Painting Style Transfer

Adam Gregor, xgrego18
Zdeněk Jelínek, xjelin47

29. května 2021

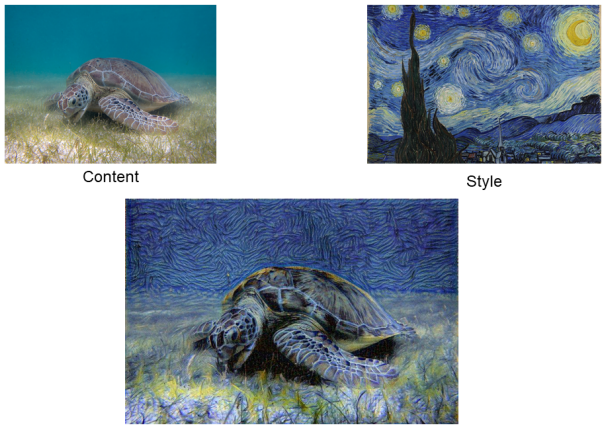
Úvod

Se zvyšováním výpočetního výkonu roste množství 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 tvorba obrazů.

Právě druhé uvedené disciplíně, v oboru strojového učení nazývané Painting Style Transfer, se bude tento text věnovat. Nejdříve uvedeme definici úlohy, následně popíšeme datovou sadu a poté uvedeme experimenty a jejich vyhodnocení.

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.

Jedním z prvních článků, věnující se této úloze za použití neuronových sítí, a zároveň článkem, z nějž vycházeli autoři tohoto textu je A Neural Algorithm of Artistic Style od Leona A. Gatyse et al. [1] Gatys ve svém článku používá 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 upravoval 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 $\overrightarrow{x}$, *content* $\overrightarrow{p}$ a *style* $\overrightarrow{a}$, má Loss funkce L_{total} tvar

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

kde $L_{content}$ a L_{style} jsou funkce počítající, jak se generovaný obrázek $\overrightarrow{x}$ liší obsahem od obsahového obrázku *content* resp. stylem od stylového obrázku *style* a α a β 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 $\overrightarrow{x}$ a $\overrightarrow{p}$ porovnává aktivační mapy na l–té konvoluční vrstvě. Má-li tato vrstva N_l filtrů a aktivační mapy mají M_l hodnot, budou porovnávané 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, mámeli matice F a P obsahující tyto matice obrázků $\overrightarrow{x}$ a $\overrightarrow{p}$, 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 $\overrightarrow{x}$ a $\overrightarrow{a}$ 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. mělčí vrstvy budou detekovat puze barvy, zatímco hlubší vrstvy mohou detekovat komplexnější entity jako tahy štětce) a pro kvalitní výsledek je podle Gatyse potřeba kombinace několika různě hlubokých vrstev. L_{style} je počítána pomocí gram matic. 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 obrazu modrá. Samotná gram matice je opět vypočtena pomocí matice Q. Pro l–tou vrstvu tedy $Q \in R^{N_l \times M_l}$. Q je poté vynásobena samou sebou v transponované formě a tedy výsledná gram matice $G \in R^{N_l \times N_l}$.

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

[1] https://arxiv.org/pdf/1508.06576.pdf

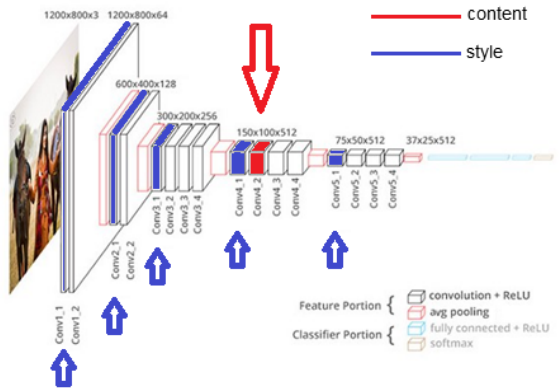
V l–té vrstvě je pak odchylka stylů $\overrightarrow{x}$ s gram maticí G^l a *style* s gram maticí A^l spočtena jako

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

Napříč všemi vrstvami je poté L_{style} spočtena jako

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

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: Znázornění použité architektury. Jsou zde také vyznačena místa, z nichž je počítána loss funkce. Původní obrázek převzat z https://towardsdatascience.com/art-with-ai-turning-photographs-into-artwork-with-neural-style-transfer-8144ece44bed

Datová sada

Naše datová sada se dělí do dvou sekcí. První je sekce *style* obrázků, obsahující díla známých autorů, napříč styly (například Wassily Kandinsky – Composition VII, nebo Vincent van Gogh – Starry Night), které byly vybrány tak, aby zastupovaly co možná nejširší záběr výtvarného projevu. V této části datové sady se nalézá 6 děl.

Druhá část sady obsahuje 5 *content* obrázků. Tentokrát se jedná o fotografie různých objektů. Opět jsme se snažili o co možná nejpestřejší výběr. V sadě se tak nalézá například hromada kovových součástí strojů, nebo fotografie rozvětveného stromu.

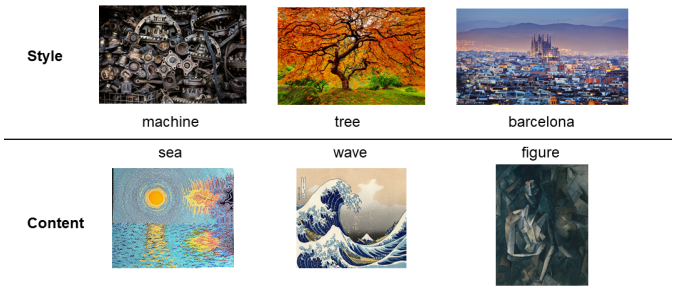
Celá datová sada je dostupná na GitHubu tohoto projektu[2]. Zde jsou také uvedeny zdroje, z nichž bylo čerpáno.

[2]https://github.com/Zjelin/paint_style_transfer

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 α a β 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 opticky nejlepší výsledky.

V tomto textu uvádíme výsledky na třech *content-style* dvojicích obrázků (viz obrázek 3), ačkoliv všechny experimenty byly provedeny na celém datasetu. Všechny ostatní výsledky jsou dostupné v repozitáři projektu.

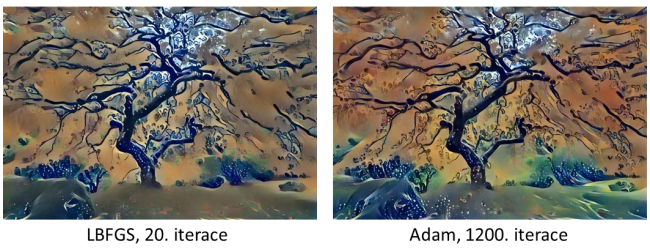


Obrázek 3: Vstupní obrázky. *Machinery-sea*, *tree-wave* a *barcelona-figure*

Test optimalizátorů

V rámci prvotní implementace, která byla vytvořena pomocí frameworku TensorFlow jsme zkoušeli téměř všechny její základní optimalizátory[3] z hlediska rychlosti konvergence a shledali jsme, že nejrychleji konverguje optimalizátor Adam. Když jsme však chtěli použít optimalizátor LBFGS, nedařilo se nám jej v TensorFlow použít. 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 6.



Obrázek 4: Porovnání výsledků optimalizátorů.

Z grafů lze vyčíst, že rychlost konvergence při použití optimalizátoru LBFGS je řádově větší, než při použití Adama. Již po 20. iteraci dokázal LBFGS vygenerovat přijatelný výsledek,

[3]https://www.tensorflow.org/api_docs/python/tf/keras/optimizers

kdežto Adamovi trvalo vygenerování vizuálně zajímavějšího výsledku okolo 1000 iterací, viz obrázek 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 počínán kromě původních vrstev, navrhovaných Gatysem, počítán z Conv1_1, Conv1_2, Conv2_1, Conv2_2, a Conv3_1. Jejich srovnání je z hlediska konvergence Loss funkce zobrazeno na obrázku 7 a porovnání kvality výstupu je na obrázku 8. Pro porovnání byly opět použity stejné tři dvojice obrázků.

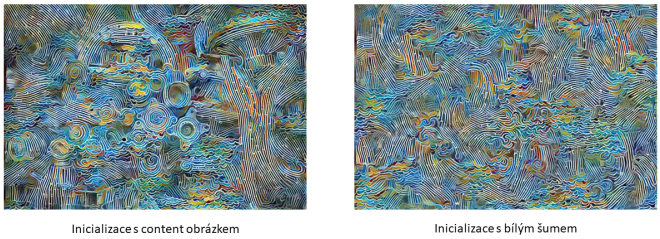
Z grafu vývoje Loss funkce obrazů, jejichž styl byl generován z mělčích vrstev, lze vypozorovat, že ve všech případech skončila hodnota Loss po poslední iteraci níž než v obrazech generovaných za pomocí původních vrstev. Dle autorů by to mohlo být způsobeno jednoduššími entitami, které jsou v těchto 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 například barva (low-level features), než komplexnější vzorce. Např. na dvojici *tree-wave* (obrázek 8) si lze všimnout, že při použití originálních vrstev se projevily vzorce vln ve větší míře (porovnejme levé spodní části obrázků).

Test počátečního generovaného obrázku

Výsledný obrázek, který je napříč iteracemi výpočtu upravován tak, aby minimalizoval Loss hodnotu je v původním článku od Gatyse inicializován na bílý šum. Během optimalizace se pak snižuje jak $L_{content}$, tak L_{style}, když z náhodného šumu vzniká výsledný obrázek. Naše řešení však za výchozí výstup používá *content* obrázek. Tato idea na změnu výchozího stavu byla přejata ze serveru medium.com[4], podle nějž byla tvořena původní implementace pomocí frameworku TensorFlow. V tomto experimentu si klademe za cíl určit, zda má tato změna vliv na kvalitu výstupu/rychlost konvergence.

Pro generování výsledků s použitím bílého šumu byl použit optimalizátor LBFGS při 100 iteracích. Příklad porovnání výstupů, je na obr. 5. Z výsledků lze usoudit, že inicializace s *content* obrázkem je výhodná, neboť konverguje rychleji. Toto je podle autorů dáno tím, že content je již ve výstupu přítomen a není tak třeba jej tam zanášet (tj. snižovat $L_{content}$ hodnotu).



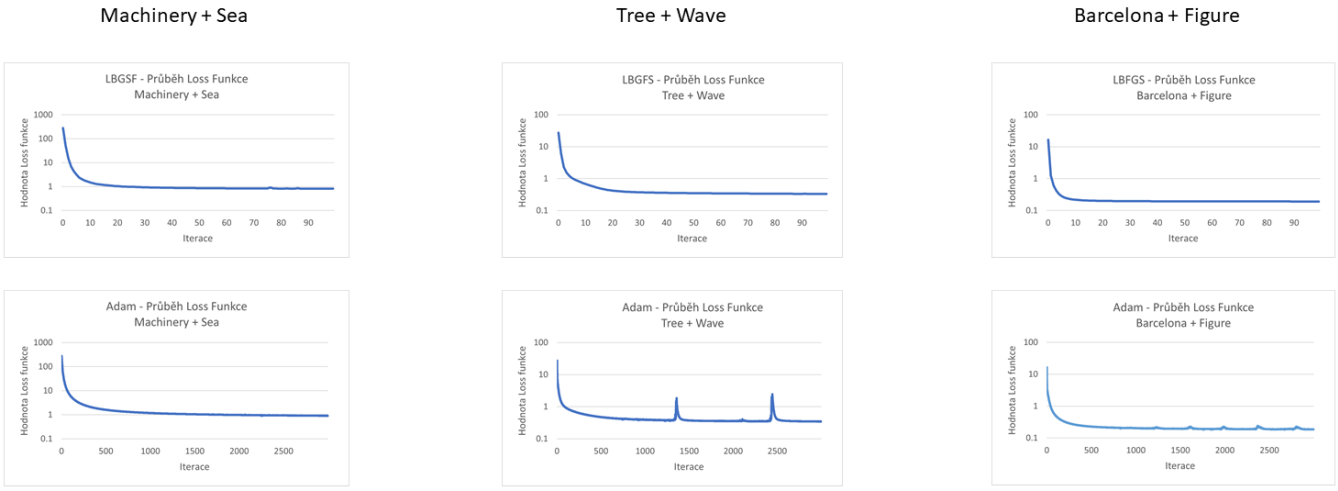
Obrázek 5: Porovnání výsledků s různými inicializačními obrázky.

Závěr

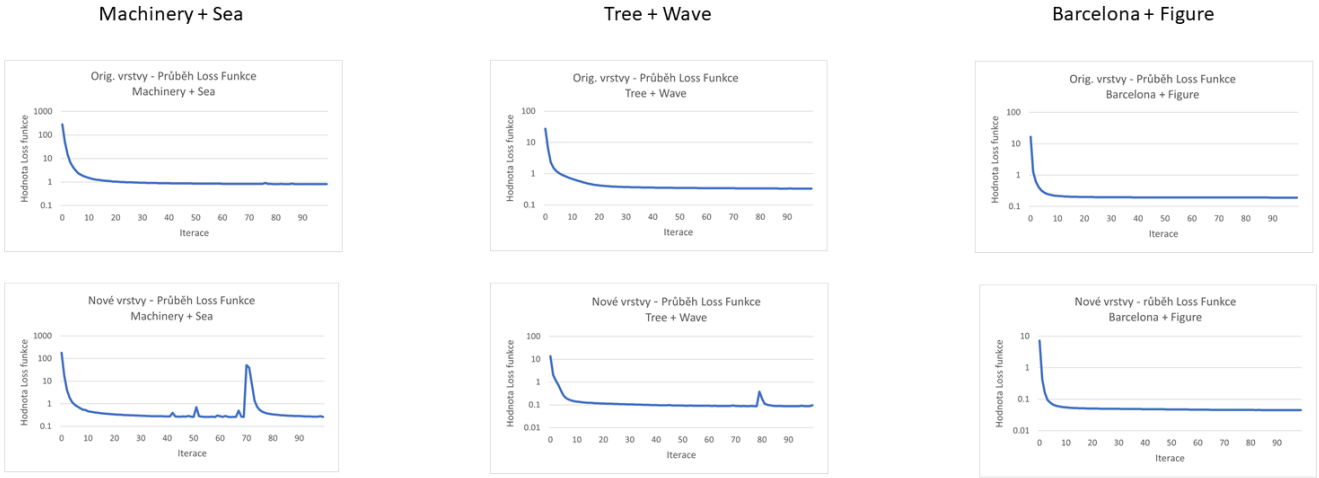
V tomto projektu jsme realizovali řešení úlohy Paint Style Transfer navržené Gatysem. Implementace byla řešena v jazyce Python3, s využitím frameworku PyTorch.

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. Prokázali jsme, že inicializace výstupního obrázku je výhodnější za použití *content* obrázku, namísto bílého šumu používaného Gatysem v původním článku.

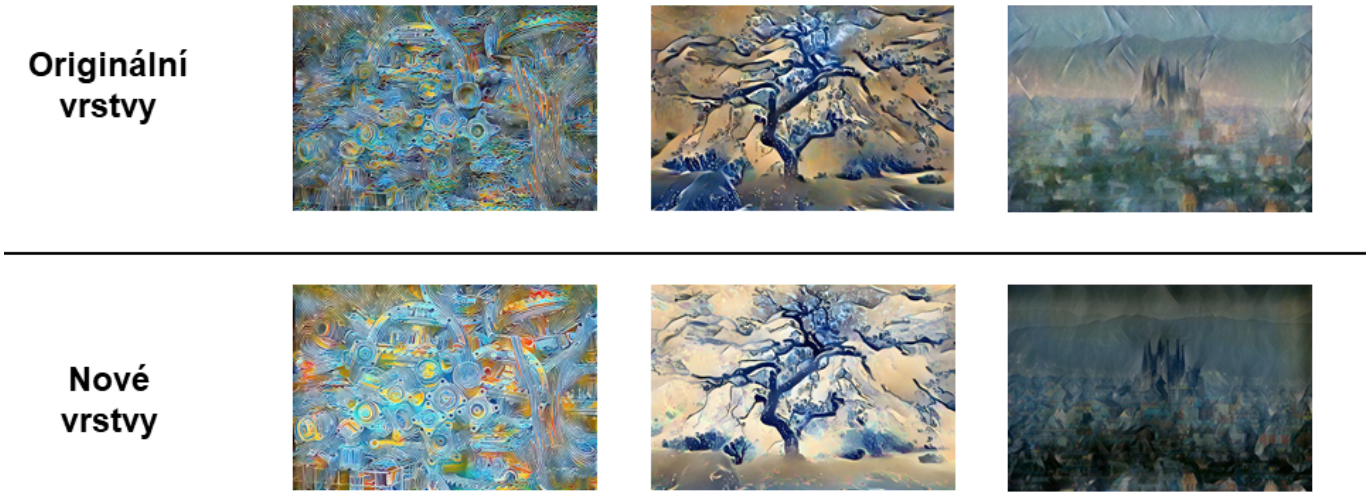
[4] https://medium.com/tensorflow/neuralstyletransfercreatingartwithdeep-learningusingtfkerasandeagerexecution7d541ac31398



Obrázek 6: Průběh Loss funkce při použití různých optimalizátorů. První řádek reprezentuje použití optimalizátoru LBFGS, druhý použití Adama.



Obrázek 7: 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.



Obrázek 8: Obrázky generované pomocí původních vrstev (nahoře) a pomocí nově zvolených stylových vrstev (dole).