VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

GENERATIVNÍ NEURONOVÉ SÍTĚ PRO RUČNĚ PSANÉ PÍSMO

GENERATIVE NEURAL NETWORKS FOR HANDWRITTEN TEXT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. PAVEL ŠEVČÍK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MICHAL HRADIŠ, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2021/2022

Zadání diplomové práce

Student:	**Ševčík Pavel, Bc.**
Program:	Informační technologie a umělá inteligence
Specializace:	Strojové učení
Název:	**Generativní neuronové sítě pro ručně psané písmo** **Generative Neural Networks for Handwritten Text**
Kategorie:	Zpracování obrazu

Zadání:

1. Prostudujte základy konvolučních sítí, generativních sítí typu GAN.
2. Vytvořte si přehled o současných metodách schopných generovat obraz ručně psaného písma.
3. Vyberte nebo navrhněte metodu schopnou generovat ručně psané písmo v kvalitě, která by mohla být vhodná i pro trénování systémů pro rozpoznávání ručně psaného písma.
4. Obstarejte si databázi vhodnou pro experimenty.
5. Implementujte navrženou metodu a proveďte experimenty nad datovou sadou. Vyhodnoťte i jestli přidání generovaných dat do trénovací sady OCR systému může zlepšit jeho úspěšnost.
6. Porovnejte dosažené výsledky a diskutujte možnosti budoucího vývoje.
7. Vytvořte stručné video prezentující vaši práci, její cíle a výsledky.

Literatura:

- Alexander Mattick, Martin Mayr, Mathias Seuret, Andreas Maier and Vincent Christlein. SmartPatch: Improving Handwritten Word Imitation with Patch Discriminators. ICDAR, 2021.
- Kai Brandenbusch, Eugen Rusakov and Gernot A. Fink. Context Aware Generation of Cuneiform Signs. ICDAR, 2021.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Hradiš Michal, Ing., Ph.D.**
Vedoucí ústavu:	Černocký Jan, doc. Dr. Ing.
Datum zadání:	1. listopadu 2021
Datum odevzdání:	18. května 2022
Datum schválení:	1. listopadu 2021

Abstrakt

Cílem této práce bylo vytvořit model pro generování řádků ručně psaného písma. Model na základě očekávaného stylu a libovolně dlouhého textu vytváří odpovídají obrázek s písmem. Navržené řešení překonává existující přístupy v kvalitě generovaného písma a umožňuje generování jak samostatných slov, tak i řádků. Kombinuje vyhledávání příznaků pro jednotlivé symboly pomocí attention a jejich rozmístění na řádek pomocí vkládání mezer. Nový přístup umožňuje specifikovat pozice symbolů na řádku jemněji než celými čísly, a tak vytvářet plynulejší interpolace mezi různými styly. Na rozdíl od předchozího řešení tento přístup využívá Gaussův filtr pro rozšíření jednotlivých příznaků symbolů do blízkého okolí. Současně tento přístup otevírá množnost trénování modelu pro odhad pozic symbolů na řádku adversariální chybou (GAN). Navíc byly vytvořeny anotace horizontálních pozic symbolů na řádcích datové sady ručně psaného písma IAM.

Abstract

The aim of this study was to create a generative neural network for handwritten text lines. The model produces variable-sized images of handwritten text lines based on the expected style. The proposed method exceeds existing models in the image quality and can be used to generate both individual words and entire lines of handwritten text. It combines the use of the attention mechanism to extract the features for each character from the text query and their arranging on the line by inserting spaces between them. The new approach allows more granular control of the symbol positions on the line, which leads to smoother style interpolations. In contrast to the previous approach, the proposed method uses the Gaussian filter to spread the individual symbols features to the surrounding area. This approach also allows to train the model for symbols position predictions using the adversarial loss (GAN). In addition, annotations of symbol horizontal positions on the lines of the IAM dataset of handwritten text have been created.

Klíčová slova

Příprava trénovacích dat, ručně psané písmo, generativní neuronové sítě, GAN, AdaIN, Transformer.

Keywords

Generating training data, handwritten text, generative neural networks, GAN, AdaIN, Transformer.

Citace

ŠEVČÍK, Pavel. *Generativní neuronové sítě pro ručně psané písmo.* Brno, 2022. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Hradiš, Ph.D.

Generativní neuronové sítě pro ručně psané písmo

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Michala Hradiše, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Pavel Ševčík
15. května 2022

Poděkování

Rád bych poděkoval mému vedoucímu práce Ing. Michalovi Hradišovi, Ph.D. za jeho ochotu, cenné rady a čas, který mi věnoval při vedení této práce.

Obsah

Kapitola 1

Úvod

Ručně psané písmo se používá již tisíce let. Až do vynálezu knihtisku v 15. století se všechny významné dokumenty, kroniky nebo knihy přepisovaly ručně. I když v současné převažují již digitální dokumenty, ručně psané písmo nalézá velké uplatnění i dnes. Psaní rukou je jedna ze základních dovedností, které se vyučují ve školách. Umožňuje formulování myšlenek, zaznamenávání nejdůležitějších informací a přitom vyžaduje jen tužku a papír. Přesto jej moderní technologie dodnes neumí zcela bezchybně zpracovávat. Počítače stále nedokáží vždy přečíst a přepsat ručně psané texty do digitální podoby, což by umožnilo další digitální zpracování. Za celou historii vzniklo nezměrné množství knih a záznamů, které by mohly být analyzovány moderními postupy. S rozvojem umělé inteligence vznikají stále úspěšnější přístupy pro analýzu ručně psaného písma. Jedním z velkých problémů těchto metod je skutečnost, že vyžadují rozsáhlé trénovací sady. Řešení nabízí modely pro umělé generování ručně psaného písma, které lze využít pro rozšíření existujících datových sad, a dosahovat tím vyšší úspěšnosti ostatních modelů. Zároveň se tyto modely mohou uplatnit i přímo pro generování písma, které se značně liší od běžných digitálních fontů.

Tato práce cílí na vylepšení moderních přístupů generování ručně psaného písma pomocí hlubokých neuronových sítí a jejich aplikaci pro rozšíření existujících datových sad. Zaměřuje se rovněž na jejich principy a podstatné vlastnosti, a to především na jejich schopnost napodobit očekávaný styl a generovat písmo pro libovolně dlouhý text. Pracuje s výřezy slov a řádků datové sady IAM a vyhodnocuje možnosti rozšíření existujících datových sad pro trénování OCR systému s datovou sadou z projektu Pero. Prezentuje nový model pro generování ručně psaného písma, který se neomezuje pouze na samostatná slova, ale dokáže vytvářet celé řádky písma. Navazuje tak na předchozí výzkum, který se zabýval rozmístěním symbolů v generovaných řádcích. V práci je navržen nový způsob rozmístění symbolů, který umožňuje specifikovat pozice symbolů na řádku jemněji než celými čísly, a tak vytvářet plynulejší interpolace mezi různými styly. Tento přístup využívá Gaussův filtr pro rozmazání příznaků jednotlivých symbolů do blízkého okolí. Kvalita generovaného písma navíc překonává existující postupy, které jsou v práci detailně analyzovány. Mezi zásadní přínosy práce tak patří: analýza existujících řešení, rozšíření state-of-the-art modelu pro generování samostatných slov o schopnost generovat celé řádky, anotace horizontálních pozic symbolů na řádcích datové sady IAM a návrh nového přístupu k rozmístění znaků, který umožňuje sjednocené trénování všech částí modelů pro generování ručně psaného písma.

Kapitola 2

Generativní neuronové sítě

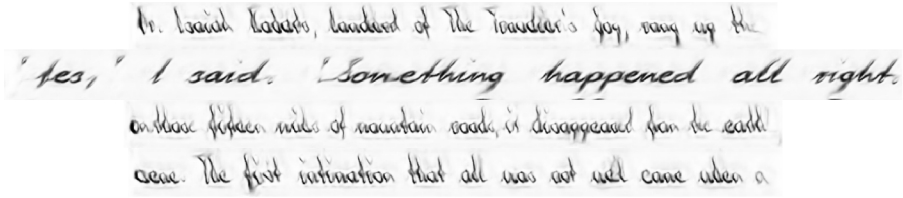
Konvoluční neuronové sítě jsou od roku 2012, kdy model se sítí *AlexNet* [26] zvítězil v každoroční soutěži *ImageNet Large-Scale Visual Recognition Challenge* [36], součástí většiny modelů, které cílí na získání informací z obrazu. Ať už se jedná o klasifikaci, detekci nebo segmentaci [38, 28, 15].

Jejich společnou vlastností je použití *konvolučních vrstev* [19], které na rozdíl od použití plně propojených vrstev využívají lokální souvislosti v datech (např. velký rozdíl v hodnotách sousedních pixelů může znamenat, že v daném místě se nachází hrana). V jiných aspektech se ale modely v různých oblastech značně liší. Cílem je, aby architektura reflektovala úkol sítě. Pokud se očekává, že model na základě krátkého textového popisu předmětu vytvoří jeho možnou fotografii, bude se tento model odlišovat od jiné neuronové sítě, která cílí na detekci předmětů v obrázku. V jednom případě je vstupem sekvence znaků a výstupem dvourozměrné pole (popř. třírozměrné pro barevné fotografie), v druhém případě je vstupem obrázek a výstupem souřadnice předmětu. Každý z příkladů vyžaduje jiné přístupy ke zpracování vstupů, a proto se architektury sítí liší.

V této práci se věnuji generování ručně psaného písma. Pro tento úkol se také dnes používají specifické přístupy. Konkrétně se často využívá architektura modelu se sítěmi, které spolu soutěží (GAN – *Generative Adversarial Networks*) [11]. Dále, pro aplikaci stylů se používá adaptivní normalizace (AdaIN – *Adaptive Instance Normalization*) [18] a pro zpracování sekvencí proměnné délky se využívá sítí typu *transformer* [41]. Nezbytnou součástí jsou i modely pro optické rozpoznání znaků (OCR – *Optical Character Recognition*), které jsou optimalizovány pomocí chybové funkce CTC – *Connectionist Temporal Classification* [13].

Vytváření nových snímků se může na první pohled zdát jako jednoduchý problém. Pokud je k dispozici datová sada s anotacemi $D = (X, Y)$, kde $X = (x_1, x_2, \ldots, x_n)$ jsou anotace a $Y = (y_1, y_2, \ldots, y_n)$ jsou obrázky, řešením může být typické učení s učitelem, kdy se od modelu G očekává, že k anotaci snímku vytvoří totožný snímek. Potom G je funkcí, která je reprezentována neuronovou sítí a která může být trénována za cílem snížení vzdálenosti mezi odhadovaným snímkem $G(x_i) = \tilde{y}_i$ a skutečným snímkem y_i.

Tento přístup však není ideální, protože v situaci, kdy ke stejnému nebo velmi podobnému vzoru existuje více možných výstupů, bude síť v ideálním případě generovat průměrný snímek (např. pro $x_1 = x_2 = x_3$ a rozdílné y_1, y_2 a y_3 bude výstupem sítě po trénování $G(x_1) = (y_1 + y_2 + y_3) \,/\, 3$). V praxi se ukazuje, že snímky bývají rozmazané, viz obrázek 2.1.



Obrázek 2.1: Ukázka generovaného písma modelem, který je trénovaný přístupem učení s učitelem s metrikou *L1 loss*. Výsledky jsou rozmazané, i když trénovací sada žádné takové neobsahuje. Ukázky převzaty z článku [7].

Generativní modely založené na soutěžení

Tento problém řeší modely založené na soutěžení (GAN – *Generative adversarial networks*). Ty připouští, že model generuje snímky, které nejsou totožné se snímky v datové sadě. Jedinou základní podmínkou je, že není možné rozlišit mezi páry trénovací množiny, tj. (x_i, y_i), a páry vytvořenými generativním model, tj. $(x_i, \tilde{y}_i)$.

Přesněji, popsaný příklad se věnuje podmíněnému generování (*Conditional generative adversarial nets*) [32], výstup modelu je podmíněn nějakou informací x_i. V základní variantě modelu GAN [11] je cílem generovat takové výstupy, které jsou nerozlišitelné od trénovací množiny, často označovanou anglicky *real data*. Množina výstupů generovaných modelem se pak často označuje anglicky jako *fake data*.

Základní model GAN se skládá ze dvou sítí – *generátoru* G a *diskriminátoru* D [11]. Každá ze sítí má v modelu jiný úkol. Generátor tvoří pouze výstupy $\tilde{y}$ a úkolem diskriminátoru je rozlišovat mezi daty z trénovací množiny y a výstupy generátoru $\tilde{y}$. Cílem modelu je vytvářet data, která jsou nerozlišitelná od dat trénovací množiny. Za tímto účelem se při učení upravují parametry G tak, aby D nebyl schopný rozlišovat mezi $\tilde{y}$ a y. Sítě tak spolu soutěží [11]:

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [\log (1 - D(G(\mathbf{z})))], \tag{2.1}$$

kde p_{data} reprezentuje rozdělení hustoty pravděpodobnosti dat trénovací množiny, $p_{\mathbf{z}}$ je funkce rozdělení hustoty pravděpodobnosti náhodného šumu, který slouží jako vstup generátoru G. Dále D označuje diskriminátor, jehož výstupem pro vstupní vektor $\mathbf{x}$ je odhadovaná pravděpodobnost, že $\mathbf{x}$ je z trénovací množiny.

Při trénování se jako u ostatních hlubokých neuronových sítí používá algoritmus *backpropagation* [19]. K trénování diskriminátoru se přistupuje jako k učení binárního klasifikátoru [11]. Pro data z trénovací množiny se očekává pravděpodobnost 100 % a pro data vytvořená generátorem 0 %. V praxi se ukázalo, že optimalizovat generátor za cílem minimalizovat chybovou funkci $\log (1 - D(G(\mathbf{z})))$ není vhodné, protože na začátku trénování, kdy G produkuje velmi špatné výstupy $\tilde{y}$, si je diskriminátor D velmi jistý ve svých rozhodnutích, tj. pro $\tilde{y}$ vrací pravděpodobnost velmi blízkou nule [11]. To způsobí, že gradient parametrů G je velmi slabý. Proto už autoři v původním článku [11] navrhují jiný cíl trénování G, a to maximalizovat hodnotu funkce $\log D(G(\mathbf{z}))$. Tato změna způsobí, že gradient parametrů G je na začátku trénování silnější, a síť se tak učí rychleji.

Autoři článku dokázali, že pro ideální diskriminátor D_G^*, který dokonale rozeznává mezi umělými a reálnými daty, funkce rozdělení hustoty pravděpodobnosti dat výstupů generátoru p_g je stejná jako funkce rozdělení hustoty pravděpodobnosti reálných dat p_{data}, protože platí, že pro pevně zvolený generátor G (tj. s fixními parametry) existuje optimální diskriminátor [11]:

$$D_G^*(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})}. \tag{2.2}$$

Dosazením optimálního diskriminátoru D_G^* do rovnice (2.1) se získává ztrátová funkce $C(G) = V(D_G^*, G)$. Po úpravě pak platí [11]:

$$C(G) = -\log(4) + 2 \cdot JSD(p_{\text{data}} \,\|\, p_g), \tag{2.3}$$

kde JSD je *Jensenova–Shannonova divergence* [33]. Je zřejmé, že ztrátová funkce nabývá minimální hodnoty, pouze pokud $p_{\text{data}} = p_g$. I když je tak zaručeno, že cílem G je zachytit funkci rozdělení hustoty pravděpodobnosti dat trénovací množiny, v praxi je získání optimálního D obtížné. Každá úprava parametrů G při trénování by vyžadovala nalezení nových parametrů optimálního D, což by vedlo k velmi pomalému učení. Proto se omezuje počet iterací aktualizací parametrů D na zvolenou konstantu k. Ta typicky nabývá hodnot v rozmezí od 1 do 10. V důsledku toho pak není nikdy cíl $p_{\text{data}} = p_g$ zaručen, přesto v praxi modely dosahují kvalitních výsledků.

Adaptivní normalizace

Základním cílem generativních modelů je zachytit celé rozdělení p_{data}. To vyžaduje, aby výstup generování byl stochastický. Například, pokud trénovací množina obsahuje páry snímků v odstínech šedé a barevných snímků stejných scén, úkolem generování je pak obarvení snímku v odstínech šedé. Pokud by na snímku byla košile, model by měl správně zachytit skutečnost, že barva košile může být různá.

Pro zavedení náhodnosti se do modelů vkládá šum. Dnes běžně používaný způsob je pomocí *adaptivní normalizace* [18]. Jedná se o rozšíření tzv. dávkové normalizace (angl. *batch normalization*) [20], přesněji její varianty *instanční normalizace* (angl. *instance normalization*) [40]. Ta se v modelech původně používala pro rychlejší a stabilnější učení. Autoři modelu *StyleGAN* [24] použili adaptivní normalizaci k aplikaci naučených stylů pro transformace. Oddělili prostor stylu, tj. stochastických vlastností, od samotného obsahu snímků. To lze využít pro interpolace mezi různými vektory příznaků reprezentující styly, a získat tak nové nebo sledovat vliv jednotlivých příznaků na výstup.

Formálně lze popsat funkci vrstvy adaptivní normalizace takto [24]:

$$\text{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i}, \tag{2.4}$$

kde $\mathbf{x}_i$ je vstupní i-tá vrstva příznaků, $\mu(\mathbf{x}_i)$ je střední hodnota vrstvy $\mathbf{x}_i$ a $\sigma(\mathbf{x}_i)$ je směrodatná odchylka vrstvy $\mathbf{x}_i$. Dále pak $\mathbf{y}_{s,i}$ je směrodatná odchylka po transformaci a $\mathbf{y}_{b,i}$ je střední hodnota po transformaci. Jako styl se v takovém případě označuje vektor všech hodnot $\mathbf{y}$ z celé sítě.

Transformer

Pro zpracování vektorů proměnné délky, tj. sekvencí, lze využít rekurentních neuronových sítí (RNN). Ty se od běžných neuronových sítí liší tím, že jejich součástí je alespoň jedna rekurentní neuronová vrstva, např. *Long short-term memory* (LSTM) [17] nebo *Gated recurrent unit* (GRU) [5]. Typickou vlastností těchto vrstev je využití tzv. skrytého stavu h_t, který je funkcí předchozího stavu h_{t-1} a vstupu v čase t. Tato sekvenční závislost mezi stavy zamezuje využití paralelního zpracování při výpočtu odezvy sítě pro jeden vstupní vektor, což je velký problém především pro velmi dlouhé vstupy [41]. Řešení tohoto problému nabízí sítě typu *transformer* [41]. Další řešení nabízí i sítě, které namísto rekurentních vrstev využívají konvoluční. Ty však vyžadují větší výpočetní výkon a pro modelování závislostí mezi vzdálenými prvky vstupního vektoru potřebují zároveň více vrstev [41].

Sítě typu transformer používají *attention* vrstvy [41]. Ty lze popsat jako funkci [41], jejímž vstupem jsou vektory *query* q, *key* k a *value* v a výstupem jsou vektory y. Vektory q a k mají stejnou délku d_k a vektory v a y mají délku d_v. Hodnota jednoho vektoru y_i je vyjádřena jako vážený součet vstupních vektorů $v_1, v_2, \ldots, v_n$. Váha vektoru v_j pro výstupní vektor y_i je určena škálovaným skalárním součinem vektorů q_i a k_i. Aby součet všech vah pro výstupní vektor y_i byl roven jedné, jsou transformovány funkcí *softmax*. V praxi se pro větší efektivitu počítají všechny váhy pro všechny výstupní vektory y současně pomocí maticového násobení. Proto platí [41]:

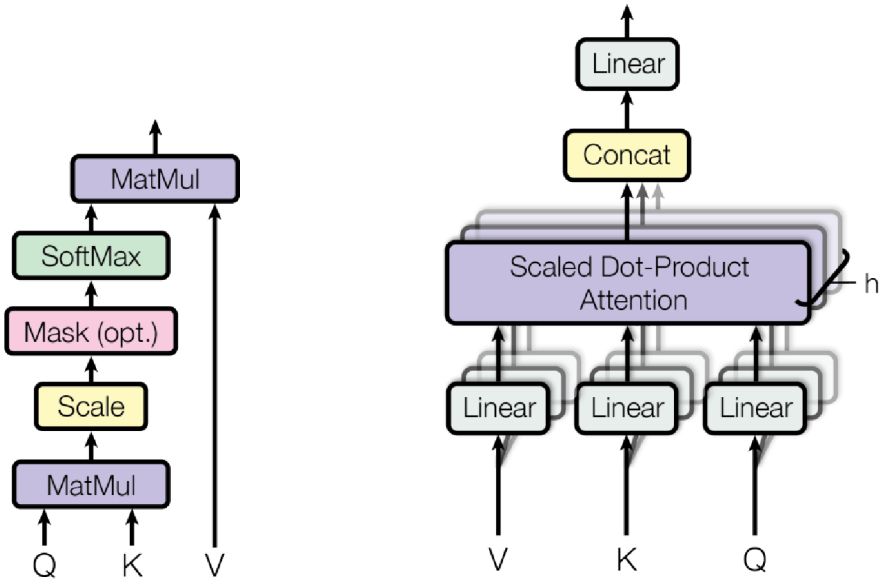
$$\text{Attention}\left(Q, K, V\right) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{2.5}$$

kde Q je matice složená z vektorů q, K je matice složená z vektorů k a V je matice složená z vektorů v. Ukázalo se, že při použití škálovaného skalárního součinu (v tomto případě skalárního součinu vynásobeného konstantou $\frac{1}{\sqrt{d_k}}$) modely dosahují lepších výsledků než při použití obyčejného skalárního součinu.

Častým použitím *attention* vrstvy je *multi-head attention* [41]. Vstupní vektory q, k, v jsou transformovány n lineárními vrstvami. Výsledkem transformace je n variant q, k, v, které jsou vstupem n *attention* vrstev. Jejich výstupy jsou spojeny do jednoho vektoru a transformovány další lineární vrstvou, aby byla zachována očekávaná dimenze výstupu. Model se tak může zaměřit na různé souvislosti ve vstupních vektorech. Porovnání výpočetních grafů *attention* a *multi-head attention* vrstev je na obrázku 2.2.

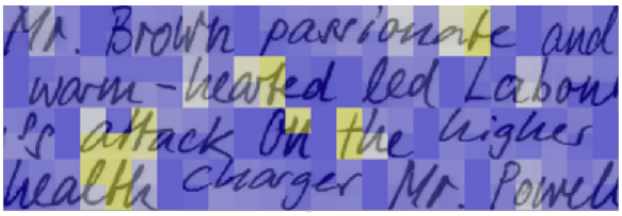
V případě, kdy je cílem nalezení závislostí mezi prvky jedné vstupní sekvence, se používá varianta *self-attention* [41]. Vstupní vektory q, k a v jsou v tomto případě všechny stejné.

Nevýhodou zpracování sekvencí pomocí sítí typu transformer je kvadratická časová složitost vzhledem k délce vstupní sekvence [2]. Tato vlastnost je problémem pro zpracování obrázků, které se běžně skládají z tisíců pixelů. Obrázky jsou proto často zpracovávány ve dvou krocích: získání příznaků s menšími rozměry pomocí konvoluční sítě a následné zpracování těchto příznaků pomocí sítě typu transformer [2]. Takto je možné využít mechanizmu *attention* i pro rozměrná data, jak ukazuje obrázek 2.3



(a) Attention (b) Multi-head attention

Obrázek 2.2: Porovnání vrstvy *attention* (a) [41] a vrstvy *multi-head attention* [41]. Vstupem obou vrstev jsou matice složené z vektorů: Q – *queries*, K – *keys*, V – *values*. Hlavním rozdílem je použití lineárních vrstev u varianty (b). Obrázky převzaty z [41].



Obrázek 2.3: Ukázka použití *attention* pro rozměrná data. Znázorněny jsou oblasti, na které se model soustředí pro získání informace o písmenu „t“. Model se soustředí na oblasti příslušící k příznakům, nikoli na jednotlivé pixely. Značně se tak snižuje výpočetní čas. Obrázek je převzat z [2].

Kapitola 3

Tvorba ručně psaného písma

Modely založené na soutěžení sítí, viz kapitola 2, jsou úspěšné v řadě úkolů generování obrazu, např. vytvoření obrazu na základě krátkého popisu [35] nebo generování realistických fotografií tváří [24]. Tyto modely nalezly uplatnění i v generování ručně psaného písma [1, 2, 7, 10, 23, 30].

Často se tvorba ručně psaného písma dělí na *online* a *offline* [10, 30]. Každý uvažuje jiný formát výstupních dat a má jak své výhody, tak i nevýhody. Offline přístupy lze dále rozdělit podle maximální délky výstupů.

Přístup *online* pracuje se sekvencemi tahů a polohami pera během psaní [10]. Naopak *offline* přístup pracuje pouze s obrazem výsledného textu [10]. Je zřejmé, že při trénování modelu, který používá online přístup, je k dispozici více informací. Navíc netrpí na nedostatky rastrových snímků, které často kromě textu obsahují různé textury pozadí a nepřesnosti [10]. Má však i značné nedostatky. Pořizování datové sady je náročnější a vyžaduje speciální vybavení [30]. Na rozdíl od offline přístupu není použitelný pro stará písma z historických kronik, tabletů atd. [10]. Protože cílem této práce je navrhnout a implementovat model pro tvorbu ručně psaného písma na základě historických dokumentů, věnuje se dále především offline přístupům.

Offline přístupy lze dále dělit podle způsobu, jak vytvářejí výstupní obrázek, a s tím související maximální délku vstupního textu. Některé metody vyžadují, aby byl dopředu jasně určen maximální počet písmen, které může generátor zpracovat, a na jejich základě generovat snímek, který má zpravidla pevně zvolené rozměry. Mezi příklady těchto modelů patří *GANwriting* [23], *SmartPatch* [30] nebo model z článku *Context Aware Generation of Cuneiform Signs* [4]. Naopak model *ScrabbleGAN* [10] a z článku [7] umožňují generování písma neomezeně dlouhých slov nebo různě dlouhých řádků textu.

Definice řešeného problému

Práce se věnuje problému podmíněného generování ručně psaného písma. Proces generování je podmíněn vstupním textovým řetězcem, který definuje textový obsah výstupního obrázku, a navíc je podmíněn stylem psaní, který je určen množinou obrázků ručně psaného písma od stejného autora.

Tj. nechť $\{\mathcal{X}, \mathcal{Y}, \mathcal{W}\}$ je trénovací množina, která obsahuje obrázky v odstínech šedé $\mathcal{X}$, jejich textové obsahy $\mathcal{Y}$ a identifikátory autorů $\mathcal{W} = \{w_i\}_{i=1}^{N}$ [23]. Nechť $X_{w_i} = \{x_{w_i,j}\}_{j=1}^{K} \subseteq \mathcal{X}$ je podmnožina K náhodně vybraných obrázků ručně psaného písma od stejného autora $w_i \in \mathcal{W}$. Nechť Σ je abeceda všech povolených znaků (např. písmen, číslic nebo interpunkč-

ních znamének). Potom je generativní model G definován jako [23]:

$$\tilde{x} = G\left(t, X_{w_i}\right) = G\left(t, \{x_1, x_2, \ldots, x_K\}\right), \tag{3.1}$$

kde $\tilde{x}$ je uměle vytvořený obrázek ručně psaného písma, jehož obsahem je řetězec $t \in \Sigma^*$.

Na rozdíl od autorů modelu *GANwriting* [23], v navrženém řešení této práce není jazyk povolených řetězců omezen žádnou maximální délkou textového obsahu. V práci se tedy věnuji problému generování ručně psaného písma pro předem neomezenou maximální délku vstupního textu, což umožňuje kromě generování jednotlivých slov i generování celých řádků textu (podobně jako autoři modelu *ScrabbleGAN* [10]).

Vyhodnocení

Vyhodnocování generativních modelů je problematické. Neexistuje jediná přesná metrika, která by umožňovala zcela objektivně posoudit kvalitu výstupních obrázků. Místo takové metriky se používá několik různých přístupů, které se zaměřují na různé vlastnosti výstupů. Patří mezi ně metrika *Fréchet inception distance* (FID) [16], statistické vyhodnocení posouzení vizuální kvality skupinou uživatelů [7, 23, 30] a porovnání úspěšnosti modelu, který byl trénován se základní trénovací datovou sadou, a stejného modelu, který byl trénován s trénovací datovou sadou rozšířenou o uměle vygenerované příklady [1, 10].

Cílem generativních modelů je zachytit podstatu dat trénovací množiny [11]. To mimo jiné znamená, že funkce rozdělení hustoty pravděpodobnosti pozorování příkladů trénovací množiny by měla být stejná jako funkce rozdělení hustoty pravděpodobnosti, kterou se řídí generování umělých dat.

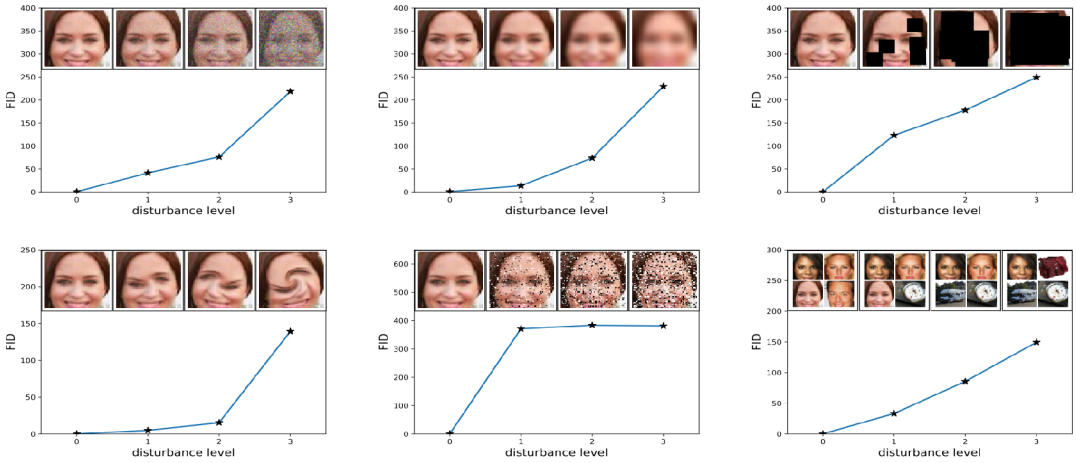
Metrika *Fréchet inception distance* [16] vychází z dříve používané metriky *Inception score* [37]. Ta k vyhodnocení generativního modelu využívá klasifikační síť *Inception v3* [39] plně natrénovanou na datové sadě ImageNet [36]. Metrika předpokládá, že u obrázků, které obsahují smysluplné objekty, si je síť jistá v klasifikaci, a tak odhadovaná pravděpodobnostní rozdělení tříd pro jednotlivé vstupy mají nízkou entropii. Naopak různé vstupní obrázky by měly být klasifikovány rozdílně, tj. entropie mezi obrázky by měla být vysoká [16]. Tento přístup k vyhodnocení koreluje s hodnocením lidí [16]. Není však ideální, v experimentech se ukázalo, že různé poškození obrázků (např. přidání šumu nebo rozmazání) není reflektované ve výsledném hodnocení [16]. Právě tyto nedostatky odstraňuje metrika Fréchet inception distance.

Zatímco Inception score k vyhodnocení vyžaduje pouze umělá data, FID k vyhodnocení používá jak umělá data, tak i reálná data. Namísto klasifikačního výstupu sítě Inception v3 vyhodnocuje příznaky z předchozí vrstvy. Jak pro umělá data, tak i pro reálná data zaznamenává aktivace. Pro výpočet výsledné hodnoty metriky pak používá metriku vzdálenosti mezi normálními rozděleními [16]:

$$d^2\left((\mathbf{m}, \mathbf{C}), (\mathbf{m}_w, \mathbf{C}_w)\right) = \|\mathbf{m} - \mathbf{m}_w\|_2^2 + \operatorname{Tr}\left(\mathbf{C} + \mathbf{C}_w - 2\left(\mathbf{C}\mathbf{C}_w\right)^{1/2}\right), \tag{3.2}$$

kde $(\mathbf{m}, \mathbf{C})$ je střední hodnota a kovarianční matice aktivací reálných dat a $(\mathbf{m}_w, \mathbf{C}_w)$ je střední hodnota a kovarianční matice aktivací umělých dat.

Pro ověření, že metrika pracuje, jak se očekává, provedli autoři řadu experimentů, ve kterých sledovali výslednou hodnotu pro různá poškození datové sady (např. rozmazání, přidání Gaussova šumu, překrytí černými čtverci nebo přidáním šumu typu sůl a pepř), viz obrázek 3.1.



Obrázek 3.1: Ukázka chování metriky FID [16] pro různá poškození obrázku. Tato poškození jsou reflektována jejím nárůstem. Obrázek převzat z [16].

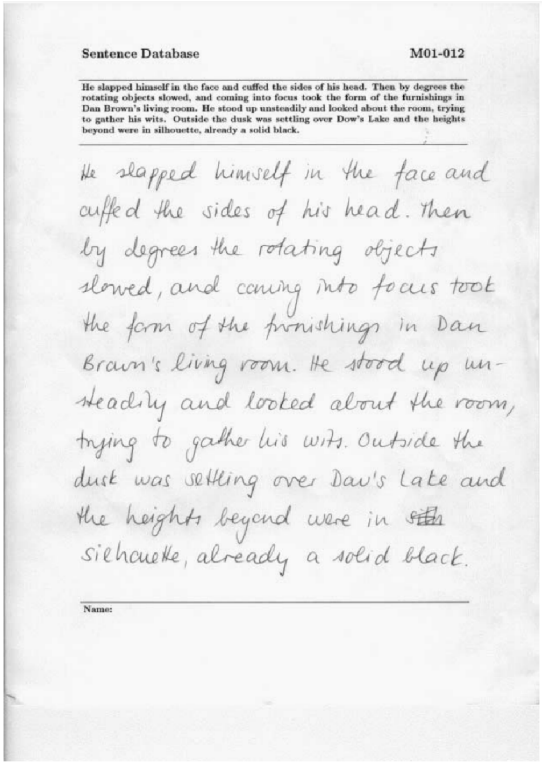
Dalším způsobem vyhodnocení generativních modelů je vyhodnocení odpovědí skupiny uživatelů. Dotazníky mohou mít mnoho podob. Například se lze ptát na ohodnocení body nebo lze použít Turingův test, kdy účastníci šetření odpovídají na otázku, zda si myslí, že daný příklad byl uměle vygenerovaný, nebo je z trénovací množiny [23]. Nevýhodou tohoto způsobu vyhodnocení je skutečnost, že vyžaduje vypracování dotazníků a ruční vyhodnocení. Výhodou je, že s dostatečně velkou skupinou lze využít statistických testů, a dosáhnout tak vypovídajících závěrů.

Augmentace dat je běžně používanou technikou k rozšíření trénovací sady [19]. Pomáhá stabilizovat proces trénování a navíc vede k vyšší kvalitě natrénovaného modelu. Uměle vygenerovaná data lze využít jako další příklady pro trénování modelů. Generativní model pak lze vyhodnotit na základě porovnání úspěšnosti modelu, který měl přístup pouze k původní trénovací množině, a modelu, který při trénování využil i nová uměle vygenerovaná data generativním modelem. V případě tvorby ručně psaného písma lze využít modely pro optické rozpoznání znaků (OCR – *Optical Character Recognition*) [1, 10].

3.1 Datové sady ručně psaného písma

Trénování hlubokých neuronových sítí vyžaduje velké datové sady. Pro generování ručně psaného písma lze využít datové sady běžně používané pro optické rozpoznání znaků. Mezi nejvíce využívané v oblasti generování patří datová sada IAM – *Institut für Informatik und angewandte Mathematik* [29], složená z anglického textu, a datová sada RIMES [14] – *Reconnaissance et Indexation de données Manuscrites et de fac similÉS*, z francouzského textu.

Datová sada IAM – *Institut für Informatik und angewandte Mathematik* [29] se skládá z ručních přepisů anglických textů z korpusu *Lancaster – Oslo/Bergen* (LOB) [21]. Obsahuje 500 textů, každý z přibližně 2 000 slov, z různých kategorií (např. novinářské články, náboženství, humorná literatura, science fiction atd.). Na vytváření datové sady IAM se podílelo 637 pisatelů. Skládá se z anotovaných 115 320 slov. Ty jsou seskupeny do 5 685 vět a 13 353 řádků [29].


Sentence Database M01-012

He slapped himself in the face and cuffed the sides of his head. Then by degrees the rotating objects slowed, and coming into focus took the form of the furnishings in Dan Brown's living room. He stood up unsteadily and looked about the room, trying to gather his wits. Outside the dusk was settling over Dow's Lake and the heights beyond were in silhouette, already a solid black.

He slapped himself in the face and cuffed the sides of his head. Then by degrees the rotating objects slowed, and coming into focus took the form of the furnishings in Dan Brown's living room. He stood up unsteadily and looked about the room, trying to gather his wits. Outside the dusk was settling over Dow's Lake and the heights beyond were in silhouette, already a solid black.

Name:

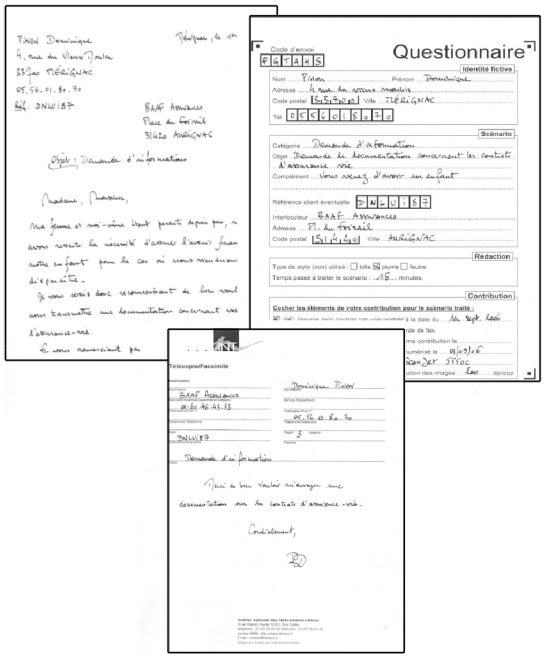
Obrázek 3.2: Ukázka jednoho formuláře použitého při vytváření datové sady IAM [29]. Skládá se z odstavce tištěného textu a jeho ručního přepisu. Obrázek převzat z [29].

Při vytváření datové sady pisatelé přepisovali odstavce textu, které byly složeny ze tří až šesti vět. Při přepisu byly jako pomůcky použity rovné rovnoběžné linky, aby písmo bylo zarovnané, a tak následné rozdělení na řádky při přepisu bylo jednoznačné [29]. To však nemusí být obecně u ručně psaných textů běžné. Obrázek 3.2 ukazuje jeden formulář včetně písemného přepisu.

Datová sada RIMES [14] je složená z 5 605 dopisů ve francouzském jazyce od více než 1 300 autorů [14]. Obsahuje tři druhy dokumentů – ručně psané dopisy, formuláře a krycí listy faxů. Kromě rozpoznávání ručně psaného písma lze využít pro řadu dalších úkolů strojového učení jako je analýza rozložení dokumentů, identifikace a verifikace autora nebo získávání informací [14]. Mimo jiné tak obsahuje 300 000 anotovaných obrázků ručně psaných slov. Proti datové sadě IAM [29] se liší ve struktuře řádků, nejsou tak zarovnané podle rovnoběžných rovných linek. Patří mezi běžně používané datové sady pro trénování a vyhodnocení uměle generovaného ručně psaného písma [1, 10]. Ukázka dokumentů, které sloužily jako podklad pro vytvoření datové sady, je na obrázku 3.3.

3.2 Existující přístupy

Tvorba ručně psaného písma není nový problém. Existuje řada modelů, které řeší, jak generovat co nejkvalitnější obrázky ručně psaného písma [1, 7, 10, 23, 30]. Starší modely pracovaly se sbírkami vzorů jednotlivých znaků, které byly postupně různě transformovány a nakonec spojeny do řetězců [12]. Takové modely však nedokázaly věrohodně reprodukovat ručně psané písmo, které by bylo nerozlišitelné od reálných příkladů.



Obrázek 3.3: Ukázka tří druhů dokumentů, které sloužily jako podklad pro datovou sadu RIMES [14]: ručně psaný dopis (vlevo), formulář (vpravo) a krycí list faxu (uprostřed). Obrázek převzat z [14].

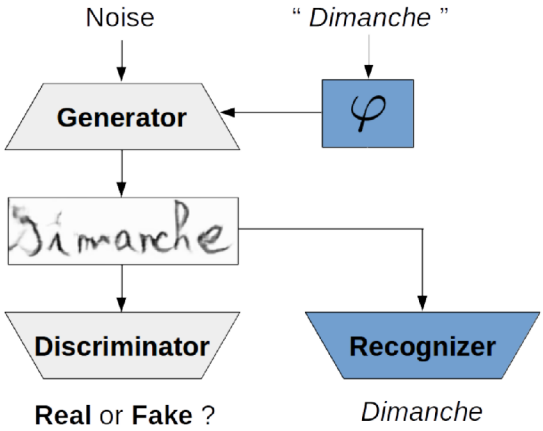
Velký přínos měl model autorů článku *Adversarial Generation of Handwritten Text Images Conditioned on Sequences* [1]. Jako první model pro tvorbu ručně psaného písma podmíněného vstupním textem vychází z generativního modelu GAN. Následovalo mnoho modelů, které staví na stejných principech, například model autorů článku *GANwriting: Content-Conditioned Generation of Styled Handwritten Word Images* [23], model *Scrabble-GAN* [10], model *SmartPatch* [30] a model autorů článku *Context Aware Generation of Cuneiform Signs* [22]. Těmto modelům se věnuji podrobně v následujícím textu.

Adversarial Generation of Handwritten Text Images Conditioned on Sequences

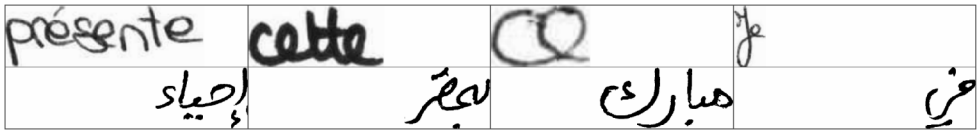
Jedná se o první model pro tvorbu ručně psaného písma podmíněného vstupním textem, který využívá architekturu modelu GAN, viz kapitola 2. Dále jako první přichází s přidáním ztrátové funkce *Connectionist Temporal Classification* (CTC) [13] ke ztrátové funkci určené diskriminátorem [1]. Tato další ztrátová funkce se používá v modelech pro optické rozpoznávání znaků. Spojením ztrátové funkce CTC a diskriminátoru z modelu GAN tak vznikl model, který byl schopný generovat realistické snímky ručně psaného písma.

Celkový model se skládá z několika částí: generátoru G, diskriminátoru D, rekurentní neuronové sítě φ a neuronové sítě pro optické rozpoznání znaků R [1]. Jejich zapojení ukazuje obrázek 3.4.

Vstupem sítě φ je text slova t, které má být vykresleno generátorem. Nejdříve je každý znak transformován do kódujícího vektoru, následně je pak celý řetězec zpracován obousměrnou rekurentní neuronovou vrstvou LSTM (*Long Short-term Memory*). Výstupem je vektor konstantní velikosti $\varphi(t)$, který je vedle náhodného šumu z vícerozměrného standardního rozdělení vstupem generátoru G [1].



Obrázek 3.4: Architektura modelu z článku [1], který se skládá, kromě generátoru a diskriminátoru, i z rekurentní neuronové sítě φ, která kóduje vstupní textový řetězec, a sítě pro rozpoznání ručně psaného písma – Recognizer. Obrázek převzat z [1].



Obrázek 3.5: Ukázka generovaného ručně psaného písma modelem z [1]. Model je schopný generovat písmo jak pro francouzská, tak i pro arabská slova. Obrázek převzat z [1].

Generátor G je konvoluční neuronová síť, která se skládá z mimo jiných částí ze tří reziduálních bloků a *self-attention* vrstvy (viz kapitola 2). Obsah generovaného slova je vložen pomocí *podmíněné dávkové normalizace* (CBN – *Conditional batch normalization*) [8], která je součástí reziduálních bloků.

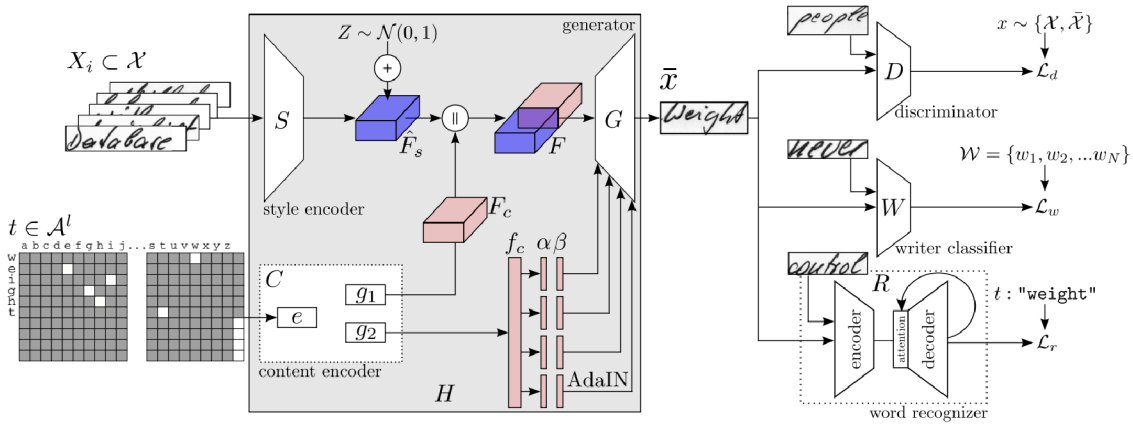
Diskriminátor D má pak standardní architekturu, tj. je složen ze sedmi reziduálních bloků, které postupně snižují rozlišení, *self-attention* vrstvy a běžné reziduální vrstvy [1]. Poslední síť R autoři článku převzali z článku [3].

Pro trénování sítě G se používají dvě chybové funkce: chyba sítě pro optické rozpoznání znaků – L_R a chyba GAN – L_{adv}. Problémem je, že hodnoty gradientů ∇L_R a ∇L_{adv} se řádově liší. Proto model tyto gradienty normalizuje tak, aby oba měly stejnou střední hodnotu a směrodatnou odchylku. Gradient ∇L_{adv} je ponechán beze změny, ale gradient ∇L_R je změněn:

$$\nabla L_R \leftarrow \left(\frac{\sigma L_{adv}}{\sigma L_R} \cdot (\nabla L_R - \mu L_R) + \mu L_{adv} \right), \tag{3.3}$$

kde σL_{adv} je směrodatná odchylka hodnot ∇L_{adv}, σL_R je směrodatná odchylka hodnot ∇L_R, μL_{adv} je střední hodnota ∇L_{adv} a μL_R je střední hodnota L_R.

Natrénovaný model je schopný generovat písmo jak francouzských slov, tak i arabských, viz obrázek 3.5. Mezi hlavní nedostatky modelu patří neschopnost generovat ručně psané písmo pro celé řádky textu, dokáže pouze generovat výsledky pro omezeně dlouhá slova. Dalším nedostatkem je, že generované písmo má velmi podobný styl [1]. Model neumožňuje podmínění výsledného stylu, tj. tloušťky čar, kurzívy atd. Přesto otevřel cestu dalším modelům a dal jim základní architekturu, ze které lze vyjít.



Obrázek 3.6: Architektura modelu *GANwriting* [23], který podmiňuje styl generovaného písma. Obrázek převzat z [23].

3.3 Generování se stylem písma

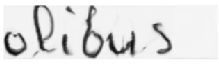
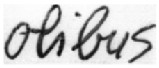
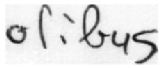
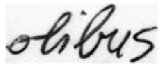
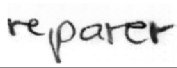
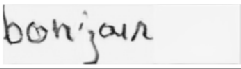
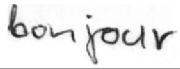
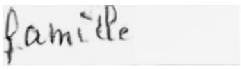
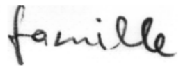
Podmínění stylu písma řeší jeden z hlavních nedostatků modelu *Adversarial Generation of Handwritten Text Images Conditioned on Sequences* [1], kterým je, že plně natrénovaný model není schopný generovat písmo s různými styly. Místo zachycení celého rozdělení pravděpodobnosti reálných příkladů model generuje pouze příklady, které jsou podobné jen určité podmnožině příkladů z trénovací datové sady. K řešení přistupují novější modely různě. Například model *GANwriting* [23] podmiňuje styl množinou obrázků písma od stejného autora nebo model *ScrabbleGAN* [10] pouze násobí příznaky symbolů v první vrstvě modelu náhodným vektorem.

GANwriting

Model *GANwriting* [23] navazuje na model autorů článku *Adversarial Generation of Handwritten Text Images Conditioned on Sequences* [1]. Odstraňuje jeho hlavní nedostatek, a to neschopnost generovat písmo s různými styly. Navrženým řešením je přidání další chybové funkce pro učení modelu a podmínění generování několika příklady písma od zvoleného autora [23]. Tyto příklady pak musí síť reflektovat v uměle generovaném písmu.

Výsledný model je postaven ze šesti částí: generátoru G, sítě pro kódování stylu autora S, sítě pro zakódování textového obsahu C, diskriminátoru D, sítě pro klasifikaci autora W a sítě pro optické rozpoznání znaků R [23]. Celkovou strukturu modelu detailněji zobrazuje obrázek 3.6.

Novou součástí modelu oproti modelu z článku [1] je konvoluční neuronová síť pro kódování stylu S [23]. Jejím vstupem je množina patnácti obrázků ručně psaného písma od zvoleného autora w_i. Síť má architekturu VGG-19-BN [38]. Všechny vstupní obrázky jsou transformovány tak, aby měly stejnou šířku a výšku, a jsou spojeny do jednoho tensoru, kde každý obrázek tvoří jeden kanál [23]. Pro modelování různých náhodných nepřesností je k výstupnímu vektoru F_s přičten vektor šumu z vícerozměrného standardizovaného normálního rozdělení [23].

Content	Alonso *et al.*	Ours		
		Style *A*	Style *B*	Style *C*
"olibus"				
"reparer"				
"bonjour"				
"famille"				

Obrázek 3.7: Srovnání kvality generovaného písma z článku [1] a písma generovaného modelem *GANwriting* [23]. Styl písma vytvořeného modelem *GANwriting* je podmíněn příkladem. Obrázek převzat z [23].

Obrázek 3.8: Ukázka vad, které může mít písmo generované modelem *GANwriting* [23]. Obrázek převzat z [30].

Část pro zakódování textového obsahu C má dva výstupy: tenzor reprezentující příznaky samostatných znaků F_c a čtyři dvojice parametrů α, β pro vrstvy AdaIN, které jsou součástí sítě G [23]. Smyslem je, aby F_c kódoval generované znaky a parametry α a β obsahovaly globální reprezentaci řetězce.

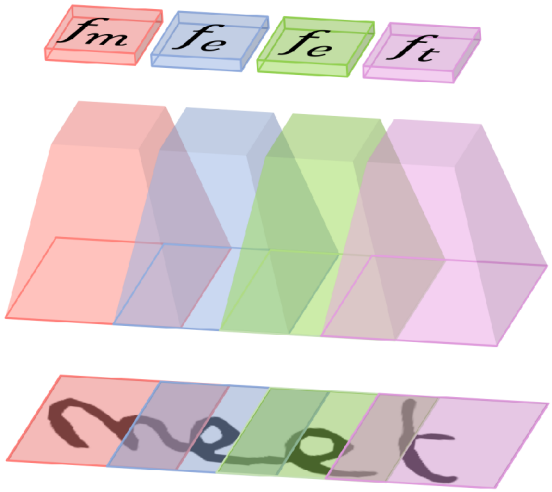
Generátor G se skládá ze dvou reziduálních bloků s normalizačními vrstvami *AdaIN* (viz kapitola 2), čtyř konvolučních bloků s vrstvami pro zvýšení rozlišení pomocí *nearest neighbor* metody [23]. Vstupem je konkatenace reprezentace stylu F_s a kódování řetězce F_c [23]. Parametry α a β jsou dalšími vstupy normalizačních vrstev *AdaIN* [23].

Diskriminátor D je složen z konvoluční vrstvy následované šesti reziduálními bloky s aktivační funkcí *LeakyReLU* [19] a *average pooling* [19] vrstvami pro snížení rozlišení [23].

Další novou podstatnou částí celého modelu je síť pro klasifikaci autora W [23]. Architekturu má stejnou jako D kromě poslední vrstvy, která obsahuje jeden neuron pro každého autora z trénovací sady. Síť W je trénována pomocí chybové funkce křížové entropie, jen pomocí reálných dat, tj. pouze dat trénovací množiny [23].

Poslední částí je síť pro optické rozpoznávání znaků R. Autoři implementovali state-of-the-art model podle [31]. Síť se skládá ze dvou částí enkodéru a dekodéru [23]. Enkodér je složen ze sítě VGG-19-BN [38] následované dvěma obousměrnými rekurentními vrstvami (B-GRU – *Bi-directional Gated Recurrent Unit*) [6]. Dekodér je pak jednosměrná rekurentní neuronová síť, která vrací v každém kroku jeden znak [23].

Model *GANwriting* dokáže generovat ručně psané písmo ve velmi vysoké kvalitě, které téměř není rozeznatelné od příkladů z trénovací množiny, viz obrázek 3.7. Generování je podmíněno nejen textovým obsahem, ale i stylem. Nevýhodou tohoto modelu je, že délka vstupního řetězce je omezena. Generování celých řádků písma tak není možné. Dalším problémem je, že ve vytvořeném písmu se objevují různé vady, například nespojité čáry nebo velmi slabě napsané znaky [30], viz obrázek 3.8.



Obrázek 3.9: Generátor modelu *ScrabbleGAN* [10] je plně konvoluční. To při dostatečně nízkém počtu vrstev způsobí omezení oblasti na výstupu, kterou může jeden prvek vstupního vektoru ovlivnit. Obrázek převzat z [10].

3.4 Generování písma pro libovolně dlouhý text

Předchozí modely jsou při generování písma omezeny maximální délkou očekávaného textového řetězce. Neumožňují tak generovat písmo pro celé věty nebo řádky. Řešení nabízí modely *ScrabbleGAN* [10] a *lineGen* [7], které na vstup plně konvolučního dekodéru vkládají sekvence místo vektorů s pevně zvolenými rozměry.

ScrabbleGAN

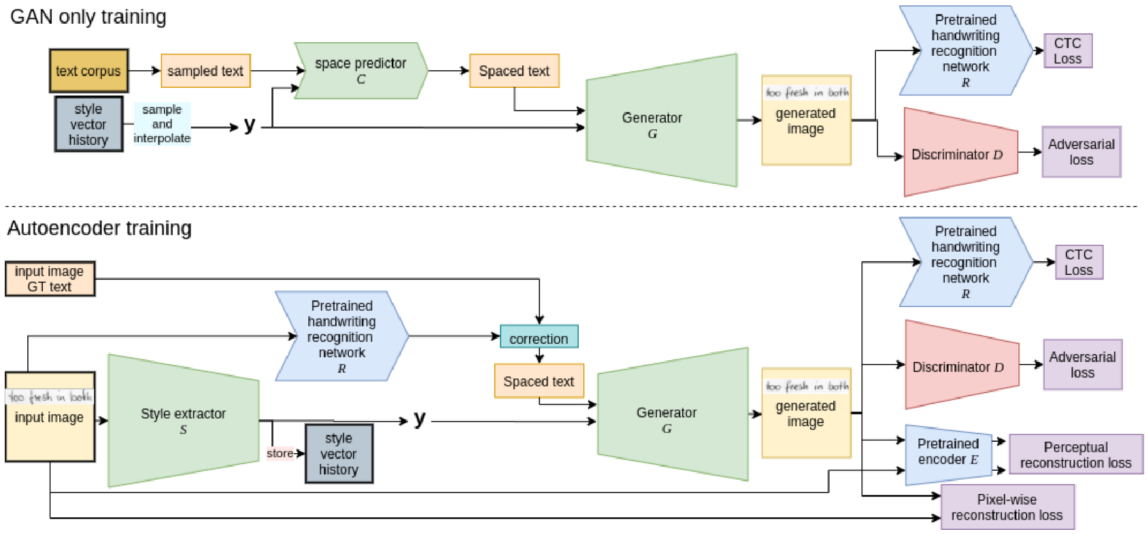
Generativní model *ScrabbleGAN* [10], podobně jako model *GANwriting* [23], přidává další podmínku generování. Kromě očekávaného textového řetězce je vstupem i vektor stylu písma. Tento vektor je však při generování volen náhodně, a není tak možné přímo styl zvolit. Novou vlastností modelu je možnost podmínit generování libovolně dlouhým řetězcem, což rozšiřuje schopnosti modelu [10]. Například dovoluje použít model ke generování celých řádků textu nebo vět.

Model se skládá z generátoru G, diskriminátoru D a sítě pro optické rozpoznávání znaků R. Generátor G i diskriminátor D jsou konvoluční sítě bez rekurentních vrstev [10].

Vstupem G je vektor příznaků F reprezentující vstupní řetězec t. Ten je vytvořen konkatenací naučených příznaků jednotlivých symbolů t [10]. Například pro vstupní řetězec „meet“ platí, že $F = [f_m, f_e, f_e, f_t]$, kde f_m, f_e, f_t jsou vektory příznaků m, e, t. Vektor F je ještě před dalším zpracováním vynásoben náhodným vektorem z, který určuje styl písma. Síť G je plně konvoluční [10]. To způsobuje vzhledem k nízkému počtu vrstev sítě, že každá část vstupního vektoru F ovlivňuje pouze část výstupu, zároveň se však každé sousední vstupní symboly mohou navzájem ovlivnit. Tuto vlastnost modelu ukazuje obrázek 3.9.

Po natrénování je model schopný generovat písmo s různými styly a libovolně dlouhými vstupními řetězci. Zároveň má však i dva značné nedostatky. Styl výstupního písma nemůže být zvolen, je určen náhodně, a každý symbol vstupního řetězce má (i přes možnou lokální interakci mezi sousedními znaky) pevně určenou šířku oblasti, kterou ovlivňuje ve výstupním obrázku. Důsledkem je viditelná vada, kdy se mezi znaky na výstupu objevují nepřirozené mezery (např. mezi dvěma znaky „l“), viz ukázka výstupů na obrázku 3.10.

Obrázek 3.10: Ukázka písma generovaného modelem *ScrabbleGAN* [10]. Jelikož každý znak vstupní sekvence ovlivňuje stejně širokou oblast ve výstupním obrázku, šířka vygenerovaného slova je pro různé styly stejná. Obrázek převzat z [10].



Obrázek 3.11: Architektura celého modelu *lineGen* z článku [7]. Pro vygenerování písma se zvolí styl **y** a spolu se vstupním textem se vytvoří pomocí sítě C zarovnaný text, který je vstupem generátoru G. Horní část obrázku ukazuje trénovaní **pouze GAN** a dolní část ukazuje krok trénování **auto-enkodér** (viz popis trénování). Obrázek převzat z [7].

Model lineGen

Další model řešící generování celých řádků ručně psaného písma je model z článku [7], v jiné literatuře označovaný jako *lineGen*. Podobně jako v modelu *ScrabbleGAN* [10] je generátor plně konvoluční a vstupem je vektor reprezentující vstupní text [7]. Liší se ale ve způsobu, jakým je tento vektor vytvořen. Zatímco *ScrabbleGAN* spojuje do řetězce naučené příznaky jednotlivých znaků, kde každému znaku připadá vždy stejná šířka bez ohledu na očekávanou oblast ve výstupním obrázku, tak model [7] volí šířku příznaků jednotlivých znaků dynamicky. Například pro písmeno „m“ lze čekat, že ve výstupním snímku bude zabírat větší oblast než písmeno „i“. Výsledkem jsou tak vizuálně kvalitnější obrázky.

Model se skládá ze šesti částí [7]: generátoru G, který vytváří obrázky na základě zarovnaného textového řetězce, vektoru stylu a šumu, diskriminátoru D, sítě pro získání stylu z obrázku písma S, sítě pro zarovnání vstupního textového řetězce C, natrénovaného modelu pro rozpoznávání ručně psaného textu R a natrénovaného enkodéru obrázků E. Jejich vzájemné propojení blíže ukazuje obrázek 3.11.

Obrázek 3.12: Srovnání vygenerovaného písma modelů *ScrabbleGAN* [10] (vlevo) a modelu *lineGen* [7] (vpravo). Zatímco první zmíněný model má pro každý znak vstupní sekvence stejně širokou oblast ve výstupním obrázku, tak druhý model volí tuto šířku dynamicky. Obrázek převzat z [7].

Podstatnou novou součástí modelu je C – síť pro zarovnání vstupního textového řetězce. C je 1D konvoluční neuronová síť, jejímž vstupem je řetězec znaků zakódovaných pomocí *one-hot encoding* a vektor stylu [7]. Pro každý znak c_i vstupního řetězce síť C odhaduje šířku mezery před c_i, tj. počet opakování symbolu *blank* před c_i, a šířku c_i ve výstupním obrázku, tj. počet opakování c_i. C je trénovaná na základě anotací trénovací datové sady pomocí ztrátové funkce MSE – střední kvadratické chyby [7].

Trénování jednotlivých modulů je rozděleno do čtyř kroků [7]:

1. **Zarovnání vstupního textu** – pro zvolený obrázek z datové sady je odhadnut styl a následně zarovnání textu v obrázku, následně jsou pomocí střední kvadratické chyby aktualizovány parametry sítí C a S.

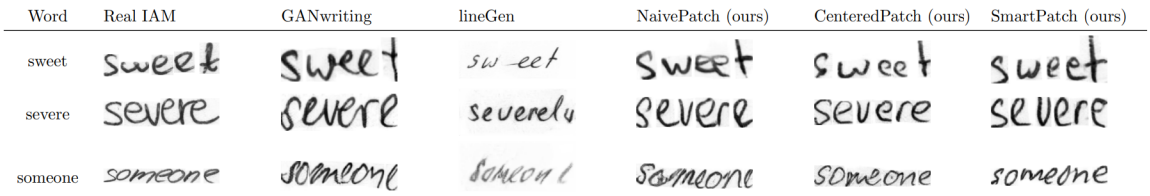
2. **Diskriminátor** – parametry sítě D jsou aktualizovány pro různé interpolace stylů, které jsou použity během jiných kroků trénování.

3. **Pouze GAN** – aktualizace parametrů na základě chybové funkce pro trénování základního modelu GAN.

4. **Auto-enkodér** – je vybrán pár snímků od stejného autora, pomocí sítě S je extrahován styl a následně jsou pro každý ze snímků vypočítány hodnoty ztrátových funkcí po rekonstrukci (L1 mezi originálním a rekonstruovaným obrázkem). Tímto krokem se aktualizují parametry sítí G a S.

Podrobnější popis jednotlivých kroků je k dispozici v původním článku [7].

Natrénovaný model je schopný generovat ručně psané písmo pro libovolně dlouhý vstupní text. Reflektuje vlastnost, že každý znak může mít ve výstupním obrázku rozdílnou šířku s ohledem na styl autora, viz ukázka výstupů modelu na obrázku 3.12. Model byl vyhodnocen skupinou uživatelů. Ti dosáhli přesnosti jen 52.2 % při rozlišování uměle vygenerovaných a reálných příkladů [7]. To ukazuje na vysokou vizuální kvalitu výsledných obrázků.



Obrázek 3.13: Srovnání písem z datové sady IAM [29] vytvořených modelem *GANwriting* [23], modelem *lineGen* [7] a modely z článku [30]. Obrázek převzat z [30].

3.5 Vylepšení kvality generovaného písma

Model *GANwriting* [23] nebo model *lineGen* [7] generují věrohodné obrázky ručně psaného písma, přesto se v nich objevují vady (např. nespojitosti tahů nebo velmi slabě napsané znaky). Novější modely se zaměřují především na odstranění těchto nedostatků a další vylepšení kvality generovaného písma.

Improving Handwritten Word Imitation with Patch Discriminators

Model *SmartPatch* [30] rozšiřuje základní architekturu modelu pro generování ručně psaného písma přidáním dalšího diskriminátoru, který sleduje jen výřezy výstupu generátoru.

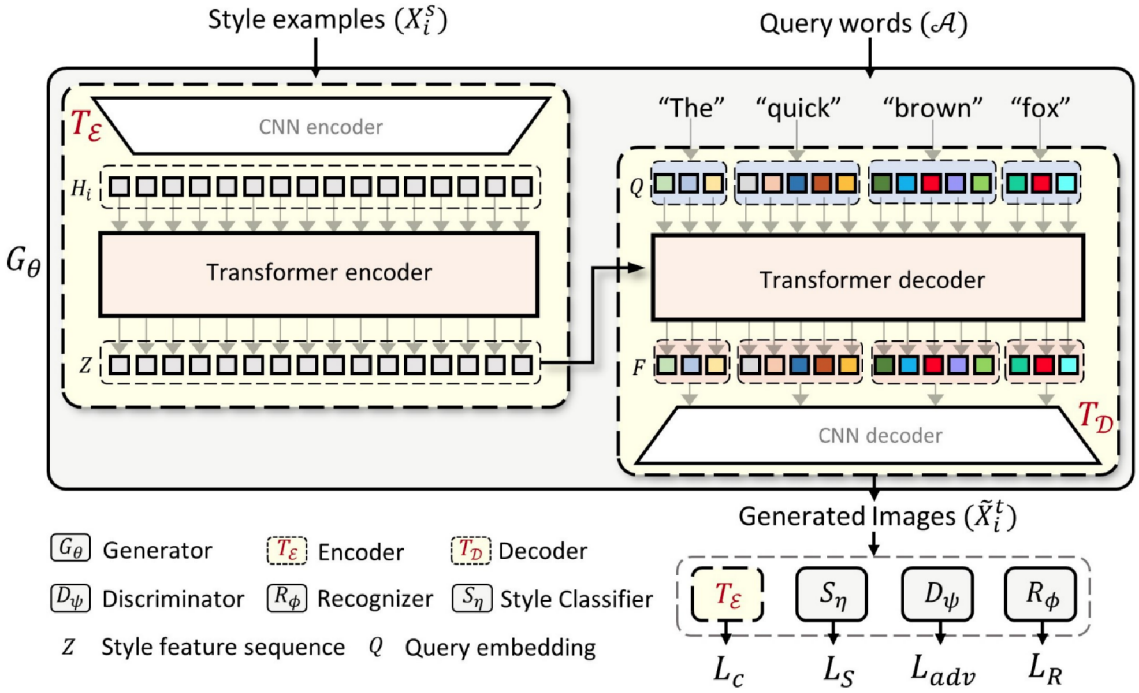
Autoři předpokládají, že vady jsou způsobeny nedostatečnou citlivostí diskriminátoru, který se více zaměřuje na celkový obrázek místo detailů [30]. Navrženým řešením je přidání diskriminátoru, který se zaměřuje na jednotlivé znaky, a tak klade větší důraz na lokální nedostatky.

Nejvhodnější model diskriminátoru na základě experimentů využívá model pro rozpoznání ručně psaného textu, přesněji využívá jeho *attention* vrstvu (viz kapitola 2). Pro každý ze znaků v obrázku je extrahována *attention maska*, která je použita k nalezení ideálního výřezu aktuálního znaku [30]. Jednotlivé výřezy jsou pak spolu s vektorem reprezentující znak posouzeny novým diskriminátorem.

Výsledkem je znatelné zlepšení detailů generovaného písma. Tahy mají konzistentní šířku v rámci celých slov, a nevnikají tak často viditelné vady [30]. Toto dokládá i studie, ve které byli účastníci vyzváni k výběru lépe vyhlížejícího slovu ze dvou možností, kde každá odpovídala výstupu jiného modelu. Písmo generované modelem *SmartPatch* bylo preferováno nad ostatními modely (*GANwriting*, model z článku [7]), dokonce bylo s 53.3 % preferováno nad reálnými příklady z trénovací datové sady IAM [29]. Výsledky generování ukazuje obrázek 3.13.

Handwriting Transformers

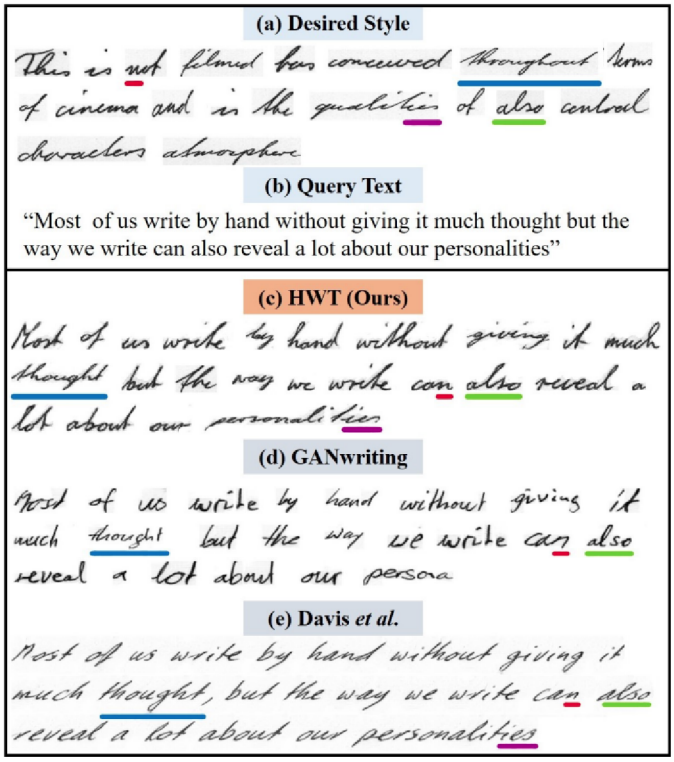
Starší modely (jako jsou *GANwriting* [23] nebo *lineGen* [7]) reprezentují styl písma jako vektor s předem zvolenými rozměry. Model *Handwriting Transformers* (HWT) [2] přišel s alternativním způsobem k předání informace o stylu psaní autora do generování nového písma. Stejně jako u starších modelů je vstupem cílový text a sada obrázků s písmem. Tyto obrázky jsou ale zpracovány pomocí hybridního enkodéru, který se skládá z plně konvolučního enkodéru a transformer enkodéru [2]. Jeho výstupem je pak sekvence příznaků, ze které se následně pomocí *attention* mechanizmu získávají příznaky pro jednotlivá písmena vstupního textu. Ty jsou pak plně konvolučním dekodérem převedeny na výstupní obrázek s písmem [2].



Obrázek 3.14: Schéma modelu *Handwriting Transformers* [2]. Vstupní sada obrázků písma X_i^S je zpracována enkodérem T_ε na sekvenci Z. Dekodér T_D pak pro každý vstupní znak extrahuje pomocí *attention* příznaky F, které jsou převedeny konvolučním dekodérem na výstupní obrázek. Obrázek převzat z [2].

Celý model zachycuje obrázek 3.14. Model lze rozdělit na dvě části: enkodér $T_\varepsilon\colon X_i^s \to Z$ a dekodér $T_D\colon (Z, A) \to \tilde{X}_i^t$. Cílem enkodéru T_ε je extrahovat příznaky Z ze vstupních obrázků X_i^s [2]. Každý obrázek je samostatně zpracován konvolučním enkodérem s cílem snížit dimenze vstupu transformer enkodéru. Příznaky z obrázků jsou spojeny do jedné sekvence H_i, která je následně transformována na Z pomocí transformer enkodéru. Dekodér T_D pro každý znak očekávaného vstupního textu A extrahuje ze sekvence Z příznaky označené F. K sekvenci F je připojena vstupní sekvence Q reprezentující očekávaný text. Každý úsek sekvence je transformován stejnou plně propojenou neuronovou vrstvou na osminásobné množství příznaků (ze sekvence $F \| Q \in T \times 1024$ na sekvenci $H \in T \times 8192$, kde T je délka očekávaného textu a $F \| Q$ je spojení sekvencí F a Q v dimenzi příznaků). Tato transformace umožňuje přímý převod H na příznaky $\tilde{H} \in T \times 512 \times 4 \times 4$, tj. 2D obrázek s 512 kanály, výškou 4 a šířkou $4T$. Příznaky $\tilde{H}$ nakonec zpracovává plně konvoluční dekodér, který vytváří obrázek ručně psaného písma v odstínech šedé [2].

Trénování vedou především tři chybové funkce L_S – klasifikační chyba stylu, L_R – chyba sítě pro optické rozpoznání znaků (OCR) a L_{adv} – chyba GAN ve variantě *hinge loss* [27]. Klasifikační síť a síť OCR jsou trénovány na základě anotací datové sady (tj. pouze s reálnými daty). Při generování je model G trénován s cílem, aby klasifikace stylu generovaného obrázku byla stejná jako klasifikace vstupních obrázků pro extrakci stylu a aby síť OCR predikovala očekávaný vstupní text [2].



Obrázek 3.15: Porovnání výsledků generování modelu *Handwriting Transformers* (HWT) [2], modelu *GANwriting* [23] a modelu *lineGen* (Davis *et al.*) [7]. Výsledky modelu HWT nejvěrněji replikují očekávaný styl písma. Obrázek převzat z [2].

Pro učení je potřeba balancování gradientů z chybových funkcí. Proto autoři normalizují gradienty podle výstupního obrázku ∇L_S a ∇L_R, aby měly stejnou směrodatnou odchylku jako ∇L_{adv} [2]:

$$\nabla L_S \leftarrow \frac{\sigma L_{adv}}{\sigma L_S} \cdot \nabla L_S, \tag{3.4}$$

$$\nabla L_R \leftarrow \frac{\sigma L_{adv}}{\sigma L_R} \cdot \nabla L_R, \tag{3.5}$$

kde σL_{adv} reprezentuje směrodatnou odchylku ∇L_{adv}, σL_S směrodatnou odchylku ∇L_S a σL_R směrodatnou odchylku ∇L_R.

Tento přístup zachycuje věrněji styl písma než předchozí modely, viz obrázek 3.15. Má však i značné nedostatky: výstup generuje po samostatných slovech, nikoli po celých řádcích jako *lineGen* [7] a šířka výstupu je závislá jen na délce vstupu bez ohledu na styl písma.

3.6 Shrnutí

Zmíněné moderní modely pro generování ručně psaného písma mají řadu společných vlastností. Nejpodstatnější je, že všechny vycházejí z Conditional GAN modelu [32]. Generování je ve všech případech podmíněno očekávaným textem na výstupu. Pro vynucení důsledného dodržování této podmínky bývá součástí modelu síť pro rozpoznávání ručně psaného písma. Tuto síť lze využít dopředu natrénovanou [7] nebo ji trénovat současně s ostatními částmi modelu [2, 23].

Novější modely [2, 7, 23] zavádějí druhou podmínku generování – styl písma. Ten má za cíl zachytit různé vlastnosti písma jako tloušťku čar, naklonění a vzhled znaků atd. Vstupem modelu je v těchto případech kromě očekávaného textu i příklad písma ve formě obrázků. Příklady bývají zpracovány sítí, jejíž výstupem je vektor nebo sekvence příznaků reprezentující styl. Modely se liší ve způsobu, jakým je styl přidán do chybové funkce modelu. Prvním způsobem je doplnění modelu o klasifikační síť. Jejím úkolem je odhadnout autora na základě vygenerovaného výstupu [2, 23]. Autor by se neměl lišit od autora, jehož písmo bylo vstupem modelu. Tento přístup není ideální, protože se očekává, že styl byl součástí trénovací datové sady. Druhý přístup je síť navrhnout jako auto-enkodér [7]. V tomto případě je vstupem ukázka písma včetně jejích textového přepisu a očekává se, že generátor vytvoří na základě stylu a textu totožný obrázek.

Další společnou vlastností je architektura generátoru. I když různé modely používají různé způsoby zavedení stylu, nejčastější volbou je použití vrstev AdaIN [7, 23, 30]. Nejvěrohodněji však napodobuje styl model HWT [2], který pro každý symbol extrahuje styl samostatně pomocí *attention*. Všechny generátory jsou konvoluční sítě bez rekurentních vrstev, to i v případě, kdy není vstupní řetězec omezen maximální délkou. Výstup je tak vždy generován jako celek.

Některé modely mají pevně omezenou maximální délku očekávaného textu [23, 30]. Je-li vstupem kratší slovo, než je limit, bývá doplněno znaky reprezentujícími mezeru. Jiné modely využívají vlastností konvolučních vrstev, pro které není nutné dopředu pevně zvolit výšku a šířku vstupního tenzoru reprezentujícího obrázek. Protože počet parametrů konvoluční vrstvy nezáleží ani na šířce, ani na výšce vstupu, záleží pouze na počtu kanálů a rozměrech filtru. Navíc platí, že oblast, kterou může ovlivnit jeden příznak vstupního tenzoru, je omezená a záleží na počtu vrstev, rozměrech filtru a délce kroku. Těchto vlastností úspěšně využívají některé modely [7, 10], které umožňují generování písma na základě libovolně dlouhého textu. Důležitou vlastností tohoto přístupu je, že oblast, kterou ovlivňuje každý příznak vstupního vektoru, je vždy stejně široká. Pokud toto model správně nereflektuje, vznikají pak nepřirozené mezery mezi jednotlivými znaky, viz model *Scrabble-GAN* [10].

Nejnovější modely jsou schopné generovat celé samostatné řádky ručně psaného písma. Nejsou omezeny maximální délkou vstupního řetězce a umožňují na základě ukázky písma specifikovat styl. Předpokládají zcela rovně napsaný text, a tak generují písmo, které vypadá, že je psané na linkovaný papír. V generovaném písmu se objevují nedostatky, např. dlouhé mezery mezi znaky uvnitř slov, velmi slabě napsané znaky nebo nespojité tahy. Model *SmartPatch* [30] se snaží některé z těchto vad odstranit. Do modelu přidává druhý diskriminátor, který pracuje s výřezy výstupního obrázku. Tyto výřezy jsou získávány na základě attention masek sítě pro rozpoznávání ručně psaného písma. Cílem je zaměřit se na detaily jednotlivých písmen. Nevýhodou tohoto modelu je skutečnost, že neumožňuje generování slov s libovolnou délkou.

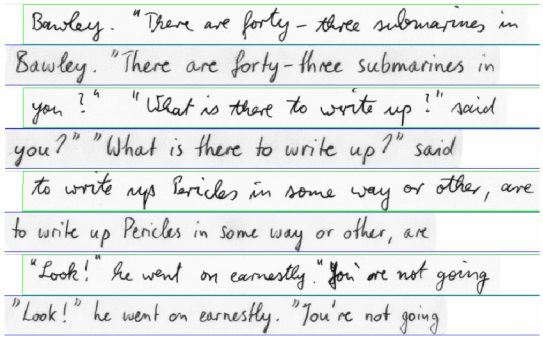
Kapitola 4

Generování řádků písma

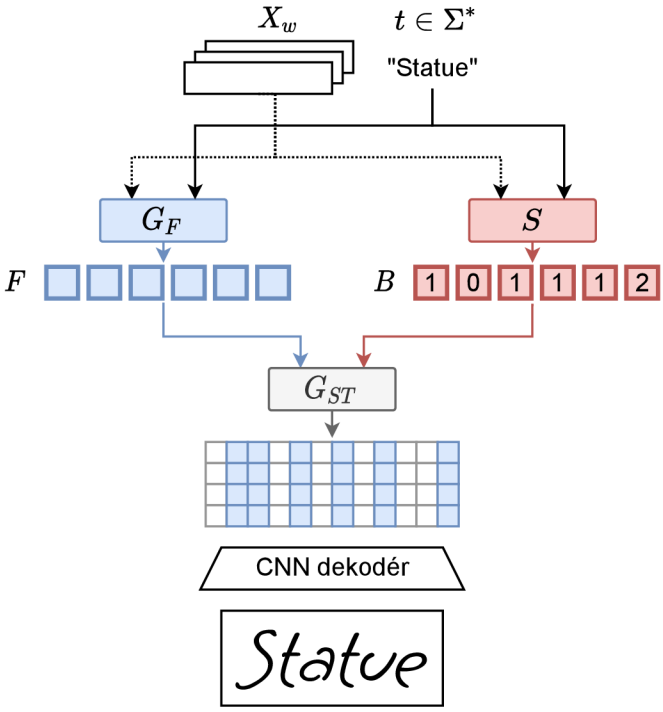
Cílem této práce je generování celých řádků písma, nikoli samostatných oddělených slov. Z existujících modelů toto umožňují modely *ScrabbleGAN* [10] a model *lineGen* [7]. Vyšší kvality výstupů dosahuje druhý ze zmíněných. Má dvě výhody: umožňuje specifikaci stylu ukázkou písma a bere i v úvahu, že v ručně psaném písmu se liší šířka některých znaků (např. šířka písmena „i“ se typicky liší od šířky písmena „m“). Nejvyšší kvality generovaného písma však dosahuje model *Handwriting Transformers* (HWT) [2], který umožňuje generování jen samostatných slov. Rozhodl jsem se proto vytvořit model, který kombinuje obě řešení jak *lineGen*, tak i HWT.

Autoři modelu *lineGen* ukázali, že správné rozmístění příznaků na řádek je zásadní pro napodobení očekávaného stylu písma pro delší vstupní textové řetězce. Bez něj vznikají nepřirozeně široké mezery mezi písmeny nebo písmena jsou roztažena do šířky, aby vyplnila mezery. Model HWT ale ukazuje, že toto není tak velký problém při tvorbě písma pro krátký text, jako jsou samostatná krátká slova [2]. Model spojuje příznaky pro jednotlivá písmena jednoduše za sebe bez mezer. Přesto je konvoluční dekodér schopný v těchto případech vyřešit problém rozmístění znaků a šířky mezer sám, bez explicitního rozmístění příznaků jednotlivých znaků očekávaného řetězce. V generování písma pro delší vstupní textový řetězec ale model selhává. Z tohoto předpokladu vychází i nově navržený model v této práci.

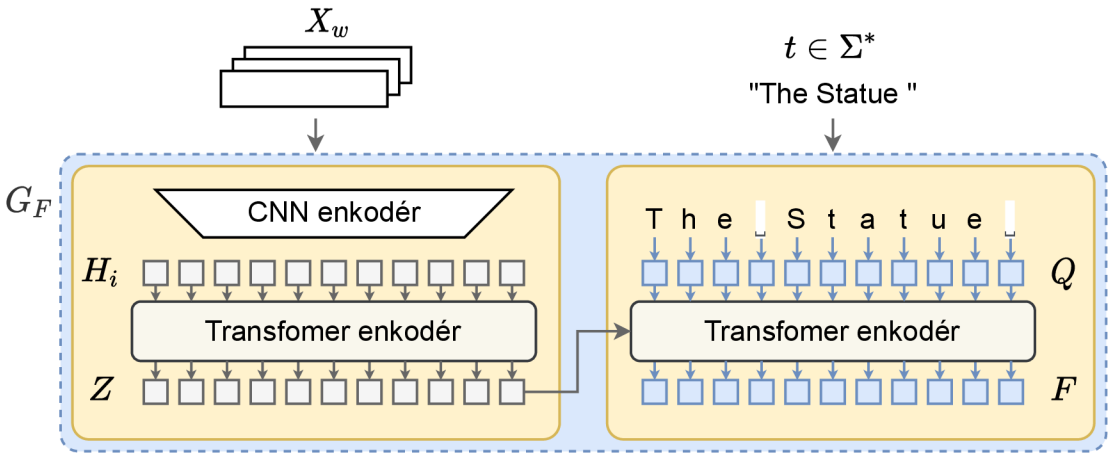
Dalším významným předpokladem je, že model HWT lépe napodobuje očekávaný styl písma díky mechanizmu *attention*, který umožňuje získání relevantních příznaků pro každý symbol vstupního textového řetězce samostatně. V předchozích modelech je styl reprezentován vektorem s konstantní velikostí. Ten je společný pro všechny symboly vstupního řetězce. Aby takový vektor s konstantní velikostí správně reprezentoval styl písma, musí zachytit informace, které jsou společné pro všechny znaky bez ohledu na očekávaný text. Tedy i v případech, kdy se modely omezují jen na tvorbu písma textových řetězců, které zpravidla obsahují jen několik různých znaků abecedy modelu. To zásadně omezuje jejich schopnost zachytit detaily očekávaného stylu některých znaků. Předpokládám, že toto je příčina, proč model *lineGen* [7] generuje téměř všechna písmena tiskacím písmem, i když vzor je napsán psacím písmem, viz obrázek 4.1.



Obrázek 4.1: Příklady, kdy model nereflektuje správně styl v generovaném písmu. Zeleně jsou vyznačeny příklady reálné datové sady a modře jejich rekonstrukce pomocí natrénovaného modelu. Ve vygenerovaných snímcích lze pozorovat, že znaky na sebe nenavazují. Obrázek převzat z [7].



Obrázek 4.2: Návrh celého procesu generování. Ten je podmíněn dvěma vstupy: X_w (množinou obrázků reprezentujících styl) a řetězcem t (očekávaným textem). Nejdříve jsou pomocí sítě G_F získány relevantní příznaky znaků F, následně je pro každý symbol řetězce t sítí S odhadnuta mezera B, která jej předchází, a nakonec část G_{ST} vytvoří 2D obrázek s příznaky, který je zpracován konvolučním dekodérem na obrázek s písmem.



Obrázek 4.3: Pro každý symbol ze vstupní sekvence jsou získány relevantní příznaky zvlášť. Množina obrázků reprezentující styl X_w je zakódována kombinací konvolučního a transformer enkodéru na sekvenci Z. Vstupní řetězec je zakódován pomocí *embedding* vrstvy na sekvenci klíčů Q. Pomocí mechanismu *attention* jsou pak získány relevantní příznaky F příslušící vstupním symbolům. Množství příznaků je tak úměrné počtu vstupních symbolů a neomezuje se na předem pevně stanovený počet. Obrázek inspirován [2].

4.1 Generativní model

Stejně jako předchozí modely pro generování ručně psaného písma, jako jsou např. *GANwriting* [23] nebo *ScrabbleGAN* [10], je součástí navrženého modelu: generátor G, diskriminátor D a model pro optické rozpoznání znaků R (OCR). Dále model obsahuje po vzoru modelu *lineGen* [7] část pro odhad rozmístění znaků S (z angl. *Spacer*). Její výstup závisí na očekávaných symbolech a stylu písma. Poslední částí modelu je síť pro klasifikaci stylu W, která se využívá k vytvoření tlaku na G, aby se snažil napodobit očekávaný styl.

Návrh a funkce generátoru G vychází z modelu HWT [2]. Stejně jako původní model, využívá kombinace plně konvolučních sítí – enkodéru a dekodéru – a sítí typu transformer pro implementaci *multi-head attention*. Liší se však ve vytváření sekvence příznaků, které jsou zpracovány v poslední části G pomocí plně konvolučního dekodéru na výstupní obrázek. Hlavním rozdílem je rozmístění příznaků na základě predikce sítě S. Příznaky jednotlivých symbolů nejsou jednoduše spojeny za sebe, ale jsou rozmazány Gaussovým filtrem a jsou umístěny na pozice. Rozmístění umožňuje generování celých řádků písma bez nepřirozených mezer mezi znaky. Rozmazání filtrem pak dovoluje jemnější přechody mezi pozicemi a v důsledku umožňuje zpětnou propagaci chyby směrem k výpočtu pozice, což u žádného předchozího modelu nebylo možné. Při interpolaci pozic tak nedochází k prudkým změnám, ale přechody jsou jemnější. Celý model zachycuje obrázek 4.2.

Činnost generátoru lze přesněji popsat následovně. Sekvence příznaků F pro symboly očekávaného řetězce $t \in \Sigma^*$ je získána stejně jako v původním modelu HWT (viz obrázek 4.3). Sekvence je ale převedena na tenzor reprezentující 2D obrázek jinak než v původním modelu. Příznaky ze sekvence F jsou transformovány na sekvenci tensorů $\tilde{F}$ o rozměrech $T \times 512 \times 8 \times 1$ (tj. sekvenci 2D obrázků s výškou 8 pixelů, šířkou 1 pixel a 512 kanály). Pro každý prvek $\tilde{f}_i \in \tilde{F}$ je určena cílová pozice p_i na základě odhadu šířky mezer B, které předchází každý symbol vstupního řetězce t. Ty odhaduje síť S na základě očekávaného stylu. Ten je pro účely sítě S reprezentován vektorem, který je získán z vypočítaných příznaků z předposlední vrstvy sítě W pro klasifikaci stylu náhodně vybraného obrázku písma z množiny X_w (množiny obrázků písma od autora w). Cílové pozice p_i jsou využity k výpočtu diskrétních hodnot filtrů K_i, podle kterého jsou f_i přičteny k původně nulovému ST – vstupu konvolučního dekodéru. Pro konkrétní hodnoty prvků sekvence ST platí:

$$ST\left(j\right) = \sum_{\forall i} K_i\left(j\right) f_i, \tag{4.1}$$

kde pro výpočet $K_i\left(j\right)$ platí:

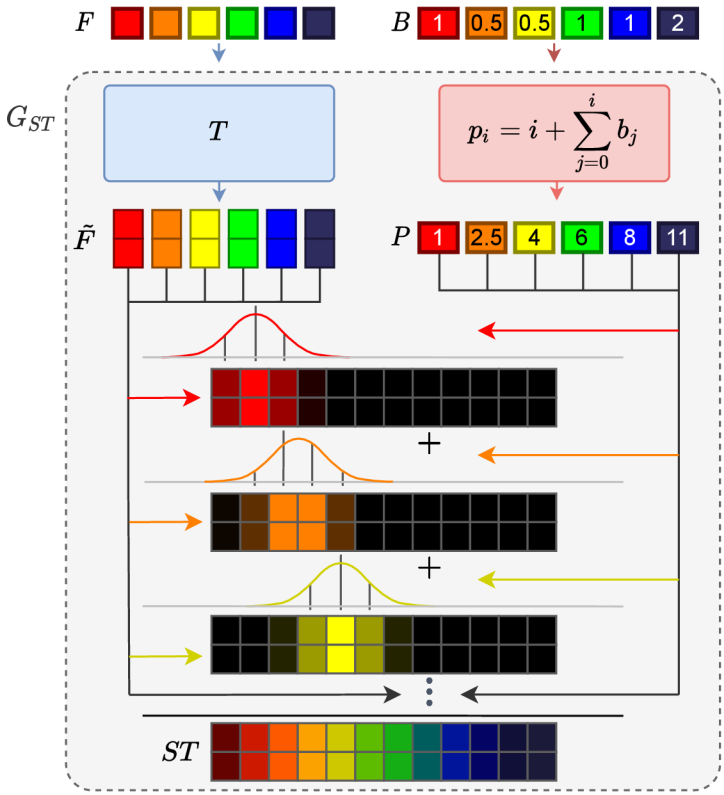
$$K_i\left(j\right) = \begin{cases} \frac{\exp\left(\frac{\left(j-p_i\right)^2}{2\sigma^2}\right)}{\sum_{k=-r}^{r} \exp\left(\frac{\left(j-p_i+k\right)^2}{2\sigma^2}\right)}, & \text{pro } p_i - r \leq j \leq p_i + r, \\ 0, & \text{jinak.} \end{cases} \tag{4.2}$$

Hyperparametry modelu jsou σ, parametr Gaussova filtru, a r, poloměr okolí, které konkrétní příznaky f_i mohou ovlivnit. Obrázek 4.4 znázorňuje tento postup.

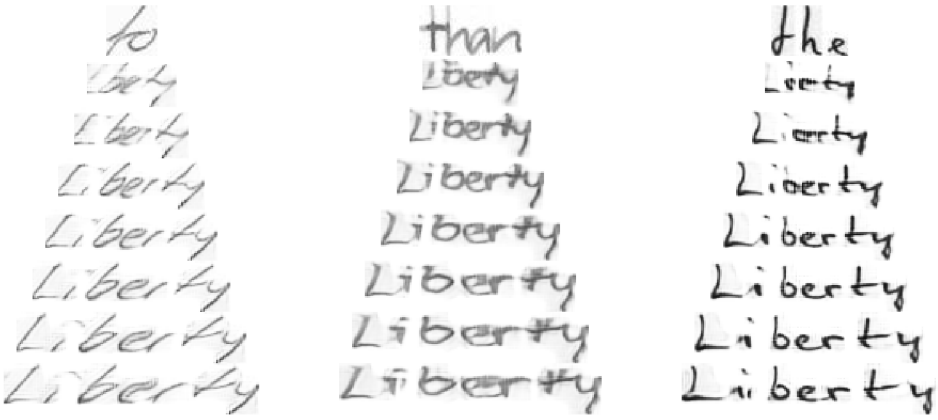
4.2 Model pro odhad rozmístění znaků

Významnou částí navrhovaného modelu je síť S, která pro každý symbol vstupní sekvence odhaduje mezeru před ním. Tímto způsobem lze určit pozice jednotlivých znaků ve výstupním obrázku s písmem. Šířka mezer mezi příznaky navíc určuje i velikost písma. Pokud jsou mezery úzké, příznaky jsou blíže u sebe. Navíc je síť vedena k tomu, aby generovala očekávaný text, nezbývá jí proto jiné řešení než přizpůsobit velikost písma mezeře a generovat ho menší. Současně, jsou-li příznaky dále od sebe, je síť tlačena k tomu, aby prostor vyplnila, a tak generuje větší písmo. Ukázka tohoto chování je znázorněna na obrázku 4.5.

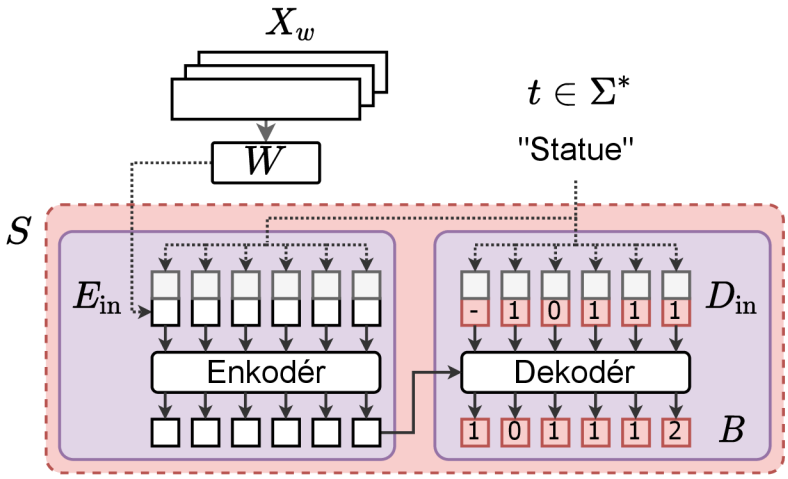
Tento postup pro generování používá i model *lineGen*. Jeho autoři ale implementovali pouze velmi jednoduchou síť S (v článku [7] označenou C, viz kapitola 3.4), která nedosahuje takové přesnosti, jaké by mohla. Vhodnějším řešením je implementace modelu jako sítě typu transformer. Vstupem transformer enkodéru je řetězec t s připojeným vektorem reprezentující styl. Mezery jsou následně autoregresivně určovány na základě vstupních znaků a šířky předchozí mezery. Výstupem je namísto jednoho čísla pro každý znak pravděpodobnostní funkce s možnými hodnotami. Tato změna lépe reflektuje možnost, kdy pro jeden vstupní řetězec a styl existuje více validních rozmístění. Tento přístup je znázorněn na obrázku 4.6. Pro dosažení maximální přesnosti je k vstupům přičteno kódování pozic (angl. *positional encoding*) [41].



Obrázek 4.4: Ukázka činnosti modulu G_{ST}, který vytváří 2D obrázek ST s příznaky pro zpracování konvolučním dekodérem. Příznaky F pro jednotlivé znaky očekávaného řetězce transformuje síť T na úseky obrázku $\tilde{F}$. Ty rozmazané Gaussovým filtrem rozmístí do ST na pozice P vypočítané z predikce šířky mezer před jednotlivými znaky B. Pozice znaků pak ve výsledném obrázku odpovídají pozicím příznaků.



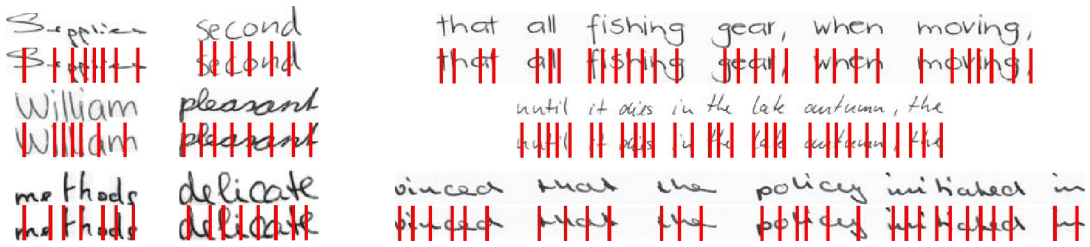
Obrázek 4.5: Rozmístění příznaků přímo ovlivňuje velikost generovaného písma. První řádek každého sloupce ukazuje jeden obrázek z množiny určující styl. Na ostatních řádcích jsou zobrazeny lineární interpolace odhadovaných pozic, tj. $50\,\%, 66.7\,\%, \ldots, 150\,\%$ odhadovaných pozic.



Obrázek 4.6: Autoregresivní model, který pro každý vstupní symbol odhaduje šířku předcházející mezery. Vstupem enkodéru je E_{in} – zakódovaný vstupní řetězec, kde je ke každému znaku připojen stejný vektor reprezentující styl. Ten je získán z aktivací předposlední vrstvy sítě W náhodně zvoleného obrázku z množiny obrázků X_w. Vstupem dekodéru je pak sekvence zakódovaného vstupního řetězce, kde je ke každému znaku připojen předchozí odhad šířky mezery.

Datová sada pozic znaků na řádku

Sít S pro odhad rozmístění znaků je trénována na základě odhadů pozic znaků získaných pomocí OCR modelu R. Tento přístup byl vybrán, protože datové sady ručně psaného písma typicky neobsahují informace o poloze znaků, ale pouze o pozicích slov, vět nebo řádků. Při rozsahu těchto datových sad čítajících desítky tisíc slov a až statisíce znaků by byla ruční anotace pozic znaků velmi náročná a nákladná. Proto je v práci využit alternativní přístup k anotaci – automatický odhad horizontálních pozic znaků na řádku na základě výstupu OCR modelu trénovaného s chybovou funkcí *Connectionist temporal classification* (CTC) [13]. Výstup takového modelu lze interpretovat jako pravděpodobnostní funkci všech možných textových řetězců (dlouhých nejvýše jako výstup modelu) podmíněnou vstupní sekvencí, tj. vstupním obrázkem. Každému úseku vstupní sekvence je přiřazena pravděpodobnostní funkce reprezentující pravděpodobnost, že se na daném úseku nalézá některý znak abecedy nebo tzv. *blank* symbol [13]. Podstatnou vlastností takového výstupu je, že pozice a pravděpodobnosti jednotlivých znaků přibližně odpovídají umístění znaků ve vstupním obrázku. Navíc lze s takovým výstupem pro libovolný možný výstupní řetězec (dlouhý nejvýše jako výstup modelu) odhadnout nejpravděpodobnější pozice jeho znaků. Tohoto je v práci dosaženo hledáním nejpravděpodobnější cesty, která reprezentuje zvolený řetězec v HMM (*Hidden Markov Model*) [34] odpovídajícího výstupu CTC sítě pomocí Viterbi algoritmu [42]. Výsledky této automatické anotace ukazuje obrázek 4.7.



(a) Slova (b) Řádky

Obrázek 4.7: Ukázka výsledků automatické anotace pozic znaků ve slovech a řádcích z datové sady IAM [29] pomocí natrénovaného OCR modelu s chybovou funkcí *CTC loss* [13]. Červené linky vyznačují predikovanou horizontální pozici. Vlevo (a) jsou znázorněny ukázky odhadů pozic pro samostatná slova a vpravo (b) jsou příklady určení pozic znaků na řádcích.

4.3 Trénování generátoru

Návrh vychází ze dvou modelů – *lineGen* [7] a HWT [2]. Podobně i trénování modelu v této práci je velmi blízké trénování zmíněných modelů. Přesto se od postupu v článcích mírně odlišuje. Liší se i od zveřejněné implementace[1] modelu HWT, která se od popisu trénování v článku také mírně liší.

Stejně jako u modelu HWT trénování generátoru G je založeno na minimalizaci tří chybových funkcí: L_S – klasifikační chyby stylu, L_R – chyby sítě pro optické rozpoznávání znaků (OCR) a L_{adv} – chyby GAN ve variantě *hinge loss* [27]. Ve zveřejněné implementaci modelu HWT se nalézá chyba při výpočtu L_R. Vstupem funkce pro její výpočet jsou výstupy vrstvy, které neodpovídají logaritmům pravděpodobností, tj. nepatří do $(-\infty; 0\rangle^n$, ale patří do $\mathbb{R}^n$. Tato chyba vede mimo jiné i na záporné hodnoty L_R. Proto probíhá trénování modelu z této práce s upraveným výpočtem výstupů R, který skutečně odpovídá logaritmům pravděpodobností. Výstupy poslední vrstvy jsou navíc transformovány funkcí LogSoftmax. Tato změna však není zásadní a nevede ke značným rozdílům kvality výstupů modelu, viz kapitola 5.2.

Významnějším rozdílem proti zveřejněné implementaci HWT je upravená trénovací smyčka. Ta je v původní implementaci rozdělena na čtyři samostatné kroky:

1. aktualizace parametrů G na základě L_{adv} a L_R^{fake} (vynechána každou lichou iteraci),
2. aktualizace parametrů D a R na základě L_{adv} a L_R^{real},
3. aktualizace parametrů G na základě L_{adv} a L_S^{fake} (vynechána každou lichou iteraci),
4. aktualizace parametrů D a W (sítě pro klasifikaci stylu) na základě L_{adv} a L_S^{real}.

Takto dochází k aktualizacím na základě L_R a L_S v odlišných krocích, což vede na pomalejší a horší trénování. Pro rychlejší a lepší trénování je vhodnější spojení 1. a 3. kroku pro aktualizaci parametrů G a 2. a 4. kroku pro aktualizaci parametrů D, R a W.

[1] Dostupné z https://github.com/ankanbhunia/Handwriting-Transformers.

Normalizace gradientů ∇L_S a ∇L_R ve zveřejněné implementaci se liší od popisu, který je v článku [2] (viz kapitola 3.5). Pro aktualizaci parametrů je využit součet původních gradientů a jejich normalizovaných variant:

$$\nabla L_S \leftarrow \nabla L_S + 2 \cdot \frac{\sigma L_{adv}}{\sigma L_S} \cdot \nabla L_S, \tag{4.3}$$

$$\nabla L_R \leftarrow \nabla L_R + 2 \cdot \frac{\sigma L_{adv}}{\sigma L_R} \cdot \nabla L_R, \tag{4.4}$$

kde σL_{adv} reprezentuje směrodatnou odchylku ∇L_{adv}, σL_S směrodatnou odchylku ∇L_S a σL_R směrodatnou odchylku ∇L_R. Výsledky generování jsou tak v průběhu trénování kvalitnější. Podobně jako normalizace se liší rychlost učení popsaná v článku [2] a použitá v implementaci. Trénování s hodnotou $5 \cdot 10^{-5}$ použitou ve zveřejněné implementaci vede na model, který lépe napodobuje očekávaný styl než model, který byl trénován s rychlostí učení $2 \cdot 10^{-4}$.

Síť S je trénována odděleně od zbytku G. Její predikce není nutná, protože během učení sítě jsou vybírány pouze ty řetězce a styly, které se nalézají v datové sadě, pozice jejich znaků jsou proto při použití konkrétního stylu předem známé. Pro trénování sítě S je využita chybová funkce křížová entropie. Očekávané hodnoty jsou získány z datové sady a vektory reprezentující styly jsou extrahovány z předposlední vrstvy natrénované sítě W (sítě pro klasifikaci stylu). Celý model G včetně části S je využíván pouze při inferenci.

Kapitola 5

Experimenty

Navržený model je vyhodnocen s použitím datové sady IAM [29] (viz kapitola 3.1) na základě experimentů jak se samostatnými slovy pro srovnání s modelem *Handwriting Transformers* (HWT) [2] (viz kapitola 3.5), tak celými řádky textu pro srovnání s modelem *lineGen* [7] (viz kapitola 3.4). Pro experimenty se slovy byly použity stejné výřezy a zvoleno stejné rozdělení sady na trénovací a validační jako v článku k modelu HWT [2]. Pro experimenty s řádky byly získány výřezy podle anotací datové sady IAM. Pro zachování přibližně stejné velikosti písma v rámci každého formuláře byly nejdříve výřezy řádků rozšířeny shora a zespodu bílou barvou a následně byly zmenšeny se zachováním poměrů stran tak, aby měly výšku 32 pixelů. Bez této úpravy se písmo z některých formulářů výrazně lišilo velikostí. Typicky se jednalo o případy, kdy jeden řádek obsahoval písmeno „y“ a druhý řádek neobsahoval žádná písmena s přesahem linky.

Implementace modelu vychází ze zveřejněné implementace[1] modelu HWT. Pro účely této práce byl však kód výrazně přepracován, aby lépe reflektoval nároky experimentů. Navíc byla značně optimalizována implementace balancování gradientů více chybových funkcí a zároveň bylo několik dalších přidáno. Většina kódu se tak výrazně liší od původní implementace. Kód je napsán v programovacím jazyce Python 3 s pomocí knihovny PyTorch[2]. Trénování probíhalo na grafické kartě RTX 2070 Super. Průměrná doba trénování každého modelu byla přibližně dva dny.

5.1 Rozmístění znaků

Hlavním cílem této práce je generování celých řádků textu, proto následuje několik experimentů s navrhovaným modelem, který kombinuje model HWT [2] a specifické vytváření vstupu konvolučního dekodéru z modelu *lineGen* [7]. Navíc je vyhodnocen i přístup s dodatečným rozmazáním příznaků a jsou porovnány různé architektury modelu pro odhad rozmístění znaků.

[1] Dostupné z https://github.com/ankanbhunia/Handwriting-Transformers.
[2] Dostupné z https://pytorch.org/.

Styl	Bez rozmístěním		S rozmístění	
returned	Statue	positive	Statue	positive
school	Statue	positive	Statue	positive
boating	Statue	positive	Statue	positive
movement	Statue	positive	Statue	positive
should	Statue	positive	Statue	positive

Tabulka 5.1: Porovnání výsledků generování původním modelem HWT [2] (bez rozmístění) a modelem rozšířeným o rozmístění příznaků. V prvním sloupci jsou ukázky z množin obrázků specifikujících styly. Oběma modely byla vygenerována slova „Statue“ a „positive“ se zvolenými styly. Výsledky obou modelů jsou srovnatelně kvalitní. Model bez rozmístění, na rozdíl od druhého modelu, však generuje pro stejný vstupní řetězec slova stejně dlouhá.

Model	IV-S	IV-U	OOV-S	OOV-U
HWT [2]	106.97	108.84	109.45	114.10
HWT (optimalizovaný)	109.33	109.95	109.20	110.18
HWT s rozmístění příznaků podle *lineGen* [7]	**100.31**	**101.55**	**100.58**	**101.34**

Tabulka 5.2: Porovnání kvality generovaného písma metrikou FID [16] modelů trénovaných na datové sadě IAM [29]. Výsledky jsou rozděleny podle generovaných slov (IV – *In vocabulary*, slova z trénovací sady a OOV – *Out of vocabulary*, slova, která nejsou v trénovací sadě) a použitých stylů (S – styly z trénovací sady a U – styly, které nejsou v trénovací sadě). Rozdělení odpovídá použitému v článcích k modelu GANwriting [23] a HWT [2]. Výsledky optimalizovaného HWT jsou srovnatelné ve všech měřeních, pouze v měření OOV-U dosahuje viditelného zlepšení (optimalizace jsou popsány v podkapitole 5.2). Nejlépe však dopadl model HWT s rozmístěním příznaků, ve všech měřeních překonává předchozí modely.

Generování slov s rozmístěním příznaků

Model HWT nepracuje s explicitním umístěním symbolů v generovaném písmu, specifikuje pouze pořadí tak, že příznaky odpovídající jednotlivým znakům spojuje jednoduše za sebe a následně je zpracovává plně konvolučním dekodérem. Oblast, kterou mohou příznaky konkrétního znaku ovlivnit, je tak předem určena, záleží pouze na pořadovém čísle znaku ve vstupním řetězci. Pro generování krátkých slov toto nepředstavuje významný problém, protože konvoluční dekodér je schopný příznaky během transformací uvnitř sítě mírně posouvat. Přesto pro delší slova nemusí být takové kompenzace dostatečné a už autoři modelu *lineGen* ukázali při porovnání s modelem *ScrabbleGAN* [10], že rozmístění příznaků při použití konvolučního dekodéru je zásadní pro dosažení věrohodného rozmístění znaků ve výstupním obrázku písma.

Po zavedení rozmístění příznaků na zvolené pozice, které odpovídají jejich očekávaným pozicím ve výstupním obrázku, je model stále schopný generovat písmo pro samostatná slova ve srovnatelné kvalitě jako původní model HWT, viz ukázky v tabulce 5.1 a hodnoty FID [16] v tabulce 5.2. Nový přístup navíc umožňuje ovlivnit, kromě pozic písmen ve slovech, i velikost písma. Při použití původního modelu HWT toto nebylo možné. Pokud jsou příznaky znaků velmi blízko, jsou znaky na výstupním obrázku menší, ale pokud jsou rozmístěny dále od sebe, jsou znaky výrazně větší.

Model	IV-S	IV-U	OOV-S	OOV-U
HWT s rozmístění příznaků podle *lineGen* [7]	12.66	12.64	12.68	12.70
HWT s rozmazáním příznaků Gaussovým filtrem	**6.43**	**6.60**	**6.42**	**6.59**

Tabulka 5.3: Porovnání kvality generovaných řádků písma metrikou FID [16] modelů trénovaných na datové sadě IAM [29]. Výsledky jsou rozděleny podle generovaného textu (OOV – *Out of vocabulary*, IV – *In vocabulary*) a použitých stylů (S – styly z trénovací sady a U – styly, které nejsou v trénovací sadě). Z výsledků vyplývá, že nejkvalitnější písmo generuje model, který používá Gaussův filtr pro rozmazání příznaků.

Generování řádků písma

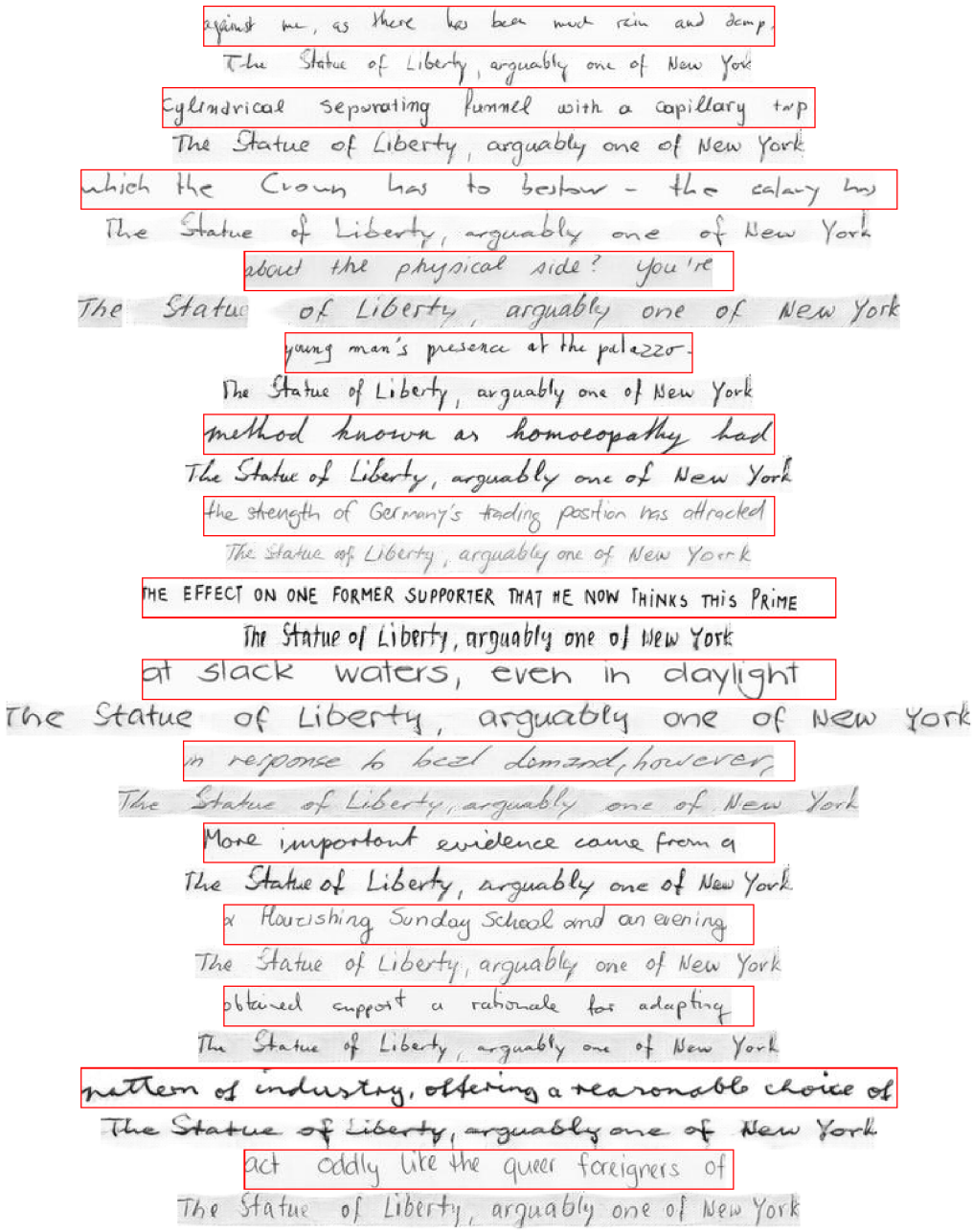
Původní model HWT je schopný generovat pouze samostatná slova. Pro vytváření řádků ve svém článku autoři pouze vkládají předem zvolenou mezeru mezi jednotlivá slova. V důsledku pak styly a šířky mezer a související rozmístění slov na řádku vůbec nereflektují očekávaný styl. Například, pokud má písmo specifické pozadí, vznikají nepřirozené bílé mezery.

Při trénování s datovou sadou obsahující řádky z datové sady IAM namísto slov, není model HWT schopný generovat realistické písmo a vůbec nereflektuje očekávaný styl. Při pokusech generovaní písma pro delší textové řetězce modelem natrénovaným pro generování samostatných slov je sice generováno písmo, které do jisté míry napodobuje očekávaný styl a je realistické, ale objevují se očekávané problémy, tj. nepřirozeně široké mezery mezi písmeny a roztažené znaky do šířky. Tyto problémy způsobuje příliš jednoduché vytváření vstupu konvolučního dekodéru, kdy jsou příznaky pro jednotlivé symboly vstupního řetězce spojeny za sebe bez mezer.

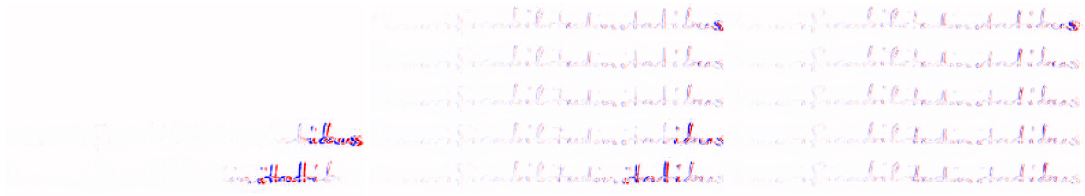
Po zavedení jednoduchého rozmístění příznaků na zvolené pozice, které odpovídají jejich očekávaným pozicím ve výstupním obrázku, je model schopný generovat nejen písmo pro samostatná krátká slova jako původní model, ale také pro celé řádky textu. Rozmístění příznaků na pozice je v jednodušší variantě dosaženo přidáním mezer, tedy nulových příznaků, stejně jako v modelu *lineGen*. V komplexnější variantě jsou příznaky po rozmístění rozmazány Gaussovým filtrem. Výsledky ukazují nejen, že rozmístění příznaků oběma způsoby dovoluje generování celých řádků textu, ale také jeho důležitost pro generování písma s delšími textovými řetězci, jako jsou řádky. Výsledky měření metrikou FID shrnuje tabulka 5.3. Ukázky vygenerovaných řádků s různými styly jsou znázorněny na obrázku 5.1.

Interpolace pozic

V základním přístupu k rozmístění příznaků symbolů, kdy jsou příznaky pouze doplněny mezerami, nejsou interpolace mezi různými rozmístěními spojité, protože tento přístup v principu vyžaduje zaokrouhlování. Při pouhém vkládání mezer neexistuje přístup, jak reprezentovat spojité pozice, když hodnoty mohou být zapsány pouze na celočíselné indexy sekvence. Významným důsledkem je skutečnost, že při použití tohoto postupu není možné při zpětném šíření chyby z L_S (chyby klasifikace stylu), L_{adv} (chyby GAN) a L_R (chyby sítě pro optické rozpoznání znaků) vypočítat jejich gradient vůči predikované pozici. Proto je síť pro odhad rozmístění symbolů trénována odděleně chybovou funkcí *MSE* na základě datové sady, a zůstává tak potenciální přínos jiných chybových funkcí nevyužitý.



Obrázek 5.1: Ukázka generovaných řádků modelem s rozmístěním příznaků symbolů včetně rozmazání Gaussovým filtrem. Ukázky jsou rozděleny do dvojic řádků, kde vždy první z nich (v rámečku) určuje styl a druhý s textem „The Statue of Liberty, arguably one of New York“ je generován navrhovaným modelem. Model napodobuje očekávaný styl a generuje celé řádky.



(a) Bez filtru (b) Gaussův filtr s $r = 2$ (c) Gaussův filtr s $r = 7$

Obrázek 5.2: Srovnání kvality interpolací pozic symbolů ve slově „Honorificabilitudinitatibus“ v závislosti na přístupu a parametrech rozmístění příznaků symbolů ve vstupu konvolučního dekodéru. Na každém řádku jsou zobrazeny diference dvou po sobě jdoucích výsledků interpolace mezi násobky odhadovaných pozic (1. řádku – 102 % − 100 %, 2. řádku – 104 % − 102 %, ...). První obrázek (a) odpovídá řešení z modelu *lineGen* [7], druhý obrázek (b) odpovídá postupu s aplikací Gaussova filtru o poloměru $r = 2$, tj. násobky každých příznaků odpovídající jednotlivým symbolům jsou přičteny i ke dvěma nižším a dvěma vyšším indexům. Obrázek (c) odpovídá postupu s aplikací Gaussova filtru se stejným parametrem σ jako v (b), ale příznaky ovlivňují namísto dvou sedm nižších a vyšších indexů. Je zřejmé, že diference jsou vyrovnanější při použití filtru, a interpolace jsou tak plynulejší.

Rozmazání příznaků Gaussovým filtrem, které je podrobněji popsané v kapitole 4.1, významně zlepšuje interpolace mezi různými rozmístěními symbolů. Dovoluje jemněji specifikovat pozice jednotlivých symbolů a navíc i počítat gradient všech chybových funkcí vůči odhadovaným pozicím, a tak je potenciálně využít k optimalizacím modelu pro odhad rozmístění symbolů. Toto potenciální využití není v práci ověřeno, ale může být součástí dalšího výzkumu. Porovnání kvality interpolací znázorňuje obrázek 5.2.

Přístupy k odhadu rozmístění znaků

Cílem tohoto experimentu bylo nalézt nejvhodnější přístup k odhadu pozic písmen ve slovech a na řádcích. Rozmístění záleží nejen na znacích, ale i na stylu písma. Píše-li autor malým písmem, budou znaky sobě blíž a naopak, píše-li velkým, budou mezery mezi znaky širší.

Styl písma proto uvažuje i model *lineGen* [7] (viz kapitola 3.4), jehož část pro odhad mezer mezi znaky je základním řešením, ze kterého navrhovaný model vychází a s kterým jsou výsledky srovnávány. Tato část modelu *lineGen* se skládá pouze ze čtyř konvolučních bloků s normalizacemi a aktivačními funkcemi. Jejím vstupem je sekvence znaků s připojeným vektorem reprezentujícím styl a výstupem je odhad šířky mezery před každým znakem.

Nevýhodou přístupu využitého v modelu *lineGen* je skutečnost, že je jen minimálně stochastický. Jedinými prvky náhody jsou vrstvy *dropout*. Proto tento model dostatečně nereflektuje existenci více validních rozmístění znaků pro jeden styl a vstupní řetězec. Navrhovaným řešením je změna formátu výstupu sítě. Místo jednoho čísla pro každý znak vstupního řetězce je výstupem pravděpodobnostní funkce s možnými hodnotami. Toto řešení pro trénování proto nepoužívá ztrátovou funkci *MSE* – střední kvadratickou chybu, ale *cross-entropy* – křížovou entropii.

Model	MSE
Regresní model z *lineGen* [7]	0.99
Klasifikační model podle *lineGen*	1.04
Klas. transformer model s pozičním kódováním	**0.93**

Tabulka 5.4: Srovnání výsledků testování modelů pro odhad rozmístění symbolů. Klasifikační varianta modelu z *lineGen* [7] dosahuje horších výsledků než původní model, ale klasifikační transformer model dosahuje vyšší přesnosti než regresní model, a navíc je stochastický.

Nevýhodou tohoto přístupu je však skutečnost, že pravděpodobnostní funkce šířky mezery před znakem není závislá na zvolené hodnotě šířky mezery pro předchozí znak. Z experimentů s rozmístěním znaků vyplývá, že tato mezera určuje i velikost písma. Proto na konkrétně zvolených šířkách mezer předchozích symbolů záleží. Pokud je na začátku zvolena určitá velikost písma, měla by zůstat konzistentní. Pro modelování této vlastnosti je možné implementovat síť jako autoregresivní.

Vyhodnoceny byly tři modely, nejlepších výsledků dosáhl model používající síť typu transformer, viz kapitola 4.2. Modely byly hodnoceny na základě testovací datové sady metrikou *MSE*. Výsledky všech testovaných modelů obsahuje tabulka 5.4.

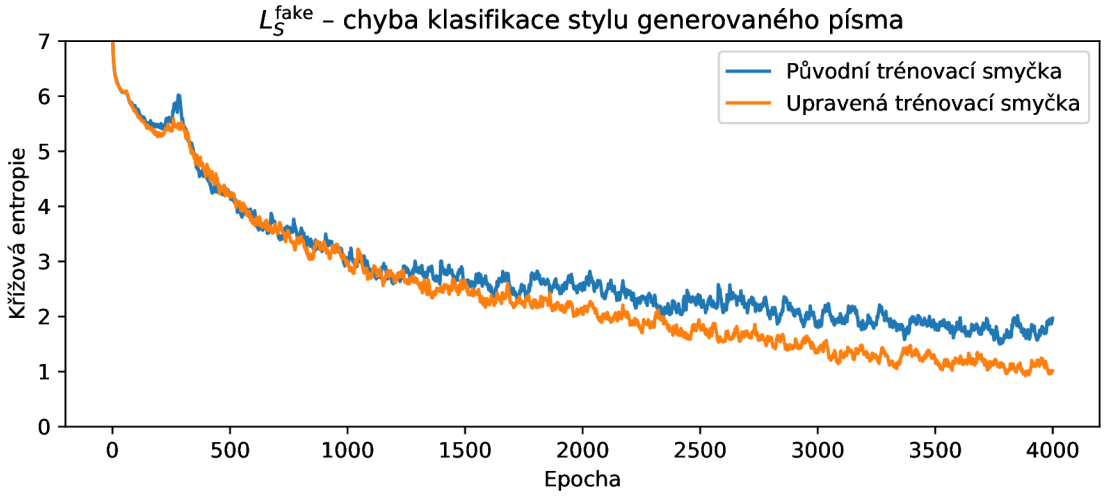
5.2 Optimalizace modelu Handwriting Transformers

Schopnost modelu *Handwriting Transformers* (HWT) [2] napodobovat očekávaný styl je ve srovnání s jinými modely větší, ale popis modelu v článku [2] ne zcela odpovídá zveřejněné implementaci. Liší se například používaná rychlost učení nebo přístup k balancování gradientů. Navíc se v implementaci nalézá i chybný výpočet vstupů chybové funkce CTC loss [13]. Proto jsou v této části vyhodnoceny různé změny a jejich důsledky.

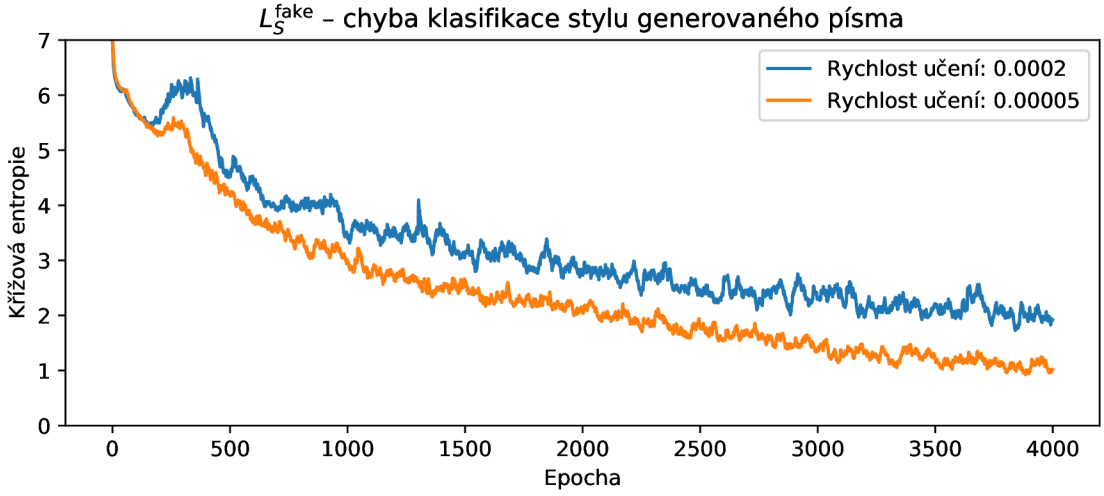
Sjednocené trénování generátoru a diskriminátoru

Původní implementace modelu HWT dělila trénovací smyčku na čtyři kroky, ve kterých byla oddělena aktualizace parametrů modelu na základě chybových funkcí L_S, klasifikační chyby stylu, a L_R, chyby sítě pro optické rozpoznávání znaků (OCR), viz kapitola 4.3. Cílem tohoto experimentu bylo porovnání průběhů původního trénování a trénování se sjednocenou aktualizací parametrů na základě gradientů získaných z obou chybových funkcí.

Po trénování s téměř totožnými parametry jako v původním článku bylo dosaženo po 4 000 epochách nižší hodnoty chybové funkce L_S^{fake}. Od původního článku se liší rychlost učení, namísto $2 \cdot 10^{-4}$ byla v experimentu použita hodnota $5 \cdot 10^{-5}$, viz experiment *Porovnání rychlostí učení*. Namísto hodnoty L_S^{fake} 1.717 bylo dosaženo hodnoty 1.065, hodnoty ostatních chybových funkcí zůstaly téměř totožné. To znamená, že model o něco lépe napodobuje očekávaný styl. Celý průběh této chybové funkce je na obrázku 5.3. Nejen, že tak trénování dosahuje lepších výsledků po stejném množství iterací, ale i dosažení 4 000 epoch trvá o 40 % času méně.



Obrázek 5.3: Porovnání průběhů L_S^{fake}, klasifikační chyby stylu generovaného písma, s původní trénovací smyčkou a upravenou, ve které je sjednocen výpočet gradientu a jeho balancování všech chybových funkcí. Je zřejmé, že trénování s upravenou smyčkou vede na model, který lépe napodobuje očekávaný styl.



Obrázek 5.4: Rychlost učení má značný vliv na schopnost modelu napodobovat očekávaný styl. S nižší rychlostí učení dosahuje model nižších hodnot L_S^{fake} v téměř celém průběhu trénování.

Porovnání rychlostí učení

Autoři modelu HWT ve svém článku [2] uvádějí, že pro optimalizaci parametrů modelu používají algoritmus *Adam* [25] s rychlostí učení $2 \cdot 10^{-4}$. Tato hodnota se však liší od hodnoty $5 \cdot 10^{-5}$ používané ve zveřejněné implementaci. Cílem tohoto experimentu bylo sledovat vliv této změny a vyhodnotit obě možnosti.

Během experimentu zůstaly ostatní parametry modelu a trénování beze změny. Na průběhy hodnot chybových funkcí L_R (chyby sítě pro optické rozpoznávání znaků) a L_{adv} (chyby GAN) neměla rychlost učení viditelný vliv. Mezi průběhy hodnot chybové funkce L_S (klasifikační chyby stylu) byl však rozdíl znatelný. Při rychlosti $2 \cdot 10^{-4}$ bylo dosaženo po 4 000 epochách hodnoty 1.914 zatímco při rychlosti $5 \cdot 10^{-5}$ jen hodnoty 0.977. Srovnání celých průběhů chybové funkce L_S s oběma rychlostmi je znázorněno na obrázku 5.4.

Pro učení modelu je proto mnohem vhodnější použití nižší rychlosti učení $5 \cdot 10^{-5}$. Takto trénovaný model dosahuje věrnějšího napodobování stylu bez zhoršení ostatních charakteristik.

Korekce vstupů chybové funkce CTC

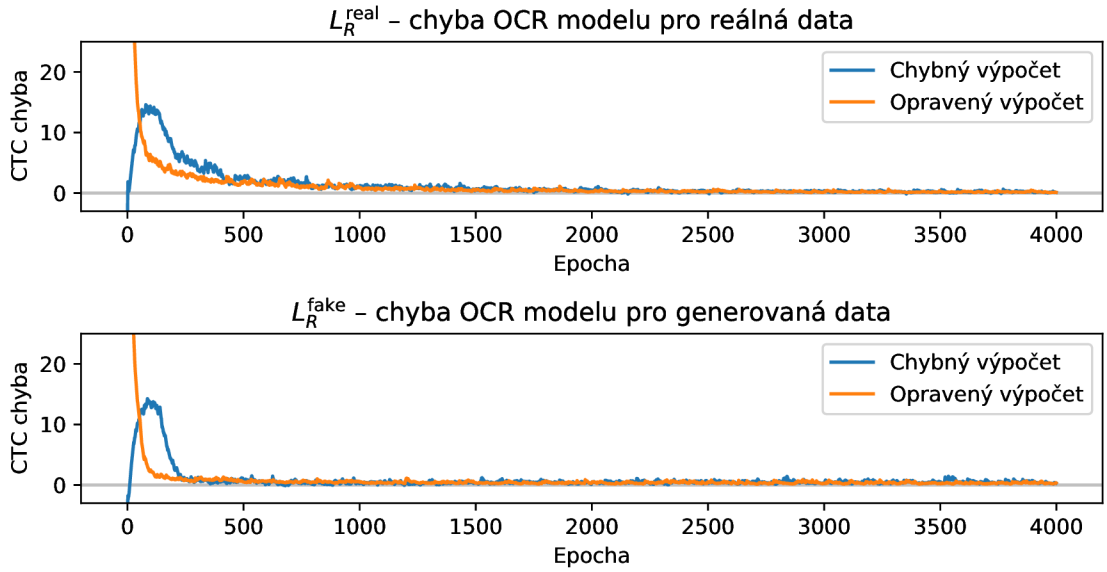
Cílem tohoto experimentu bylo sledování vlivu chyby ve zveřejněné implementaci sítě pro optické rozpoznávání znaků R, jejíž výstupy náležely do $\mathbb{R}^n$ místo do očekávaného podprostoru $(-\infty, 0\rangle^n$ odpovídajícího logaritmům pravděpodobností. Tato chyba se projevovala především skutečností, že chybová funkce občas nabývala při trénování záporných hodnot. Chyby byla odstraněna dodatečnou transformací výstupů R pomocí funkce

$$\text{LogSoftmax}\,(x_i) = \log\left(\frac{e^{x_i}}{\sum_j e^{x_j}}\right). \tag{5.1}$$

Po této změně lze interpretovat původní výstupy R jako logity a po jejich transformaci jako jim odpovídající logaritmy pravděpodobností.

Mezi důsledky této úpravy patří nižší šumění hodnot L_R, trénování zároveň nezačíná prudkým nárůstem hodnot od nuly následovaným postupným poklesem, ale rychlým poklesem z vysokých hodnot, viz obrázek 5.5. Změna neměla na kvalitu generovaných obrázků natrénovaným modelem viditelný vliv, pouze v počátcích trénování bylo generováno písmo s čitelným textem o něco dříve.

Předpokládám, že důvodem minimálních změn v chování modelu jsou vlastnosti funkce LogSoftmax. Ta se skládá z logaritmu a funkce Softmax, tedy funkcí, které jsou obě ryze rostoucí na celém svém definičním oboru. Znaménka všech prvků ∇L_R tak zůstávají beze změny, což je pravděpodobně dostatečné pro vytvoření užitečného signálu pro trénování generátoru G.



Obrázek 5.5: Porovnání průběhů L_R^{real} a L_R^{fake} s nekorektními vstupy chybové funkce *CTC loss* [13] a opravenými vstupy. Chybný výpočet vede i na záporné hodnoty a atypické průběhy hodnot chybových funkcí, zatímco při použití opraveného výpočtu mají hodnoty očekávaný průběh, tj. pouze postupný pokles a ustálení.

Přístupy k balancování gradientů

V článku [2] autoři HWT tvrdí, že balancování gradientů různých chybových funkcí při učení pomáhá trénování. Implementace balancování (viz rovnice (5.2)) se však liší od popisu v článku (viz rovnice (5.3)). Nenormalizované gradienty nejsou v implementaci nahrazeny normalizovanými variantami, ale jsou k nim přičteny.

$$\nabla L_\bullet \leftarrow \nabla L_\bullet + 2 \cdot \frac{\sigma L_{adv}}{\sigma L_\bullet} \cdot \nabla L_\bullet, \tag{5.2}$$

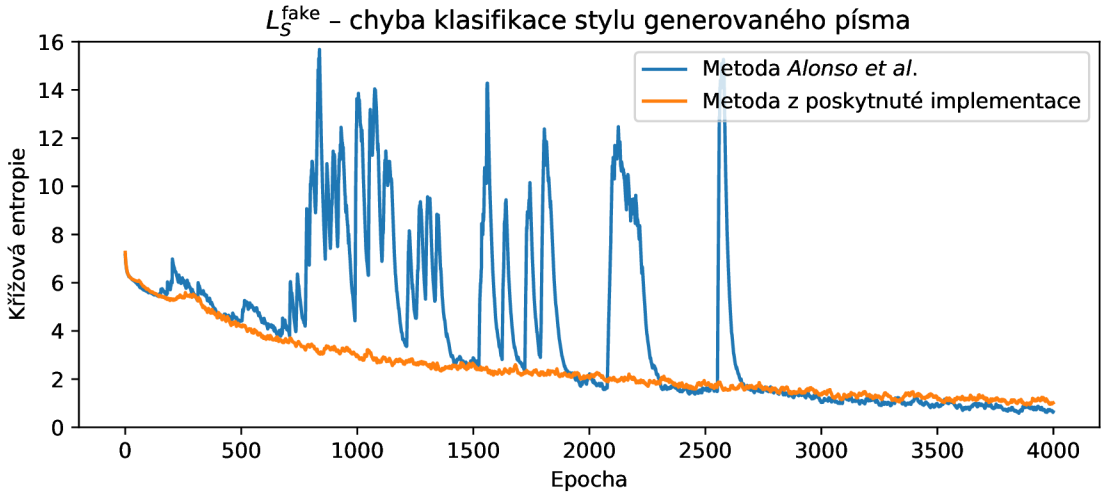
$$\nabla L_\bullet \leftarrow \frac{\sigma L_{adv}}{\sigma L_\bullet} \cdot \nabla L_\bullet, \tag{5.3}$$

kde $\nabla L_\bullet$ je gradient chybové funkce L_S nebo L_R s ohledem na generovaný obrázek, $\sigma L_\bullet$ je směrodatná odchylka hodnot takového gradientu a σL_{adv} je směrodatná odchylka hodnot gradientu chybové funkce L_{adv} s ohledem na generovaný obrázek. Postup normalizace se navíc liší od postupů využitých jinými modely, jako jsou *lineGen* [7] nebo model autorů *Alonso et al.* [1]. Normalizace modelu HWT vychází z normalizace popsané v článku autorů *Alonso et al.* [1] (viz rovnice (5.4)), neuvažuje ale střední hodnoty hodnot gradientů.

$$\nabla L_\bullet \leftarrow \left(\frac{\sigma L_{adv}}{\sigma L_\bullet} \cdot (\nabla L_\bullet - \mu L_\bullet) + \mu L_{adv} \right), \tag{5.4}$$

kde μL_{adv} je střední hodnota hodnot ∇L_{adv} a $\mu L_\bullet$ je střední hodnota hodnot $\nabla L_\bullet$.

Vyhodnoceny byly všechny tři přístupy k balancování. Kvalita písma generovaného modely trénovaných podle rovnic (5.2) a (5.3) se po ukončení trénování neliší. Především však na začátku trénování (do epochy č. 1 400 z celkových 4 000) bylo písmo generované podle rovnice (5.3) značně méně kvalitní. V průběhu dalšího trénování ale tento rozdíl zmizel.



Obrázek 5.6: Srovnání průběhů chybové funkce L_S^{fake} při použití balancování gradientů podle rovnice (5.4) (*Alonso et al.*) a podle rovnice (5.2). V prvním případě dochází ke značným výkyvům, které v důsledku ovlivňují i realističnost generovaného písma.

Při aplikování postupu podle rovnice (5.4) je trénování méně stabilní. Během trénování se kvalita písma značně mění. Hodnoty chybových funkcí L_R a L_S se občas na několik desítek epoch výrazně zvýší (viz obrázek 5.6) a následně klesnou na podobné hodnoty jako před výkyvem. Generované písmo je tak sice po většinu času trénování čitelné a z části napodobuje očekávaný styl, ale je snadno rozeznatelné od příkladů z reálné datové sady.

Nejvhodnějšími přístupy k balancování gradientů jsou tak varianty podle rovnic (5.2) a (5.3). Tyto oba přístupy vedou na generování písma, které nejlépe napodobuje očekávaný styl a není snadno rozeznatelné od příkladů z reálné datové sady. Přístup podle rovnice (5.4) není vhodnou volbou, protože trénování je méně stabilní a generované písmo není kvalitní.

Úpravy sítě pro klasifikaci stylu

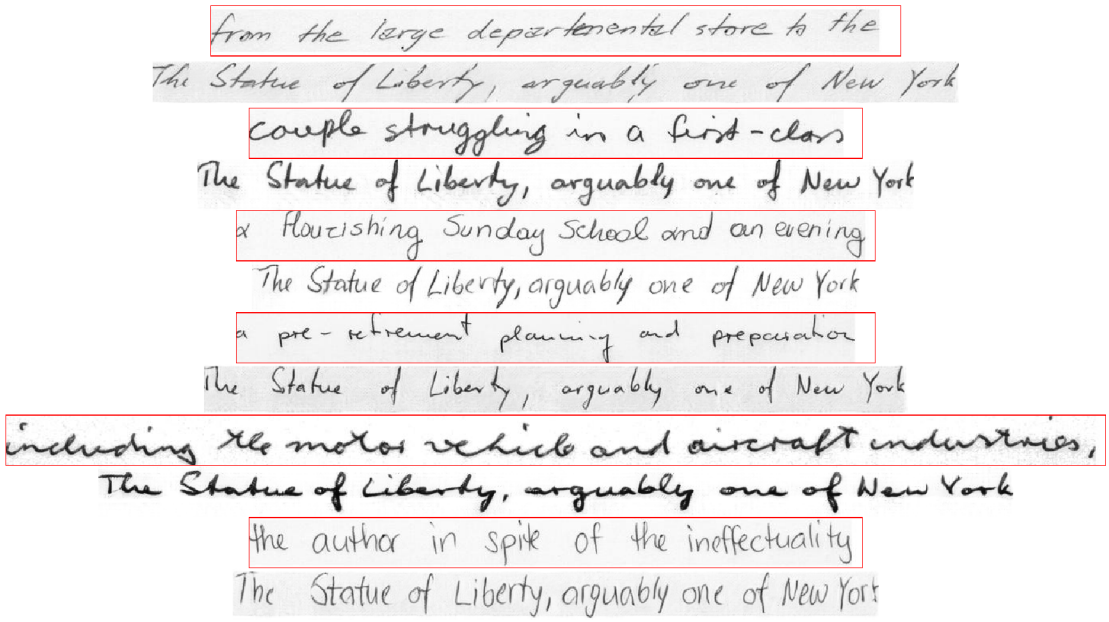
Jednou z chybových funkcí, které vedou trénování modelu HWT, je L_S – chyba klasifikace stylu. Ta je zcela zásadní pro napodobování očekávaného stylu. Diskriminátor, který slouží k výpočtu chybové funkce GAN (L_{adv}), pouze rozlišuje mezi reálnými a umělými snímky bez informace o očekávaném stylu. Při trénování modelu se tak očekává, že klasifikační síť hledá příznaky, které budou relevantní i pro předem neznámé styly. Síť je trénována s chybovou funkcí křížová entropie (angl. *cross-entropy*). Nabízí se však i alternativní způsoby, jako je použití *triplet loss* nebo použití chybové funkce *Additive Angular Margin Loss – ArcFace* [9]. Chybová funkce *ArcFace*, na rozdíl od běžného přístupu ke klasifikaci, kdy je cílem jednotlivé třídy pouze od sebe oddělit, maximalizuje vzdálenost mezi příznaky různých tříd.

Natrénován a vyhodnocen byl model, ve kterém jsou upraveny pouze poslední vrstvy sítě pro klasifikaci stylu tak, aby výpočet odpovídal *ArcFace*. Tímto způsobem bylo zaručeno, že změny kvality výsledků jsou přímým důsledkem této změny. Podle vyhodnocení metrikou FID [16] vede využití *ArcFace* ke zhoršení kvality generovaného písma, viz tabulka 5.6. Variabilita stylů je nižší, viz ukázky v tabulce 5.5.

Tabulka 5.5: Porovnání schopností modelů napodobit očekávaný styl. V prostředním sloupci je na každém řádku zobrazeno několik vybraných slov z množiny obrázků, která specifikuje očekávaný styl. Levý sloupec obsahuje ukázky slov vygenerovaných modelem trénovaným s chybovou funkcí *ArcFace* [9] a rozmístěním znaků podle *lineGen* [7]. V pravém sloupci jsou znázorněny ukázky generovaných slov modelem, který pouze nepoužívá *ArcFace*. Modely byly trénovány na samostatných slovech z datové sady IAM [29].

Model	IV-S	IV-U	OOV-S	OOV-U
HWT s rozmístění příznaků	**100.31**	**101.55**	**100.58**	**101.34**
HWT s rozm. příznaků a *ArcFace* [9]	105.57	107.07	105.54	106.86

Tabulka 5.6: Porovnání výsledků měření kvality generovaného písma metrikou FID [16]. Výsledky jsou rozděleny podle generovaných slov (IV – *In vocabulary*, slova z trénovací sady a OOV – *Out of vocabulary*, slova, která nejsou v trénovací sadě) a použitých stylů (S – styly z trénovací sady a U – styly, které nejsou v trénovací sadě). Využití chybové funkce *ArcFace* vede ke zhoršení kvality generovaného písma – pravděpodobně kvůli horší schopnosti napodobit očekávaný styl.



Obrázek 5.7: Ukázka generovaného písma s výškou obrázků 64 pixelů. Obrázky jsou rozděleny do dvojic řádků, kde vždy první z nich (v rámečku) určuje styl a druhý s textem „The Statue of Liberty, arguably one of New York“ je vygenerován modelem. Upravený model generuje realistické písmo i ve vyšším rozlišením a věrně napodobuje očekávaný styl.

5.3 Vyhodnocení

Řada navržených změn modelu HWT [2] vede k vyšší kvalitě generovaného písma pro samostatná slova, jak dokládají naměřené hodnoty metrik FID [16] na datové sadě IAM [29]. Současně nový model umožňuje generování celých řádků a trénování s nimi bez nutnosti vyřezávání slov. Nová implementace metod používaných v modelu HWT umožňuje efektivnější a rychlejší trénování, viz podkapitola 5.1. Několik ukázek generovaných řádků ve vyšším rozlišení shrnuje obrázek 5.7. Na obrázku 5.8 je znázorněna ukázka interpolací různých stylů.

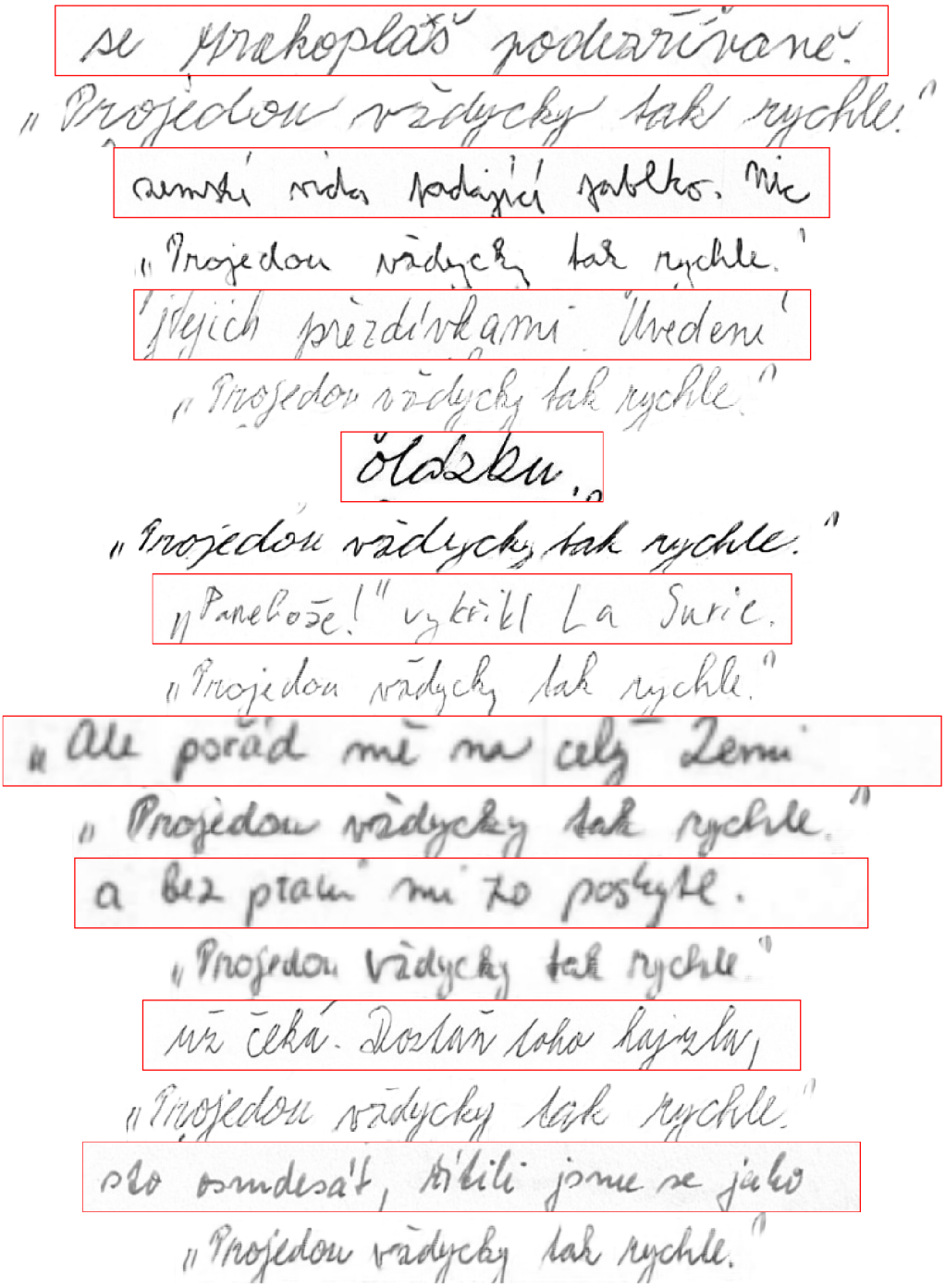
Kromě datové sady IAM byl model pro generování řádků natrénován na datové sadě z projektu Pero[3], čítající na 4 415 řádků českého textu včetně znaků s diakritikou. Tato datová sada byla rozšířena o generované řádky s cílem zvýšit přesnost OCR modelu. Přesnost modelu se však nezlepšila, naopak se zhoršila, viz hodnoty v tabulce 5.7. Toto je pravděpodobně způsobeno nedostatečnou variabilitou generovaného písma. I když je generativní model schopný generovat realistické písmo, jeho schopnost přesně napodobovat očekávaný styl písma není dostatečná. OCR model dosahuje s rozšířenou trénovací sadou značně nižší chyby než model, který je trénovaný pouze s původní sadou. To naznačuje, že generované trénovací příklady jsou pro OCR model příliš „jednoduché“ a pro učení tudíž neužitečné. Přesto generované písmo dosahuje dobré vizuální kvality a dokazuje, že navržený model dokáže vytvářet i psací písmo a pracovat s rozsáhlejší abecedou. Ukázky generovaného písma s různými styly jsou na obrázku 5.9.

[3]Dostupné z https://pero.fit.vutbr.cz/.

Obrázek 5.8: Model je schopný generovat písmo i pro různé interpolace stylů. Styly jsou získány lineární interpolací sekvencí Z, které jsou výstupem enkodéru obrázků reprezentujících styl.

Složení datové sady	p. kroků	CER trénovacích dat	CER validačních dat
5k snímků	28.5k	4.76 %	**8.89 %**
5k + 300k snímků	28.5k	2.33 %	24.18 %
5k + 300k snímků	170.5k	1.49 %	16.16 %

Tabulka 5.7: Porovnání úspěšnosti OCR modelu v závislosti na složení trénovací datové sady (ve formátu: počet reálných + počet generovaných snímků). Rozšíření trénovací datové sady generovanými snímky nepřispělo ke snížení CER (*Character error rate* – chybovosti znaků), naopak vedlo k jeho zvýšení. Pravděpodobně dochází k přetrénování modelu na generovaných snímcích, které tolik nepřispívají k učení.



Obrázek 5.9: Ukázka generovaného písma modelem natrénovaným na datové sadě z projektu Pero. Generovaným textem je věta „Projedou vždycky tak rychle“. Obrázky jsou rozděleny do dvojic řádků, kdy vždy první určuje styl a druhý je vygenerován modelem.

Kapitola 6

Závěr

Cílem této práce bylo generování řádků písma v kvalitě, která by mohla být vhodná i pro trénování systémů pro rozpoznávání ručně psaného písma. Navržený model překonává existující řešení jak v generování samostatných slov, tak i celých řádků. Tyto výsledky dokládají naměřené hodnoty metriky FID, které jsou nižší než u ostatních modelů. Přináší nový způsob rozmístění příznaků symbolů, který dovoluje specifikovat pozice jemněji než celočíselnými pozicemi, a tak i vytvářet plynulejší interpolace mezi různými styly písma. Významnou vlastností nového způsobuje je i možnost trénovat model pro odhad pozic pomocí adversariální chyby (GAN), nikoli odděleně, jak k tomuto problému přistupují jiné modely.

Navržený model navazuje na předchozí výzkum v této oblasti. Kombinuje jejich významné prvky a zároveň je vylepšuje. V úvodních kapitolách jsou podrobně analyzovány nejvýznamnější modely z minulých let. Text se zaměřuje především na offline přístupy ke generování písma. Tedy na ty přístupy, které neuvažují proces generování jako sekvenci tahů a pohybů, ale pracují pouze se snímky písma.

Pro ověření vlastností navrhovaného modelu byla provedena řada experimentů s datovou sadou IAM a datovými sadami z projektu Pero. Experimenty se zaměřují jak na ověření vlastností navrhovaného modelu, tak i na vylepšení dříve publikovaných řešení.

Další výzkum může navázat na slibné vlastnosti nového způsobu rozmístění příznaků symbolů, popřípadě se zaměřit na další vylepšení detailů generovaného písma. I když generované písmo dosahuje vysoké kvality, jeho variabilita není tak vysoká jako u reálné datové sady. Cílem dalšího výzkumu by tak mohlo také být hledání lepšího přístupu, jak přimět model generovat písmo ve specifikovaném stylu místo aktuálního použití klasifikační chyby stylu.

Literatura

[1] Alonso, E., Moysset, B. a Messina, R. Adversarial generation of handwritten text images conditioned on sequences. In: IEEE. *2019 International Conference on Document Analysis and Recognition (ICDAR)*. 2019, s. 481–486. DOI: 10.1109/ICDAR.2019.00083.

[2] Bhunia, A. K., Khan, S., Cholakkal, H., Anwer, R. M., Khan, F. S. et al. Handwriting transformers. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, s. 1086–1094. DOI: 10.1109/ICCV48922.2021.00112.

[3] Bluche, T. a Messina, R. Gated convolutional recurrent neural networks for multilingual handwriting recognition. In: IEEE. *2017 14th IAPR international conference on document analysis and recognition (ICDAR)*. 2017, sv. 1, s. 646–651. DOI: 10.1109/ICDAR.2017.111.

[4] Brandenbusch, K., Rusakov, E. a Fink, G. A. Context Aware Generation of Cuneiform Signs. In: Springer. *International Conference on Document Analysis and Recognition*. 2021, s. 65–79. ISBN 978-3-030-86549-8.

[5] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F. et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *ArXiv preprint arXiv:1406.1078*. 2014.

[6] Chung, J., Gulcehre, C., Cho, K. a Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *ArXiv preprint arXiv:1412.3555*. 2014.

[7] Davis, B., Tensmeyer, C., Price, B., Wigington, C., Morse, B. et al. Text and style conditioned gan for generation of offline handwriting lines. *ArXiv preprint arXiv:2009.00678*. 2020.

[8] De Vries, H., Strub, F., Mary, J., Larochelle, H., Pietquin, O. et al. Modulating early visual processing by language. *ArXiv preprint arXiv:1707.00683*. 2017.

[9] Deng, J., Guo, J., Xue, N. a Zafeiriou, S. Arcface: Additive angular margin loss for deep face recognition. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, s. 4690–4699. DOI: 10.1109/CVPR.2019.00482.

[10] Fogel, S., Averbuch Elor, H., Cohen, S., Mazor, S. a Litman, R. Scrabblegan: Semi-supervised varying length handwritten text generation. In: *Proceedings of the*

IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020, s. 4324–4333. DOI: 10.1109/CVPR42600.2020.00438.

[11] Goodfellow, I., Pouget Abadie, J., Mirza, M., Xu, B., Warde Farley, D. et al. Generative Adversarial Nets. In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. Cambridge, MA, USA: MIT Press, 2014, s. 2672–2680.

[12] Graves, A. Generating sequences with recurrent neural networks. *ArXiv preprint arXiv:1308.0850*. 2013.

[13] Graves, A., Fernández, S., Gomez, F. a Schmidhuber, J. Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. In: *Proceedings of the 23rd International Conference on Machine Learning*. New York, NY, USA: Association for Computing Machinery, 2006, s. 369–376. ICML '06. DOI: 10.1145/1143844.1143891. ISBN 1595933832.

[14] Grosicki[1], E., Carre, M., Brodin, J.-M. a Geoffrois[1], E. RIMES evaluation campaign for handwritten mail processing. Citeseer. 2008.

[15] He, K., Gkioxari, G., Dollár, P. a Girshick, R. Mask R-CNN. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, s. 2980–2988. DOI: 10.1109/ICCV.2017.322.

[16] 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. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2017, s. 6629–6640. NIPS'17. ISBN 9781510860964.

[17] Hochreiter, S. a Schmidhuber, J. Long short-term memory. *Neural computation*. MIT Press. 1997, sv. 9, č. 8, s. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[18] Huang, X. a Belongie, S. Arbitrary Style Transfer in Real-Time with Adaptive Instance Normalization. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, s. 1510–1519. DOI: 10.1109/ICCV.2017.167.

[19] Ian Goodfellow, Yoshua Bengio, A. C. The Deep Learning Book. *MIT Press*. 2017, sv. 521, č. 7553, s. 785. DOI: 10.1016/B978-0-12-391420-0.09987-X. ISSN 1432122X.

[20] Ioffe, S. a Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: PMLR. *International conference on machine learning*. 2015, s. 448–456. ICML'15.

[21] Johansson, S., Leech, G. N. a Goodluck, H. *Manual of information to accompany the Lancaster-Oslo/Bergen Corpus of British English, for use with digital computer*. Department of English, University of Oslo, 1978.

[22] Kang, L., Riba, P., Rusinol, M., Fornes, A. a Villegas, M. Content and Style Aware Generation of Text-line Images for Handwriting Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. IEEE. 2021. DOI: 10.1109/TPAMI.2021.3122572.

[23] Kang, L., Riba, P., Wang, Y., Rusiñol, M., Fornés, A. et al. Ganwriting: Content-conditioned generation of styled handwritten word images. In: Springer. *European Conference on Computer Vision*. 2020, s. 273–289. ISBN 978-3-030-58592-1.

[24] Karras, T., Laine, S. a Aila, T. A Style-Based Generator Architecture for Generative Adversarial Networks. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, s. 4396–4405. DOI: 10.1109/CVPR.2019.00453.

[25] Kingma, D. P. a Ba, J. Adam: A method for stochastic optimization. *ArXiv preprint arXiv:1412.6980*. 2014.

[26] Krizhevsky, A., Sutskever, I. a Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. Red Hook, NY, USA: Curran Associates Inc., 2012, s. 1097–1105.

[27] Lim, J. H. a Ye, J. C. Geometric gan. *ArXiv preprint arXiv:1705.02894*. 2017.

[28] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. et al. SSD: Single shot multibox detector. In: *European conference on computer vision*. Cham: Springer International Publishing, 2016, s. 21–37. ISBN 978-3-319-46448-0.

[29] Marti, U.-V. a Bunke, H. The IAM-database: an English sentence database for offline handwriting recognition. *International Journal on Document Analysis and Recognition*. Springer. 2002, sv. 5, č. 1, s. 39–46.

[30] Mattick, A., Mayr, M., Seuret, M., Maier, A. a Christlein, V. SmartPatch: Improving Handwritten Word Imitation with Patch Discriminators. *ArXiv preprint arXiv:2105.10528*. 2021.

[31] Michael, J., Labahn, R., Grüning, T. a Zöllner, J. Evaluating sequence-to-sequence models for handwritten text recognition. In: IEEE. *2019 International Conference on Document Analysis and Recognition (ICDAR)*. 2019, s. 1286–1293. DOI: 10.1109/ICDAR.2019.00208.

[32] Mirza, M. a Osindero, S. Conditional generative adversarial nets. *ArXiv preprint arXiv:1411.1784*. 2014.

[33] Nielsen, F. On a generalization of the Jensen–Shannon divergence and the Jensen–Shannon centroid. *Entropy*. Multidisciplinary Digital Publishing Institute. 2020, sv. 22, č. 2, s. 221.

[34] Rabiner, L. a Juang, B. An introduction to hidden Markov models. *Ieee assp magazine*. IEEE. 1986, sv. 3, č. 1, s. 4–16. DOI: 10.1109/MASSP.1986.1165342.

[35] Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C. et al. Zero-shot text-to-image generation. *ArXiv preprint arXiv:2102.12092*. 2021.

[36] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S. et al. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*. 2015, sv. 115, č. 3, s. 211–252. ISSN 15731405.

[37] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A. et al. Improved techniques for training gans. *Advances in neural information processing systems*. 2016, sv. 29, s. 2234–2242.

[38] Simonyan, K. a Zisserman, A. Very deep convolutional networks for large-scale image recognition. *ArXiv preprint arXiv:1409.1556*. 2014.

[39] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. a Wojna, Z. Rethinking the inception architecture for computer vision. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, s. 2818–2826. DOI: 10.1109/CVPR.2016.308.

[40] Ulyanov, D., Vedaldi, A. a Lempitsky, V. Improved Texture Networks: Maximizing Quality and Diversity in Feed-Forward Stylization and Texture Synthesis. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, s. 4105–4113. DOI: 10.1109/CVPR.2017.437.

[41] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L. et al. Attention is all you need. *Advances in neural information processing systems*. 2017, sv. 30.

[42] Viterbi, A. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*. IEEE. 1967, sv. 13, č. 2, s. 260–269. DOI: 10.1109/TIT.1967.1054010.

Příloha A

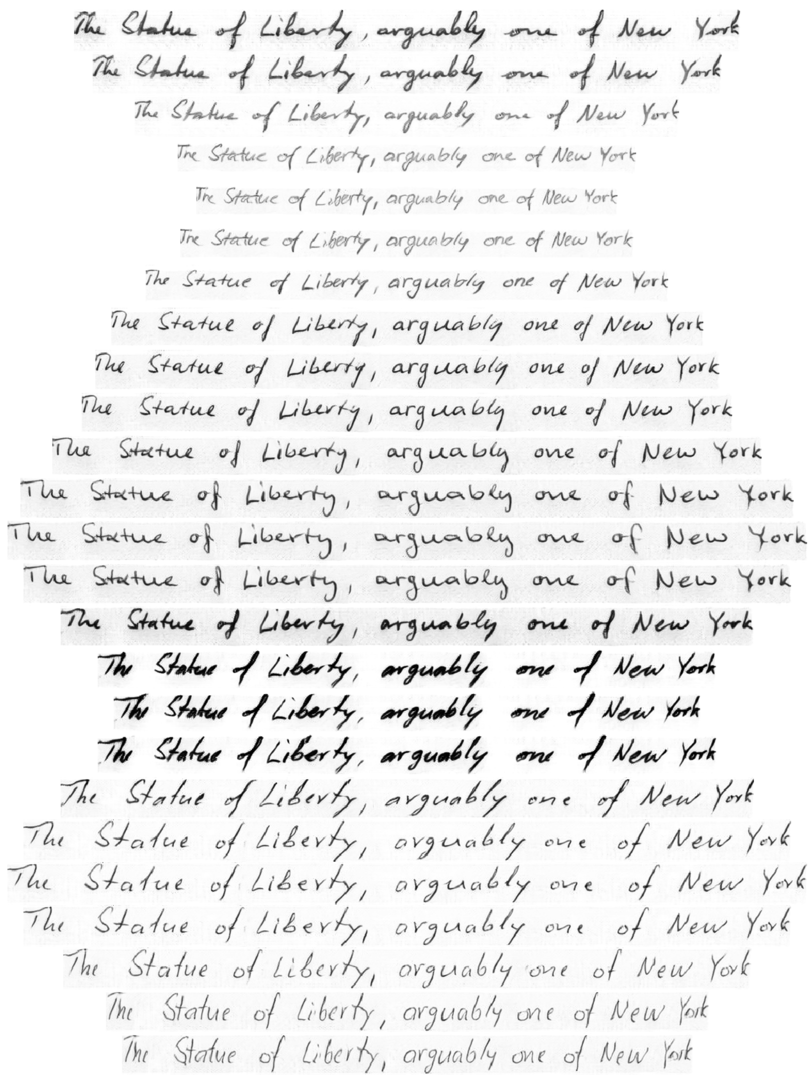
Obsah přiloženého DVD

Přiložené DVD obsahuje tyto soubory a adresáře:

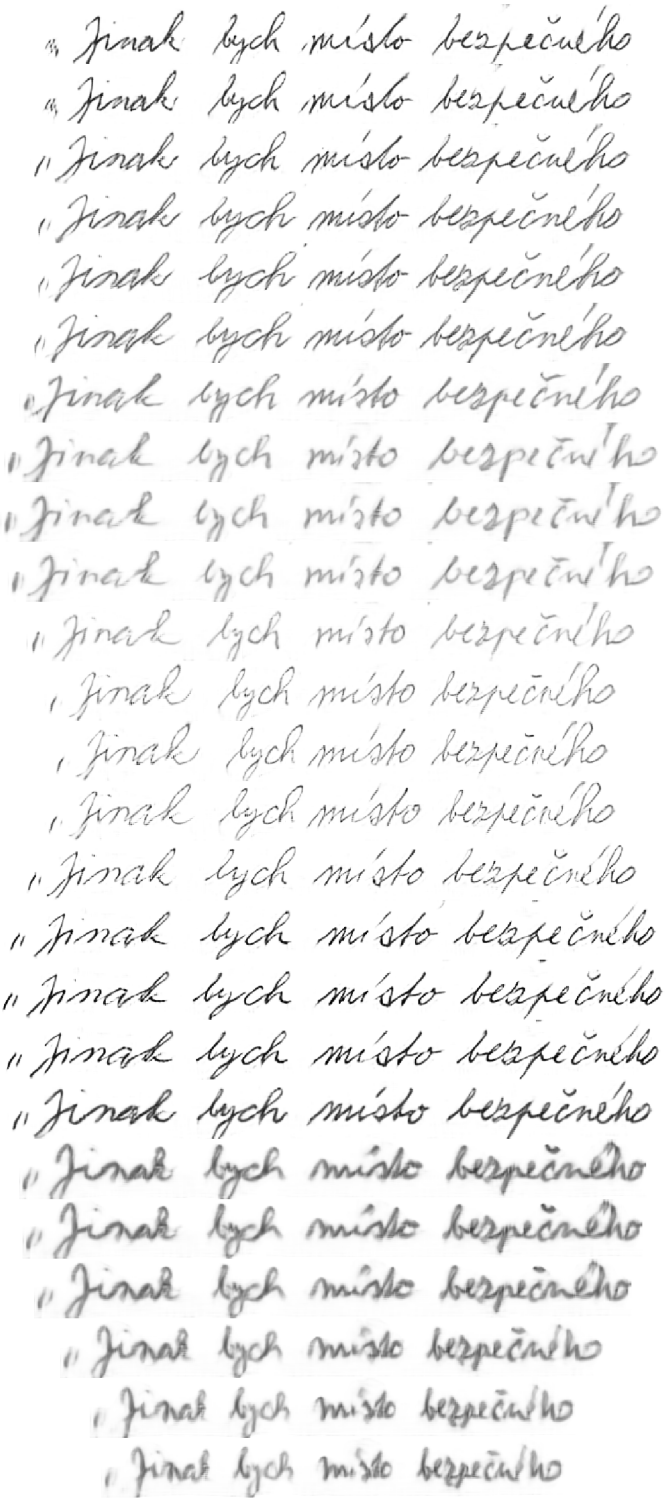
- **code/** – adresář obsahující veškeré zdrojové kódy,
- **data/** – adresář obsahující připravené datové sady pro zdrojové kódy z adresáře **code/** a výběr uložených parametrů několika natrénovaných modelů,
- **tech_report/** – adresář obsahující tuto technickou zprávu ve formátu PDF včetně všech souborů pro její kompilaci,
- **video.mp4** – video prezentující cíle a dosažené výsledky této práce,
- **README.md** – manuál pro instalaci a spuštění.

Příloha B

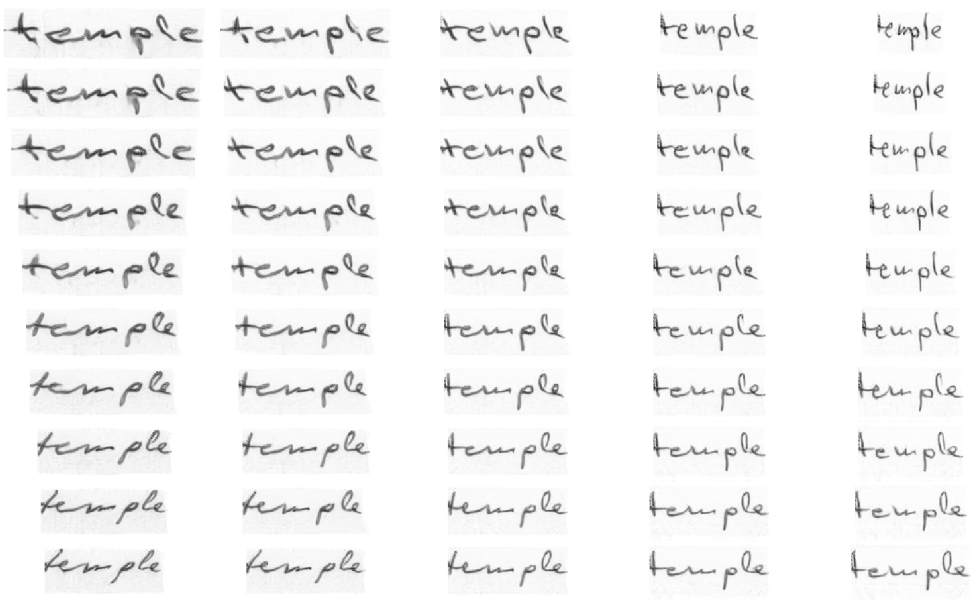
Obrázky generovaného písma



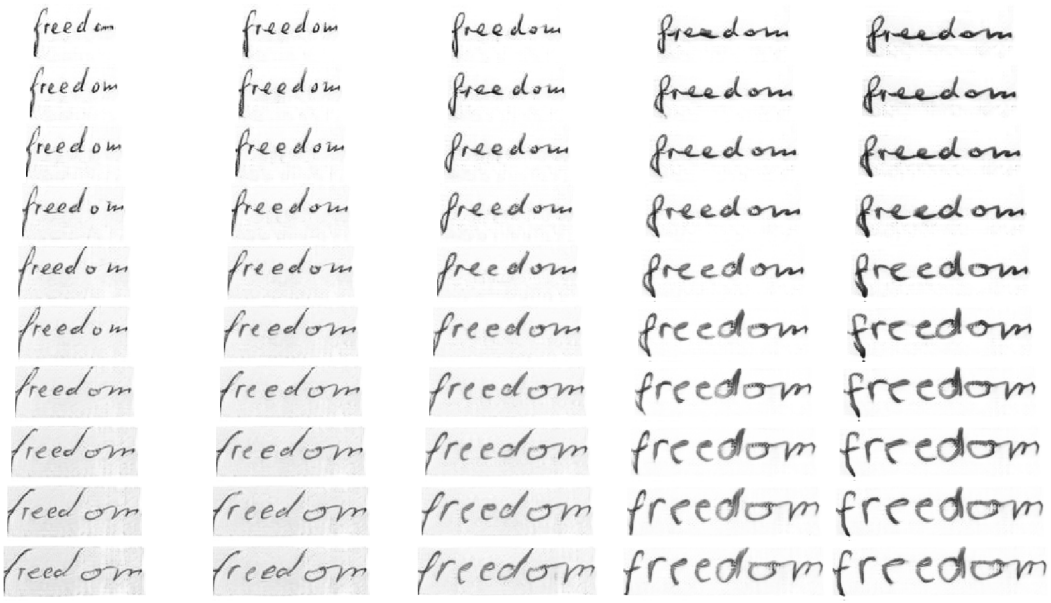
Obrázek B.1: Ukázka písem získaných interpolací různých stylů s generovaným textem „The Statue of Liberty, arguably one of New York“. Řádky jsou vygenerovány modelem natrénovaným na řádcích datové sady IAM.



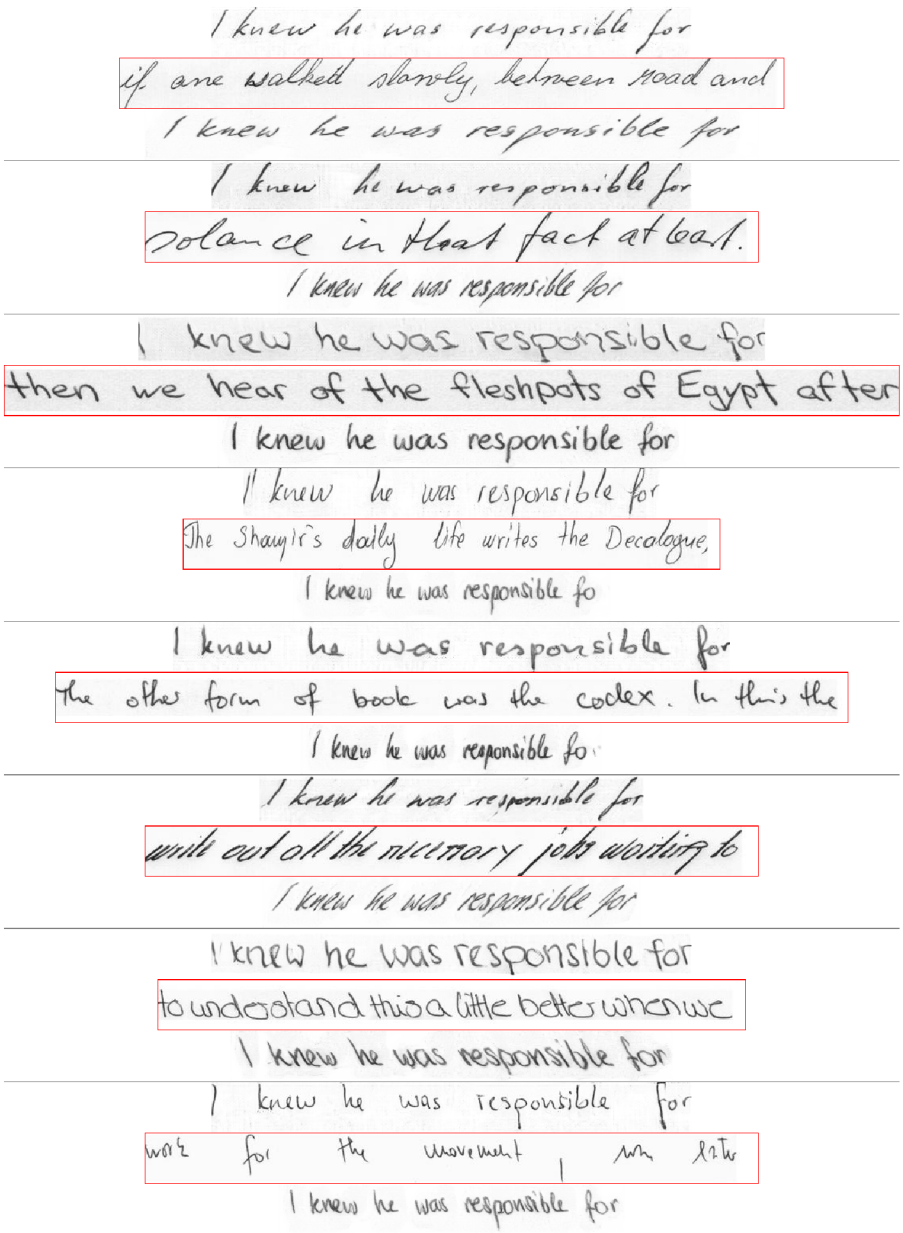
Obrázek B.2: Ukázka písem získaných interpolací různých stylů s generovaným textem „„Jinak bych místo bezpečného“. Řádky jsou vygenerovány modelem natrénovaným na řádcích datové sady z projektu Pero.



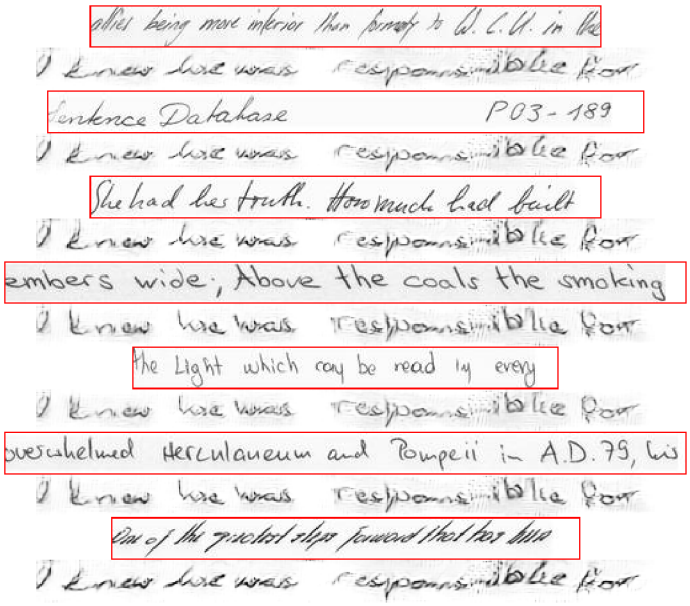
Obrázek B.3: Ukázka písem získaných interpolací čtyř různých stylů s generovaným textem „temple“. Slova jsou vygenerována modelem natrénovaným na řádcích datové sady IAM.



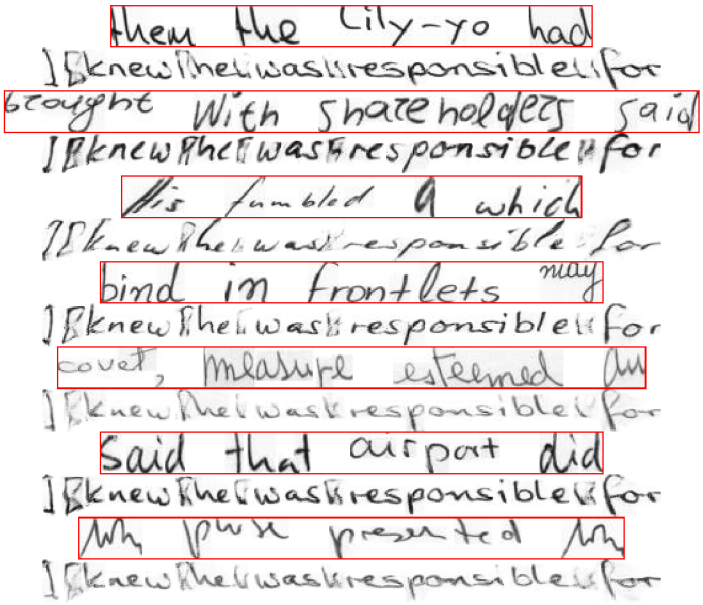
Obrázek B.4: Ukázka písem získaných interpolací čtyř různých stylů s generovaným textem „freedom“. Slova jsou vygenerována modelem natrénovaným na řádcích datové sady IAM.



Obrázek B.5: Srovnání písem generovaných modelem z této práce a modelem *lineGen* s textem „I knew he was responsible for“. Ukázky jsou rozděleny do trojic řádků, kde první z nich je vygenerován modelem z této práce, druhý (v rámečku) určuje očekávaný styl a třetí je vygenerován modelem *lineGen*. Model navržený v této práci lépe napodobuje očekávaný styl.



Obrázek B.6: Ukázka písem generovaných modelem *Handwriting Transformers* (HWT) trénovaným na řádcích datové sady IAM s textem „I knew he was responsible for“. Ukázky jsou rozděleny do dvojic řádků, kde první z nich (v rámečku) určuje styl a druhý je generován modelem. Bez rozmístění příznaků symbolů není model HWT schopný generovat realistické písmo pro delší textové řetězce.



Obrázek B.7: Ukázka písem generovaných modelem *Handwriting Transformers* (HWT) trénovaným na slovech datové sady IAM s textem „I knew he was responsible for“. Ukázky jsou rozděleny do dvojic řádků, kde první z nich (v rámečku) určuje styl a druhý je generován modelem. Generované písmo napodobuje styl a je realistické (s výjimkou mezer, které nejsou v datové trénovací datové sadě), ale objevují se nepřirozeně široké mezery mezi písmeny a roztahané znaky do šířky.