Vysoké učení technické v Brně
Fakulta informačních technologií

Konvoluční neuronové sítě – Semestrální projekt

Obarvování obrazu

Tomáš Venkrbec
xvenkr01@vutbr.cz

Kateřina Fořtová
xforto00@vutbr.cz

Jiří Dvořák
xdvora2u@vutbr.cz

30. května 2021

1 Popis problému

Zadáním projektu bylo vytvoření modelu schopného provádět automatické obarvování obrazu. Tím rozumíme úlohu přidání barev do šedotónového obrazu bez nutných zásahů uživatele. Tato úloha je ze své podstaty obecně nejednoznačná a konkrétní výstup tak záleží na dodatečných informacích, kontextu či vzorových datech. Naším cílem bylo vyvinout model využívající generativních neuronových sítí (GAN), který dokáže obarvit snímky s rozličnými typy objektů, jak snímky z námi zvoleného datasetu, tak snímky nahrané uživatelem.

2 Analýza existujícího řešení – DeOldify

DeOldify [1] je open-source model hlubokého učení, jehož první verzi implementoval Jason Antic již mezi lety 2018 a 2019. Cílem bylo vytvoření modelu pro obarvování historických šedotónových snímků. Model vychází převážně z modelu SAGAN, který oproti vanilla verzi GAN dokáže při trénování zahrnout kontext z celé plochy obrázku, nikoliv pouze z lokálního okolí. Zároveň je model inspirovaný principem modelu Progressive Growing GAN, který umožňuje dosáhnout vyššího rozlišení snímků. Generátor využívá architekturu sítě U-Net, která byla původně vyvinuta pro segmentaci biomedicínských dat, ale lze jí využít i pro tento účel. V dnešní době se DeOldify více zaměřuje na obarvování videa, námi implementovaný model se inspiroval starší verzí[1], která pracovala pouze s obarvováním šedotónových fotografií.

3 Dataset

3.1 ImageNet

ImageNet[2] je datovou sadou čítající přes 1,25 milionu snímků zařazených do 1000 kategorií [5]. Jedná se o jednu z nejznámějších datových sad využívanou zejména v úlohách strojového učení. Veliká rozličnost snímků mnoha různých objektů přináší ovšem jistou nevýhodu při trénování námi implementovaného modelu, protože trénování probíhá déle a nedosahuje tak kvalitních výsledků, než kdyby se využívalo pouze určité podkategorie snímků (např. snímky přírody nebo obličejů). Pro trénování byla využita verze datasetu se snímky rozměrů 64×64 pixelů.

3.2 COCO dataset

Ve své práci jsme se rozhodli provést experimenty i s dalším datasetem. Jedná se o COCO dataset[3], který obsahuje více jak 328 tisíc snímků zařazených do 91 objektových kategorií [3]. Každý objekt je poté i zařazen do superkategorie – např. objekty kategorií *bicycle*, *car*, *motorcycle*, *airplane*, *bus*, *train*, *truck* nebo *boat* jsou zařazeny do superkategorie *vehicle* [4]. COCO dataset jsme se rozhodli využít zvláště z důvodu snadné možnosti vybrání pouze několika kategorií obrázků, protože dataset ImageNet je příliš rozsáhlým a obsahuje obrovské množství rozličných objektů. Při trénování jsme využili jak rozlišení 64×64, tak i vyššího rozlišení 128×128 pixelů.

[1] `https://github.com/dana-kelley/DeOldify`
[2] `https://www.image-net.org/about.php`
[3] `https://cocodataset.org/#home`

4 Řešení

Kromě skriptů obsahující implementace *self-attention*[4] vrstvy a *spektrální normalizace*[5] je celá implementace řešení již pouze naší prací, nepřejímáme žádné již existující řešení. Námi řešený projekt je dostupný ve veřejném repozitáři na serveru *GitHub*[6].

4.1 Implementační detaily

Projekt byl implementován v jazyce Python, k implementaci modelů neuronových sítí byla využita knihovna *Tensorflow*, přesněji její rozhraní *Keras*. Ke zjednodušení procesu evaluace byl k vizualizaci všech relevantních trénovacích metrik využit nástroj *Tensorboard*.

4.2 Spouštěcí skript

Veškerá práce s vytvořenou neuronovou sítí je iniciována centrálně ze skriptu `run.py`. Je umožněno spustit trénování neuronové sítě od počátku, pokračovat v trénování po načtení již dříve natrénovaných vah a nebo pouze překonvertovat vlastní černobílé obrázky pomocí natrénované sítě.

Ke konverzi vlastních snímků je třeba spustit skript s argumentem `--images`, k pokračování dřívějšího trénování slouží argument `--load_weights` a základní chování bez použití argumentů je trénování modelu od počátku. Rovněž nastavitelné jsou veškeré cesty k vstupům a výstupům a většina parametrů nastavení sítě a trénování. Popis všech argumentů lze najít použitím argumentu `--help`.

4.3 Model

Naše řešení, stejně jako DeOldify, kterým jsme byli inspirováni, používá Self-Attention GAN [6], zkráceně SAGAN.

4.3.1 Diskriminátor

Diskriminátor je z dvojice sítí tou jednodušší, jedná se o klasickou konvoluční neuronovou síť využívající architekturu VGG. V každém konvolučním bloku, sestávajícím se z dvou konvolučních vrstev, využíváme za každou konvoluční vrstvou spektrální normalizaci a aktivační funkci *LeakyReLU*. Self-attention blok přidáváme pouze jednou do celé sítě, a to na konec bloku s rozlišením 32×32. Použití právě při tomto rozlišení je podle autorů této techniky nejefektivnější [6]. Diskriminátor je trénován tradičně pomocí optimalizátoru *Adam* s koeficientem učení 0.0004. Vstupem této sítě jsou tedy barevné obrázky a výstupem je pravděpodobnost, zda se jedná o skutečný či falešný obrázek.

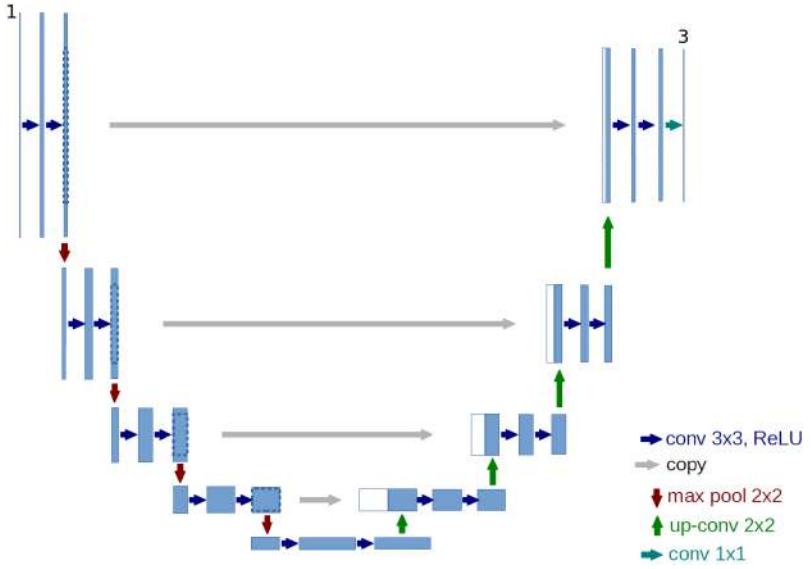
4.3.2 Generátor

Zajímavější částí generativních sítí je generátor. Tuto síť jsme vytvořili podle U-Net architektury, která je kromě segmentace, ke které byla původně vytvořená, vhodná i na úlohy jako *Image-to-image Translation* a *Super-resolution* a proto jí lze použít i na řešení tohoto problému.

[4] `https://github.com/kiyohiro8/SelfAttentionGAN`

[5] `https://github.com/IShengFang/SpectralNormalizationKeras`

[6] `https://github.com/TomasVenkrbec/knn-project`



Obrázek 1: Architektura U-Net použitá v generátoru

Konvoluční bloky v generátoru jsou podobné těm v diskriminátoru, pro každé rozlišení máme dvojici konvolučních vrstev se spektrálními normalizacemi. Čím je architektura speciální je práce s výstupy těchto konvolučních bloků. Jak je na obrázku 1 vidět, výstupy bloků jsou vstupy nejen dalších bloků se sníženým rozlišením, ale jsou také kopírovány na druhou stranu sítě do *upscaling* části sítě, na vstup bloku se stejným rozlišením, jako je rozlišení výstupu.

Rovněž podobné je využití self-attention bloků, které jsou opět využity v blocích s rozlišením 32×32 pixelů, a to jak v downscaling, tak v upscaling části neuronové sítě. Výstupem upscaling části sítě je obarvený vstupní šedotónový obrázek.

Generátor je také trénován pomocí optimalizátoru *Adam*, ovšem s menším koeficientem učení oproti diskriminátoru, a to 0.0001. Toto se používá ke stabilizaci a urychlení trénování sítí, jelikož je to alternativou více trénovacích kroků diskriminátoru než generátoru [2].

4.3.3 Chybové funkce

Diskriminátor se učí rozpoznávat původ obrázků – zda pochází z datové sady, nebo jsou výstupy generátoru. U vstupních obrázků tyto informace vždy máme, tudíž můžeme jako chybovou funkci využít *Binary Cross Entropy* mezi labely vstupných dat a labely, které byly na výstupu diskriminátoru.

Generátor využívá dvojici chybových funkcí. Za prvé, jelikož používáme generativní neuronové sítě a máme diskriminátor, máme chybovou funkci *Adversarial Loss*. Generátor se učí tím, že se snaží co nejvíce zvýšit hodnotu chybové funkce diskriminátoru. Toto motivuje generátor k tomu, aby co nejlépe obarvoval snímky, které má na vstupu. Aby obarvené snímky odpovídaly obsahem těm, které jsou na vstupu, využíváme *Perceptual Loss* (též známý jako *Feature Loss* nebo *VGG Loss*), který porovnává hodnoty aktivací ve vybraných vrstvách předtrénované VGG sítě, když je jí na vstup dán vstupní a výstupní obrázek generátoru. Kombinace těchto chybových funkcí motivuje generátor k tomu, aby neměnil obsah vstupního obrázku, ale pouze ho obarvil.

4.3.4 Trénování a vyhodnocování

Trénování probíhalo v prostředí *Google Colab*, který jsme vzhledem k velikosti modelu a z ní vycházející náročnosti trénování byli nuceni použít. Trénování bylo tedy omezeno po dobu, po kterou jsou dostupné zdarma grafické karty, což se odráží na finálních výsledcích.

Pro lepší kontrolu nad procesem trénování a vyhodnocování jsme předefinovali funkce `train_step` a `test_step`, kterými knihovna *Keras* vykonává jednotlivé kroky trénování a validace ve funkci `fit`. Díky tomu jsme mohli ručně počítat hodnoty chybových funkcí a aktualizovat váhy obou sítí. Také jsme snadněji mohli přidat vlastní callbacky k průběžnému ukládání vah modelu, vygenerovaných obrázků a všech trénovacích metrik do *Tensorboard*.

5 Výsledky

Při trénování modelu jsme využili tři přístupy. Nejprve byl model natrénovaný pouze na datasetu ImageNet (viz Sekce 3.1). Rozsáhlost datasetu však přinesla několik problémů, kdy trénování trvalo delší dobu, a přesto nedosahovalo dobrých výsledků. Zdá se, že model má při velmi rozličných snímcích problém s velkou variancí a neví, jaké rysy se přesně naučit. Proto jsme se rozhodli otestovat výsledky modelu, pokud po natrénování modelu na ImageNetu dotrénujeme tento model na specifičtější části COCO datasetu. Tento postup však ukázal, že v takovém případě má model tendenci aplikovat barevný odstín pouze na relativně velké plochy. Například jsou pak tedy plochy jako podlaha či stěna relativně dobře obarvené, ale model ignoruje další – menší – objekty na snímku.

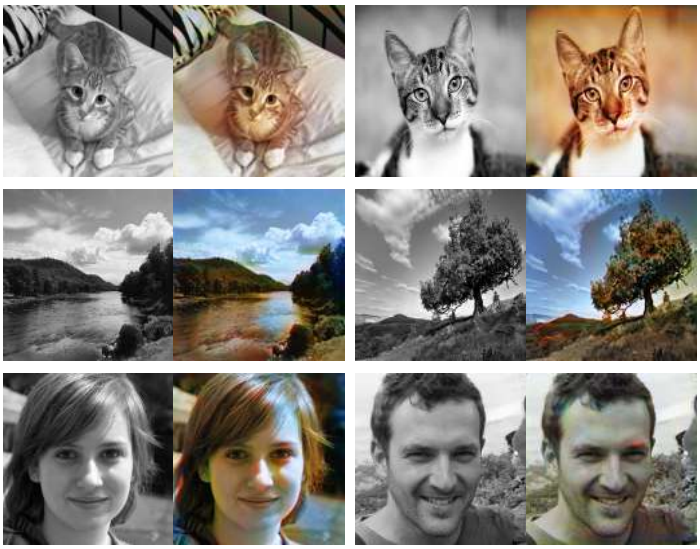
5.1 ImageNet

Výstupy modelu natrénovaného na snímcích z datasetu ImageNet po 100 epochách jsou ukázány na Obrázku 2.

Obrázek 2: Příklady výstupů modelu natrénovaného na ImageNet datasetu.

5.2 COCO dataset

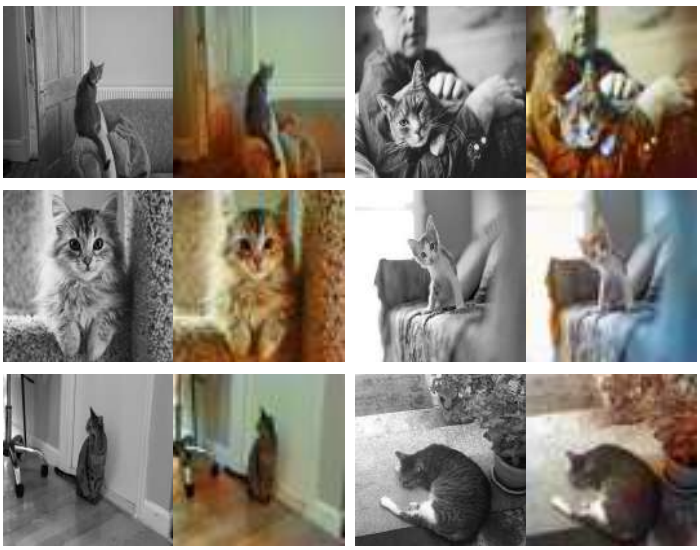
Narozdíl od ostatních přístupů byl model na COCO datasetu natrénovaný se snímky v rozlišení 128 × 128 pixelů. Obrázek 3 níže ukazuje, že model produkuje obstojné výsledky zejména v případě snímků z přírody.



Obrázek 3: Příklady výstupů modelu natrénovaného na COCO datasetu.

5.3 Kombinované trénování

Využití tohoto přístupu bohužel nepřineslo plánované výsledky. Model se v tomto případě soustředí na velké plochy, které následně obarví jedním barevným odstínem, jak jde vidět níže na obrázku 4.



Obrázek 4: Příklady výstupů modelu natrénovaného na ImageNetu s následným dotrénováním na obrázcích koček z COCO datasetu. Vstupem modelu byly zcela neznámá data nalezená na internetu.

6 Závěr

Model ukázal, že je schopný nalézt dominantní odstín v obraze a ten dále aplikovat. Bohužel však nebyl schopný naučit se tuto úlohu řešit na úrovni jednotlivých objektů. V případě datasetu ImageNet byl pro model zřejmě příliš komplexní a model nebyl schopný naučit se produkovat kvalitní výsledky. Při použití datasetu COCO se model snaží obarvovat přesnější lokality, ale stále na celou takovou oblast aplikuje jediný barevný odstín. Jako možné vylepšení modelu pro zlepšení kvality výsledků by stálo za zvážení přidání dodatečné informace pro trénování, například na základě dominantních barev v obraze.

Reference

[1] Antic, J. *DeOldify* [online]. Říjen 2018 [cit. 2021-05-28]. Dostupné z: `https://github.com/jantic/DeOldify`.

[2] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B. a Hochreiter, S. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. *ArXiv e-prints*. cerven 2017, s. arXiv:1706.08500.

[3] Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R. et al. Microsoft COCO: Common Objects in Context. únor 2015, s. arXiv:1405.0312v3.

[4] Simalango Fernandus, M. *What Object Categories / Labels Are In COCO Dataset?* [online]. Duben 2018 [cit. 2021-05-30]. Dostupné z: `https://tech.amikelive.com/node-718/what-object-categories-labels-are-in-coco-dataset/`.

[5] Simon, J. *ImageNet — part 1: going on an adventure* [online]. Září 2017 [cit. 2021-05-30]. Dostupné z: `https://julsimon.medium.com/imagenet-part-1-going-on-an-adventure-c0a62976dc72`.

[6] Zhang, H., Goodfellow, I., Metaxas, D. a Odena, A. Self-Attention Generative Adversarial Networks. *ArXiv e-prints*. kveten 2018, s. arXiv:1805.08318.