

# Painting Style Transfer

Adam Gregor, xgrego18  
Zdeněk Jelínek, xjelin47

11. května 2021

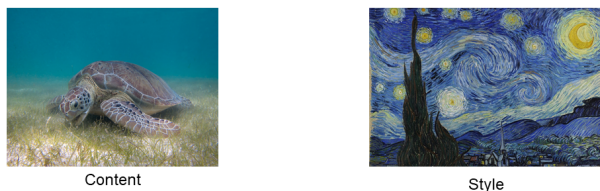
## Úvod

Se zvyšováním výpočetního výkonu roste množina úloh, které byly dříve považovány za řešitelné výhradně člověkem, avšak u kterých se později ukázalo, že přeci jen existují metody (zpravidla založené na strojovém učení), pomocí nichž může stroj produkovat výsledky, které jsou při porovnání s těmi lidskými srovnatelné, či dokonce lepší. Příkladem takové úlohy může být hra Go, nebo malba obrazů.

Právě druhé uvedené disciplíně, v oboru strojového učení nazývané Painting Style Transfer, se bude tento text věnovat. Nejdříve uvedeme definici úlohy, poté rozebereme její implementaci v podání autorů textu a na závěr uvedeme experimenty a jejich vyhodnocení spojené s touto úlohou. **TOTO MOŽNÁ SE TO ZMĚNÍ.**

## Definice úlohy

Úloha Painting Style Transfer spočívá v kombinaci dvou vstupních obrázků do jednoho obrázku výstupního. Vstupy jsou obrázek *content*, na němž je obsah obrázku, který má být na výstupu a obrázek *style*, mající výtvarný styl, který má být do výstupu předán. Příklad vstupů a jejich kombinace je uveden na obrázku 1.



Obrázek 1: Příklad Paint Style Transfer. Vstupním obrázkem nesoucí obsah byla želva, stylem byla malba Starry Night. Výsledkem je obrázek želvy, jako by byla namalovaná ve stylu Starry Night.

neuronových sítí, a zároveň článkem, z něž vycházeli autoři tohoto textu je A Neural Algorithm of Artistic Style od Leona A. Gatysa et al.<sup>1</sup> Gatys ve svém článku používal předtrénovanou síť VGG-19, v níž pozměnil max-pooling vrstvy za avg-pooling. Vstupem takovéto sítě pak byl obrázek, zpočátku bílý šum, který se během jednotlivých iterací aktualizoval na základě hodnot Loss funkce. Při tvorbě výstupního obrázku tedy nejsou aktualizovány parametry neuronové sítě, ale jednotlivé pixely toho obrázku a to tak, aby byla minimalizována hodnota Loss funkce. Označíme-li průběžně generovaný výstupní obrázek  $\vec{x}$ , *content*  $\vec{p}$  a *style*  $\vec{a}$ , má Loss funkce  $L_{total}$  tvar

$$L_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha L_{content}(\vec{p}, \vec{x}) + \beta L_{style}(\vec{a}, \vec{x})$$

kde  $L_{content}$  a  $L_{style}$  jsou funkce počítající, jak se generovaný obrázek  $\vec{x}$  liší obsahem od obsahového obrázku *content* resp. stylem od stylového obrázku *style* a  $\alpha$  a  $\beta$  jsou váhy těchto hodnot. V článku se používaly poměry  $\alpha/\beta = 10^{-3}$  nebo  $10^{-4}$ .

Dílčí Loss funkce  $L_{content}$ , měřící rozdíl obsahu mezi  $\vec{x}$  a  $\vec{p}$  porovnává aktivační mapy na  $l$ -té konvoluční vrstvě. Má-li tato vrstva  $N_l$  filtrů a aktivační mapy mají  $M_l$  pixelů, budou porovnávány matice  $Q \in R^{N_l \times M_l}$ . Jejich řádky budou představovat jednotlivé konvoluční filtry vrstvy a sloupce aktivační mapy těchto filtrů, převedené na vektor. Nyní, máme-li matice  $F$  a  $P$  obsahující tyto matice obrázků  $\vec{x}$  a  $\vec{p}$ , je  $L_{content}$  spočtena jako

$$L_{content} = \frac{1}{2} \sum_{i,j} (F_{ij} - P_{ij})^2$$

kde  $i$  je index filtru a  $j$  je index pixelu. V původním článku byla pro výpočet použita vrstva Conv4\_2.

$L_{style}$ , měřící rozdíl stylu mezi  $\vec{x}$  a  $\vec{a}$  se oproti  $L_{content}$  nepočítá pouze nad jednou vrstvou ale nad několika. Je to z toho důvodu, že jinak hluboké vrstvy detekují jinak komplexní prvky stylu (např. mělké vrstvy budou detekovat pouze barvy, zatímco hlubší vrstvy mohou detekovat komplexnější entity jako tahy štětce) a pro kvalitní výsledek je podle Gatysa potřeba kombinace několika různě hlubokých vrstev.  $L_{style}$  je počítána pomocí gram matice. Ta počítá korelaci mezi filtry a míru jejich aktivace na jedné vrstvě. To si lze představit na příkladu, kdy máme dva filtry, jeden detekující modrou barvu a druhý detekující spirály. Pokud by tyto filtry měly vysokou korelaci, znamenalo by to, že bude každá spirála v obraze modrá. Samotná gram matice je vypočtena opět pomocí matice  $Q$ . Pro

$l$ -tou vrstvou tedy  $Q \in R^{N_l \times M_l}$ .  $Q$  je poté vynásobena sebou samou v transponované formě a tedy výsledná gram matice  $G \in R^{N_l \times N_l}$ .

$$G = Q \times Q^T$$

V  $l$ -té vrstvě je pak odchylka stylů  $\vec{x}$  s gram maticí  $G^l$  a style s gram maticí  $A^l$  spočtena jako

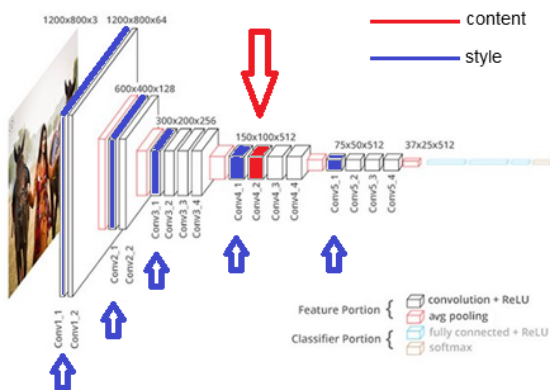
$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - A_{ij}^l)^2$$

Napříč všemi vrstvami je poté  $L_{style}$  spočtena jako

$$L_{style} = \sum_{l=0}^L w_l E_l$$

kde  $E_l$  jsou dílčí chyby z různých vrstev a  $w_l$  jsou váhy těchto chyb. Gatys používal rovnoměrně rozdělené váhy, které v součtu dají hodnotu 1. V článku se používají chyby z vrstev Conv1\_1, Conv2\_1, Conv3\_1, Conv4\_1 a Conv5\_1.

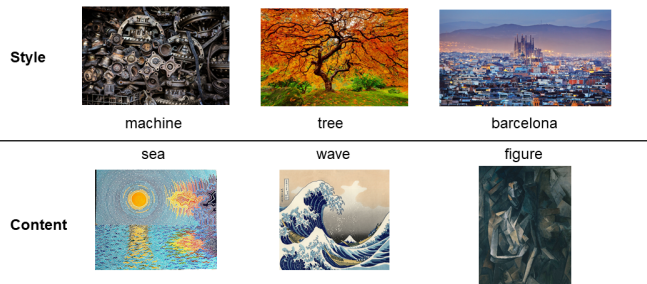
Na obrázku 2 je vyobrazena používaná síť VGG-19 spolu s místy, z nichž se počítá chyba.



Obrázek 2: Znárodnění použité architektury. Jsou zde také vyznačena místa, z nichž jsou počítány chyby. Původní obrázek převzat z <https://towardsdatascience.com/art-with-ai-turning-photographs-into-artwork-with-neural-style-transfer-8144ece44bed>

## Experimenty

V rámci experimentů s naší implementací jsme se rozhodli porovnat rychlost konvergence různých optimalizátorů, vyzkoušet vliv výběru odlišných vrstev pro výpočet  $L_{style}$  a vyzkoušet vliv změny počátečního obrázku z bílého šumu na content obrázek. Hodnoty  $\alpha$  a  $\beta$  byly nastaveny na  $\alpha = 10^{-1}$   $\beta = 10^6$  a tedy poměr  $\alpha/\beta = 10^{-7}$ . Tento poměr byl vybrán z toho důvodu, že produkoval  $L_{total}$  hodnoty blízko 0 a dle autorů na něm šly dobře pozorovat rozdíly v chování během jednotlivých experimentů. V rámci experimentů používáme 3 style a 3 content obrázky, které jsou vyobrazeny na obrázku 3. Tyto obrázky používáme v content-style párech machinery-sea, tree-wave a barcelona-figure.

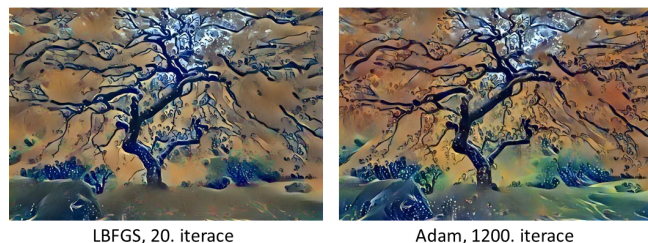


Obrázek 3: Obrázky použité v rámci experimentů. Jejich zdroje jsou uvedeny na gitu projektu.

## Test optimalizátorů

V rámci prvotní implementace, která byla vytvořena pomocí frameworku TensorFlow jsme zkoušeli téměř všechny její základní optimalizátory<sup>2</sup> z hlediska rychlosti konvergence a shledali jsme, že nejrychleji konverguje optimalizátor Adam. Když jsme však chtěli použít optimalizátor LBFGS, avšak v TensorFlow se nám jej nedařilo implementovat. To byl hlavní důvod migrace naší implementace do frameworku PyTorch.

V rámci experimentu jsme porovnávali optimalizátory Adam a LBFGS. Learning rate Adama byl nastaven na 0,01, pro LBFGS byl nastaven na 0,8. Výsledky experimentů zobrazují grafy na obrázku 5.



Obrázek 4: Porovnání výsledků optimalizátorů.

Z grafů lze vyčíst, že rychlost konvergence při použití optimalizátoru LBFGS je řádově rychlejší, než při použití Adama. Už po 20. iteraci dokázal LBFGS vygenerovat přijatelný výsledek, kdežto Adamovi trvalo vygenerování vizuálně zajímavějšího výsledku okolo 1000 iterací, viz 4.

Dalším zajímavým jevem je, že v případě optimalizátoru Adam občas docházelo k nenadálému nárůstu Loss hodnoty. Pro tento úkaz nemají autoři vysvětlení.

## Test stylových vrstev

Dalším experimentem, který byl vykonán bylo testování, nakolik závisí rychlost konvergence Loss funkce a vizuální kvalita výstupu na výběru vrstev, na nichž se počítá  $L_{style}$ . Při experimentu byl ponechán počet iterací výpočtu. Style loss byl pro účely experimentu počítán kromě původních vrstev, navrhovaných Gatysem počítán z Conv1\_1, Conv1\_2, Conv2\_1,

<sup>2</sup>[https://www.tensorflow.org/api\\_docs/python/tf/keras/optimizers](https://www.tensorflow.org/api_docs/python/tf/keras/optimizers)

Conv2\_2, a Conv3\_1. Jejich srovnání je z hlediska konvergence Loss funkce zobrazeno na obrázku ?? a porovnání kvality výstupu je na obrázku 7. Pro porovnání byly opět použity stejné tři dvojice obrázků.

Z grafu vývoje Loss funkce obrazů, jejichž styl byl generován z mělčích vrstev, lze vypožorovat, že ve všech případech skončila hodnota Loss po poslední iteraci níž než v obrazech generovaných za pomoci původních vrstev. Dle autorů by to mohlo být způsobeno jednoduššími entitami, které jsou v těchto vrstvách detekovány. Dále se u vývoje Loss hodnoty pro nově vybrané vrstvy opět objevují nenadálé výkyvy.

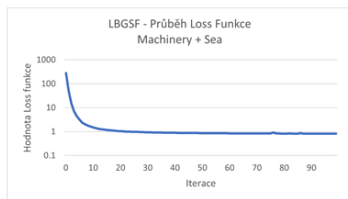
Z vizuálního hlediska mají obrazy generované mělčími vrstvami barevně výraznější schéma, které přejali ze stylového obrázku. Při použití mělčích vrstev se propaguje spíše barva (low-level features), než komplexnější vzorce. Např. na dvojici *tree-wave* si lze všimnout, že při použití originálních vrstev se projevíly vzorce vln ve větší míře (porovnejme levé spodní části obrázků).

## Závěr

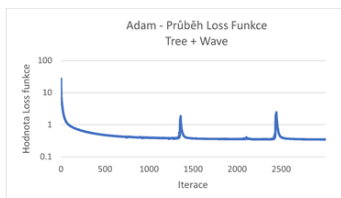
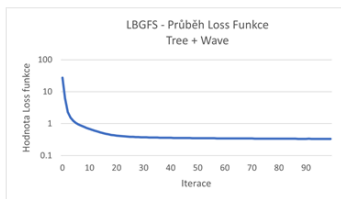
V tomto díle jsme realizovali řešení úlohy Paint Style Transfer navržené Gatysem. Implementace byla řešena v jazyce Python3, s využitím frameworku PyTorch.

V rámci experimentů jsme porovnali efektivitu optimalizátorů, z nichž nejlépe vyšel LBFGS. Dále jsme navrhli alternativní volbu stylových vrstev, kde jsme vybrali vrstvy mělčí. Alternativní vrstvy dosahovaly rozdílných vizuálních výsledků. Nelze jednoznačně určit, která volba je vhodnější, neboť ani jedna série výsledků není signifikantně horší a vkus je subjektivní.

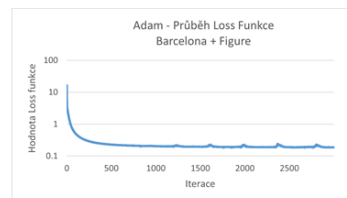
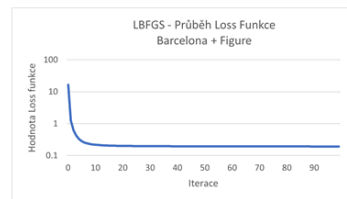
Machinery + Sea



Tree + Wave

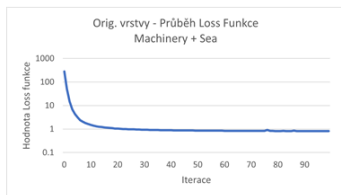


Barcelona + Figure

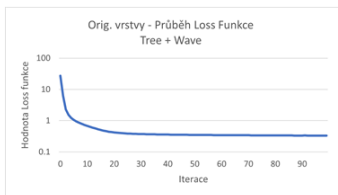


Obrázek 5: Průběh Loss funkce při použití různých optimalizátorů. První řádek reprezentuje použití optimalizátoru LBFGS, druhý použití Adama.

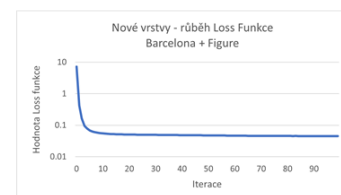
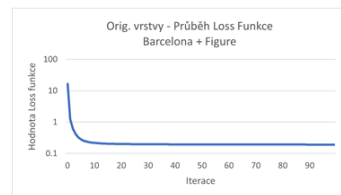
Machinery + Sea



Tree + Wave

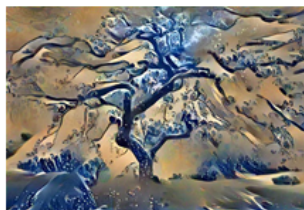


Barcelona + Figure



Obrázek 6: Průběh Loss funkce při počítání  $L_{style}$  z různých vrstev. Nahoře Conv1\_1, Conv2\_1, Conv3\_1, Conv4\_1 a Conv5\_1. Dole Conv1\_1, Conv1\_2, Conv2\_1, Conv2\_2, a Conv3\_1.

Originální  
vrstvy



Nové  
vrstvy



Obrázek 7: Obrázky generované pomocí původních vrstev (nahore) a pomocí nově zvolených stylových vrstev (dole).