SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM
MŰSZAKI ÉS HUMÁNTUDOMÁNYOK KAR,
MAROSVÁSÁRHELY
SZOFTVERFEJLESZTÉS SZAK

Párhuzamos képstílus átruházás konvolúciós neuronhálókkal

MESTERI DISSZERTÁCIÓ

TÉMAVEZETŐ:
dr. Iclănzan Dávid
Egyetemi tanár

SZERZŐ:
Szilágyi Ervin

2017 Július

UNIVERSITATEA SAPIENTIA TÂRGU-MUREŞ
FACULTATEA DE ŞTIINŢE TEHNICE ŞI UMANISTE
SPECIALIZAREA DEZVOLTARE DE SOFTWARE

DeepArt

Lucrare de master

Coordonator ştiinţific:
dr. Iclănzan Dávid

Absolvent:
Szilágyi Ervin

2017 Iulie

SAPIENTIA UNIVERSITY TÂRGU MUREŞ
FACULTY OF TECHNICAL AND HUMAN SCIENCES
SOFTWARE DEVELOPMENT SPECIALIZATION

Parallel artistic style transfer using deep convolutional neural networks

Master Thesis

Advisor:
dr. Iclănzan Dávid

Student:
Szilágyi Ervin

2017 July

eredetisegi nyilatkozat

KIVONAT

kivonat

Szilagyi Ervin,
.....

ABSTRACT

abstract

Szilagyi Ervin,

.....

ABSTRACT

english abstract

Szilagyi Ervin,

.....

Tartalomjegyzék

Cuprins

Table Of Contents

1. fejezet

Bevezető

Napjainkban a képfeldolgozás egy eléggé elterjedt kutatási terület. A kutatások célja főleg az információ kinyerésére, gépi látás kivitelezésére irányult. Minderre kiváló megoldást jelentett a mély konvolúciós hálók (ConvNets)[1][2] sikeres használata növelve ezzel ezek népszerűségét. Fontos megjegyezni, hogy a konvolúciós neuron hálók felfedezése már pár évtizede történt, tehát maga a technológia már régebb ismert volt. Az újrafelfedezésüket és hirtelen népszerűség növekedését annak köszönhetik, hogy az utóbbi években olyan hardveres megoldások jelentek meg, amik lehetővé teszik az ilyen típusú hálók létrehozását és működtetését.

Az Nvidia cég 2007-ben bevezette az Nvidia CUDA platformot[3]. Ez egy komoly, használható fejlesztő környezetett jelentett olyan fejlesztők számára akik nagy méretű adatpárhuzamos algoritmusokat szerettek volna fejleszteni. A CUDA környezet direkt elérhetőséget nyújt a videókártya utasításkészletéhez megengedve ezzel ennek a programozását. Ugyanakkor számos olyan videókártya került piacra ami egyre komolyabb számítási készségekkel bírt. Ezt a lehetőséget értelemszerűen a kutatók ki is használták így számos újabb publikáció és javaslat jelent meg amik neuron hálókat használnak az illető probléma megoldására.

A deep konvolúciós hálók népszerűségének növekedésével egyre több olyan fejlesztői környezet jelent meg amiknek célja a mesterséges intelligencia feladatok megoldása. Ilyen könyvtárak például a Caffe[4], Keras[5], Theano[6], Tensorflow[7], Torch[8] stb. Ezek a környezetekben, habár különböző stílusban de egyazon problémákra hivatottak gyors és egyszerű megoldásokat ajánlja ugyanúgy mezei szoftverfejlesztők, mint kutatók számára.

Az gépi látás egyik fontos alkalmazási területe a képen levő tárgyak, élőlények emberek felismerése. Ilyen területen a konvolúciós hálók kimagasló teljesítményt nyújtanak, olyannyira, hogy egyes kisérletek szerint ez már nemhogy az emberi látással megegyező, hanem azt felülmúló teljesítményt nyújtanak[9]. Feltevődik a kérdés, hogyha ennyire szofisztikált a gépi látás, akkor nem-e lehetne használni arra, hogy új képeket alkosson. Amint kiderült erre is alkalmasak. Az általam bemutatandó dolgozat is ezt a témát próbálja megcélozni. A gépi látás a tárgyak, élőlények mellett képes felismerni maga a kép

művészeti stílusát. Ez elsősorban kihasználható arra, hogy híres művészek alkotásait csoportosítsuk, rendszerezzük[10], de amint e dolgozatból ki fog derülni, ki lehet használni arra is, hogy egy művészeti stílust egy adott festményről átvigyük egy mindennapi képre, fotóra.

A dolgozatom célja magyar híres festőművészek festészeti stílusát átvenni és ezt alkalmazni mindennapi képekre illetve mozgóképekre. Eddigiekben, ahhoz hogy egy mindennapi fényképből művészeti képet varázsoljunk, képszerkesztő szoftverek segítségével lehetett elérni manuálisan. Mindezt egy olyan egyén végezhette, akinek képszerkesztési illetve képmanipulálási szakismere volt adott képszerkesztési szoftverkörnyezetben. Magától értődik az, hogy ez mozgóképek esetében egy időigényes folyamat. Dolgozatom mindezekre megoldást próbál adni, azáltal, hogy az általam elkészített szoftvert bárki használhatja, nincs szükség különböző képszerkesztői szakértelemre, emellett a folyamat ideje jelentősen csökkenni fog.

2. fejezet

Hasonló rendszerek feltérképezése

A neuron hálók használata a számítástechnikában nem egy újonnan kialakult terület. Frank Rosenblatt 1958-ban publikált egy olyan mintafelismerő algoritmust[11], ami egyszerű összeadást és kivonást használva képes volt "tanulni". A rendszer képes volt finomhangolni állapotát a bekövetkező iterációk során. Ezt az algoritmust perceptronnak nevezzük. 1975-ben Paul Werbos bevezette a backpropagation algoritmust[12], amit a perceptronnal együtt használva megoldotta a perceptron azon problémáját miszerint az csak lineárisan elválasztható osztályokat volt képes kategorizálni. Habár a neuron hálók tanulmányozása eléggé ígéretesnek látszott, számítási igényük, komplexitásuk és lassú válasz idejük miatt a kutatók arra következtetésre jutottak, hogy a gyakorlatban még nem lehet alkalmazni őket.

Yann LeCun professzor és csapata 1998-ban egy újabb topológiájú hálót vezetett be[13]. A LeNet-5 elnevezésű háló konvolúciós rétegeket is tartalmazott ezért konvolúciós neuron hálónak nevezzük. A publikáció célja kézzel írott számjegyek kategorizálása volt, létrehozva ezáltal a MNIST adatbázist, ami 60000 28x28-as felbontású kézzel írott számjegyet tartalmaz, emellett tartalmaz egy 10000 tagból álló teszthalmazt. A dolgozatban bemutatott LeNet-5 háló 0,7%-os hiba aránnyal volt képes kategorizálni a számjegyeket, ami messze felülmúlta a többrétegű perceptronos megoldást.

Dave Steinkraus, Patrice Simard és Ian Buck 2005-ben publikált dolgozata[14] letette az alapjait a neuronhálók videokártyán történő programozásának. A videókártyán történő adatpárhuzamos programozás hatalmas performancia növekedést jelentett a processzoron futó neuronhálókkal szemben. Előtérbe kerül a deep learning és a mély konvolúciós hálók használata[1][2].

Eddigiekben sikerült nagyon pontos felismerő illetve osztályozó rendszereket alkotni. A mély konvolúciós hálók használata azonban nem merül ki ennyiben. 2015-ben publikálásra került egy olyan deep learning-et használó algoritmus, ami képes képek illetve festmények művészeti stílusát átvinni egy másik digitális képre[15]. Mostani dolgozatom is erre a publikációra alapoz, az ebben bemutatott módszereket próbálja alkalmazni illetve továbbfejleszteni. A tanuláshoz egy korábban bevezetett és gépi látáshoz használt, előre

betanított neuron hálót használnak fel, a VGG-19-et. Yaroslav Nikulin és Roman Novak tudományos kutatása[16] ezzel szemben eddig ugyanezt a módszert alkalmazta más ismertebb előre betanított hálókra, mint például AlexNet, GoogLeNet vagy VGG-16. Ugyanúgy a VGG-16 háló használata is kiváló eredményeket mutatott míg a GoogLeNet és az AlexNet architektúrájuk miatt komolyabb információvesztéshez vezetnek így a végeredmény nem lesz annyira látványos. Ugyanúgy kísérletek irányultak az eredi eljárás optimalizálására, megjelentek olyan rendszerek amik sajátos, erre a célre betanított neuron hálókat alkalmaznak[17][18][19].

2016-ban a Prisma labs inc. kiadta mobilos applikációját Prisma név alatt[20]. Az aplikáció előre megadott ismert festői/grafikai stílusokat alkalmazza a telefon kamerája által készített képekre. Az applikáció az előbbiekben bemutatott kutatásokra alapoz. Ugyanakkor fontos megjegyezni, hogy maga a stílus alkalmazását a különböző fotókra nem az okostelefon végzi. A szerkeszteni kívánt képet a telefon felkülde egy szervergépre ami majd válaszként a szerkesztett képet küldi vissza.

Maga stílusátvitel nem csak állóképekre alkalmazható, ezt bizonyította Manuel R., Alexey D., Thomas B. tudományos dolgozata[21], valamit ezt próbalja megoldani a jelenlegi dolgozatom is. Értelemszerűen egy adott videót több álló képkocka alkot. Viszont ahhoz, hogy látványos művészeti mozgóképet gyártsunk, nem elegendő maga a videót darabokra vágni és minden képkockára alkalmazni a stílust. Erre adott megoldást Manuel R. és társainak kutatása.

3. fejezet

A rendszer

3.1. Áttekintés

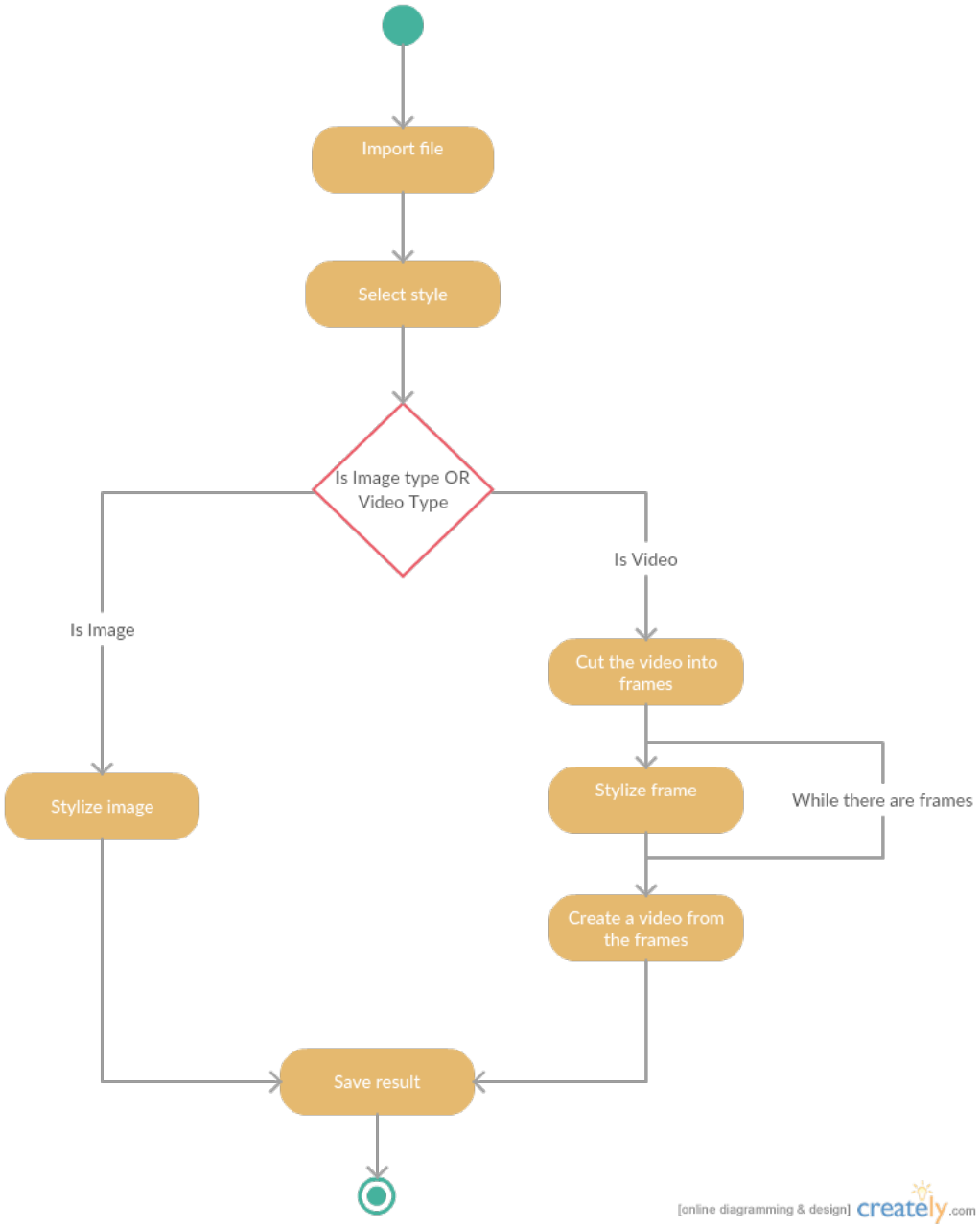
Dolgozatom célja egy olyan multiplatform számítástechnikai szoftver tervezése és fejlesztése ami deep learning-et használva képes híres magyar festők festményeinek a stílusát átvenni és alkalmazni a felhasználó által megadott digitális képekre illetve videókra. A fejlesztett szoftver könnyen használható grafikus felhasználói felülettel rendelkezik és támogatja a Linux valamint a Microsoft Windows alapú operációs rendszereket. A szoftver futtatásához a felhasználónak rendelkeznie kell egy olyan videókártyával ami támogatja az Nvidia CUDA platformot.

A szoftver fejlesztése Python3.5[23] programozási nyelvben történt, viszont egyes esetekben felhasználásra kerülnek egyes előre legyártott önállóan is futtatható állományok. Emellett még használva vannak a következő Python könyvtárak:

- numpy[24]: használata elengedhetetlen akkor, ha többdimenziós tömbökkel szeretnénk dolgozni. Nagyon sok matematikai problémára tartalmaz előre definiált megoldást és efelett tökéletesen használható a tensorflow könyvtár mellett
- tensorflow[7]: talán egyik legismertebb deep learning és mély konvolúciós hálók tanítására kifejlesztett könyvtár. Teljes mértékben támogatja a videókártyán történő programozást.
- PyQt[25]: a szoftver grafikus felhasználói felületének a megvalósításához használatos, emellett komoly feladat orientált párhuzamosítást tud biztosítani.
- OpenCV[26]: képfeldolgozásra szakosodott könyvtár, főleg a különböző kiterjesztésű képek beolvasására és mentésére volt használva.

A rendszer működése felülnézetből nagyon egyszerű (3.1 ábra). A felhasználó kiválaszt egy bemeneti állományt, ami kép kiterjesztésű (.jpg, .png) vagy mozgókép kiterjesztésű (.gif, .avi, .mp4) lehet valamint kiválaszt egy stílust a megadottak közül. A rendszer

annak függvényében, hogy milyen bemenetet adtunk, eldönti, hogy kép vagy mozgóképpel kell dolgoznia. Kép esetén egyszerűen lefuttatja a tanulási algoritmust amely során az átveszi a művészeti kép stílusát. Mozgókép esetén képkockákra bontja azt, majd minden képkockára alkalmazva lesz a tanítási algoritmus. Ha összes képkocka szerkesztve lett, akkor a rendszer felépíti a képkockákból a kimeneti videót. Ezek után, függetlenül, hogy kép vagy videó lett a végeredmény, a rendszer kimenti azt egy megadott folderbe.



3.1. ábra. A rendszer működése felülnézetből

3.2. A tanítási módszer állóképek esetében

A rendszer tanításához állokép esetében két legalább képet bemenet szükséges, az eredeti kép, amire át szeretnénk ruházni a stílust és a stílust tartalmazó kép, aminek a stílusát át szeretnénk ruházni. Mindkét bemenet esetében felírunk egy veszteség függvényt amik részei a végső nagy tanítási függvénynek.

3.2.1. Az eredeti kép tanításának a veszteségi függvénye

A mély neurálos hálók (Deep Neural Networks) azon típusai amik a legeredményesebbek a képfeldolgozási feladatok elvégzésben a konvolúciós hálók. Mesterséges intelligencia területén a konvolúciós hálók olyan feed-forward típusú neuronhálók, amiket a biológiai elsődleges látókéregről mintáztak. A háló kifejezetten arra volt tervezve és kifejlesztve, hogy kétdimenziós formákat ismerjen fel. A háló alapértelmezetten több rétegből tevődik össze:

- konvolúciós réteg: a konvolúciós hálók alap kövei. A réteg súlyai úgynevezett konvolúciós szűrők alkotják, amelyek a forward pass lépés során tanítva vannak. Ez a tanítási lépés úgy történik, hogy a szűrőt végig toljuk a bemeneten és konvolúciónak nevezett műveletet végzünk.
- pooling layer: arra használatos, hogy a bemeneti adathalmazt méretét leszűkítsük úgy, hogy a halmaz adott értékein valamely matematikai műveletet végzünk (átlagszámolás, maximum számolás, minimum számolás). Erre azért van szükségünk, hogy növeljük a tanulás gyorsaságát, csökkentsük a számítások komplexitását megakadályozva ezzel az "overfitting" bekövetkezését.
- fully connected layer: minden bemenet mindem más bemenettel kapcsolatban áll.

A konvolúciós hálók súlyzókként konvolúciós szűrőket tartalmaznak. Ezek, tárgyfelisrmerés tanítása esetében, az adott bemeneti kép különböző tulajdonságait fogják tartalmazni. Annak függvényében, hogy a háló topológiájában egy adott réteg hol helyezkedik el, más tulajdonságokat fognak raktározni a szűrök. Az bemeneti réteghez közel álló alacsony szintű rétegek az adott kép pixelinformációit próbálják megjegyezni ezzel szemben a magasabb szinten levő rétegek szűrői különböző tárgyakat, formákat próbálnak megjegyezni[27][28]. A hálók által tartalmazott információ vizualizálható azáltal, hogy rekonstruáljuk a bemeneti képet a súlyzók alapján. Alacsony rétegek esetében apró módosításokkal visszanyerhető az eredeti kép míg magasabb rétegek esetében objektumok, formák nyerhetők vissza.

Az eredi kép tanításához egy már előre betanított mély konvolúciós neuronhálót használunk fel. A háló tudományos kontextusban VGG-19 név alatt terjedt el amit Simonyan

K. és Zisserman A. vezetett be publikációjukban[29] ahol még a "model E" nevet viselte. Ez a háló gép látás és tárgyfelismerés céljából volt bevezetve olyan eredményeket produkálva e téren amik az emberi látással versengenek. Rétegei és topológiájának részletes bemutatása a [29] publikációban történik, valamint a 3.2 figurán is látható. Fontos megjegyezni, hogy a háló rétegeiből használva volt a 16 konvolúciós réteg, valamint az 5 pooling réteg, a teljesen összekötött rétegek nem voltak használva.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

3.2. ábra. VGG-19 (Model E) háló topológiája[29]

A választás azért esett erre az előre betanított hálóra, mivel Yaroslav N. és Roman N. kutatása alapján[16] összehasonlítva a AlexNet, GoogLeNet VGG-16 és VGG-19 halókat, a VGG-19 használata során sikerült a leglátványosabb képeket készíteni.

A konvolúciós neuron háló minden rétege tartalmaz egy adott számú szűrőt, amik különböző tulajdonságokat tartalmaznak a bementi képről. Ebből kifolyólag maga a bemenet

kódolva van a szűrőkben. Egy N tulajdonságót tartalmazó réteg N szűrővel rendelkezik amelyek mérete M. Maga a réteg kimenete lementhető egy $N * M$ -es mátrixban. Egy adott réteg válasza egy bemeneti képre vizualizálható, ha gradient descent módszert alkalmazunk egy fehér zajt tartalmazó bemeneti képen. Tehát, legyen R^l egy adott az l haló válasza egy adott bemeneti képre. Legyen W^l ugyanaz az l háló válasza egy adott fehér zajt tartalmazó bemeneti kép esetében. Akkor felírható a veszteség függvény mint:

$$L_{content}(\vec{x}, \vec{r}, l) = \frac{1}{2} \sum R_{ij}^l - W_{ij}^l \tag{3.1}$$

Ahol $\vec{x}$ a bemeneti képet jeleni, $\vec{r}$ pedig azt a kimeneti képet jelenti amit a rendszer generál a rétegek tulajdonságaiból. Mindez Tensorflow környezetben a következőképpen nézne ki:

```
for layer_name in CONTENT_LAYER:
        content_loss += content_weight * (2 * tf.nn.l2_loss(
                all_layers[layer_name] - content_features[layer_name]) /
                content_features[layer_name].size)

content_loss /= len(CONTENT_LAYER)
return content_loss
```

A $CONTENT\_LAYER$ tuple típusú ami tartalmazza azoknak a rétegeknek az azonosítóját amik részt vesznek a veszteség függvény kiszámításában. A $content\_features$ valtozó egy lista, ez tartalmazza az összes réteg válaszát a bemeneti eredeti képre. A visszaküldött érték egy tensor típusú, egyik részét fogja képezni a végső optimalizálandó veszteségfüggvénynek.

3.2.2. Az stílus kép tanításának a veszteségi függvénye

Az előbbiekben az eredeti bemeneti kép tartalmára voltunk kíváncsiak. A stílus kép esetében viszont nem maga a kép tartalma a fontos. Ebben az esetben egy mintázatot szeretnénk kinyerni a stílusképből. Ehhez fontos megismerni maga a Gramm mátrix fogalmát. Hogyha adott egy vektorhalmazunk $v1...vn$, akkor a G Gramm mátrixot a következő eljárás szerint határozzuk meg[30]:

$$G_{ij} = v_i \cdot v_j \tag{3.2}$$

Tehát hogyha az A mátrixunk oszlopait maga a $v1, v2...vn$ vektorok képezik, akkor a G Gramm mátrix a következőképpen kapható meg:

$$G_{ij} = AA^T \tag{3.3}$$

A Gramm mátrix egy szorzatot jelen egy adott vektorhalmaz összes elemei között. A mi esetünkben ez a vektorhalmaz jelentheti egy adott konvolúciós réteg által kiszűrt tulajdonságokat az adott bemeneti stílusképből. Amint már említettem, maga a konvolúciós réteg szűrői egy adott kép tulajdonságait tartalmazzák. A Gramm mátrix ij pozíciójában elhelyezkedő elem megadja, hogy egy adott réteg i-dik tulajdonsága mennyire teljesül a j-dik tulajdonság jelenlétében, tehát a két tulajdonság milyen mértékben aktiválódik egyszerre. Ha az ij pozícion levő elem közelít a 0-hoz, akkor az azt jelenti, hogy a két tulajdonság nem aktiválódik egyszerre, ezzel ellentétben, ha az érték nagy, akkor az azt jelenti, hogy a két tulajdonság nagy valószínűséggel aktiválódik egyszerre.

Hogyha az l rétegnek N szűrője van, akkor $G \in R^{N_l * N_l}$, ahol:

$$G^l_{ij} = \sum_k F^l_{ik} \cdot F^l_{jk} \tag{3.4}$$

Ahhoz, hogy megkapjuk egy adott háló által generált mintát, textúrát, hasonlóan mint az előzőekben, fehér zaj képet adunk bemenetként. A veszteségfüggvény egyetlen hálóra felírható mint a fehér zaj képre adott válasz és a stílus képre adott válasz átlagos négyzetes hibájaként. A G jelenti a fehér zaj képre adott választ, az A pedig a stílus képre adott választ.

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij} - A_{lj})^2 \tag{3.5}$$

Az összes stílus réteg válasza felírható, mint egy összeg, ahol w_l egy súlyzó faktort jelent:

$$L_{style}(\vec{a}, \vec{x}) = \sum w_l E_l \tag{3.6}$$

3.2.3. A stilizált kép tisztítása

A két veszteségfüggvényt alkalmazva sikeresen át tudjuk ruházni a stílust a művészi képről a bemeneti mindennapi képre. Viszont észlelhetjük, hogy az eredmény kép eléggé zajos. Ennek érdekében bevezetünk egy újabb veszteségfüggvényt a zaj csökkentésére, ami a Total Variation Denoising algoritmusra alapszik. Az algoritmus szerint vesszük a stilizált képet és eltoljuk X koordináta mentén egy pixellel, majd az Y koordináta mentén is eltoljuk egy pixellel. Az eltolt képeket kivonjuk az eredi képekből, majd az eredmények abszolút értékeit pixelenként összeadjuk. Ezáltal egy újabb veszteségi függvényt alítunk elő, amit ugyancsak minimalizálni kell.

$$L_{tv}(\vec{a}, \vec{x}) = \sum_{i,j} |(X_{ij} - A_{i+1j})| + \sum_{i,j} |(X_{ij} - A_{ij+1})| \tag{3.7}$$

3.2.4. A teljes veszteségfüggvény felírása és tanítása statikus képek esetében

Eddig bemutatásra került a stílus kép tanításának veszteségfüggvénye, az eredeti kép veszteség függvényének tanítása valamint egy veszteségfüggvény a kimeneti kép zajtalanítására. A végső veszteségfüggvény egyszerűen felírható a három veszteségfüggvény összegeként.

$$L = L_{content} + L_{style} + L_{tv} \tag{3.8}$$

A teljes veszteségfüggvény tanítására az Adam[32] optimalizáló algoritmus alkalmaztuk. Az Adam (Adaptive Moment Estimation) momentum algoritmus egy sztochasztikus gradiens alapú optimalizáló algoritmus. Az algoritmus felhasználja a elsődleges, valamint a másodlagos gradiensek átlagát ahhoz, hogy a veszteségfüggvényt minimalizálja. Az optimalizálási lépések a következő eljárás szerint történnek:

$$\begin{aligned}(m_t)_i &= \beta_1(m_{t-1})_i + (1-\beta_1)(\nabla L(W_t))_i \\ (v_t)_i &= \beta_2(m_{t-1})_i + (1-\beta_2)(\nabla L(W_t))_i^2 \\ (W_{t+1})_i &= (W_t)_i - \alpha\frac{\sqrt{1-(\beta_2)_i^t}}{1-(\beta_1)_i^t}\frac{(m_t)_i}{\sqrt{(v_t)_i}+\varepsilon}\end{aligned} \tag{3.9}$$

[33]

Ahol, m és v jelentik az elsőfokú valamint másodfokú gradienseket, W jelenti a súlyzókat, β_1 és β_2 jelentik a momentumokat, emellet ε egy nagyon kis értékű szám annak érdekében, hogy elkerüljük a 0-val való osztást.

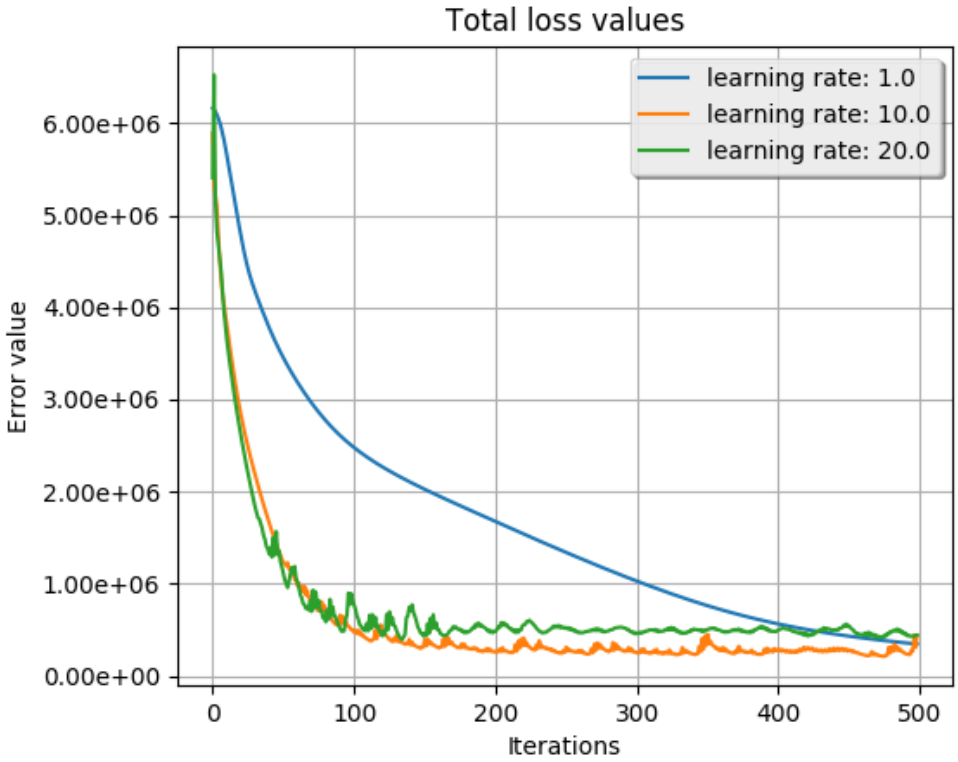
Értelemszerűen a tensorflow deep learning könyvtár beépítetten tartalmazza a Adam optimalizációs algoritmust, ezért ennek a leprogramozására nincs szükség.

3.3. A tanítási módszer mozgóképek esetében

3.3.1. Naiv megközelítés

Eddigiekben bemutatásra keszült miképpen vihető át a stílus egyik statikus képről a másikra. Mindenki tisztába van azzal, hogy egy videó statikus képek sorozatából tevődik össze, amiket képkockáknak (frame) nevezünk. Ennek megfelelően a naiv megközelítés az lenne, hogy ha adott egy videó, akkor vágjuk azt darabokra, pontosabban statikus képkockákra, majd az összes képkockára alkalmazzuk a stílus átruházási módszert.

A bemeneti videót frame-ekre való bontásához az ffmpeg nyílt forráskodú alkalmazást



3.3. ábra. Tanítási függvény optimalizálása különböző tanítási ráták esetében

használtuk. Az ffmpeg parancssorból futtatható, a következő paraméterek szükségesek, ahhoz, hogy a mozgóképet statikus képek sorozatává alakítsa:

```
ffmpeg -i <input_video_path> -f image2 frame%05d.<ext>
```

A $frame\%05d. < ext >$ egy mintát jelent ami szerint a képkockák elnevezését határozza meg. Az $< ext >$ a képkockák kiterjesztését jelenti.

Miután a stílus átruházása az összes képkockára megtörtént, vesszük az összes stilizált képkockát és felépítjük belölük a kimeneti videót. Ez ugyancsak megoldható az ffmpeg alkalmazással.

```
ffmpeg "-i" frame%05d.<ext> output_name.<ext>
```

A naiv megközelítés működik, a kimenet egy olyan videó lesz, amely tartalmazza a bemeneti stíluskép jellegzetességeit. Ugyanakkor észlelhető, hogy az eredmény nem valami esztétikus. Észlelhető, hogy a képkockák közötti átmenet nem valami folyamatos. Emellett különböző zajokat, úgynevezett artifact-eket észlelhetünk.

3.3.2. A haló inicializálása képkockák esetében

Amint említettük, a háló inicalizálására egy fehér zaj képet használtunk statikus bemeneti képre való stílusátruházáskor. Több képkocka esetében ezzel az a gond, hogy a

képkockák nem fognak egyanabba a lokális minimumba konvergálni tanításkor. Végeredményképp az átmenet egyik képkockáról a másikra nem lesz folyamatos.

Egy megoldás kísérlet erre az lenne, hogy a ha egy képkocka stilizálása történik, a hálót ne fehér zaj képpel inicializáljuk, hanem az előtte levő stilizált képkockával. Ez simább átmenetet fog eredményezni abban az esetben, ha a képkockákon nincs mozgás. Ennek a módszernek a használata szükségszerű viszont nem elégséges ahhoz, hogy szemnek is kellemes eredményt kapjunk.

3.3.3. Optical flow bevezetése

Az optical flow egy mozgóképen megjelenő objektum különböző sebességvektorait jelenti. A képt képkocka közötti optical flow megesztimálásával mérhető az adott objektum valós mozgási sebessége. A kamerához távolabb levő objektumok kisebb sebességvektorokkal rendelkeznek mint azon objektumok amelyek ugyanazon sebességgel mozognak, viszont közelebb vannak a kamerához[34]. Az optical flow meghatározásához a DeepMatching - Deepflow algoritmust használtuk. Ennek a kimenete egy .flo kiterjesztésű fájl ami tartalmazza adott pixelek esetében a horizontális és vertikális elmozdulás vektorokat.

Az optical flow meghatározásával egy mozgóképen meghatározhatók azok a régiók, amik statikusak (adott időn belül az illető régióban nincs változás, úgymond nincs mozgás) valamint azokat a régiókat ahol változás történik. Ami még fontos nekünk meghatározni egy adott mozgó objektum/személy esetében azt az kétdimenziós intervallumot, amelyben mozog. Meghatórozzuk a mozgási határait (lásd:). Ennek a metódusát Sundaran és társai írják le a ... publikációjukban.

Két egymést követő képkocka esetében meghatározható az optical flow előre, tehát az képkockák időrendi sorrendjében, valamint meghatározható visszafele, az időrendi sorrend ellentett irányában. Jelölje $w(i, j)$ az előre történő optical flow-t, valamint $\hat{w}(i, j)$ a visszafele történő optical flow-t. Ekkor felírható a w torzítása a w alapján:

$$\tilde{w}(i, j) = w((i, j) + \hat{w}(i, j)) \tag{3.10}$$

A mozgás néküli zónák meghatározhatóak a következőképpen:

$$|\tilde{w} + \hat{w}|^2 > 0.01(|\tilde{w}|^2 + |\hat{w}|^2) + 0.5 \tag{3.11}$$

A mozgási határok pedig meghatározhatóak:

$$|\nabla\hat{u}| + |\nabla\hat{v}| > 0.01|\tilde{w}|^2 + 0.002 \tag{3.12}$$

A 3.11 és 3.13 képletek alapján alkotható egy olyan mátrix ami tartalmazza a mozgás néküli zonánkat valamint a mozgási határokat. Ezt a mátrixot átalakítjuk olyan módon,

hogy a mozgási határokra 1-est rakunk, a többi érték 0-ás lesz, jelöljük ez a mátixot $c^{(i-1,i)}$-vel az $i-1$ és i-dik képkocka között. Ezt felhasználva felírhatjuk egy újabb veszteség függvényt:

$$L_{temporal}(x, w, c) = \frac{1}{D} \sum_{k=1}^{D} c_k \cdot (x_k - w_k)^2 \tag{3.13}$$

3.4. A rendszer tervezése és kivitelezése

3.4.1. A rendszer interakciós és viselkedési modellje

A rendszer interakciós modellje (3.4. ábra) a lehető legegyszerűbb. Egyetlen aktort határozunk meg, aki maga a felhasználó. A rendszer esetében nem beszélhetünk bármilyen adminisztrációs felületről mivel nem rendelkezik semmiféle háttértárral valamint nem pillanatnyilag nem definiál semmifel hálózati komponenst.

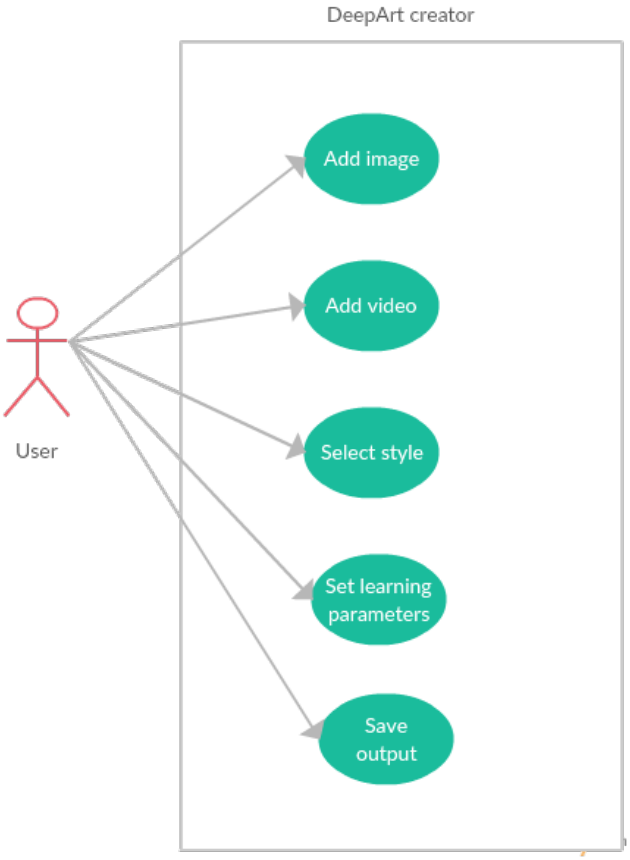
A felhasználónak lehetősége van új kép illetve videó anyag bevitelére. Ez a funkcionalitást egyazon komponens végzi aminek a egyik feladata eldönteni, hogy a kijelölt bemenet kép vagy videó típusú. A felhasználónak lehetősége van a meghatározott kiterjesztésű fájlok közül választani bemenethez.

A felhasználónak lehetősége van a rendszer által biztosított stílusok közül választani. A felhasználó nem határozhat meg új stílust. A rendszer nem biztosít erre adminisztrációs felületet.

A felhasználó meghatározhatja az átruházási folyamat tanítási paramétereit. A rendszer erre biztosít alapértelmezett értékeket amiket a felhasználó változtathat. A rendszer megjegyzi az új paramétereket, ezek új indításkor is megmaradnak. A rendszer az alapértelmezett paramétereket is megjegyzi, ezeket a felhasználó bármikor visszálíthatja az erre kifejlesztett opció használatával.

A felhasználó meghatározhatja az utvonalat ahova a kimenet le lesz mentve. A mentést a rendszer automatikusan el fogja végezni az átruházási folyamat után.

A rendszer interakciós modelljét az 3.1. ábra tartalmazza. A felhasználó kiválaszt egy bemeneti állományt a rendszer által meghatározott kiterjesztések közül(.jpg, .png, .gif, .avi, .mp4), majd kiválaszt egy stílust a megadottak közül. A rendszer annak függvényében, hogy milyen bemenetet adtunk, eldönti, hogy kép vagy mozgóképpel kell dolgoznia. Kép esetén lefuttatja a 3.2-es fejezetben ismertetett taítási algoritmust amely során az átruházza a művészeti kép stílusát. Mozgókép esetén képkockákra bontja azt, majd alkalmazva lesz a 3.3-as fejezet tanítási metódusa. Ha összes képkocka szerkesztve lett, akkor a rendszer felépíti a képkockákból a kimeneti videót. Végezetül a rendszer lementi a megadott helyre a kimeneti állományt.



3.4. ábra. A rendszer viselkedési modellje

3.4.2. A rendszer strukturális modellje

A rendszer a következő komponensekből tevődik össze (3.5. ábra):

- Main Window: az aplikáció fő ablaka, ez az ablak jelenik meg az applikáció indításakor, valamint erről az ablakról lehet kiválasztani a bemeneti adatot és a stílus képet. Az ablak Start gombjának a lenyomásával indítható az átruházási folyamat. Az ablak tartalmaz két beágyazott területet, ahol a bemeneti adat, valamint a végeredmény tekinthető meg.

- Settings Dialog: a főablak menüsorából (Menu Bar) jeleníthető meg. Fő feladata egy grafikus felület biztosítása a különféle beállítások és tanítási paraméterek megadásához. A felhasználó megadhatja az útvonalat ahhoz a mappához, ahova a kimenet lesz tárolva, megadhatja a bemeneti VGG19 háló útvonalat valamint a tanítási paramétereket.

- ProgressBar Dialog: a feladata kimutatni a felhasználó számára, hogy az átruházás milyen státusban van, mennyi van még hátra amig a folyamat befejeződik. Erre azért van szükség, mivel az átruházási folyamat időigényes, ezért fontos a felhasználó tudtára adni, hogy milyen állapotban van a folyamat. A folyamat befejeztekor egy

jelzést küld a Main Window komponensnek, ami ki fogja jelezni a felhasználónak a végeredményt.

- Artistic Video creator: ez képezi az aplikáció magját, ez az a komponens, ami elvégzi maga az átruházási folyamatot. A bemeneti paramétereket/adatokat a Main Window komponenstől kapja. Az átruházási folyamat a videó kártyán fog futni, de ettől függetlenül a komponens jelzéseket fog küldeni a ProgressBar komponensnek.

- Image handler: feladata a bemeneti képek beolvasása és kiíratása a merevlemezre. Beolvasáskor egy preprocesszálás műveletet végez, majd kimenetkor egy poszprocesszálás művelet lesz elvégezve. Fontos megjegyezni, hogy ezt a komponens a processzálások miatt nem ajánlott használni applikáción belüli képkijelzésre, például a Main Window komponens esetében.

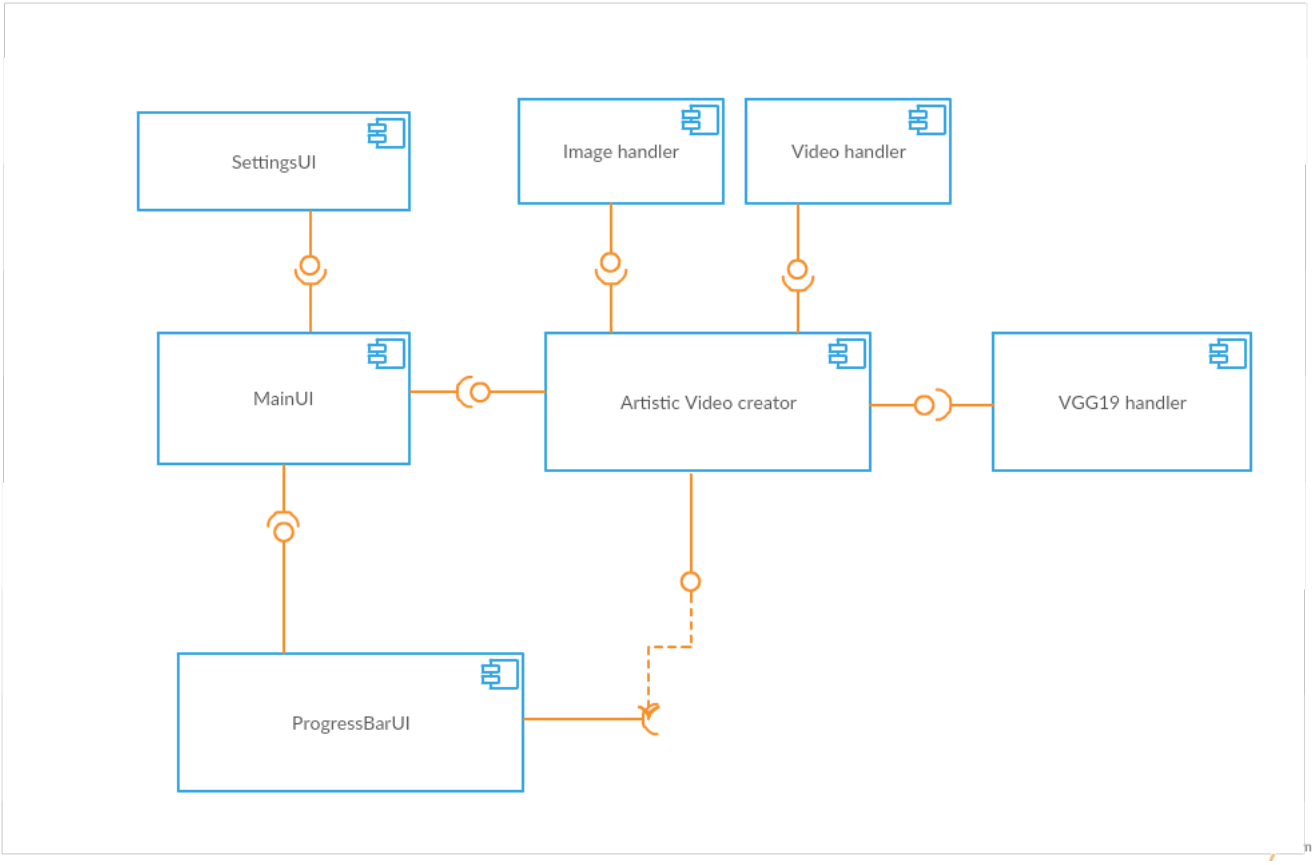
- Video handler: feladata a bemeneti videot képkockákra vágni és a kimeneti képkockákból videót készíteni. A képkockák egy temporális folderbe lesznek elmentve vágás után, ezeknek a beolvasását a Image handler komponens végzi majd.

- VGG19 handler: a komponens feladata ez előre betanított és kimentett VGG19 háló beolvasása és átalakítása egy olyan hálóvá amit a Tensorflow könyvtár fel tud dolgozni majd.

3.4.3. Többszálas megoldás és kommunikáció a komponensek között

A egy bemeneti képre való stílus átruházás időigényes folyamat. Mozgókép esetében ez a folyamat időigénye lineárisan növekedik a képkockák számának szorzatával. Annak érdekében, hogy az alkalmazásunk reszponzív legyen elhagyhatatlan aspektus egy olyan modell kialakítása ami igénybe veszi a modern processzorok többszálas működési tulajdonságát. Az Artistic video creator komponens működése során ugyan igénybe veszi a videó kártya számítási kapacítását, de ez semmiképp sem jelenti azt, hogy a futási idő elhanyagolható. Ugyanakkor fontos kiemelni azt is, azon függvényhívások sorozata ami a videókártyán fog elvégződni, szinkron módon történik. Valójában a programunk meghívja az adott függvényt majd addig várakozik, amig megoldás nem érkezett erre.

Alkalmazásunk esetében megkülönböztetünk egy főszálat valamint egy mellékszálat (3.6. ábra). A főszálon fog futni a Main Window és a köréje csoportosulú felhasználói felülettel rendelkező komponensek, mint például a Settings vagy a ProgressBar. Az alkalmazás indításakor csak a főszál indul el ami létrehozza a Main Window és Settings komponenseket. Ezt követően, ha a felhasználó megadja a bemeneti állományt, létrejön a ProgressBar komponens. A ProgressBar azért jön létre csak ebben a pillanatban, mivel

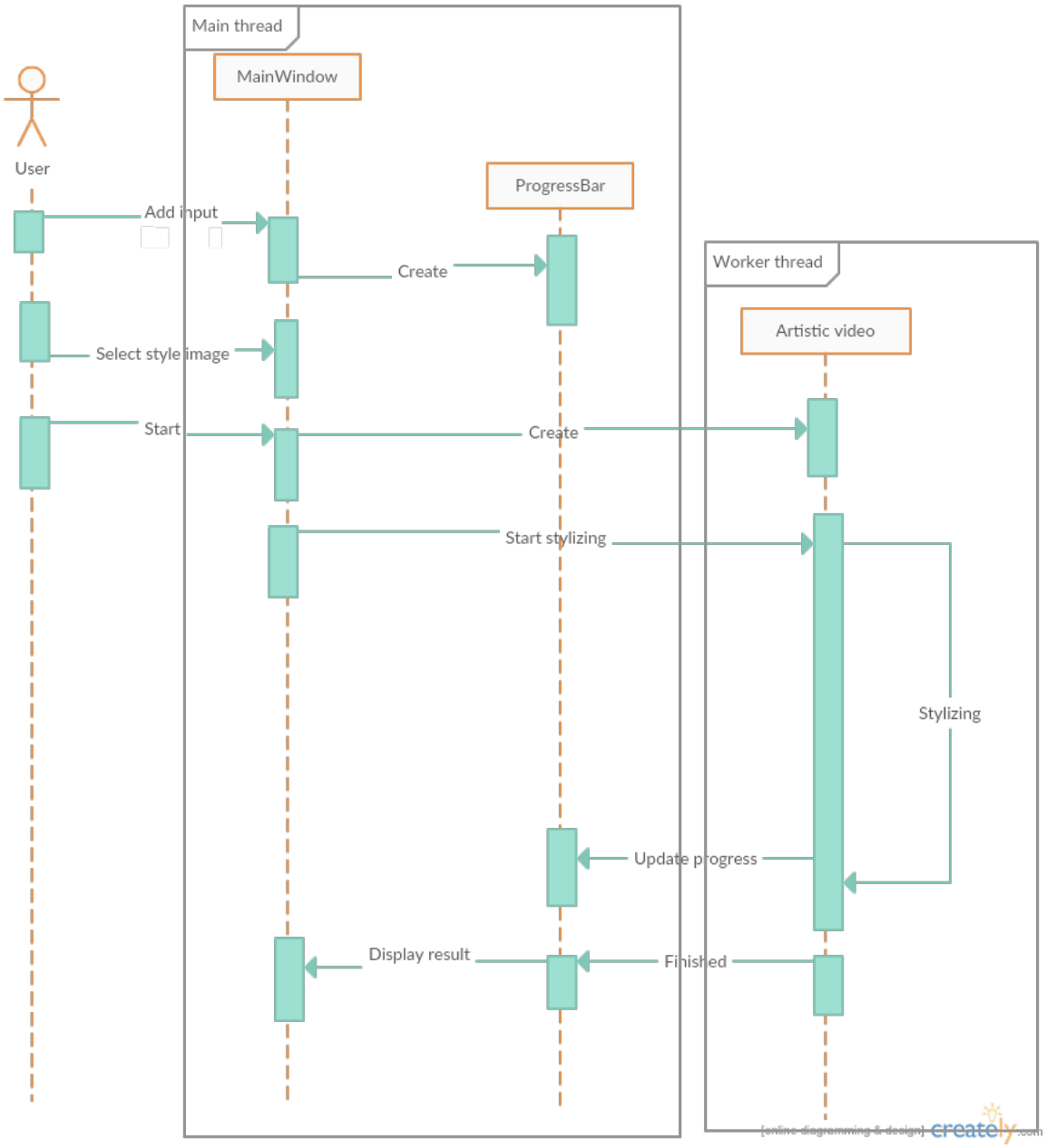


3.5. ábra. A rendszert alkotó komponensek

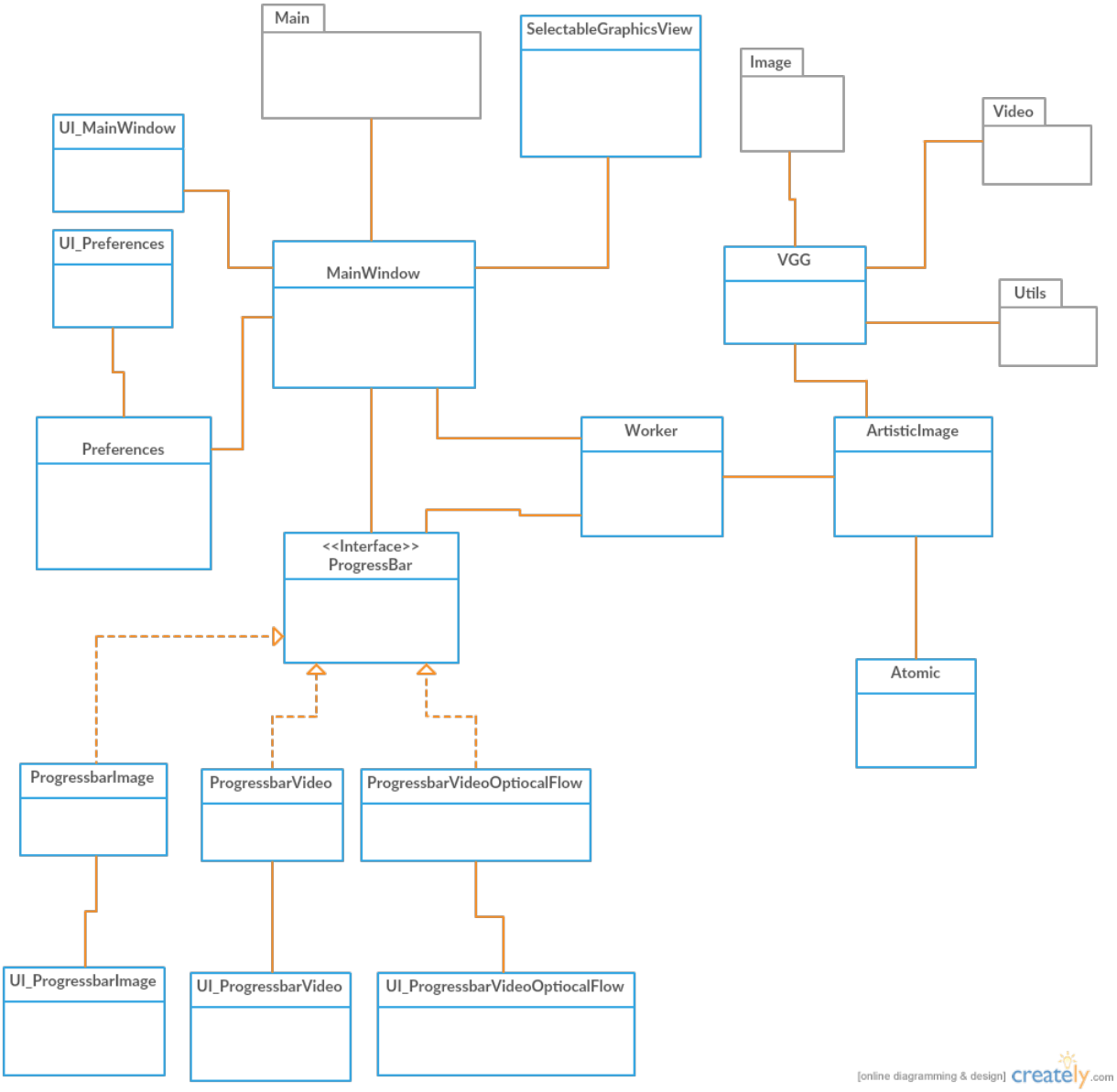
a rendszernek el kell döntenie a bemenet típusát és ennek függvényében egy factory modell segítségével dönti el, hogy milyen típusú komponens jön létre. Ebben a pillanatban azonban a Progressbar komponens még nem látható a felhasználó számára.

Ha felhasználó úgy dönt, hogy a megfelelő bemenetet választotta ki, elindíthatja az átruházási folyamatot. Ekkor egy külön szál jön létre (Worker thread), ami átveszi a bemeneti adatokat és elkezdi a folyamatot. A terhelés ebben a pillanatban átkerül a másodlagos szálra, ennek eredményeképp a főszál nem fog blokkolódni. Így az felhasználói felület aktív marad a felhasználó számára és ki fogja tudni szolgálni annak kéréseit. A munkafolyamat közben az Artistic Video creator komponens kapcsolatban áll a ProgressBar komponenssel és jelzéseket küld a munkafolyamat státusáról így ez ki tudja jelezni, hogy mennyi munka van még hátra. A munkafolyamat bármikor leállítható még azelőtt, hogy az átruházás befejeződött volna. Ilyenkor a ProgressBar üzenetet küld az Artistic Video creator komponensnek, ami beállít egy saját leállításra szánt flag-et. A folyamat két iteráció között állítható le, ha az illető flag be volt állítva. Miután a munkafolyamat sikeresen leállt, egy jelzés fog érkezni a ProgressBar komponenshez ami majd nyugtázni fogja. Ha a munkafolyamat bevégződött, vagy a felhasználó által megszakításra került, a másodlagos szál el fog halni, az ezáltal igényelt erőforrások pedig fel fognak szabadulni.

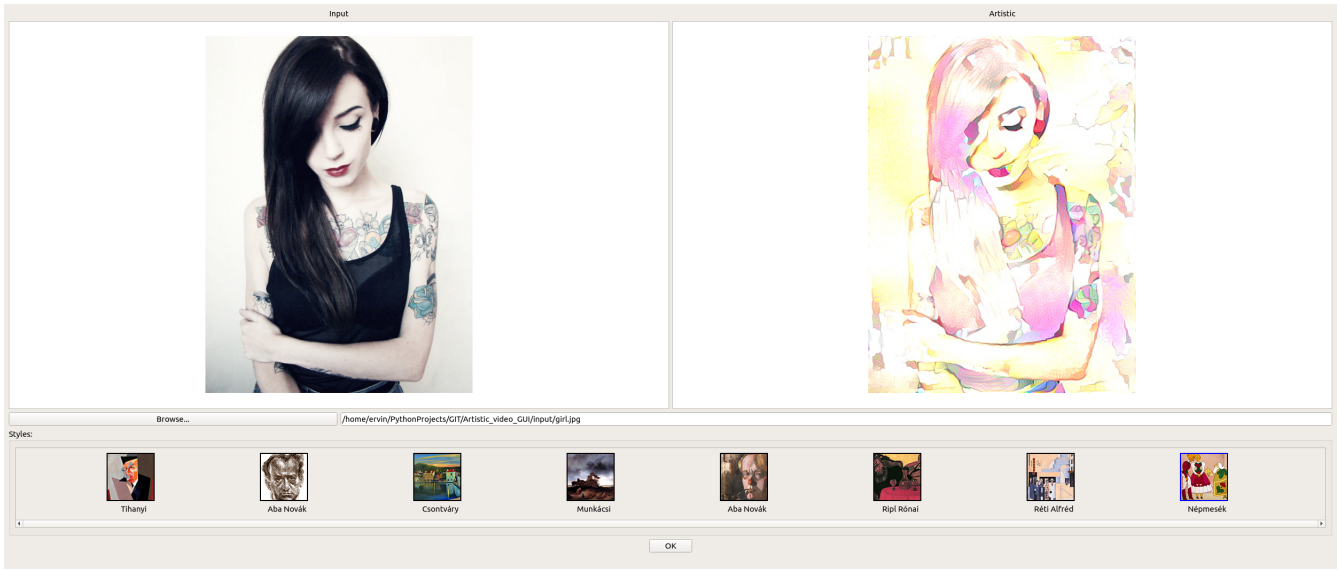
3.4.4. Az implementáció során felépített objektumok bemutatása



3.6. ábra. Átruházás szekvencia diagrammja



3.7. ábra. A rendszer osztálydiagrammja



3.8. ábra. A rendszer felhasználói felülete

4. fejezet

A rendszer tesztelése

testing

5. fejezet

Összefoglaló

összefoglaló

Ábrák jegyzéke

Irodalomjegyzék

[1] Krizhevsky, A., Sutskever, I., and Hinton, G. E. ImageNet classification with deep convolutional neural networks (2012)

[2] Zeiler, M. D. and Fergus, R. Visualizing and understanding convolutional networks (2013)

[3] `https://en.wikipedia.org/wiki/CUDA` (2017.04.24)

[4] `http://caffe.berkeleyvision.org` (2017.04.24)

[5] `https://keras.io/` (2017.04.24)

[6] `http://deeplearning.net/software/theano/` (2017.04.24)

[7] `https://www.tensorflow.org/` (2017.04.24)

[8] `http://torch.ch/` (2017.04.24)

[9] `https://computerstories.net/microsoft-computer-outperforms-human-image-recogniti` (2017.04.29)

[10] Kevin Alfianto, Mei-Chen Yeh, Kai-Lung Hua - Artist-based Classification via Deep Learning with Multi-scale Weighted Pooling (2016)

[11] Rosenblatt F. - The Perceptron: A Probabilistic Model For Information Storage And Organization In The Brain (1958)

[12] Werbos, P.J. - Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences (1975)

[13] LeCun, Yann, Léon Bottou, Yoshua Bengio, Patrick Haffner - Gradient-based learning applied to document recognition (1998)

[14] Dave Steinkraus, Patrice Simard. Ian Buck - Using GPUs for Machine Learning Algorithms (2005)

[15] Gatys, L. A., Ecker, A. S., Bethge - A neural algorithm of artistic style (2015)

[16] Yaroslav Nikulin, Roman Novak - Exploring the Neural Algorithm of Artistic Style (2016)

[17] Justin Johnson, Alexandre Alahi, Li Fei-Fei - Perceptual Losses for Real-Time Style Transfer and Super-Resolution

[18] Ulyanov, D., Lebedev, V., Vedaldi, A., and Lempitsky - Texture networks: Feed-forward synthesis of textures and stylized images

[19] Ulyanov, D., Lebedev, V., Vedaldi, A., and Lempitsky - Instance Normalization: The Missing Ingredient for Fast Stylization

[20] `https://en.wikipedia.org/wiki/Prisma_(app)` (2017.04.29)

[21] Manuel Ruder, Alexey Dosovitskiy, Thomas Brox - Artistic style transfer for videos (2016)

[22] asd

[23] `https://www.python.org/` (2017.04.30)

[24] `http://www.numpy.org/` (2017.04.30)

[25] `https://www.riverbankcomputing.com/software/pyqt/intro` (2017.04.30)

[26] `http://opencv.org/` (2017.04.30)

[27] Gatys, L. A., Ecker, A. S., Bethge, M. Texture synthesis and the controlled generation of natural stimuli using convolutional neural networks (2015)

[28] `https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html` (2017.05.01)

[29] Simonyan, K., Zisserman, A. - Very Deep Convolutional Networks for Large-Scale ImageRecognition (2015)

[30] `http://mathworld.wolfram.com/GramMatrix.html` (2017.05.02)

[31] `https://en.wikipedia.org/wiki/Total_variation_denoising` (2017.05.03)

[32] Kingma D. P., Lei Ba, J. - ADAM: A method for stochastic optimization (2015)

[33] `http://caffe.berkeleyvision.org/tutorial/solver.html` (2017.05.14)

[34] `https://www.mathworks.com/discovery/optical-flow.html` (2017.05.18)