kezében van, ami egy ilyen kreatív alkalmazás esetén egészen nagy korlátozás lehet.

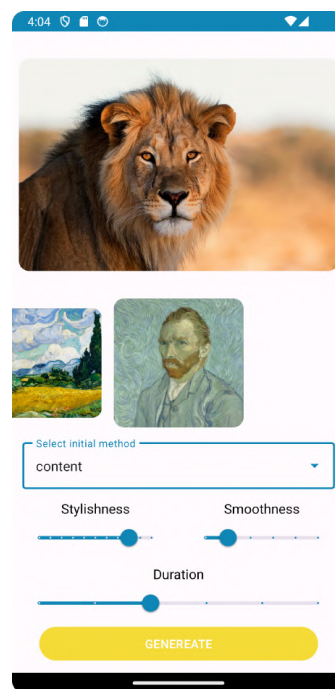
A projekttemmel erre a problémára szeretnék megoldást találni azáltal, hogy az alapértelmezetten megadott stílusok mellett a felhasználó bármikor hozzáadhassa az általa kedvelt festő alkotásait és későbbiekben azzal ruházza fel a képeit. Emellett pedig a létrehozott képeket a felhasználó meg tudja osztani a főoldalon és más artisztikus képeket és azon paramétereit meg tudja tekinteni. Amennyiben talál a főoldalon egy új stílust, hozzá is tudja adni az ő stílus listájához. Így az emberek között tud gyarapodni a művészet és képesek egymást motiválni az alkotásokban.



(a) Létrehozott képek



(b) Stílusok



(c) Paraméterek kiválasztása

5. ábra. Alkalmazás áttekintés

4. Használt Technológiák

Az alkalmazás megvalósítása érdekében **Android Studiót** [11] használok, az Iguana 2023.2.1 Patch 2 verzióját. A kinézetet az **XML**-ben [12] van megtervezve és a mögöttes futó logika a **Kotlin** [13] programozási nyelvben lett megalkotva.

Adatbázisként a **Firestore**-t [14] alkalmazom, amit igen egyszerű módon össze lehet kapcsolni Android Stúdióban és könnyed módon lehetővé teszi az adatbázisba való írás/olvasást, emellett a felhasználók hitelesítését és jelszavak kódolását is könnyedséggel oldja meg.

A képek generálása érdekében **Python 3.10.11** [15] programozási nyelvet alkalmaztam a következő függőségekkel:

- **NumPy** [16]: egy Python-könyvtár, amelyet tömbökkel való munkavégzésre használnak. Funkciói is vannak a lineáris algebra, a Fourier-transzformáció és a mátrixok területén történő munkához.
- **Matplotlib** [17]: egy átfogó könyvtár statikus, animált és interaktív vizualizációk létrehozásához Pythonban.
- **PyTorch** [18]: egy Python csomag, amely két magas szintű szolgáltatást biztosít: tenzorszámítás (mint a NumPy [16]) erős GPU-gyorsítással; mély neurális hálózatok, amelyek szalagos autograd rendszerre épülnek.
- **OpenCV** [19]: (*Open Source Computer Vision Library*) olyan programozási funkciók könyvtára, amelyek főként valós idejű számítógépes látást biztosítanak.
- **FastAPI** [20]: egy modern webes keretrendszer, amelyet először 2018-ban adtak ki RESTful API-ok Pythonban való építésére. A Python 3.8+ rendszerű API-ok építésére használják szabványos Python-típusú tippek alapján.
- **WandB** [21]: A *Weights & Biases* egy gépi tanulási fejlesztőplatform, amely lehetővé teszi a felhasználók számára, hogy valós időben nyomon kövessék és megjelenítsék modellképzési folyamatuk különböző aspektusait.

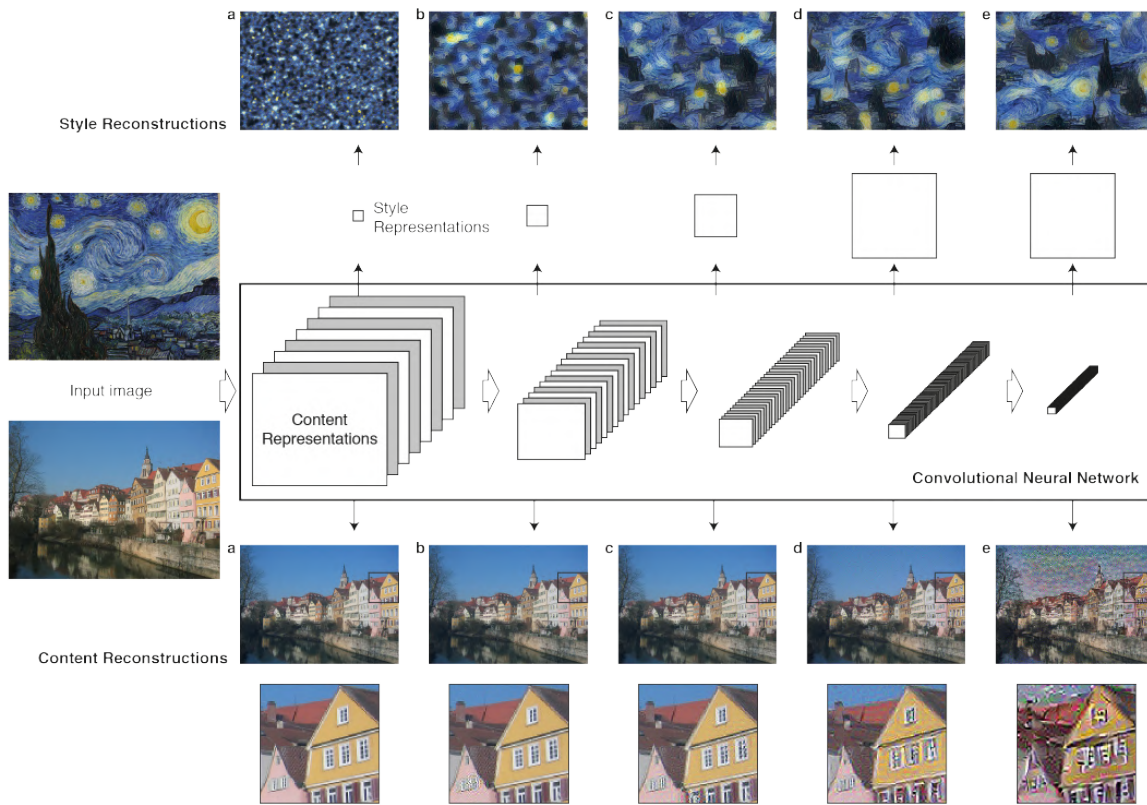
5. Neural Style Transfer

Egy kép szemantikus tartalmának megjelenítése különböző stílusokban egy nehéz képfeldolgozási feladat. A régebbi módszerek fő korlátozása az volt, hogy nem voltak olyan képreprezentációk, amelyek világosan kifejezték volna a kép szemantikus információját, így nem lehetett hatékonyan elválasztani a képtartalmat és a stílust egymástól. Ebben az esetben olyan képreprezentációk vannak használatba véve, amelyek Konvolúciós Neurális Hálózatokból származnak, ezek objektumfelismeréshez lettek optimalizálva, és explicit módon szemantikus képinformációt tartalmaznak. A *”A Neural Algorithm of Artistic Style”* [22] módszerből kiindulva vált lehetővé, hogy természetes képek tartalmát és stílusát tudjuk szétválasztani és újra kombinálni. Az algoritmus lehetővé teszi új, magas észlelési minőségű képek előállítását, amelyek összekapcsolják egy tetszőleges fénykép tartalmát és számos ismert műalkotás megjelenítését. Eredményül pedig betekintést nyújtanak a Konvolúciós Neurális Hálózatok által megtanult mély képreprezentációkba, és bemutatják potenciáljukat a magas szintű képszintézis és manipuláció terén.

Egy kép stílusának átvitele egy másik képre tekinthető egy textúraátviteli problémának. A textúraátvitel célja, hogy szintetizáljon egy textúrát egy forrásképből, miközben korlátozza a textúraszintézist annak érdekében, hogy megőrizze a célkép szemantikus tartalmát. Ideális esetben egy stílusátviteli algoritmusnak képesnek kell lennie arra, hogy kinyerje a célkép szemantikus képtartalmát (például az objektumokat és a közös tájat) és informálja a textúraátviteli eljárást a célkép szemantikus tartalmának megjelenítésére a forráskép stílusában.

A Mély Konvolúciós Neurális Hálózatok fejleményei olyan erős számítógépes látási rendszereket hoztak létre, amelyek megtanulják kinyerni a magas szintű szemantikus információkat a természetes képekből. Kimutatták, hogy a Konvolúciós Neurális Hálózatok, amelyeket elegendő címkézett adatokkal finomhangolnak, különféle látási feladatokban jobban teljesítenek, mint az előző kézzel tervezett jellemzők. Emellett a Konvolúciós Neurális Hálózatok lehetővé teszik, hogy explicit módon befolyásolva legyenek az általuk megtanult reprezentációkra azáltal, hogy újrainicializálva a hálózatot különböző osztályozási címkékkel stílusosító célokra finomhangoljuk.

Eme módszert alkalmazva elérhetővé válik, hogy kinyerjük a szemantikus képtartalmat a Konvolúciós Neurális Hálózatokból, és aztán alkalmazzuk azt egy új eljáráshoz, amely szétválasztja és újra kombinálja a képtartalmat és a stílust. A módszer egy neurális algoritmuson alapul, amely képes szétválasztani és újra kombinálni a természetes képek tartalmát és stílusát, ami új, magas észlelési minőségű képek előállítását teszi lehetővé, amelyek összekapcsolják egy tetszőleges fénykép tartalmát és számos ismert műalkotás megjelenését.



6. ábra. **Konvolúciós Neurális Hálózat (CNN).** Egy adott bemeneti képet a CNN minden feldolgozási szakaszában szűrt képek halmazaként ábrázolunk. Miközben a különböző szűrők száma növekszik a feldolgozási hierarchia mentén, a szűrt képek mérete valamilyen mintavételezési mechanizmus által csökken (pl. max-pooling), ami a hálózat rétegenkénti összes egységének számának csökkenéséhez vezet.

Tartalomrekonstrukciók. Különböző feldolgozási szakaszokban a CNN-ben található információt megjeleníthetjük a bemeneti kép rekonstrukciójával, anélkül hogy csak egy adott rétegben ismernénk a hálózat válaszait. A bemeneti képet rekonstruáljuk a conv1_1' (a), conv2_1' (b), conv3_1' (c), conv4_1' (d) és conv5_1' (e) rétegekből az eredeti VGG-hálózatból. Az alsó rétegekből történő rekonstrukció majdnem tökéletes (a,b,c). A hálózat felső rétegeiben a részletes képinformáció elveszik, miközben megőrződik a kép magas szintű tartalma (d,e). **Stílusrekonstrukciók.** Az eredeti CNN-reprezentációkra épült egy új jellemzőtér, amely rögzíti egy bemeneti kép stílusát. A stílus reprezentáció korrelációkat számol ki a CNN különböző rétegeiben található különböző jellemzők között. A bemeneti kép stílusát különböző CNN rétegek részalmazain alapuló stílus reprezentációkból rekonstruáljuk (conv1_1' (a), conv1_1' és conv2_1' (b), conv1_1', conv2_1' és conv3_1' (c), conv1_1', conv2_1', conv3_1' és conv4_1' (d), conv1_1', conv2_1', conv3_1', conv4_1' és conv5_1' (e)). Ez olyan képeket hoz létre, amelyek megfelelnek egy adott kép stílusának egy növekvő skálán, miközben elhagyja a jelenet globális elrendezésének információját.

5.1. Mély képreprezentációk

Az imént bemutatott eredmények a VGG hálózaton alapulnak, amelyet objektumfelismerésre és lokalizálásra képeztek ki [23], ez részletes leírásra került az eredeti munkában [24]. A hálózat az emberi teljesítménnyel vetekszik egy általános vizuális objektumfelismerési teszten. A VGG hálózat 19 rétegből áll, amiből 16 konvolúciós és 5 poolozási réteg által nyújtott jellemzőtér (*feature space*) lett felhasználva. A teljesen összecsatolt rétegek elhanyagolásra kerültek elvégre nem osztályozási problémával kellett szembeállni. A modell bárki számára nyilvánosan elérhető és felfedezhető a caffe-frameworkben [25]. A kép szintézis érdekében a maximális poolozási művelet átlag poolozásra lett lecserélve, amely kissé vonzóbb eredményeket nyújtott.

5.1.1. Tartalom reprezentáció

Általában mindegyik rétege a hálózatnak egy nemlineáris szűrőbankot határoz meg, amelynek a komplexitása növekszik a hálózatban való mélységének függvényében. Tehát egy adott bemeneti kép $\vec{x}$ kódolva van minden rétegben a CNN szűrőinek válaszaival erre a képre. Egy N_l különböző szűrőt tartalmazó réteg N_l jellemző térképet tartalmaz, mindegyik mérete M_l , ahol M_l a jellemző térkép magasságának és szélességének szorzata. Tehát egy l réteg válaszai tárolhatók egy $F^l \in \mathcal{R}^{N_l \times M_l}$ mátrixban, ahol F_{ij}^l a j -edik pozícióban lévő i -edik szűrő aktivációja a l rétegben.

Azoknak az információknak a megjelenítéséhez, amelyek különböző rétegekben vannak kódolva a hierarchiában (lásd a 6. ábrát, tartalom rekonstrukciók), gradiens visszafejtést végzünk egy fehér zaj képen annak érdekében, hogy megtaláljunk egy másik képet, amely megfelel az eredeti kép jellemzői válaszáinak. Legyen $\vec{p}$ és $\vec{x}$ az eredeti kép és a generált kép, valamint P^l és F^l azok megfelelő jellemző reprezentációi az l rétegben. Ezután definiáljuk a négyzetes hibát a két jellemző reprezentáció között.

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} (F_{ij}^l - P_{ij}^l)^2 . \quad (1)$$

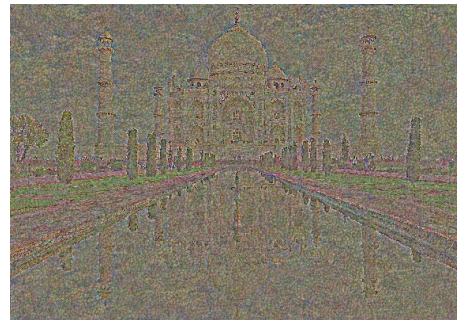
A veszteség deriváltja az l réteg aktivációi szerint megegyezik

$$\frac{\partial \mathcal{L}_{content}}{\partial F_{ij}^l} = \begin{cases} (F^l - P^l)_{ij} & \text{if } F_{ij}^l > 0 \\ 0 & \text{if } F_{ij}^l < 0 . \end{cases} \quad (2)$$

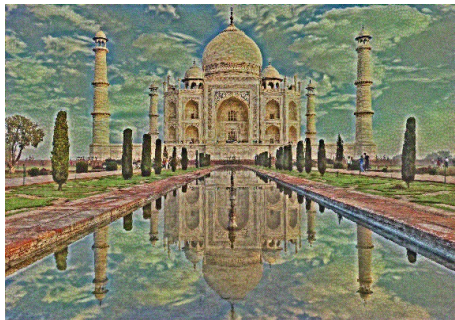
ezáltal a kép $\vec{x}$ -hoz tartozó gradienst kiszámíthatjuk a szokásos hiba visszatérjesztés (back-propagation) módszerrel (2). Így tehát változtathatjuk az eredetileg véletlenszerű képet $\vec{x}$ -t, amíg ugyanazt a választ nem kapjuk egy bizonyos rétegben a Konvolúciós Neurális Hálóban, mint az eredeti kép $\vec{p}$ esetében.



(a) Eredeti kép



(b) 5 iteráció



(c) 25 iteráció



(d) 150 iteráció

7. ábra. A VGG19-es model konvolúciós rétegei felhasználásával megpróbáljuk rekonstruálni az eredeti képet (Taj Mahal). Megfigyelhető, hogy 5 iteráció után még teljes mértékben zajjal teli képet nyert vissza, 25 iteráció után már kivehető a képből a Taj Mahal formája, és 150 iterációt meghaladván szinte teljesen rekonstruálta az eredeti képet.

5.1.2. Stílus reprezentáció

A bemeneti kép *stílusának* reprezentációjához egy olyan jellemzőteret használunk, amelyet a textúrainformációk megragadására terveztek [26]. Ez a jellemzőtér felépíthető a háló bármely rétegében található szűrőválaszokra. Ez a különböző szűrőválaszok közötti korrelációkra épül, ahol az elvárt érték a jellemzőtér térbeli kiterjedésére vonatkozik. Ezeket a jellemző korrelációkat a Gram-mátrix $G^l \in \mathcal{R}^{N_l \times N_l}$ adja meg, ahol G_{ij}^l az l rétegben az i és j vektorizált jellemzőtérképek közötti belső szorzat.

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l. \quad (3)$$

Több réteg jellemző korrelációinak bevonásával állíthatunk elő egy állandó, több skálás reprezentációt a bemeneti képről, amely megragadja annak textúráját, de nem a globális elrendezést. Ismét megtekinthetjük ezeknek a stílusjellemző tereknek az általuk rögzített információját, amelyeket a háló különböző rétegein felépítettünk egy olyan kép létrehozásával, amely megegyezik egy adott bemeneti kép stílus reprezentációjával. Ehhez egy fehér zaj képből kiindulva gradiens csökkentéssel minimalizáljuk az eredeti kép Gram-mátrixainak bejegyzéseinek négyzetes távolságát az előállítandó kép Gram-mátrixainak bejegyzéseitől [26].

Legyen $\vec{a}$ és $\vec{x}$ az eredeti kép és a generált kép, valamint A^l és G^l a megfelelő stílusreprezentációjuk az adott l rétegben. A l réteg hozzájárulásával a veszteség ekkor

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - A_{ij}^l)^2 \quad (4)$$

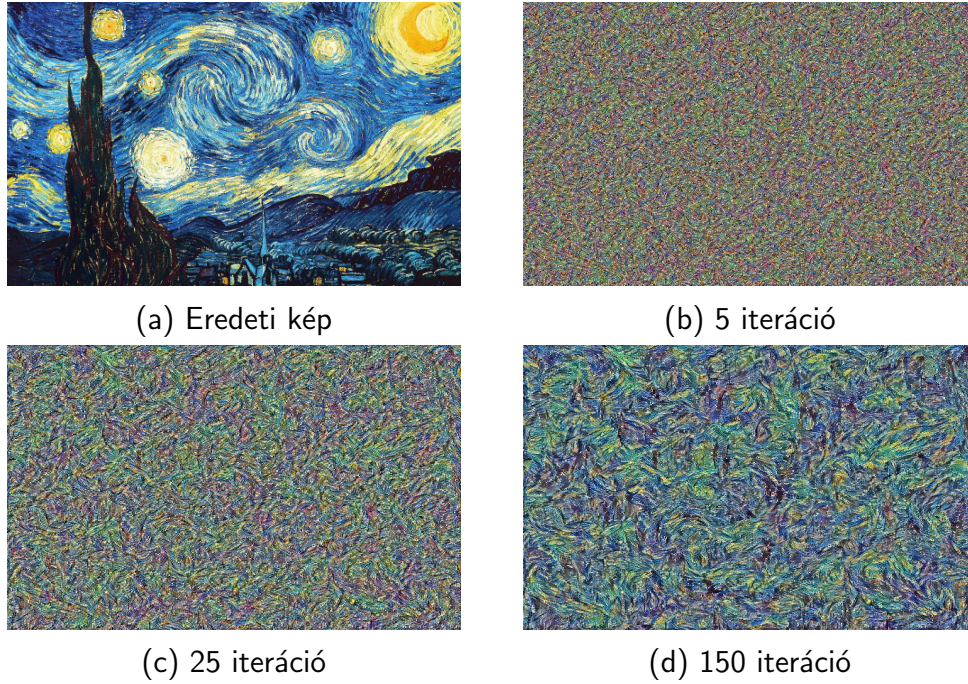
és a teljes veszteség

$$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^L w_l E_l \quad (5)$$

ahol w_l a súlyozási tényezők, amelyek meghatározzák minden réteg hozzájárulását a teljes veszteséghez. Az E_l veszteségnek az l réteg aktivációira vonatkozó deriváltja analitikusan kiszámítható

$$\frac{\partial E_l}{\partial F_{ij}^l} = \begin{cases} \frac{1}{N_l^2 M_l^2} ((F^l)^T (G^l - A^l))_{ji} & \text{if } F_{ij}^l > 0 \\ 0 & \text{if } F_{ij}^l < 0. \end{cases} \quad (6)$$

Az E_l veszteség gradiensei a pixelértékekkel $\vec{x}$ kapcsolatban könnyen kiszámíthatók a standard hibavisszaterjesztés segítségével.



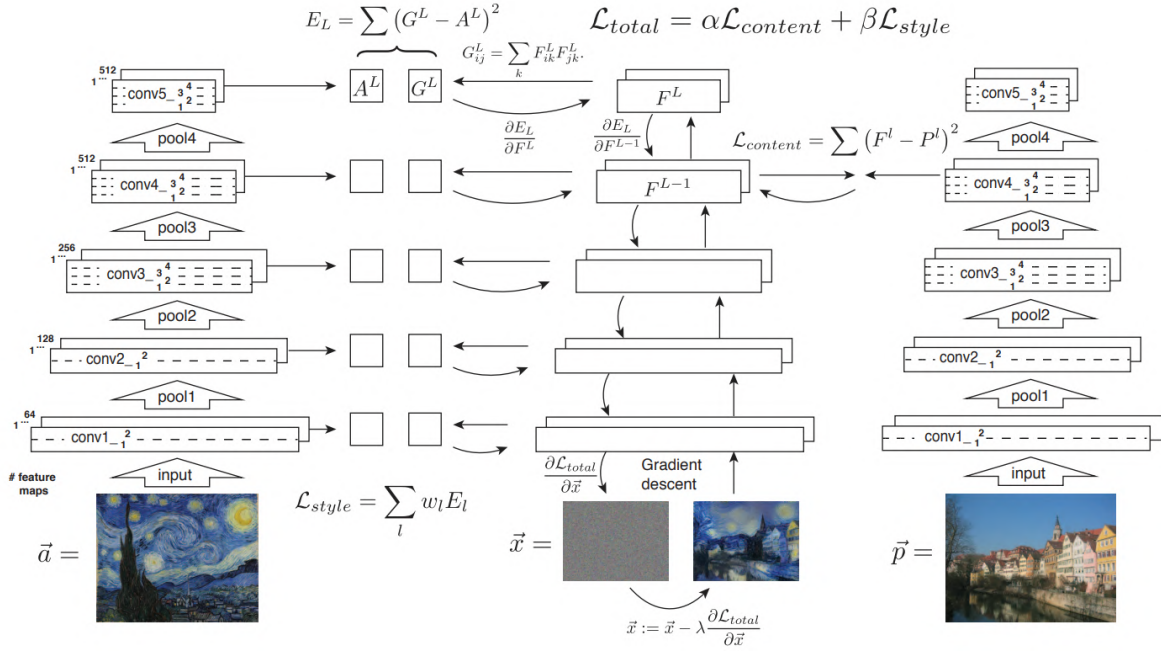
8. ábra. A VGG19-es model konvolúciós rétegei felhasználásával megpróbáljuk kinyerni a kép stílusát (Starry Night - Van Gogh). Megfigyelhető, hogy amíg kezdetben csak egy zajjal teli képet nyer vissza, addig az iterációk növekedésével tisztán kivehető az eredeti kép textúrája.

5.1.3. Style Transfer

Ahhoz, hogy egy műalkotás stílusát $\vec{a}$ egy fotóra $\vec{p}$ átültessük, egy új képet szintetizálunk, amely egyszerre illeszkedik a fotó tartalom reprezentációjához és a festmény stílus reprezentációjához (9). Így egyidejűleg minimalizáljuk egy fehér zajú kép jellemző reprezentációjának távolságát a fotó tartalom reprezentációjától egy rétegben, valamint a festmény stílus reprezentációját egy adott számú rétegen át a Konvolúciós Neurális Hálózatban. A minimalizálandó veszteségfüggvényünk:

$$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{style}(\vec{a}, \vec{x}) \quad (7)$$

ahol α és β a tartalom és stílus rekonstrukció súlyozási tényezői. A pixelértékekkel $\frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$ kapott gradienst bemenetként használhatjuk néhány numerikus optimalizációs stratégiához. Ahhoz, hogy összehasonlítható méretű képinformációkat nyerjünk ki, mindig azonos méretűvé alakítottuk a stílus képet a tartalom képhez, mielőtt a jellemző reprezentációkat számítottuk volna.



9. ábra. A stílusátviteli algoritmus során először kinyerjük és tároljuk a tartalmi és stílusjegyeket. A stílus kép $\vec{a}$ áthalad a hálózaton, és stílus reprezentációját A_l minden rétegre kiszámítjuk és tároljuk (bal oldal). A tartalmi kép $\vec{p}$ áthalad a hálózaton, és a tartalom reprezentációját P_l egy rétegben tároljuk (jobb oldal). Ezután egy véletlenszerű fehér zaj kép $\vec{x}$ áthalad a hálózaton, és kiszámítjuk annak stílusjegyeit G_l és tartalmi jegyét F_l . Minden stílus reprezentációban szereplő rétegre kiszámítjuk az G_l és A_l közötti elemenkénti négyzetes különbséget, hogy megkapjuk a stílusveszteséget $\mathcal{L}_{style}$ (bal oldal). Ugyanígy kiszámítjuk az F_l és P_l közötti négyzetes különbséget, hogy megkapjuk a tartalmi veszteséget $\mathcal{L}_{content}$ (jobb oldal). A teljes veszteség, $\mathcal{L}_{total}$, egy lineáris kombinációja a tartalmi és a stílus veszteségnek. A pixelértékekhez tartozó deriváltja az error back-propagation segítségével számolható ki (középső rész). Ezt a gradienst iteratívan alkalmazzuk az $\vec{x}$ képen, amíg egyszerre meg nem felel a stílus kép $\vec{a}$ stílusjegyeinek és a tartalmi kép $\vec{p}$ tartalmi jegyeinek (középső, alsó rész).



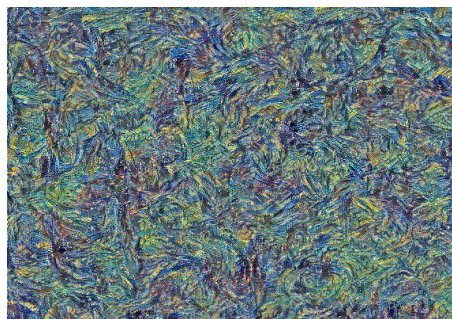
(a) Tartalmi kép



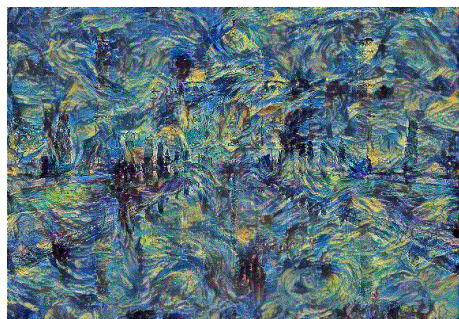
(b) Stílus kép



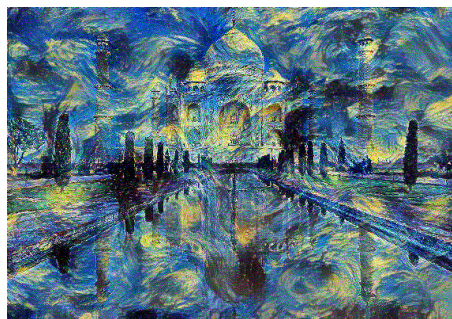
(c) 0 iteráció



(d) 500 iteráció



(e) 1500 iteráció



(f) 3000 iteráció

10. ábra. A tartalmi és stílus képnek az előzőekben is használt Taj Mahal és Starry Night képek lettek kiválasztva. Az algoritmus futásához Adam optimalizáló került felhasználásra, amely 3000 iteráción keresztül végezte el a gradiens hibafejtést annak érdekében, hogy az (a) kép fel tudja öltetni a (b) képnek a stílusát. Az $\alpha/\beta = 10^2$ arányt képvisel. Megfigyelhetjük, hogy kezdetben egy zajos képből kiindulva egyre jobban magára ölti a kép a stílusjegyeket és későbbiekben sikeresen ráilleszkedik a Taj Mahal formája a képre.

5.2. Eredmények

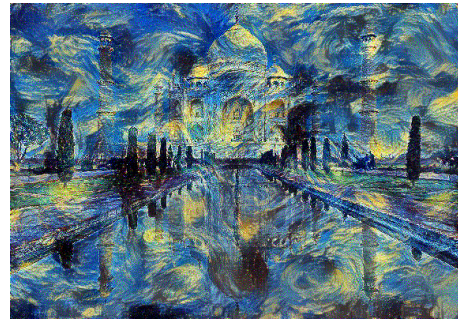
Ebben a tanulmányban a kulcsfontosságú megállapítás az, hogy a tartalom és a stílus reprezentációi a Konvolúciós Neurális Hálózatban jól szétválaszthatók. Tehát képesek vagyunk mindkét reprezentációt függetlenül manipulálni annak érdekében, hogy új, érzékelhetően jelentős képeket hozzunk létre.

5.2.1. Kompromisszum a tartalom és a stílus megfeleltetése között

Természetesen az képtartalom és a stílus nem teljesen szétválasztható. Amikor olyan képet szintetizálunk, amely összekapcsolja egy kép tartalmát egy másik kép stílusával, általában nincs olyan kép, amely tökéletesen megfelel mindkét korlátozásnak egyszerre. Azonban, mivel az általunk minimalizált veszteségfüggvény a tartalom és a stílus veszteségfüggvényeinek lineáris kombinációja, simán szabályozhatjuk a hangsúlyt a tartalom vagy a stílus rekonstrukciójára (11). A stílus erős hangsúlyozása olyan képeket eredményez, amelyek megfelelnek a műalkotás megjelenésének, gyakorlatilag texturált verzióját adják, de alig mutatnak valamit a fotó tartalmából. Amikor erős hangsúlyt fektetünk a tartalomra, egyértelműen azonosítható a fotó, de a festmény stílusa kevésbé illeszkedik. Egy adott tartalom- és stílus képpár esetén lehet módosítani a tartalom és a stílus közötti egyensúlyt annak érdekében, hogy vizuálisan vonzó képeket hozzunk létre.



(a) $\alpha/\beta = 10^3$



(b) $\alpha/\beta = 10^2$

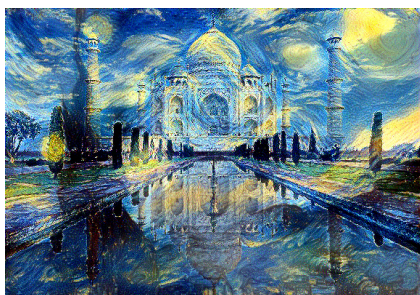


(c) $\alpha/\beta = 10^1$

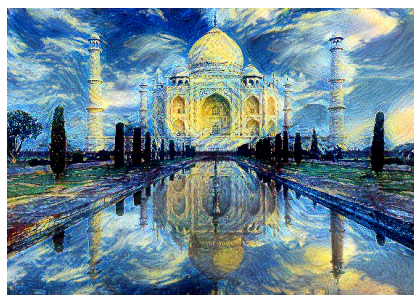
11. ábra. Az α/β arányait változtatva megfigyelhető, hogy az (a) képen nem jelennek meg megfelelő módon a stílus képből származó vonások, amíg a (c) kép esetében az uralmat teljes mértékben átveszi a stílus és elveszni látszik a tartalom. A (b) kép esetében viszont egy harmonia figyelhető meg a stílus és tartalom között.

5.2.2. A kép inicializálása

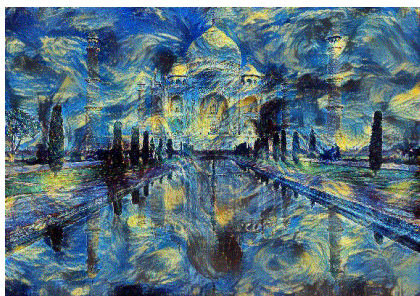
Eddig az összes megjelenített képet fehér zajjal inicializáltuk. Azonban az kép szintézis inicializálható lehet a tartalmi képpel vagy a stílus képpel is. Megvizsgáltuk ezt a két alternatívát: bár kissé befolyásolhatják a végleges képet az inicializálás térbeli szerkezete felé, a különböző inicializálásoknak nincs erős hatásuk a szintézis folyamatának eredményére. Ami viszont megfigyelhető, hogy amennyiben stílusból inicializálunk, megmaradhatnak maradványok az eredeti stílus képből. Fontos megjegyezni, hogy csak zajjal történő inicializálás teszi lehetővé új képek tetszőleges számú létrehozását. A rögzített képpel történő inicializálás mindig meghatározott eredményhez vezet (a gradiens descent eljárásának sztochasztikus jellegétől eltekintve).



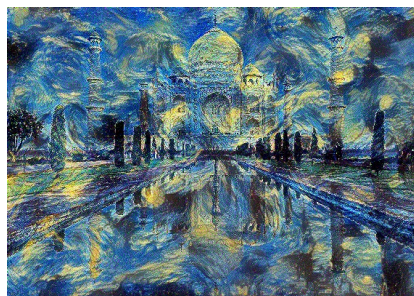
(a) stílus kép



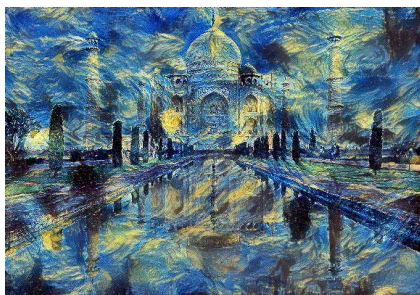
(b) tartalmi kép



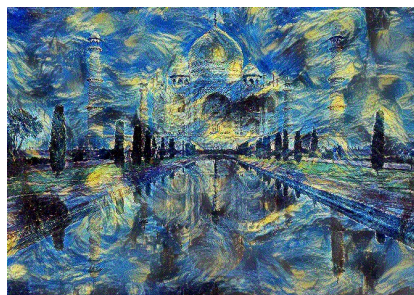
(c) zajos kép



(d) zajos kép



(e) zajos kép



(f) zajos kép

12. ábra. Amennyiben a stílus képet használjuk kiindulási pontnak (a), akkor megfigyelhető, hogy maradnak maradványok az eredeti festményből (mint például jobb felső sarokban lévő sárga Hold). A (b) esetben, mikor a kiindulási pont a tartalom, akkor nagyon gyöngéd módon helyezkedik rá a stílus és sokkal tisztább módon kivehető belőle a tartalom. A zajból való kiindulás (c, d, e, f) azt eredményezi, hogy minden egyes lefutásnál egy teljesen új kép jön létre, így végtelen megoldással bír ez a probléma.

5.2.3. Teljes variáció veszteség

A teljes variáció veszteség (*total variation loss*) egy fontos komponense a Neural Style Transfer eljárásnak. Ez a veszteség segít fenntartani a generált képek simaságát és textúráját. A *total variation* veszteség méri a kép intenzitásának változását a térbeli koordináták mentén. Minél kisebb ez a változás, annál simábbnak tekintjük a képet. A veszteség képlete az egymást követő pixelintenzitások közötti különbségek abszolút értékeinek összege.

$$\mathcal{L}_{tv}(\vec{x}) = \sum_{i,j} (|x_{i,j} - x_{i+1,j}| + |x_{i,j} - x_{i,j+1}|) . \quad (8)$$

A *total variation* veszteség hozzájárul a generált képek vizuális minőségének javításához, mivel segít csökkenteni azokat a zajokat és kis részleteket, amelyek gyakran megjelenhetnek a neurális stílusátvitel során. A *total variation* veszteség összeadódik a többi veszteségekkel, a stílusveszteséggel és a tartalmi veszteséggel, hogy megfelelő egyensúlyt teremtsen a képek stílusa és tartalma között.

$$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{style}(\vec{a}, \vec{x}) + \gamma \mathcal{L}_{tv}(\vec{x}) . \quad (9)$$



(a) $\gamma = 1$



(b) $\gamma = 10^4$



(c) $\gamma = 10^5$



(d) $\gamma = 10^6$

13. ábra. Látható, hogy amennyiben a γ értéke növekedik, annak függvényében lesz simább a végleges kép.

6. Alkalmazás megvalósítása

Azért, mert okostelefonszerveres része a mindennapjainknak, úgy döntöttem, hogy egy telefonos applikáció formájában valósítom meg az alkalmazást. Ezek mellett manapság az okostelefonok szinte olyan jó minőségű képeket készítenek, mint egy professzionális kamera. A szoftvert architektúráját 3 fő elemre lehet felbontani.

6.1. Szerver oldal

Abból kifolyólag, hogy a képek generálása erős hardvert igényel, a videokártyák játszanak kiemelt szerepet ebben a folyamatban. Ez azzal magyarázható, hogy több magon képesek párhuzamosítani és Tensor [27] műveleteket elvégezni. Úgy döntöttem, hogy a szerver oldalt lokálisan fogom futtatni a saját számítógépemem, mert cloud platformokon a videokártya igénylése igen csak magas anyagi költségekkel jár.

A szerverre Ubuntu 22.04-et [28] telepítettem, mert a szükséges driverek telepítése sokkal egyszerűbb módon történhetett meg, mint Windows alapú operációs rendszeren. Ezek mellett a számítógépet egy Geforce RTX 3060-as [29] videokártya hajtja meg, amely megfelelő gyorsasággal végezi el a szükséges számításokat a képek szintéziséhez. Az izoláció érdekében Conda [30] szoftver segítségével virtuális környezetet hoztam létre a megfelelő Python [15] verzióval és telepítettem a szükséges függőségeket, mint pl.: PyTorch [18] és FastAPI [20].

Az NST [31] algoritmus implementálásához a következő git repo-ból indultam ki: <https://github.com/gordicaleksa/pytorch-neural-style-transfer> [32].

A `neural_style_transfer.py` szkript a következő argumentumokat várja el:

- `--content_img_name`: amely elvárja a tartalmi kép nevét. Ennek a képnek a `data/content-images/` könyvtár alatt kell szerepelnie, hogy sikeresen be tudja olvasni.
- `--style_img_name`: amely elvárja a stílus kép nevét. Ennek a képnek a `data/stlye-images/` könyvtár alatt kell szerepelnie, hogy sikeresen be tudja olvasni.
- `--height`: megadja a kiadott kép magasságát, ebből pedig a program kiszámolja a kép szélességét úgy, hogy megtartsa a tartalmi kép arányait. Alapértelmezetten ez az érték 400, ami 400 pixel magasnak felel meg.
- `--content_weight`: az $\mathcal{L}_{content}$ súlyát adja meg. Ezt α -val ábrázoljuk amikor számoljuk az $\mathcal{L}_{total}$ -t.
- `--style_weight`: az $\mathcal{L}_{style}$ súlyát adja meg. Ezt β -val ábrázoljuk amikor számoljuk az $\mathcal{L}_{total}$ -t.

- `--tv_weight`: az $\mathcal{L}_{tv}$ súlyát adja meg. Ezt γ -val ábrázoljuk amikor számoljuk az $\mathcal{L}_{total}$ -t.
- `--optimizer`: itt lehet választani az Adam [33] vagy az L-BFGS [34] optimalizálok között. Az Adam-nek sokkal több időbe telik, amíg elvégzi a kép szintézist, viszont az L-BFGS esetén elég gyakran előfordult, hogy néhány iteráció után befejezte az optimalizálást sikertelen eredménnyel.
- `--model`: itt lehet választani a VGG19-es és VGG16-os között, a VGG19-es modell sokkal pontosabb eredményt ad, de sokkal több időt és memóriát vesz igénybe amíg a képet kigenerálja. [24]
- `--saving_freq`: megadja, hogy milyen sűrűségben mentse el a generált képeket az iterációk között. Alapértelmezetten a végleges képet menti el.

6.1.1. Weights and Biases

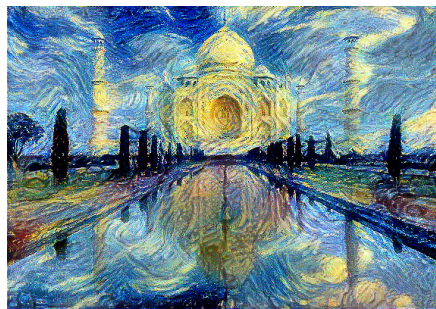
Ezek után elvégeztem egy futtatást a fent említett paraméterek különböző permutációjára, így sikeresen meg tudtam vizsgálni azt, hogy a paraméterek, milyen kombinációban adják a legjobb eredményt. A logolás a WandB-ra [21] történt, ahol egyszerű módon tudtam szűrni a különböző hiperparaméterek között, ezek mellett futás után is el lesznek mentve az eredmények a felhőben. Összesen 540 permutációban próbáltam ki a paramétereket és néhány kiválasztott képet, amelynek futási ideje 23 óra volt.



(a) tartalmi kép

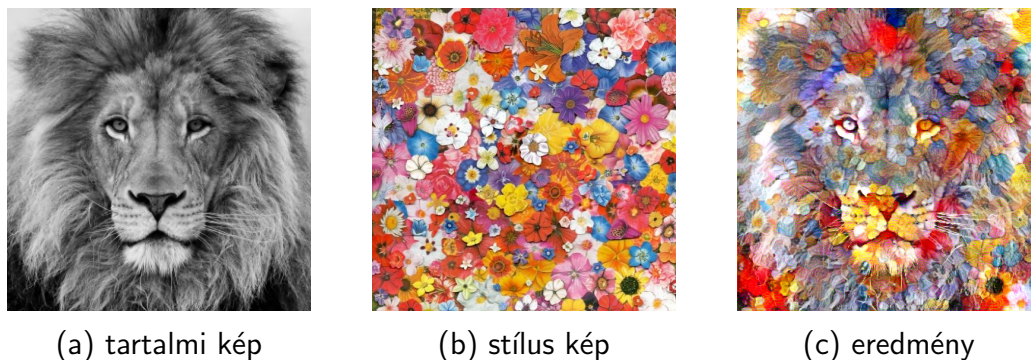


(b) stílus kép

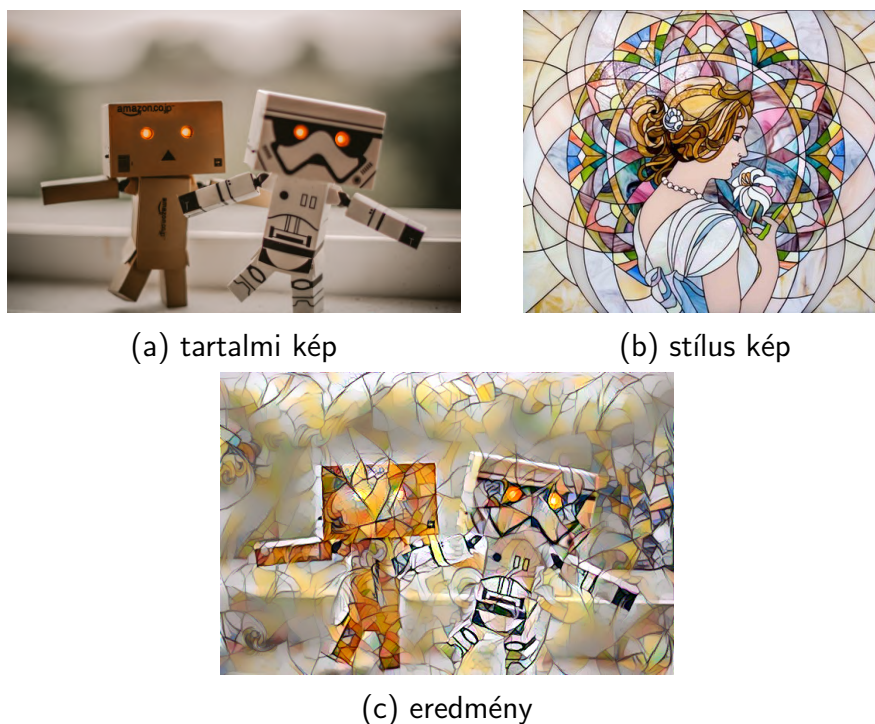


(c) eredmény

14. ábra. A legjobb eredményt a Taj Mahal és Starry Night-ra a következő paraméterek adták: `content_weight`: 10^5 , `style_weight`: 10^4 , `tv_weight`: 1, `init_method`: *content*



15. ábra. A legjobb eredményt a oroszlán képre és Ben Giles festményére a következő paraméterek adták: `content_weight: 105`, `style_weight: 103`, `tv_weight: 1`, `init_method: style`



16. ábra. A legjobb eredményt a figures képre és egy mozaikra a következő paraméterek adták: `content_weight: 105`, `style_weight: 102`, `tv_weight: 10`, `init_method: random`

Az eredmények (14., 15., 16. ábrák) bebizonyítják, hogy nincsenek kőbevésett paraméterek, a tesztelések adják ki, hogy éppenséggel mely adja a legjobb eredményt a felhasználó számára. Amennyiben szeretné, hogy a stílusból maradványok maradjanak a végső képen akkor érdemes a stílus képből kiindulni, amennyiben azt szeretné, hogy a tartalom jobban kivehető legyen, érdemes a tartalmi képpel kezdeni. Amennyiben pedig a kíváncsiság hajtja, meg lehet próbálni zajjal kiindulni. Ámbár észrevehető, hogy annak függvényében, hogy melyik képet vesszük alapul megváltozik súlyok aránya is, annak érdekében, hogy ne a stílus vagy tartalom uralja a végső képet.

6.1.2. Endpoint-ok

Mivel nem szeretnénk terhelni a telefonok erőforrásait a számításokkal, ezért a képeket egy helyi szerveren tároljuk majd, és ott végezzük el a képalkotáshoz szükséges számításokat. Viszont ezt hálózaton keresztül kell elérni, szóval létre kell hozni olyan endpoint-okat, amelyekkel képeket lehet feltölteni a szerverre, le lehet őket kérni és indítani lehet képszintézist adott paraméterekkel. Ennek megvalósítására FastAPI-t [20] vettem igényben, amely lehetővé teszi, hogy RestAPI-okon keresztül kommunikálni tudjon a készülék a szerverrel.

A következő endpoint-okra volt szükség:

- `http://server.domain/content/upload/`: felel azért, hogy sikeresen fel tudjunk tölteni tartalmi képet.

```
@app.post('/content/upload/')
async def upload_content_image(file: UploadFile = File(...)):
```

- `http://server.domain/style/upload/`: felel azért, hogy sikeresen fel tudjunk tölteni stílus képet.

```
@app.post('/style/upload/')
async def upload_style_image(file: UploadFile = File(...)):
```

- `http://server.domain/image/content/{image_name}`: felel azért, hogy adott tartalmi képet el tudjunk érni.

```
@app.get('/image/content/{image_name}')
async def get_content_image(image_name: str):
```

- `http://server.domain/image/style/{image_name}`: felel azért, hogy adott stílus képet el tudjunk érni.

```
@app.get('/image/style/{image_name}')
async def get_style_image(image_name: str):
```

- `http://server.domain/image/generated/{image_name}`: felel azért, hogy adott generált képet el tudjunk érni.

```
@app.get('/image/generate/{image_name}')
async def get_generated_image(image_name: str):
```

- `http://server.domain/generate`: szerepe, hogy a feltöltött képek kiválasztásával és megadott paraméterekkel elindítson egy képszintézist.

```
@app.post('/generate')
async def generate(content_img: str, style_img: str, init_method: str,
style_weight: int, tv_weight: int, iterations: int):
```

6.2. Adatbázis

Adatbázisul Firebase-t [14] használtam, azért mert egyszerűen oldható meg az összekötés. Ezek mellett olyan funkciókkal bír amelyek nagyon egyszerűvé teszik egy telefonos applikáció lefejlesztését. Két fő funkciót használtam fel a szolgáltatások közül.

Elsősorban a **Authentication**-t vettem hasznul amely felelős a felhasználók bejelentkezéséért. Több féle bejelentkezési módszer is elérhető, mint például Google fiókkal való bejelentkezés a jelszó megadása nélkül. Az egyszerűség kedvéért email és jelszó megadásával tettem lehetővé, hogy a felhasználók regisztrálni tudjanak az alkalmazásba. Természetesen a jelszavakat kódolva tárolja el az adatbázis, így a felhasználók jogai tiszteltben vannak tartva. Ezek mellett még igényel egy email-es megerősítést, a kamu felhasználók megelőzése érdekében.

Másodsorban a **Firestore Database** funkciót használtam fel, amely biztosít egy NoSQL adatbázist. Itt a egyszerű módon meg tudtam tervezni az adatbázis struktúráját. 3 *collection*-t hoztam létre az adatok tárolására

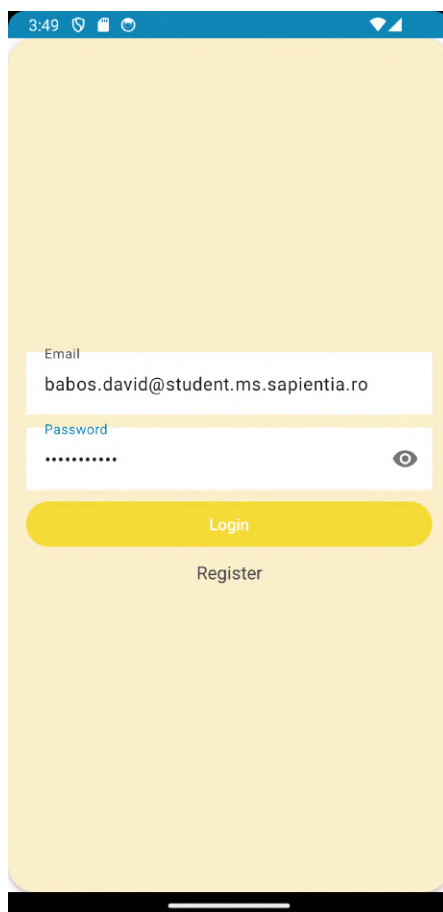
- `/styles` collection: itt tároltam el az alapértelmezetten megadott stílusok adatait: stílus neve, stílus URL-je.
- `/users` collection: itt tároltam el az alap információkat a felhasználóról, mint például a felhasználó beceneve. És mindegyik felhasználó dokumentum alá létrehoztam egy új `/styles` collection, ahova a felhasználó az általa feltöltött stílusokat tárolta el.
- `/gens` collection: ide kerültek a generált képek dokumentumai és hozzá tartozó adatok, mint például a kiválasztott stílus, a tartalmi kép URL-je, melyik felhasználó hozta létre és egyéb képszintézishez szükséges paraméterek amik fennebb is voltak említve (6.1 felsorolás).

6.3. Android alkalmazás

A könnyed felhasználói élmény érdekében egy mobil alkalmazást fejlesztettem Android platformokra. A fejlesztéshez Android Stúdiót [11] használtam, ahol a *layout*-ok megtervezéséhez XML-t [12] és logika implementáláshoz Kotlin [13] használtam. Az applikáció 6 *activity*-t tartalmaz, amelyek mind különböző funkciókat látnak el. Az alkalmazás számára elérhetőséget adtam az interneten való kommunikációhoz, hogy olvasni tudjon a külső tárhelyről. Fejlesztés során a 12-es Android verziójú telefonokat céloztam meg.

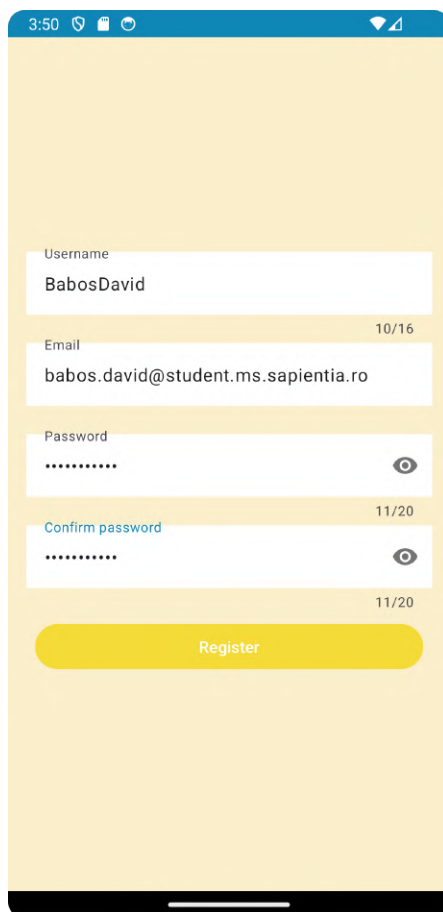
6.3.1. Bejelentkezés és Regisztráció

Amennyiben a felhasználó még nem volt bejelentkezve kezdetekben a Bejelentkezési oldalt (*Login Activity*, 17. ábra) dobja fel az alkalmazás, ahol e-mail cím és jelszó segítségével be tud lépni. Itt a Firebase [14] ellenőrzi, hogy a felhasználó megfelelő e-mailt/jelszót adott-e meg és, hogy megerősítette-e már a felhasználóját az email-en kapott megerősítő kóddal.



17. ábra. Login Activity

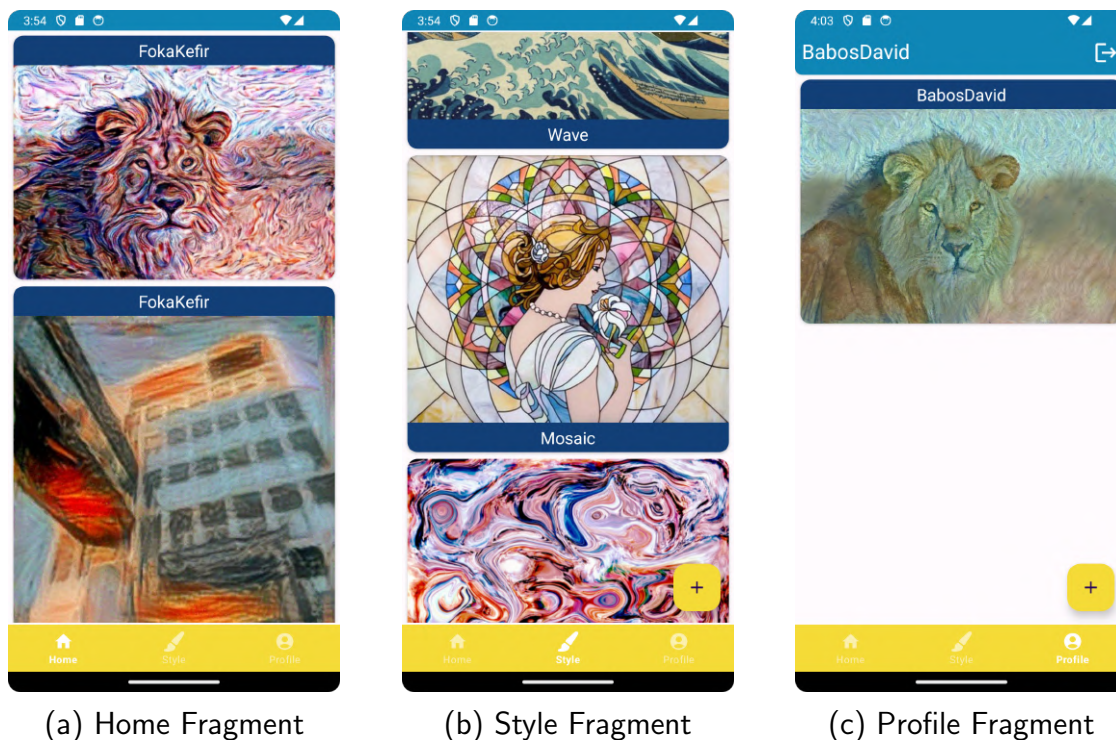
Amennyiben a felhasználó még nem volt regisztrálva az alkalmazásra a **Register** szövegre kattintva át tud menni a Regisztrációs felületre (*Register Activity*, 18. ábra), ahol név, email és jelszó megadásával tud regisztrálni az alkalmazásba. Az email és jelszó ellenőrzése Regex-ek [35] segítségével történik meg. Amennyiben helyes formátumban vannak megadva a bemeneti paraméterek, az adatbázisba is regisztrálja a felhasználót.



18. ábra. Register Activity

6.3.2. Főképernyő

Rögtön miután bejelentkezett a felhasználó, megtekintheti a főképernyőt (19. ábra), ahol lehetősége van navigálni különböző oldalak (*fragment*-ek) között. Az oldalak közötti váltakozást a Alsó Navigációs Sáv (*Bottom Navigation View*) és Navigációs Gráf (*Navigation Graph*) segítségével oldottam meg.



19. ábra. Main Activity

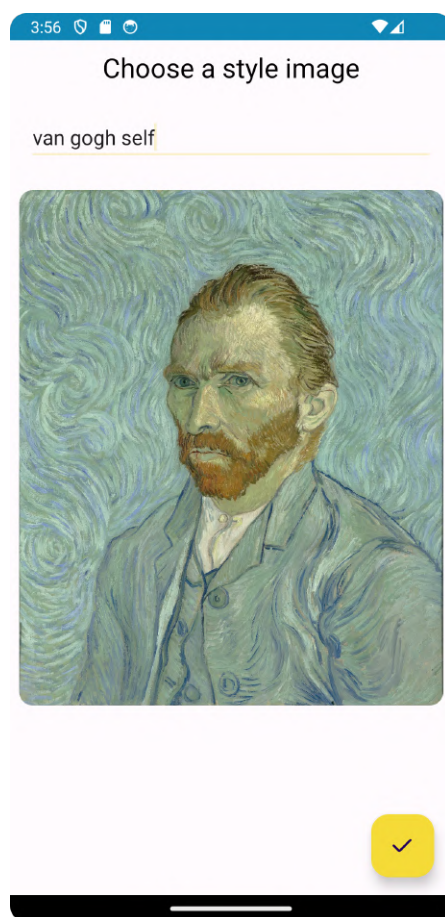
Az elsőként felugró **Home Fragment**-en (19.a) meg lehet tekinteni az összes felhasználó által generált képeket. Ennek megvalósítására egy *RecyclerView*-t alkalmaztam, amely az adatbázisból lekéri a generált posztokat (`/gens` collection, 6.2), majd pedig a szerverről (6.1.2) betölti a generált képeket. Azáltal, hogy a felhasználónak lehetősége van más felhasználók által létrehozott képeket böngészni, inspirációt tud nyerni az elkövetkező képek létrehozásához.

Amennyiben a felhasználó átnavigál **Style Fragment**-re (19.b) lehetősége van megtekinteni az alapértelmezetten megadott stílusokat, valamint az ön maga által hozzáadott stílusokat is. Ezeket ugyancsak az adatbázis (`/styles` és `/users/{user_id}/styles` collection, 6.2) és szerver (6.1.2) felhasználásával tölti be. Amennyiben a felhasználó nincs megelégedve az alaphoz adott stílus képekkel lehetősége van más stílusok hozzáadásához (6.3.3). Ez megadja a felhasználónak azt a lehetőséget, hogy a fantáziáját szabadon tudja engedni és olyan stílusokban is tudjon képet generálni ami alaphoz nem érhető el, más alkalmazásokkal szemben, mint pl. a Prisma [10] vagy Deep Art Effects [9], ahol a stílusok száma még akkor is korlátozva van, ha a felhasználó előfizet.

A harmadik fülre kattintva megtekintheti a **Profile Fragment**-et (19.c). Itt a **Home Fragment**-hez (19.a) hasonlóan meg tudja tekinteni a generált képeket, annyi különbséggel, hogy csak az általa létrehozott képek jelennek meg. Ezek mellett, ha a felhasználó ki szeretne lépni az alkalmazásból azt itt tudja megtenni. Amennyiben rákattint a + gombra lehetősége lesz új képeket alkotni (6.3.4).

6.3.3. Stílus hozzáadása

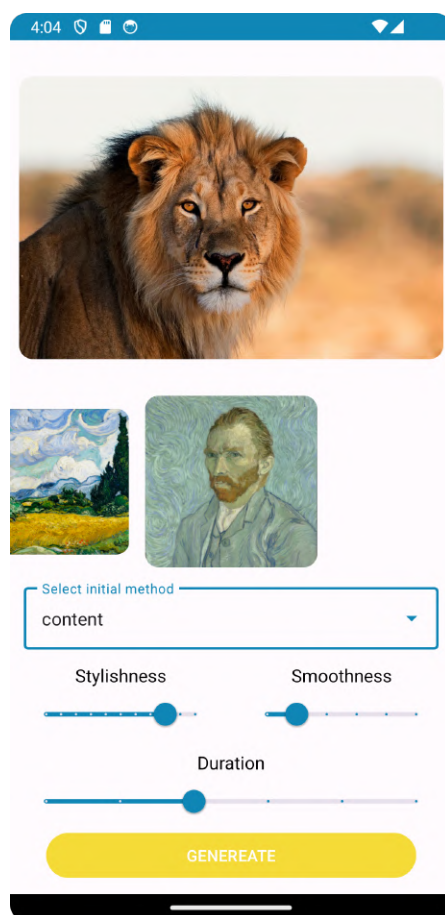
Amennyiben a felhasználó hozzá szeretne adni új stílusokat a könyvtárához azt a Style Fragment-en (19.b) a + gombra kattintva teheti meg. Ezután az alkalmazás átirányítja a következő oldalra, ahol a könyvtárból kiválaszthat egy általa tetszőleges képet és annak egy nevet is adhat. Majd pedig a pipára kattintva ezt véglegesíthet is. A felhasználó az újonnan hozzáadott képet meg tudja tekinteni a Style Fragment-en (19.b)



20. ábra. Add Style Activity

6.3.4. Kép generálása

Amikor a felhasználó megnyomja a + gombot a Profile Fragment-en (19.c), az alkalmazás átnavigálja a **Generate Image Activity**-re. Itt a felhasználó ki tudja választani a tartalmi képet az eszköztéről, ezek után a stílus könyvtárból ki tudja választani a számára tetszőleges stílus képet egy kép-csúszka segítségével. Ezek után a többi paramétert is be tudja állítani, mint például, hogy melyik képből induljon ki (5.2.2). A *stylishness* csúszka segítségével állítja az α/β közötti arányt (5.2.1). A *smoothness* csúszka segítségével tudja állítani a *total variation loss* erősségét (5.2.3). A *duration*-el pedig azt tudja beállítani, hogy hány iteráción keresztül generálja a képet, annak függvényében, hogy melyik képet veszi kiindulási pontnak (5.2.2), nem lesz annyi iterációra szükség amíg elvégződik a szintetizálást.

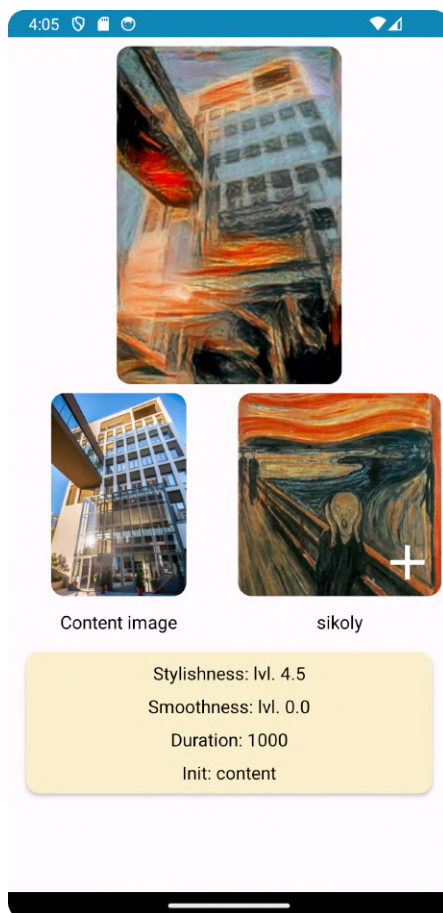


21. ábra. Generate Image Activity

6.3.5. Generált képek megtekintése

A felhasználónak lehetősége van egy generált kép részleteinek megtekintésére, amennyiben a Profile Fragment-en (19.c) vagy Home Fragment-en (19.a) rákattint egy generált képre. Itt megtekintheti a végeredményt és a kiindulási alapon vett képeket. Ezek mellett meg tudja nézni, hogy milyen paraméterek beállításával generálódott a kép (6.3.4).

Amennyiben a stílus még nincs elmentve a felhasználó stílusai között, lehetősége van a képen a **+**-ra kattintva elmenteni azt a könyvtárába. Ez a funkció is arra szolgál, hogy a felhasználók egymás számára inspirációt nyújtsanak az újabb képek megalkotásában.



22. ábra. Generation Activity

7. Összefoglaló

A szoftverről elmondható, hogy nagyon könnyed felhasználói élménnyel teszi elérhetővé bárki számára a Neural Style Transfer algoritmust (5). A képek szintetizálása könnyed módon valósítható meg és azáltal, hogy nincs korlátozva a stílusok száma, mint más alkalmazásokban, teljes mértékben a felhasználó kreativitására van bízva, hogy milyen képeket alkothat. Ezek mellett, mivelhogy egy közösségi média funkcióival is bír, organikus módon tudja terjeszteni az emberek között a művészetet és a felhasználók hatással tudnak lenni egymásra inspirációs jelleggel.

A használt algoritmus a 2016-os ”Image style transfer using convolutional neural networks” publikációból [31] lett felhasználva, amely megreformálta a képszintézis kutatási területét. Ezek mellett az algoritmus megalkotása érdekében Aleksa Gordiő példaprogramját [32] vettem alapul.

Ennek a projektnek a megalkotásának folyamán nagyon sokat tanultam a Konvolúciós Neurális Hálózatokról [24] és, hogy mily formában képesek információkat eltárolni a képekről a mély rétegekben. Ebben a kutatásban nagyon sok irányt látok amelyben még tovább lehetne haladni és mégjobban beleásni a képek szintézisébe és alkotásába.

8. Továbbfejlesztési lehetőségek

Ebben a projektben nagyon sok potenciált látok és véleményem szerint nagyon sok irányban lehet továbbfejleszteni ezt a szoftvert.

8.1. Közösségi média

Mivelhogy az alkalmazás egy közösségi média formájában körvonalazódik, így több manapság használt funkciót is át lehetne emelni már létező közösségi médiákból (pl. Instagram, Facebook, TikTok, stb.)

- Lehetőség, hogy ismerősöket jelöljünk be és csak azoknak a személyeknek a posztjai jelenjenek meg a főoldalon akiket ismerünk.
- Csoportokat létrehozni, ahova adott stílus szerint lennének csoportosítva a generált képek.
- Leírást adni a generált képekhez, így jobban körbe lehetne írni az elképzelt művet.
- Tag-elni a generált képeket, így eszerint lehetne szortírozni.
- Kommentelni más felhasználók képe alá.
- Like-olni a képeket.
- Megadni a lehetőséget, hogy ne csak a stílust mentse el egy kigenerált képnél, hanem egyenesen a beállított paraméterekkel ki is próbálni.

8.2. Üzleti logika

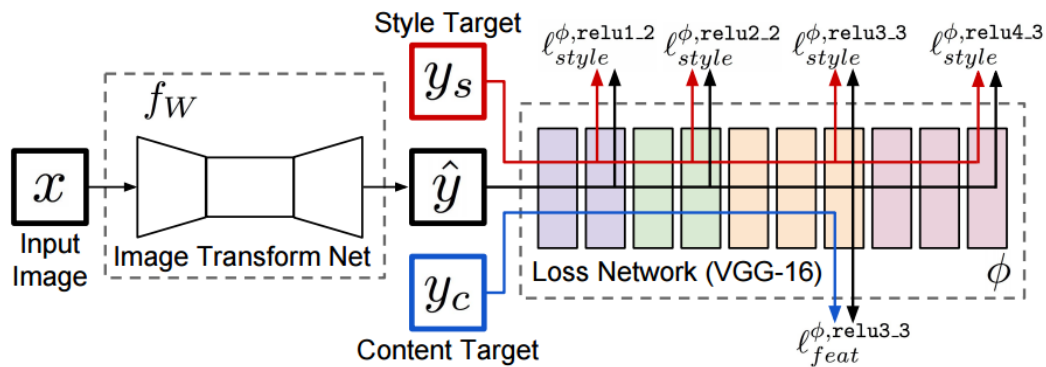
Bár sokan azt vallják, hogy a művészet mindenki számára elérhető és nem kellene pénzügyi hasznot húzni belőle, mégis vannak millió dolláros értékű festmények egyes emberek otthonaiban. Sokan élnek a művészeti alkotások kereskedelméből. Tehát ez a gondolatmenet is figyelembe vehető lenne az alkalmazás tervezésénél. Az egyik ilyen megközelítés az NFT-k (*Non-fungible token*) [36] gyártása lehet, ami a mesterséges intelligencia mellett nagy népszerűségnek örvendhet. Egy jó opció lenne, hogy minden felhasználó heti egy képét kiválaszthasson és későbbiekben NFT-ként [36] kereskedjen vele. Manapság hatalmas összegek keringenek eme alkotások körül.

Mindezek mellett egy design-er munkáját is művészetnek lehet nevezni, az alkalmazás eme személyek munkáját is meg tudja könnyebbíteni. Manapság logók megtervezésében sokan használnak MI-t. Az alkalmazás képes lenne poszter gyártók vagy logó tervezők munkáját megkönnyebbíteni.

8.3. Neural Style Transfer

A Neural Style Transfer a 2015-ös évektől kezdve rengeteg kutató fantáziáját megmozgatta és nagyon sok publikáció íródott arról, hogy hogyan lehetne még magasabbra emelni képszintézist javított NST-vel.

Mint megfigyelhetjük egy kép generálása akár 3000 iterációba is kerülhet, ami egy 400x400 pixel felbontású kép esetén és egy RTX 3060-as [29] videokártyát felhasználva is 3 percbe telhet. Erre megoldást nyújthat a Feed-Forward módszer [37], aminek a felhasználásával néhány másodperc alatt képes kigenerálni egy stilizált képet. Ennek a módszernek viszont az a hátránya, hogy hozzá mindegyik stílusra be kell tanítani egy modellt amelynek az az egyetlen szerepe, hogy az adott stílusban képet generáljon. Így elvesztené pont azt a lehetőséget, hogy a felhasználók önmaguk töltsék fel a stílusokat. De egy remek opció lehet, hogy az alapértelmezetten megadott képeket az imént említett módszerrel [37] generáljuk ki néhány másodperc alatt, de az újonnan behozott stílusokat a régi módszer [31] felhasználásával hozza létre. Amennyiben pedig egy újdonsült stílus nagyon népszerű lesz egy patch formájában arra is elérhetővé válhat a Feed Forward módszer.



23. ábra. Az Feed-Forward módszerrel működő NST háló rendszerének vázlata [37].

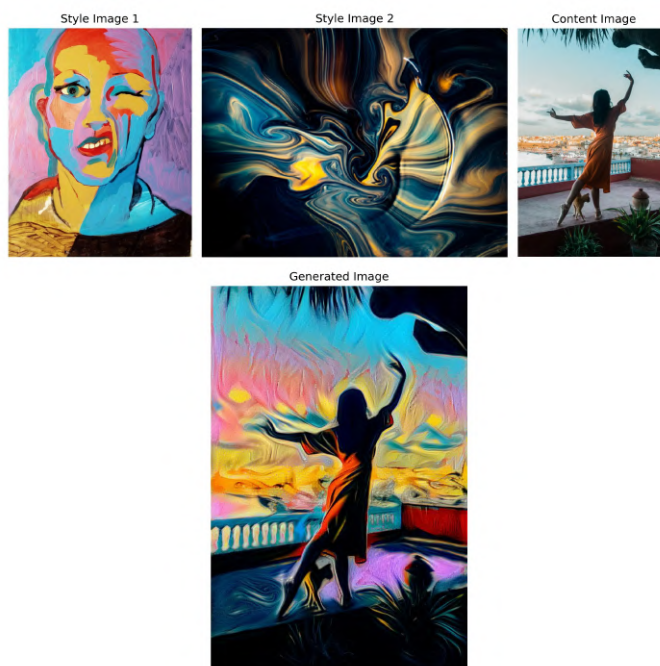
A Feed Forward módszer alkalmazása más kapukat is képes lenne megnyitni, mint például videók stilizálása, abból kifolyólag, hogy már szekundumok alatt képes lenne az algortimus új képek megalkotására, nem esne nehezére, egy betanított stílust alkalmazni egy videó frame-jeire. Így akár élő videókban is alkalmazni lehetne a NST-t.

Előfordul, hogy a képen látható tartalom, csak egy részét szeretnék módosítani, mint például egy személy vagy egy tárgy. Erre a problémára megoldást nyújt a szegmentáció, ami a tartalmi képből kiemel egyes objektumokat. Így akár azt is meg lehetne valósítani, hogy a fókuszban lévő objektum egy féle stílust vegyen fel, míg a háttér egy teljesen másikat (24. ábra).



24. ábra. A háttér stílusa a Starry Night-ot veszi alapul, míg a személy egy mozaik stílusát veszi fel [38].

Ami viszont igazán meg tudná mozgatni a felhasználók fantáziáját, ha lehetőség lenne egy tartalmi képet több stílussal felruházni. Előfordul, hogy egy stíusból remek mintázatokat képes kivonni az algoritmus, viszont a stílus kép önmagában nem elég színes, ezért egy második stílus képről venné át a színeket (25. ábra).



25. ábra. Két stílus alkalmazása egy tartalmi képre. Az elsőből átveszi a színeket a másodikból pedig a vonásokat [39].

9. Hivatkozások

- [1] TensorFlow, „Neural style trasfer,” https://www.tensorflow.org/tutorials/generative/style_transfer.
- [2] OpenAI, „Chatgpt,” <https://openai.com/blog/chatgpt>.
- [3] —, „Dall·e 2,” <https://openai.com/dall-e-2>.
- [4] M. W. Diederik P Kingma, „Variational autoencoder,” <https://arxiv.org/abs/1312.6114>.
- [5] M. M. B. X. D. W.-F. S. O. A. C. Y. B. Ian J. Goodfellow, Jean Pouget-Abadie, „Generative adversarial network,” <https://arxiv.org/abs/1406.2661>.
- [6] P. A. Jonathan Ho, Ajay Jain, „Denoising diffusion probabilistic models,” <https://arxiv.org/abs/2006.11239>.
- [7] A. Mordvintsev, „Deepdream,” <https://blog.research.google/2015/06/inceptionism-going-deeper-into-neural.html>.
- [8] Tensorflow, „Deepdream at tensorflow,” <https://www.tensorflow.org/tutorials/generative/deepdream>.
- [9] „Deep art effects,” <https://www.deeparteffects.com/>.
- [10] I. F. A. U. A. H. Alexey Moiseenkov, Oleg Poyaganov, „Prisma,” <https://prisma-ai.com/>.
- [11] Google, „Android studio,” <https://developer.android.com/studio>.
- [12] W. W. W. Consortium, „Xml,” <https://www.w3.org/XML/>.
- [13] JetBrains, „Kotlin,” <https://kotlinlang.org/>.
- [14] Google, „Firebase,” <https://firebase.google.com/>.
- [15] G. van Rossum, „Python,” <https://www.python.org/>.
- [16] T. Oliphant, „Numpy,” <https://numpy.org/>.
- [17] J. Hunter, „Matplotlib,” <https://matplotlib.org/>.
- [18] M. AI, „Pytorch,” <https://pytorch.org/>.
- [19] Intel, „Opencv,” <https://opencv.org/>.
- [20] S. Ramírez, „Fastapi,” <https://fastapi.tiangolo.com/>.

- [21] C. V. P. Lukas Biewald, „Weights biases,” <https://wandb.ai/site/>.
- [22] M. B. Leon A. Gatys, Alexander S. Ecker, „A neural algorithm of artistic style,” <https://arxiv.org/abs/1508.06576>.
- [23] H. S. J. K. S. S. S. M.-Z. H. A. K. A. K. M. B. A. C. B. L. F.-F. Olga Russakovsky, Jia Deng, „Imagenet large scale visual recognition challenge,” <https://arxiv.org/abs/1409.0575>.
- [24] A. Z. Karen Simonyan, „Very deep convolutional networks for large-scale image recognition,” <https://arxiv.org/abs/1409.1556>.
- [25] J. D. S. K. J. L. R. G.-S. G. T. D. Yangqing Jia, Evan Shelhamer, „Caffe: Convolutional architecture for fast feature embedding,” <https://arxiv.org/abs/1408.5093>.
- [26] M. B. Leon A. Gatys, Alexander S. Ecker, „Texture synthesis using convolutional neural networks,” https://papers.nips.cc/paper_files/paper/2015/hash/a5e00132373a7031000fd987a3c9f87b-Abstract.html.
- [27] Wikipedia, „Tensor,” <https://en.wikipedia.org/wiki/Tensor>.
- [28] M. Shuttleworth, „Ubuntu,” <https://ubuntu.com/>.
- [29] Nvidia, „Geforce rtx 3060,” <https://www.nvidia.com/en-us/geforce/news/gfecnt/20211/geforce-rtx-3060/>.
- [30] A. Inc., „Anaconda,” <https://www.anaconda.com/>.
- [31] M. B. Leon A. Gatys, Alexander S. Ecker, „Image style transfer using convolutional neural networks,” <https://ieeexplore.ieee.org/document/7780634>.
- [32] A. Gordić, „pytorch-neural-style-transfer,” <https://github.com/gordicaleksa/pytorch-neural-style-transfer>.
- [33] J. B. Diederik P. Kingma, „Adam: A method for stochastic optimization,” <https://arxiv.org/abs/1412.6980>.
- [34] P. L. C. Zhu, R. H. Byrd and J. Nocedal, „Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization,” <https://dl.acm.org/doi/10.1145/279232.279236>.
- [35] S. C. Kleene, „Regex,” https://en.wikipedia.org/wiki/Regular_expression.
- [36] A. D. Kevin McCoy, „Non-fungible token,” <https://www.nft.com/>.
- [37] L. F.-F. Justin Johnson, Alexandre Alahi, „Perceptual losses for real-time style transfer and super-resolution,” <https://arxiv.org/abs/1603.08155>.

-
- [38] A. Gordić, „pytorch-naive-video-nst,” <https://github.com/gordicaleksa/pytorch-naive-video-nst>, 2020.
- [39] A. Chadha, „Visualizing and understanding,” 2020, <https://aman.ai>.