UNIWERSYTET GDAŃSKI
Wydział Matematyki, Fizyki i Informatyki

Paweł Luszuk
nr albumu: 235425

Transfer Stylu przy użyciu Uczenia Maszynowego w Blenderze

Praca magisterska na kierunku:
INFORMATYKA
Promotor:
dr Piotr Arłukowicz

Gdańsk 2020

Streszczenie

W niniejszej pracy przedstawiony zostanie proces implementacji tzw. Transferu Stylu (ang. Style Transfer) w programie do grafiki 2D/3D Blender.

Pierwszy rozdział poświęcony jest omówieniu podstawowych pojęć związanych z sztuczną inteligencją. Następnie omówione zostały sieci neuronowe i widzenie komputerowe jako procesy służące do interpretacji oraz przetwarzania obrazów 2D. Aktualnie dostępnych jest szereg narzędzi umożliwiających analizę obrazu, omówione one zostały w rozdziale 4. Dla wcześniej omówionych narzędzi istnieją biblioteki, które pozwalają na wykorzystanie pełnego potencjału uczenia maszynowego i zostały one omówione w rozdziale 5.

Implementacja transferu stylu została szczegółowo omówiona w rozdziale 6. Algorytmu Neural-Style opracowany przez Leona A. Gatysa, Alexandra S. Eckera i Matthiasa Bethge w środowisku Blender. W tym rozdziale został pokazany cały proces od założeń, przez stworzenie modelu, poprzez szkolenie. Pokazano, że transfer stylu pozwala na analizować zdjęcia i odtwarzać je w nowym stylu artystycznym, ale również ma pewne ograniczenia.

Słowa kluczowe

Blender, Transfer Stylu, Python, PyTorch, Uczenie Maszynowe, Głębokie Uczenie, Sztuczna Inteligencja, Neuron, Konwolucyjne sieci Neuronowe, Metoda Gradientu Prostego, Wsteczna Propagacja, RGB, Tensor, PyCharm, Jupyter Notebook, CUDA, PIP, BPY, Torchvision, Pillow, OS, Matruca Gram, VGG19

Spis treści

Wprowadzenie

Żyjemy w czasach niespotykanego nigdy w historii ludzkości tempa rozwoju nauki i technologii. Komputery są obecnie w stanie wykonywać zadania, które jeszcze 20 lat temu mogły pojawić się tylko w umysłach Philipa K. Dicka czy Wiliama Gibsona.

Zadania, które jeszcze kilka lat temu wymagały wyższego poznania, są rozwiązywane przez maszyny o niemal nadludzkim poziomie wydajności. Zawody, bez których ludzkość dziesięciolecia temu nie mogła się obejść nie istnieją. Do zadań takich jak wykrywanie znaków drogowych i pieszych przez autonomiczne samochody. Analiza obrazu czy obróbka dźwięku może odciążyć człowieka w prostych powtarzalnych czynnościach.

Uczone maszynowo generatory obrazu, takie jak style transfer mogą tworzyć nieograniczone bazy inspiracji dla designerów czy grafików. Są one trudne w implementacji ale banalne w użyciu, dlatego poczatkujący graficy mogą osiągnąć ciekawe efekty wizualne bez szczegółowej wiedzy. Na chwilę pisania tej pracy takie narzędzia nie są popularnymi narzędziami dla grafików, gdyż wymagają dużej wiedzy teoretycznej do własnej implementacji. Dużym wkładem w popularyzację programów oraz rozszerzeń bazujących na uczeniu maszynowym. Możliwości jaki stoją przed obrazami generowanymi maszynowo są nieograniczone zarówno do prototypowania i tworzenia "bazy" pomysłów, jak również do odciążania człowieka z zadań które jeszcze kilka lat temu były wykonalne tylko dla najlepszych specjalistów w branży.

ROZDZIAŁ 1

Podstawy

Sztuczna inteligencja, uczenie maszynowe oraz głębokie uczenie znajdują zastosowanie w coraz większej ilości aplikacji. W tym rozdziale zostaną omówione podstawowe pojęcia związane ze sztuczną inteligencją. Na schemacie 1.1 zostały przedstawione relacje, które zachodzą pomiędzy przedstawionymi pojęciami.

1.1. Sztuczna inteligencja

Sztuczna inteligencja (ang. Artificial Intelligence) to dziedzina nauki[20, 21], zajmująca się tworzeniem modeli zachowań inteligentnych oraz programów i systemów symulujących te zachowania, zarówno w dziedzinie oprogramowania jak i sprzętu. Prostą sztuczną inteligencję reprezentuje nawet kalkulator. Przyjęto też, iż wyznacznikiem jakości sztucznej inteligencji jest umiejętność „udawania" ludzkiego mózgu. Już w roku 1950 Alan Turing opracował test [1], określający w jakim stopniu maszyna jest w stanie myśleć jak człowiek.

1.2. Uczenie maszynowe

Uczenie maszynowe (ang. Machine Learning) to dziedzina nauki łącząca ze sobą informatykę, matematykę oraz statystykę. Kładzie ona szczególny nacisk na analizę danych i dostosowywanie logiki programu w zależności od otrzymanych informacji. Głównym celem jest praktyczne zastosowanie osiągnięć w dziedzinie sztucznej inteligencji do stworzenia automatycznego systemu, który można ulepszyć za pomocą zgromadzonego doświadczenia i zdobycia nowej wiedzy na tej podstawie [9].

Uczenie maszynowe polega na pobraniu dużej ilości danych, poddaniu ich

analizie i zbudowaniu na tej podstawie modelu. Dzięki temu, podając programowi nowy zestaw danych, będzie mógł on określić wynik, na podstawie zebranego doświadczenia.

1.3. Głębokie uczenie

Głębokie uczenie (ang. Deep Learning) jest podkategorią uczenia maszynowego. Polega na znajdowaniu powiązań między elementami w zbiorze danych niedostrzegalnym przez człowieka. Zajmuje się głównie przetwarzaniem dźwięku np. rozpoznawanie "głosu", przetwarzaniem języka naturalnego np. tłumaczenia oraz analizą obrazu np. identyfikowanie osób. Charakteryzuje się to przetwarzaniem dużej ilości danych wejściowych, których wynikiem są funkcje numeryczne, które ułatwiają algorytmowi niższego rzędu, takim jak klasyfikator, uzyskanie poprawnych wyników na nowych danych. [2, 17, 18, 19] Głębokie uczenie ma na celu pobranie oryginalnych danych i przedstawienie reprezentacji tych samych danych, które można podać algorytmowi w celu rozwiązania problemu. Głębokiego uczenia można użyć w przypadku modeli ogólnych, które nie są zaprojektowane do rozwiązania określonego zadania, ale mogą być automatycznie dostosowywane do specjalizacji w problemie.

ROZDZIAŁ 2

Widzenie komputerowe

Możliwośc interpretacji i przetwarzania obrazu przez programy zostało zrewolucjonizowane przez wprowadzenie konwolucyjnych sieci neuronowych, a systemy oparte na obrazach zyskały nowy zestaw możliwości.

W tym rozdziale zostaną omówione podstawowe definicje związane, a następnie przedstawiony zostanie przykład obrazujący według jakiego wzorca konstruowane są sieci neuronowe. Omówione zostanie również możliwości odczytu, przetwarzania oraz zapisu obrazów.

2.1. Neuron

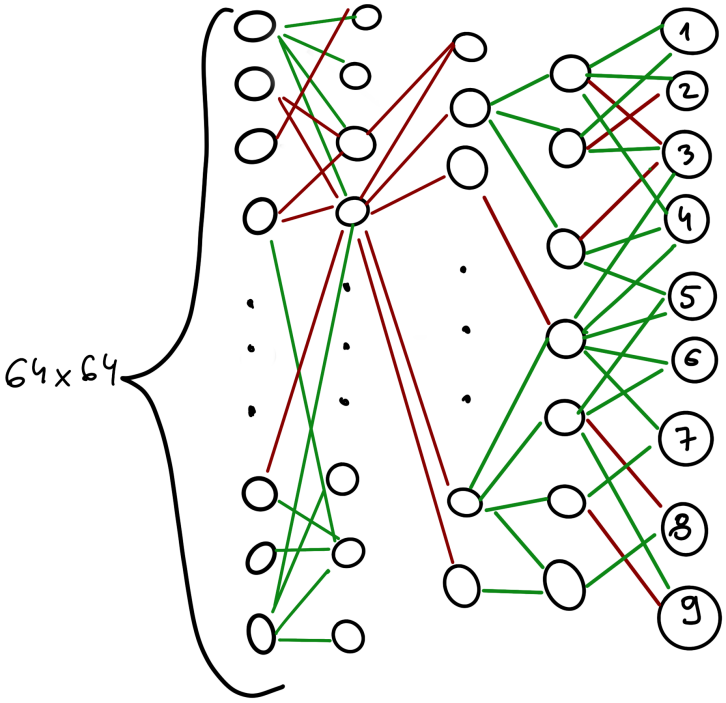
Neuronem określamy najbardziej elementarną cząstkę modelu uczenia maszynowego, który jest jest podstawowym budulcem sieci neuronowej. Często jest on pojedynczą liczbą np. odcień szarości (ang. Grayscale Value) dla pojedynczego piksela w przypadku czarno-białego obrazka. Połączenia miedzy neuronami obrazowane się za pomocą grafu (Rysunek 2.1).

2.2. Sieci Neuronowe

Sieci Neuronowe (ang. Neural Networks) to teoretyczny paradygmat struktur matematycznych inspirowany układem synaps i połączeń między nimi w mózgu. Składowymi sieci neuronowych są neurony oraz połączenia między nimi. Sieci neuronowe otrzymują liczbowe dane wejściowe i przekazują liczbowe dane wyjściowe. Może to być np. klasyfikacja, tłumaczenie lub obliczenie. Sieci neuronowe dzieli się na warstwy (ang layers), w których każda przetwarzana jest z uwzględnieniem poprzedniej [7, 8].

2.3. Przykład

Posłużmy się przykładem prostej sieci neuronowej, która ma za zadanie rozpoznać pisane cyfry. Jest to najbardziej elementarny przykład w dziedzinie uczenia maszynowego. W przypadku analizy obrazka 64x64 piksele jedna warstwa sieci neuronowej sieć neuronowa składać będzie się z 4096 neuronów, czyli łącznej liczby pikseli w macierzy. Odniesienia do tego przykładu znajdą się w dalszych podrodziałach.



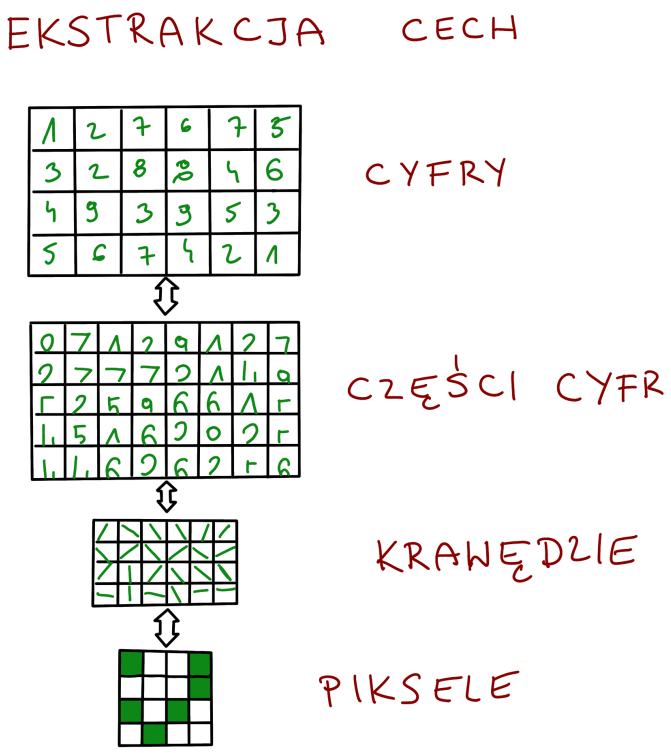
Rysunek 2.1. Układ neuronow dla obrazka 64x64 px

Źródło: Opracowanie własne

2.4. Konwolucyjne Sieci Neuronowe

Konwolucyjne Sieci Neuronowe (ang. Convolutional neural network) zwane też splotowymi to charakterystyczny dla głębokiego uczenia typ sieci neuronowych, w których liczba warstw w środku sieci jest ukryta (ang. Hidden Layers).

Konwolucyjcym Sieci Neuronowe działają na zasadzie ekstrakcji cech (ang. feature extraction) na zbiorach danych. Zbudowane są z sekwencyjnych warstw, gdzie każda warstwa dokonuje ekstrakcji cech coraz to wyższego poziomu.



Rysunek 2.2. Przykład ekstrakcji cech w konwolucyjnej sieci neuronowej

Źródło: Opracowanie własne

Sieci konwolucyjne poprzez trening są w stanie nauczyć się, jakie cechy szczególne obrazu pomagają w jego klasyfikacji. Jest to możliwe dzięki zastosowaniu filtrów badających relacje pikselli będących w sąsiedztwie. Warto jednak wspomnieć o innych sieciach np. Long short-term memory network, które swoje zastosowanie ma w analizie dźwięku. Konwolucyjna sieć neuronowa składa się z jednej lub wielu warstw klastrów połączeń.

Sprawdzają się one przy analizie obrazów. Mogą być one automatycznie dostosowywane, aby specjalizować się w danym problemie.

Dane są często dzielone na osobne zestawy próbek szkoleniowych i próbek walidacyjnych, co pozwala na ocenę modelu na podstawie danych, na których nie był szkolony.

2.5. Funkcja Aktywacji

Najprostszą jednostką w (głębokich) sieciach neuronowych jest operacja liniowa (skalowanie + przesuwanie), po której następuje funkcja aktywacji (ang. Activation function).

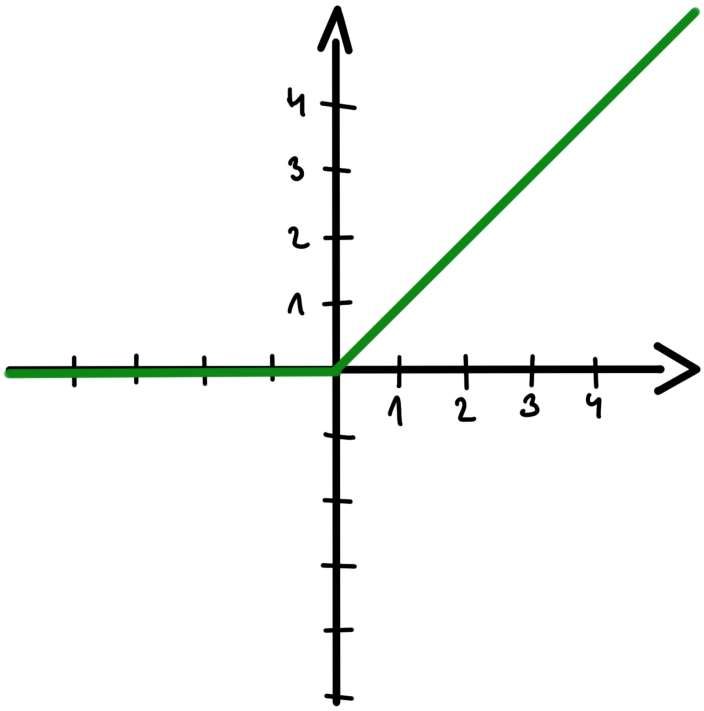
Ostatnią warstwą jest output cyfr 0 - 10 i ta z największą wartością aktywacji zostaje wybrana.

Dokładniej mówiąc neuron to funkcja, która przyjmuje dane wejściowe ze wszystkich neuronów z poprzedniej warstwy i przetwarza to na wartość od 0 do 1, 0 to czarny 1 to biały wartość ta zwana jest “aktywacją” lub “funkcją aktywacji”.

Funkcja aktywacji ma na celu skoncentrowanie wyników poprzedniej operacji liniowej w danym zakresie. Aktywacja z poprzedniej warstwy, determinuje aktywację w kolejnych aż do samego końca. Dla każdego neuronu wykonywana jest operacja matematyczna:

$$neuron = (w * x + b) \tag{2.1}$$

- “x” jest wkładem do obliczeń pojedynczego neuronu , w przypadku rozpoznawania cyfr x jest wartością 0 - 1 odcieniem szarości dla danego piksela



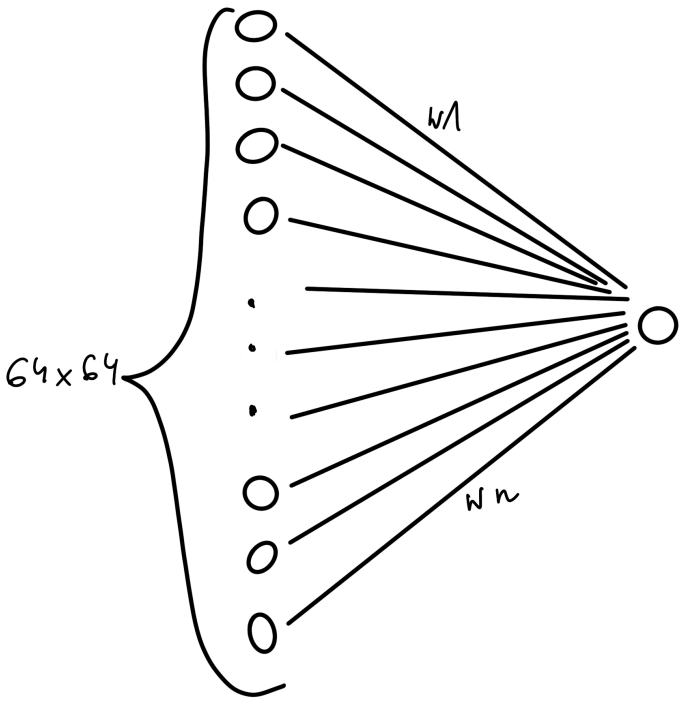
Rysunek 2.3. Funkcja ReLU: $f(x) = max(0, x)$

Źródło: Opracowanie własne

- “w” jest wagą, Jest to wartość, która na swój sposób określa jak istotny danych neuronów w przypadku rozpoznawania cyfr pisanych piksele na skraju obrazka będą miały mniejsze znaczenie niż te w okolicach jego centrum. Wagi mówią, jaki wzór pikselowy odbiera neuron w kolejnej warstwie.

- ”b” to odchylenie (bias), który mówi, jak wysoka musi być suma ważona, zanim neuron zacznie być znacząco aktywny.

W każdej warstwie wartość w neuronie po dodaniu odchylenia i przemnożeniu przez wagę w celu identyfikacji tylko “ważnych” wartości całość



Rysunek 2.4. Przypisywanie wag

Źródło: Opracowanie własne

przeliczamy jeszcze przez funkcję aktywacyjną. Tak zwana warstwa ReLU usuwa ujemne wartości z mapy aktywacji, ustawiając je na zero, co zwiększa nieliniowe właściwości funkcji decyzyjnej i całej sieci.

Wagi mnożymy z aktywacją , aby wagi pomagały w określeniu stopnia aktywacji, zachowujemy je w przedziale od 0 do 1.

2.6. Funkcja Strat

Funkcja strat (ang. loss function) to miara błędu w wykonywaniu zadania, takiego jak błąd między przewidywanymi wyjściami a mierzonymi wartościami. Celem jest, aby funkcja straty była jak najniższa.

2.7. Metoda Gradientu Prostego

Metoda Gradientu Prostego (ang. Gradient descent) to opracowany przez Cauchy'ego w 1847 r. [6] iteracyjny algorytm optymalizacji pierwszego rzędu do znajdowania lokalnego minimum funkcji różniczkowalnej. Aby znaleźć lokalne minimum funkcji za pomocą opadania gradientu, wykonujemy kroki proporcjonalne do ujemnego gradientu (lub przybliżonego gradientu) funkcji w bieżącym punkcie. Ale jeśli zamiast tego podejmujemy kroki proporcjonalne do dodatniego gradientu, zbliżamy się do lokalnego maksimum tej funkcji; procedura jest wówczas nazywana wynurzaniem gradientowym [16].

Im więcej kroków optymalizacji tym bardziej efektowny będzie transfer stylu. Przykład użycia takiego optymalizatora znajduje się w rozdziale 5.3.

2.8. Wsteczna Propagacja

W uczeniu maszynowym propagowanie wsteczne (ang. backpropagation) jest szeroko stosowanym algorytmem w uczeniu sieci neuronowych przekazywanych do nadzorowanego uczenia. Przy dopasowywaniu sieci neuronowej propagacja wsteczna oblicza gradient funkcji utraty w odniesieniu do wag sieci dla przykładu z pojedynczym wejściem i wyjściem i robi to skutecznie, w przeciwieństwie do naiwnego bezpośredniego obliczania gradientu w odniesieniu do każdej masy indywidualnie. [16] Ta wydajność umożliwia stosowanie metod gradientowych do szkolenia sieci wielowarstwowych, aktualizując wagi w celu zminimalizowania strat. Algorytm wstecznej propagacji działa poprzez obliczenie gradientu funkcji utraty w odniesieniu do każdej masy według reguły łańcuchowej, obliczanie gradientu pojedynczej warstwy na raz, iterowanie wstecz od ostatniej warstwy.

2.9. RGB

W przypadku algorytmu rozpoznjającego cyfry, obraz interpretowany przez komputer był w postaci dwuwymiarowej macierzy, z wartościami od 0 do 1. W przypadku algorytmów do innych zastosowań rozposnawanie gatunków kwiatów lub znaków drogowych może okazać się to za mało. Analiza obrazów kolorowych zwiększa złożonośc sieci neuronowych.

Istnieje kilka sposobów kodowania liczb na kolory. Najczęstszym jest RGB, który określa kolor za pomocą trzech liczb reprezentujących intensywność czerwieni, zieleni i niebieskiego.

W tym momencie kolorowy obrazek jest obiektem podobnym do tablicy o trzech wymiarach: dwóch wymiarach przestrzennych (szerokość i wysokość) i trzecim wymiarze odpowiadającym kanałom czerwonym, zielonym i niebieskim.

Każdy kolor będzie reprezentowany jako 8-bitowa liczba całkowita, jak w większości formatów fotograficznych ze standardowych aparatów konsumenckich.

Wczytany obraz (np. w formacie *.jpg) należy przekrztałcić w tensor, który układa obraz w sposób zgodny z oczekiwaniami danego API uczenia maszynowego.

2.10. Tensor

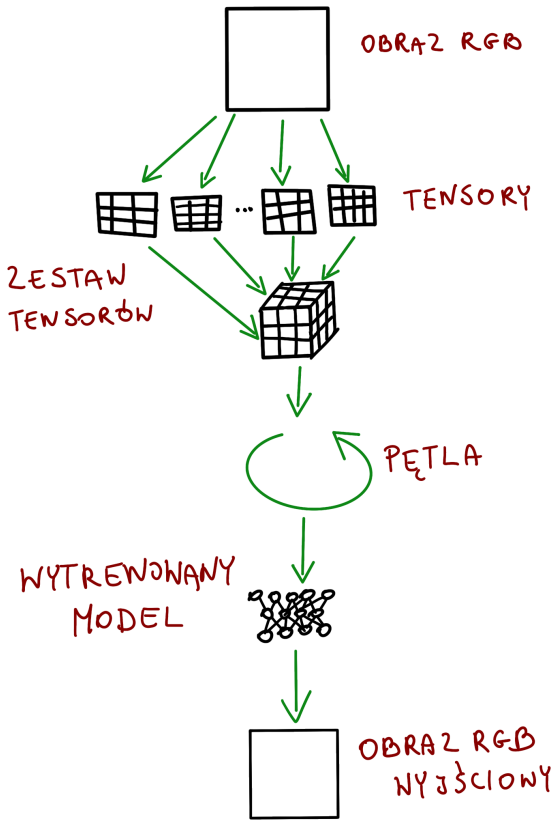
Tensor[4], jest to wielowymiarowa tablica. Ponownie mieszasz definicje z wykorzystaniem. Najpierw tłumaczysz, że to wielowymiarowa tablica, później dajesz wykorzystanie w sieciach neuronowych, a dalej znowu piszesz co to jest.

Tensor to struktura danych przechowująca zbiór liczb, które są dostępne indywidualnie za pomocą indeksu i które mogą być indeksowane za pomocą wielu indeksów.

Termin tensor jest dołączony do pojęcia przestrzeni, układów odniesienia i transformacji między nimi. Dla wszystkich innych tensor odnosi się do uogólnienia wektorów i macierzy do dowolnej liczby wymiarów, jak pokaza-

no na rysunku. Tensor reprezentujące wartości w poszczególnych pikselach są często kodowane za pomocą liczb 8-bitowych, na przykład w kamerach konsumenckich. W zastosowaniach medycznych, naukowych i przemysłowych nierzadko można znaleźć piksele o większej precyzji numerycznej, takie jak 12-bitowe i 16-bitowe.

Ta precyzja zapewnia szerszy zakres lub zwiększoną czułość w przypadkach, w których piksel koduje informacje dotyczące właściwości fizycznej, takiej jak gęstość kości, temperatura lub głębokość.



Rysunek 2.5. Schemat przetwarzania obrazu

Źródło: Opracowanie własne

Sieci neuronowe pobierają tensory na wejściu i wytwarzają tensory jako wyjścia. W rzeczywistości wszystkie operacje w sieci neuronowej i podczas

optymalizacji są operacjami między tensorami, a wszystkie parametry (takie jak wagi i odchylenia) w sieci neuronowej są tensorami.

Sieci neuronowe zwykle działają z wejściowymi tensorami zmiennoprzecinkowymi. Sieci neuronowe wykazują najlepszą wydajność treningu, gdy dane wejściowe mieszczą się w przedziale od około 0 do 1 lub –1 do 1 (efekt definiowania ich bloków konstrukcyjnych). Sieci neuronowe wymagają przedstawienia danych jako wielowymiarowych tensorów numerycznych, często 32-bitowych liczb zmiennoprzecinkowych.

Na rysunku 2.4 został przedstawiony schemat procesu operacji na obrazie.

Na tensorach można wykonać kilka innych operacji na danych wejściowych, w tym przekształcenia geometryczne, takie jak obrót, skalowanie i kadrowanie. Operacje te mogą pomóc w szkoleniu lub mogą być wymagane, aby dowolne dane wejściowe były zgodne z wymaganiami wejściowymi sieci, takimi jak rozmiar obrazu.

2.11. Podsumowanie

Podsumowując można uważać za poprawne stwierdzenie, iż uczenie się sieci to tylko modyfikowanie wag oraz odchyleń tak, aby osiągać najlepsze rezultaty, czyli uczenie się jest swego rodzaju oszacowaniem parametrów.

Często w przypadku gdy sieć neuronowa ma rozpoznawać zależności bazujące na kształtach a rozpoznawanie cyfr pisanych właśnie takie jest, często redukuje się dane wejściowe do minimum. Zamienia się wtedy kolorowy obrazek RGB na czarno-biały. Dzięki takiemu zabiegowi sieć neuronowa nie zawiera niepotrzebnych danych takiech jak kolor i może to dać lepszą rozpoznawalność jak również wzrost wydajności zarówno w procesie szkolenia jak i działania klasyfikatora.

ROZDZIAŁ 3

Przegląd Wykorzystanych Narzędzi

Przetwarzanie obrazu przy użyciu sieci neuronowych to interdyscyplinarne zagadnienie łączącę zagadnienia z matematyki, graficzki czy programowania. Przy tworzeniu implementacji tych zagadnień można sięgnąć po wiele dostępnych narzędzi. W tym rozdziale opisane zostaną natomiast te, użyte w implementacji Transferu Stylu opisanego szczegółowo wraz z przykładowym kodem w rozdziale 5.

3.1. Python

Język Python powstał już w latach dziewięćdziesiątych, między innymi dzięki rewolucji uczenia maszynowego przeżywa swoją drugą młodość. Jest używany w przez największe frameworki do uczenia maszynowego takie jak Tensorflow czy PyTorch opisany w rodziale 4.3. Jest to język, z którego w większości składa się interfejs Blendera. Każda wersja blendera ma wbudowaną już własną wersję pythona. Na potrzeby tej pracy cała implementacja Transferu Stylu została napisana w tym języku.

3.2. Blender

Blender jest to wolnym i otwartym oprogramowaniem do modelowania i renderowania obrazów oraz animacji trójwymiarowych.Pozwala on na pisanie w języku python skryptów, które poszerzają podstawowe funkcjonalności blendera.

3.3. Wtyczki

Jedną z wielu zalet środowiska Blender jest jego otwartość. Większość kodu jest otwarta i konfigurowalna, ułatwia to tworzenie rozszerzeń (wtyczek).

Wtyczki w Blenderze mogą istnieć w postaci pojedynczych skryptów

$$*.py \tag{3.1}$$

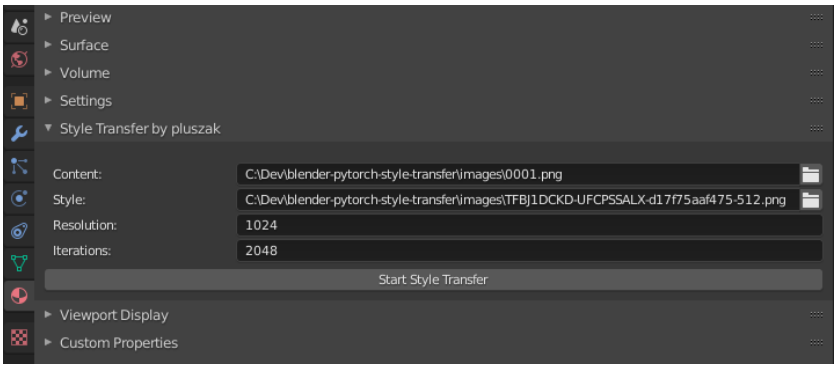
lub zestawu skryptów pakowanych w

$$*.zip \tag{3.2}$$

.zip. Uruchamiany wtedy jest plik

$$\_\_init\_\_.py \tag{3.3}$$

Zarówno wtyczki do Blendera jak i same operacje przy użyciu PyTorch'a mogą być tworzone na dowolnym systemie operacyjnym. Te narzędzia znacznie ułatwiają pracę nad tego typu oprogramowaniem. Do tego programu na potrzeby tej pracy utworzone zostało rozszerzenie, implementujące Transfer Stylu.



Rysunek 3.1. Implementacja Transferu Stylu jako wtyczki w Blenderze

Źródło: Opracowanie własne

3.4. PyCharm

PyCharm to zintegrowane środowisko programistyczne (IDE) dla języka programowania Python firmy JetBrains. Zapewnia m.in.: edycję i analizę kodu źródłowego, graficzny debugger, uruchamianie testów jednostkowych, integrację z systemem kontroli wersji.

3.5. Jupyter Notebook

Jupyter Notebook to iInteraktywne środowisko pozwalająca na wykonywanie kodu uczenia maszynowego po stronie serwera. Pozwala na szybkie debugowanie i testowanie kodu między innymi biblioteki PyTorch.

3.6. CUDA

CUDA (ang. Compute Unified Device Architecture) jest to opracowana przez firmę Nvidia uniwersalna architektura procesorów wielordzeniowych (głównie kart graficznych) umożliwiająca wykorzystanie ich mocy obliczeniowej do rozwiązywania ogólnych problemów numerycznych w sposób wydajniejszy niż w tradycyjnych, sekwencyjnych procesorach ogólnego zastosowania. Charakteryzuje się ona wyszszym współczynnikiem FLOPS (ang. floating point operations per second) czyli ilości operacji zmiennoprzecinkowych na sekundę.

ROZDZIAŁ 4

Przegląd Wykorzystanych Bibliotek

Podobnie jak w przypdaku narzędzi jest wiele bibliotek w języku Python rozwiązujące te same zagdanienia. W tym rozdziale opisane zostaną natomiast te, użyte w implementacji Transferu Stylu opisanego szczegółowo wraz z przykładowym kodem w rozdziale 5.

4.1. PIP

PIP (ang. Package In Python) to standardowy system zarządzania rozszerzeniami i bibliotekami w Pythonie. Wszystkie inne zewnętrzen bibilioteki pythona instalowane są przy pomocy biblioteki PIP.

Listing 4.1. Przykład instalacji PIP-a na danym Pythonie

```
python.exe -m ensurepip
```

4.2. PyTorch

PyTorch to biblioteka, która ułatwia budowanie projektów głębokiego uczenia się. [3] Podkreśla elastyczność i pozwala wyrazić modele głębokiego uczenia się w idiomatycznym języku Python. Ta przystępność i łatwość użycia znalazły wczesnych użytkowników w społeczności badawczej, a od lat od wydania biblioteki stała się jednym z najważniejszych narzędzi do głębokiego uczenia się dla szerokiego zakresu aplikacji.

Instalujemy ją przy pomocy wcześniej wspomnianego PIP-a:

Listing 4.2. Instalacja PyTorch przy użyciu PIP-a

```
python.exe -m pip install torch
```

Dzięki integracji bibliotek PyTorch ze standardową biblioteką Python i otaczającym ekosystemem, ładowanie najpopularniejszych rodzajów danych i konwertowanie ich do tensorów PyTorch jest wygodne. Biblioteki takie jak PyTorch pozwalają efektywnie budować i trenować modele sieci neuronowych.

PyTorch zapewnia podstawową strukturę danych, Tensor, wielowymiarową tablicę, która ma wiele podobieństw z tablicami NumPy więcej o Tensorach w rozdziale 2.10.

Tensory przyspieszają operacje matematyczne (przy założeniu, że obecna jest odpowiednia kombinacja sprzętu i oprogramowania), a PyTorch ma pakiety do rozproszonego szkolenia, procesy robocze dla efektywnego ładowania danych oraz obszerną bibliotekę wspólnych funkcji głębokiego uczenia.

PyTorch stanowi zarówno doskonałe wprowadzenie do głębokiego uczenia się, jak i narzędzie przydatne w profesjonalnych kontekstach do pracy na wysokim poziomie w świecie rzeczywistym.

[3]PyTorch ma "Py"z Pythona, ale jest w nim dużo kodu innego niż Python. Ze względu na wydajność większość PyTorch jest napisana w C++. Zarówno tensory, jak i powiązane operacje mogą działać na CPU lub GPU.

Uruchomienie na GPU powoduje ogromne przyspieszenie w porównaniu z procesorem, a dzięki PyTorch nie wymaga więcej niż dwóch dodatkowych wywołań funkcji. Druga podstawowa rzecz, którą zapewnia PyTorch, pozwala tensorom śledzić wykonywane na nich operacje i obliczać pochodne wyjścia w odniesieniu do dowolnego z jego danych wejściowych w sposób analityczny poprzez wsteczną propagację.

PyTorch może być wykorzystywany do fizyki, renderowania, optymalizacji, symulacji, modelowania itp.

Modele głębokiego uczenia automatycznie uczą się kojarzyć dane wejściowe i pożądane wyniki z przykładów.

W najprostszym przypadku model wykona wymagane obliczenia na lokalnym procesorze lub na pojedynczym GPU, więc gdy pętla treningowa zawiera

dane, obliczenia mogą rozpocząć się natychmiast. [3, 10, 12] Częściej jednak trzeba korzystać ze specjalistycznego sprzętu, takiego jak wiele procesorów graficznych lub aby wiele maszyn włączyło swoje zasoby w szkolenie modelu.

Listing 4.3. Funkcja boolowska, sprawdzająca czy dana maszyna wspiera architekturę CUDA

```
torch.cuda.is_available()
```

PyTorch domyślnie przyjmuje model natychmiastowego wykonania (tryb eager). Ilekroć instrukcja dotycząca PyTorch jest wykonywana przez interpreter Pythona, odpowiednia operacja jest natychmiast wykonywana przez bazową implementację C++ lub CUDA.

Listing 4.4. Przykład utworzenia przykładowego tensora na GPU

```
gpu = torch.tensor([[2.1, 3.7], [4.2, 0.0], [6.9, 6.9]], device='cuda')
```

Czasami wyamgane jest utworzenie Tensorów po stronie CPU a dopiero później przekazanie go do GPU :

Listing 4.5. Przykład kopiowania tensora utworzonego z CPU do GPU

```
gpu = points.to(device='cuda')
```

Ten kod zwraca nowy tensor, który ma te same dane liczbowe, ale jest przechowywany w pamięci RAM GPU, a nie w zwykłej pamięci RAM systemu.

Te reprezentacje zmiennoprzecinkowe są przechowywane w tensorach. Tensory to tablice wielowymiarowe i podstawowa struktura danych w PyTorch. PyTorch ma wszechstronną bibliotekę standardową do tworzenia tensorów i operacji matematycznych. Tensory można uszeregować na dysk i ładować z powrotem. Wszystkie operacje tensora w PyTorch mogą być wykonywane zarówno na CPU, jak i na GPU bez zmiany kodu.

4.3. BPY

BPY (ang. Blender Python) to biblioteka do komunikowania się z Blenderem, a konkretnej z Pythonem wbudowanym w Blendera. Wszystkie operacje, które można wykonać przy pomocy intefejsu gragicznego Blendera można też przedstawić jako kod wykonywany przy odwołaniu do BPY [14, 15].

```
bpy.ops.mesh.primitive_monkey_add(size=2, enter_editmode=False, location=(0, 0, 0))
bpy.context.space_data.context = 'MATERIAL'
bpy.ops.material.new()
bpy.data.materials["Material.001"].node_tree.nodes["Principled BSDF"].inputs[0].default_value = (0.0481753, 0.8, 0.0433253, 1)
bpy.data.materials["Material.001"].node_tree.nodes["Principled BSDF"].inputs[0].default_value = (0.0481753, 0.8, 0.0433253, 1)
bpy.context.space_data.context = 'MODIFIER'
bpy.ops.object.modifier_add(type='SUBSURF')
bpy.context.object.modifiers["Subdivision"].show_expanded = False
bpy.ops.object.modifier_add(type='DECIMATE')
bpy.context.object.modifiers["Decimate"].show_expanded = False
```

Rysunek 4.1. Przykład prostych operacji w Blenderze, przedstawone w postaci kodu z odwołaniem do BPY. Między innymi znajedzimy tu dodanie nowego materiału, zminę koloru, dodoanie modyfikatora Subdivision.

Źródło: Opracowanie własne

4.4. NumPy

NumPy jest zdecydowanie najpopularniejszą biblioteką wielowymiarową. PyTorch oferuje płynną interoperacyjność z NumPy, co zapewnia integrację z resztą bibliotek naukowych w języku Python, takie jak SciPy, Scikit-learn i Pandas.

Tensory PyTorch, w przeciwieństwie z macierzami NumPy, mają zdolność do wykonywania szybkich operacji na graficznych jednostkach przetwarzających (GPU), do dystrybucji operacji na wielu urządzeniach lub maszynach oraz do śledzenia wykresu utworzonych obliczeń im. Wszystkie te funkcje są ważne przy wdrażaniu nowoczesnej biblioteki do głębokiego uczenia się.

Tensory PyTorch można konwertować na tablice NumPy i odwrotnie. W ten sposób można wykorzystać ogromną liczbę funkcji w szerszym ekosystemie Pythona, który zbudował się wokół typu tablicy NumPy. Ta zerowa

kopia współdziałania z tablicami NumPy wynika z systemu pamięci, który współpracuje z protokołem buforującym Pythona. Ze względu na jego wszechobecność w ekosystemie nauki danych w Pythonie łatwa zamiana tensora na tablicę NumPy, aby użyć charakterystycznych dla NumPy metod bywa przydatne.

4.5. Torchvision

Torchvision składa się z popularnych zestawów danych, architektur modeli i typowych transformacji obrazu. Torchvision udostępnie wiele wytrenowanych zawczasu modeli, które są gotowe do użycia. Między innymi VGG19 używane w tej pracy.

Trzy ostatnie biblioteki są nieproporcjonalnie mało opisane. Dodałbym w jakim kontekście będą wykorzystane w pracy chociaż.

Listing 4.6. Przykład pobrania modelu VGG19 przez torchvision, więcej o VGG19 w rozdziale 5.4

```
models.vgg19(pretrained=True).features
```

4.6. Pillow

Pillow (ang. Python Image Library PIL) dodaje obsługę grafiki np. otwieranie, modyfikowanie, zapisywanie plików graficznych.

4.7. OS

Biblioteka impermentująca interfejsy systemu operacyjnego. W przypadku tej pracy pozwala wykonać instalację paczek w tle poprzez wykonanie procesu terminalowego w tle.

Listing 4.7. Przykład pobrania modelu VGG19 przez torchvision, więcej o VGG19 w rozdziale 5.4

```
os.popen(cmd)
```

ROZDZIAŁ 5

Transfer Stylu

5.1. Ogólne założenia

Algorytm Neural-Style opracowany został przez Leona A. Gatysa, Alexandra S. Eckera i Matthiasa Bethge. Neural-Style lub Neural-Transfer pozwala robić zdjęcia i odtwarzać je w nowym stylu artystycznym. Algorytm pobiera trzy obrazy, obraz wejściowy, obraz treści i obraz stylu. Następnie zmienia dane wejściowe, aby przypominały treść obrazu i styl artystyczny obrazu stylu.

Jak zostało wspimniane we wcześniejszych rodziałach sieci neuronowe zbudowane są z sekwencyjnych warstw, gdzie każda warstwa dokonuje ekstrakcji cech coraz to wyższego poziomu.

Na początku możliwe musi być analiza obrazu "stylu" i odzielenie stylu obrazka od jego zawartości. Następnym krokiem jest tranfer tego stylu z jednego obrazka do drugiego.[4]

Na podstawowym poziomie może odnosić się wyłącznie do kolorystyki (np. dominanata czerwonego) oraz tekstury (fale na całości obrazu). Może też odnosić się do bardziej specyficznych właściwosći np. stylu ruchu pędzla lub techniki malowania. Oczywiście treści i stylu obrazu nie można całkowicie rozdzielić. Podczas syntezy obrazu, który łączy zawartość jednego obrazu ze stylem drugiego, zwykle nie istnieje obraz, który idealnie pasuje do obu ograniczeń jednocześnie. Ponieważ jednak funkcja straty, którą minimalizujemy podczas syntezy obrazu, jest liniową kombinacją funkcji straty odpowiednio dla treści i stylu, możemy płynnie regulować nacisk na każdą rekonstrukcję treści lub stylu.

W przykładzie, w którym algorytm rozpoznawał cyfry, neuronem był pojedynczy piksel ze skalą szarości (ang. grayscale), w przypadku Transferu Stylu mamy dwa obrazki o pełnym spektrum RGB (red, green, blue), a

co za tym idzie neuron będzie bardziej skomplikowany. Podobnie jak w Rysunku 2.1. Tworzymy teoretyczny model warstw sieci neuronowych. W tym przypadku pierwszą warstwą jest tablica wszystkich pikseli z wartościami RGB z obrazka wejściowego RGB. Ostatnią warstwą są pixele w tej samej ilości i układzie, ale ze zmienionymi wartościami RGB.

Rysunek 5.1. Przykład zastosowania tranferu stylu Obrazy łączące treść zdjęcia ze stylem kilku znanych dzieł sztuki. Obrazy zostały utworzone przez znalezienie obrazu, który jednocześnie pasuje do reprezentacji treści fotografii i stylu kompozycji. Oryginalne zdjęcie przedstawiające wizualizację nowego budynku wydziału Informatyki Uniwersytetu Gdańskiego. Obraz, który zapewnił styl dla odpowiedniego wygenerowanego obrazu, Gwiaździsta noc Vincenta van Gogha, 1889. Na samym końcu widzimy obraz wyjścia po wykonianiu 2048 iteracji.

Źródło: Opracowanie własne

Oddzielenie treści od stylu w obrazach jest niezwykle trudnym problemem. Jednak ostatni postęp w głębokich konwolucyjnych sieciach neuronowych [4,5,7] stworzył potężne komputerowe systemy interpretacji obrazu, które uczą się wyodrębniać informacje semantyczne na wysokim poziomie z obrazów naturalnych.

Koncepcyjnie jest to algorytm przesyłania tekstur, który ogranicza metodę syntezy tekstur poprzez reprezentacje cech z supernowoczesnych sieci neuronowych konwergentnych.[4,5,7] Ponieważ model tekstury opiera się również na głębokich reprezentacjach obrazu, metoda transferu stylu elegancko ogranicza się do problemu optymalizacji w ramach jednej sieci neuronowej. Nowe obrazy są generowane przez wykonanie wyszukiwania przed obrazem w celu dopasowania reprezentacji cech przykładowych obrazów.

Kiedy konwolucyjne sieci neuronowe są szkolone w zakresie rozpoznawania obiektów, rozwijają reprezentację obrazu, która sprawia, że informacje o obiektach stają się coraz bardziej wyraźne wzdłuż hierarchii przetwarzania. Dlatego wzdłuż hierarchii przetwarzania sieci obraz wejściowy przekształca się w reprezentacje, które są coraz bardziej wrażliwe na rzeczywistą treść obrazu, ale stają się względnie niezmienne dla jego dokładnego wyglądu. Zatem wyższe warstwy w sieci wychwytują zawartość wysokiego poziomu w kategoriach obiektów i ich rozmieszczenia w obrazie wejściowym, ale nie ograniczają bardzo dokładnie wartości pikseli w rekonstrukcji. Natomiast rekonstrukcje z niższych warstw po prostu odtwarzają dokładne wartości pikseli oryginalnego obrazu Dlatego też opis funkcji w wyższych warstwach sieci nazywamy reprezentacją treści.

5.2. Model

Modelem określamy sieć neuronową z dopasowanymi wagami osiągającą wysoką skuteczńść np. w przypadku klasyfikacji. Treningiem natomiast określamy proces, w którym wagi modyfikowane są w oparciu o pewną sumę kontrolną lub inny system oceniania.

Aby trenować model, potrzeba:

- źródła danych treningowych
- optymalizatora do dostosowania modelu do danych treningowych
- pętli
- sposobu uzyskania modelu i danych sprzętu, które będą wykonywać obliczenia potrzebne do wyszkolenia modelu

Transfer stylu to złożona technika wymagająca bardzo złożonego wytrenowanego modelu. Tutaj przydatne okazują się wytrenowane już modele. VGG19 jest bardzo efektywny przy ekstrakcji cech.

5.3. Matryca Gram

Aby osięgnąć efektywną jak i efektowną ekstrakcję stylu. Minimalizujemy różnicę między rozkładami funkcji. Wyciągnięty styl wciąż przecowuje informację niepotrzebne, takie jak pozycje elementów czy ich strukturę na obrazie trzeba to wyczyścić. Trzeba wykonać przetwarzanie wstępne (ang. preprocesing) na macierzy cech, która została wyciągnięta:

$$Gram = V^T V \tag{5.1}$$

,gdzie V to dowony wektor będący jedną z cech (ang. feature) w modelu, natomiast T jest transpozycją tego wektora.

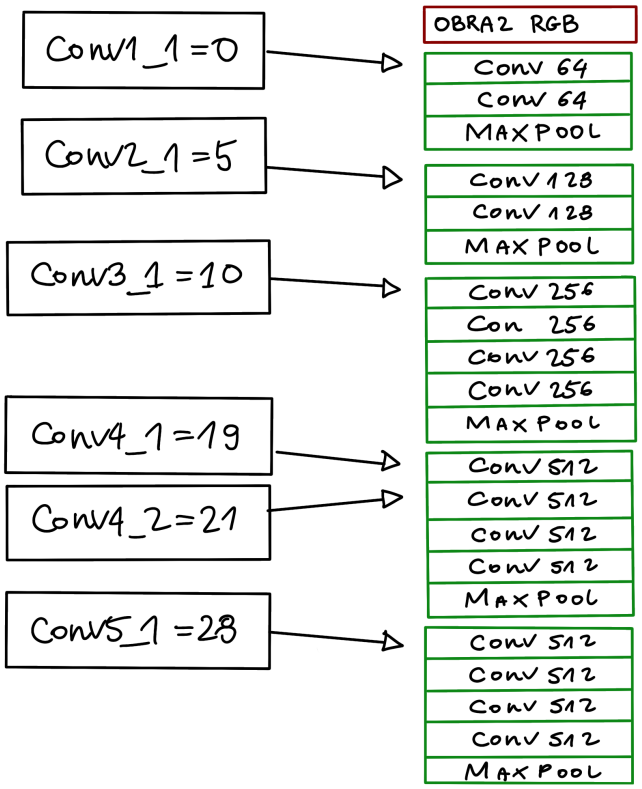
Listing 5.1. Przykład implementacji funkcji, która przetwarza tensor wedle opisanego powyżej wzoru.

```
# Gram Matrix
        def gram(tensor):
                _, d, h, w = tensor.size()
         tensor = tensor.view(d, h * w)
         gram = torch.mm(tensor, tensor.t())
         return gram
```

$$gram = torch.mm(tensor, tensor.t) \tag{5.2}$$

5.4. VGG19

VGG (ang. Visual Geometry Group) to pretrenowany model konwolucyjnej sieci neuronowej. Wytrenowana na uniwersytecie Oksfordzkim została powszechnie udostępniona. Została ona przeszkolona w zakresie rozpoznawania i lokalizacji obiektów na obrazie 2D jest szczegółowo opisana w oryginalnej pracy [5].



Rysunek 5.2. Struktura VGG19

Źródło: Opracowanie własne

Kluczowe w tym przypaku jest klasyfikacja oraz wyodrębnianie funkcji. VGG19 Wykorzystuje przestrzeń cech, jaką zapewnia znormalizowana wersja 16 konwojacyjnych i 5 warstw pulujących 19-warstwowej sieci VGG. Znor-

malizowaliśmy sieć, skalując wagi tak, aby średnia aktywacja każdego filtra konwolucyjnego względem obrazów i pozycji była równa jeden. Takie ponowne skalowanie można wykonać dla sieci VGG bez zmiany jej mocy wyjściowej, ponieważ zawiera ona jedynie prostujące liniowe funkcje aktywacyjne i nie ma normalizacji ani łączenia nad mapami cech. Nie używamy żadnej z w pełni połączonych warstw. W przypadku syntezy obrazu stwierdzono, że zastąpienie maksymalnej operacji łączenia przez średnie łączenie w pulę daje nieco bardziej atrakcyjne wyniki, dlatego pokazane obrazy zostały wygenerowane ze średnim zestawieniem w puli.

Jako, iż VGG19 to wilozadaniowy model, nie wszystkie warstwy będą przydatne przy implementacji Transferu Stylu.

Do transferu stylu potrzebne są tylko te warstwy używane do znajdywania cech. Jest to tylko 6 warstw: warstwy 0, 5, 10, 19, 28 do ekstrakcji stylu oraz warstwa 21 do ekstrakcji kontentu.

Listing 5.2.]Przykład implementacji modelu VGG19 [13]

```
# VGG19
model.vgg19(pretrained=True)

def features(image, model):

        layers = {
        '0': 'conv1_1', '5': 'conv2_1',
        '10': 'conv3_1', '19': 'conv4_1',
        '21': 'conv4_2', '28': 'conv5_1'
        }

        features = {}

        for name, layer in model._modules.items():
                image = layer(image)
                if name in layers:
                        features[layers[name]] = image

        return features
```

5.5. Szkolenie

Aby przenieść styl kompozycji z jednego obrazka na drugi, syntezujemy nowy obraz, który jednocześnie pasuje do reprezentacji treści i reprezentacji stylu. W ten sposób wspólnie minimalizujemy odległość reprezentacji cechy białego szumu od reprezentacji treści zdjęcia w jednej warstwie oraz stylowej reprezentacji obrazu zdefiniowanej na kilku warstwach konwolucyjnej sieci neuronowej.

Reprezentacje treści i stylu w konwolucyjnej sieci neuronowej są dobrze rozdzielne. Oznacza to, że można niezależnie manipulować obiema reprezentacjami, aby wytwarzać nowe, sensownie postrzegalne obrazy. Aby zademonstrować to odkrycie, generujemy obrazy, które mieszają reprezentację treści i stylu z dwóch różnych obrazów źródłowych.

Innym ważnym czynnikiem w procesie syntezy obrazu jest wybór warstw pasujących do treści i reprezentacji stylu. Jak opisano powyżej, reprezentacja stylu jest reprezentacją wieloskalową, która obejmuje wiele warstw sieci neuronowej. Liczba i położenie tych warstw determinuje lokalną skalę, w której dopasowany jest styl, co prowadzi do różnych wrażeń wizualnych [4]. Zauważyliśmy, że dopasowanie reprezentacji stylu do wyższych warstw w sieci zachowuje struktury obrazów lokalnych na coraz większą skalę, co zapewnia płynniejsze i bardziej ciągłe wrażenia wizualne. Dlatego najbardziej atrakcyjne wizualnie obrazy są zwykle tworzone przez dopasowanie reprezentacji stylu do wysokich warstw w sieci, dlatego dla wszystkich wyświetlanych obrazów dopasowujemy cechy stylu na warstwach „conv1 1”, „conv2 1”, „conv3 1 ”,„ conv4 1 ”i„ conv5 1 ”sieci.

5.6. Optymalizacja

Adam to optymalizator bazujący na metodzie gradientu prostego.

Listing 5.3. Przykład użycia optymalizatora Adam w PyTorch, więcej informacji w rodziale 5.3

```
# ustawienie optymalizatora
optimizer = optim.Adam([target], lr=0.003)

# reset gradientu
optimizer.zero_grad()

# aktualizacja
optimizer.step()
```

5.7. Ograniczenia

Prawdopodobnie najbardziej ograniczającym czynnikiem jest rozdzielczość zsyntetyzowanych obrazów. Zarówno wymiar problemu optymalizacji, jak i liczba jednostek w konwolucyjnej sieci neuronowej rosną liniowo wraz z liczbą pikseli. Dlatego szybkość procedury syntezy zależy w dużej mierze od rozdzielczości obrazu. Obrazy przedstawione w tym artykule zostały zsyntetyzowane w rozdzielczości około 512 × 512 pikseli, a procedura syntezy może zająć nawet godzinę na GPU Nvidia K40 (w zależności od dokładnego rozmiaru obrazu i kryteriów zatrzymania spadku gradientu). Podczas gdy ta wydajność obecnie zabrania stosowania online i interaktywnych aplikacji naszego algorytmu transferu stylu, prawdopodobne jest, że przyszłe ulepszenia w głębokim uczeniu również zwiększą wydajność tej metody.

Inną kwestią jest to, że zsyntetyzowane obrazy są czasami poddawane szumom o niskim poziomie. Chociaż jest to mniej problem w przypadku transferu stylu artystycznego, problem staje się bardziej widoczny, gdy zarówno treść, jak i obrazy w stylu są fotografiami i wpływa na fotorealizm zsyntetyzowanego obrazu. Jednak szum jest bardzo charakterystyczny i wydaje się przypominać filtry jednostek w sieci.

Zatem możliwe byłoby skonstruowanie skutecznych technik usuwania szumów w celu późniejszego przetwarzania obrazów po procedurze optymalizacji. Oprócz pracy nad transferem tekstur, powszechnie stosowane podejścia róż-

nią się koncepcyjnie od naszej pracy, ponieważ zapewniają wyspecjalizowane algorytmy do renderowania obrazu źródłowego w jednym określonym stylu.

Oddzielenie treści obrazu od stylu nie jest koniecznie dobrze zdefiniowanym problemem. Wynika to głównie z tego, że nie jest jasne, co dokładnie określa styl obrazu. Mogą to być pociągnięcia pędzlem na obrazie, mapa kolorów, niektóre dominujące formy i kształty, ale także kompozycja sceny i wybór podmiotu obrazu - i prawdopodobnie jest to połączenie ich wszystkich i wiele więcej.

5.8. Podsumowanie

System ten pozwala - przynajmniej w pewnym stopniu - na oddzielenie treści obrazu od stylu. Jednym z wyjaśnień może być to, że podczas uczenia się rozpoznawania obiektów sieć musi stać się niezmienna dla wszystkich odmian obrazu, które zachowują tożsamość obiektu. Reprezentacje, które uwzględniają zmienność treści obrazu i zmienność jego wyglądu, byłyby niezwykle praktyczne dla tego zadania. W świetle uderzających podobieństw między zoptymalizowanymi pod kątem wydajności sztucznymi sieciami neuronowymi a wizją biologiczną, kuszące jest spekulowanie, że ludzka zdolność do wyodrębniania treści ze stylu - a zatem nasza zdolność do tworzenia i cieszy się sztuką - może być również przede wszystkim znakiem rozpoznawczym potężnych możliwości wnioskowania naszego systemu wizualnego.

ROZDZIAŁ 6

Zakończenie

Możliwości jaki stoją przed obrazami generowanymi maszynowo są nieograniczone zarówno do prototypowania i tworzenia ”bazy” pomysłów, jak również do odciążania człowieka z zadań które jeszcze kilka lat temu były wykonalne tylko dla najlepszych specjalistów w branży.

W pracy tej zostały przedstawione podstawowych pojęć związanych z sztuczną inteligencją. Następnie omówione zostały sieci neuronowe i widzenie komputerowe jako procesy służące do interpretacji oraz przetwarzania obrazów 2D.

Aktualnie dostępnych jest szereg narzędzi umożliwiających analizę obrazu, omówione one zostały w rozdziale 4. Dla wcześniej omówionych narzędzi istnieją biblioteki, które pozwalają na wykorzystanie pełnego potencjału uczenia maszynowego i zostały one omówione w rozdziale 5.

Implementacja transferu stylu została szczegółowo omówiona w rozdziale 6. Algorytmu Neural-Style opracowany przez Leona A. Gatysa, Alexandra S. Eckera i Matthiasa Bethge w środowisku Blender. W tym rozdziale został pokazany cały proces od założeń, przez stworzenie modelu, poprzez szkolenie. Pokazano, że transfer stylu pozwala na analizować zdjęcia i odtwarzać je w nowym stylu artystycznym, ale również ma pewne ograniczenia.

Napisz co zostało pokazane w pracy, że podstawowe pojęcia dotyczące SI, sieci neuronowe, omówione zostały biblioteki. Po co zostały te rzeczy omówione. Że został pokazany transfer stylu, co pozwala, że zostały pokazane także jego ograniczenia. Jakie można podjąć kolejne kroki, aby usprawnić proces transferu stylu? (albo jakiś inny algorytm).

ROZDZIAŁ 7

Bibliografia

- [1] *Alan Turing*, "*Computing Machinery and Intelligence*", *Mind*, *vol.LIX*, *no.* 236, *Padziernik*1950
- [2] *Ian Goodfellow and Yoshua Bengio and Aaron Courville*, "*DeepLearningAnMITPressbook*", *https* : *//www.deeplearningbook.org*
- [3] *Vishnu Subramanian Deep Learning with PyTorch* *https* : *//arxiv.org/pdf/*1508.06576.*pdf*
- [4] *Leon A. Gatys*, *Alexander S.Ecker*, *Matthias Bethge* "*A NeuralAlgorithm of ArtisticStyle*"
- [5]*https* : *//www.cv−foundation.org/openaccess/content_cvp*2016/*papers/* *Gatys_Image_Style_Transfer_CVPR_*2016*_paper.pdf*
- [6] *Lemarchal*, *C.*(2012)."*Cauchy and the Gradient Method*"(*PDF*). *Doc Math Extra* : 251–254.
- [7] *https* : *//papers.nips.cc/paper/*5633 − *texture* − *syanthesis* − *using* − *convolutional* − *neural* − *networks.pdf*
- [8] *K. Simonyan and A. Zisserman.Very Deep Convolutional Networks for LargeScale Image Recognition.* *arXiv* : 1409.1556[*cs*], *Sept*.2014.*arXiv* : 1409.1556.
- [9] *GrokkingDeepLearning*, *byAndrewW.Trask*
- [10] *DeepLearningwithPyTorchbyEliStevensandLucaAntiga*
- [12]*https* : *//pytorch.org/tutorials/advanced/neural*$_s$*style*$_t$*utorial.html*
- [13] *https* : *//github.com/rrmina/neural* − *style* − *pytorch*

- [14] $https://polycount.com/discussion/205872/creating-images-with-python-in-blender$
- [15]$https://cloud.blender.org/p/scripting-for-artists/5993ed908119170ebb57164b$
- [16] $https://www.3blue1brown.com$
- [17] $https://www.manning.com/books/grokking-deep-learning$
- [18] $https://www.deeplearningbook.org$
- [19]$DeepLearning, byIanGoodfellow, YoshuaBengio, andAaronCourville.$
- [20] $https://pl.wikipedia.org/wiki/Sztuczna_inteligencja$
- [21]$http://www-users.mat.umk.pl/ rudy/wsn/wyk/wsn-wyklad-17-convnets1.pdf$

DODATEK A

Tytuł załącznika jeden

Treść załącznika jeden.

DODATEK B

Tytuł załącznika dwa

Treść załącznika dwa.

Oświadczenie

Ja, niżej podpisany(a) oświadczam, iż przedłożona praca dyplomowa została wykonana przeze mnie samodzielnie, nie narusza praw autorskich, interesów prawnych i materialnych innych osób.

.....	.....
data	podpis