Poprawa obrazów po kompresji z użyciem sieci neuronowych

Klaudia Pawluć (160681), Ryszard Knop (160452)

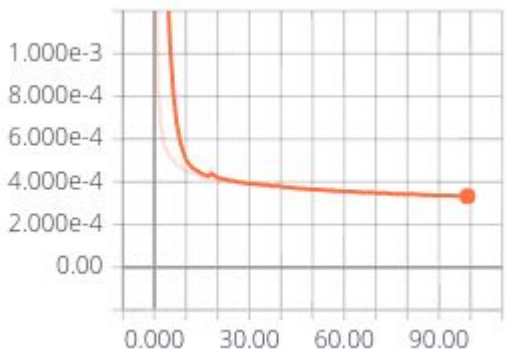
Kompresja obrazów jest powszechnie używana, aby zmniejszać rozmiar obrazów, dzięki czemu można szybciej przesłać pliki oraz zajmują mniejszy obszar w pamięci. Najbardziej powszechnym i wydajnym sposobem kompresji jest JPEG, jednakże powoduje on powstawanie artefaktów w obrazach. Nasz projekt stara się zminimalizować te artefakty i przywrócić oryginalną jakość zdjęcia na ile jest to możliwe. Artefakty te wyglądają następująco (nieskompresowane źródło po lewej, skompresowane po prawej):



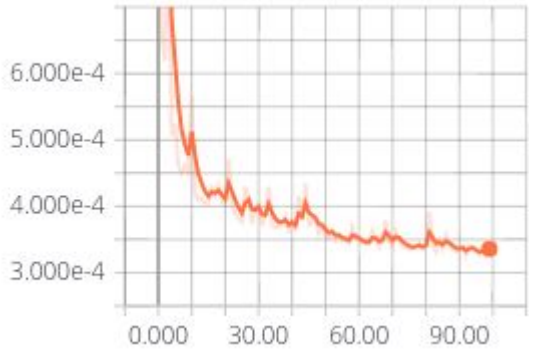

Projekt sieci:

Rozwiązanie to powstało z połączenia pomysłów zaczerpniętych z kilku innych sieci realizujących podobne zadania (odszumianie, powiększanie, naprawa) w intuicyjny dla nas sposób. Ze znanych sieci wyciągnęliśmy przybliżony sposób na naprawę i zaimplementowaliśmy siecią podaną powyżej (po sugestiach prowadzącego w trakcie konsultacji). Niestety, sieć nie odzyskuje wszystkich brakujących informacji z oryginalnego obrazka, ale dość dobrze maskuje rażące artefakty kompresji - działa lepiej niż “blur i wyostrzenie” często używane przy poprawie jakości silnie skompresowanych obrazów.

Implementacja wykonana została z użyciem biblioteki Keras (z backendem Tensorflow). Sieć uczona była optymalizatorem Adam, z funkcją straty MSE. Trening trwał 100 epok (104s każda), razem 2h50m.



Wart. funkcji straty dla zest. uczącego

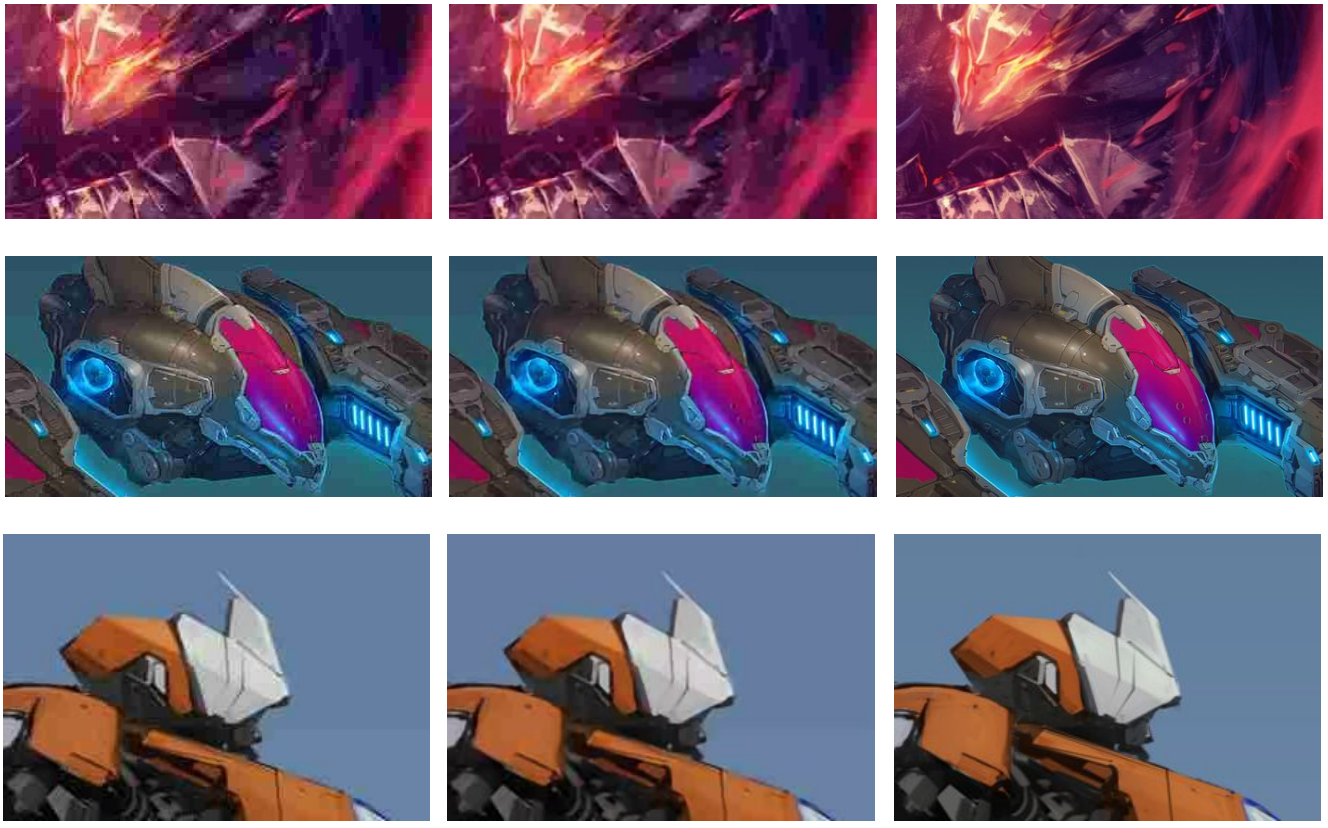


Wart. funkcji straty dla zest. testowego

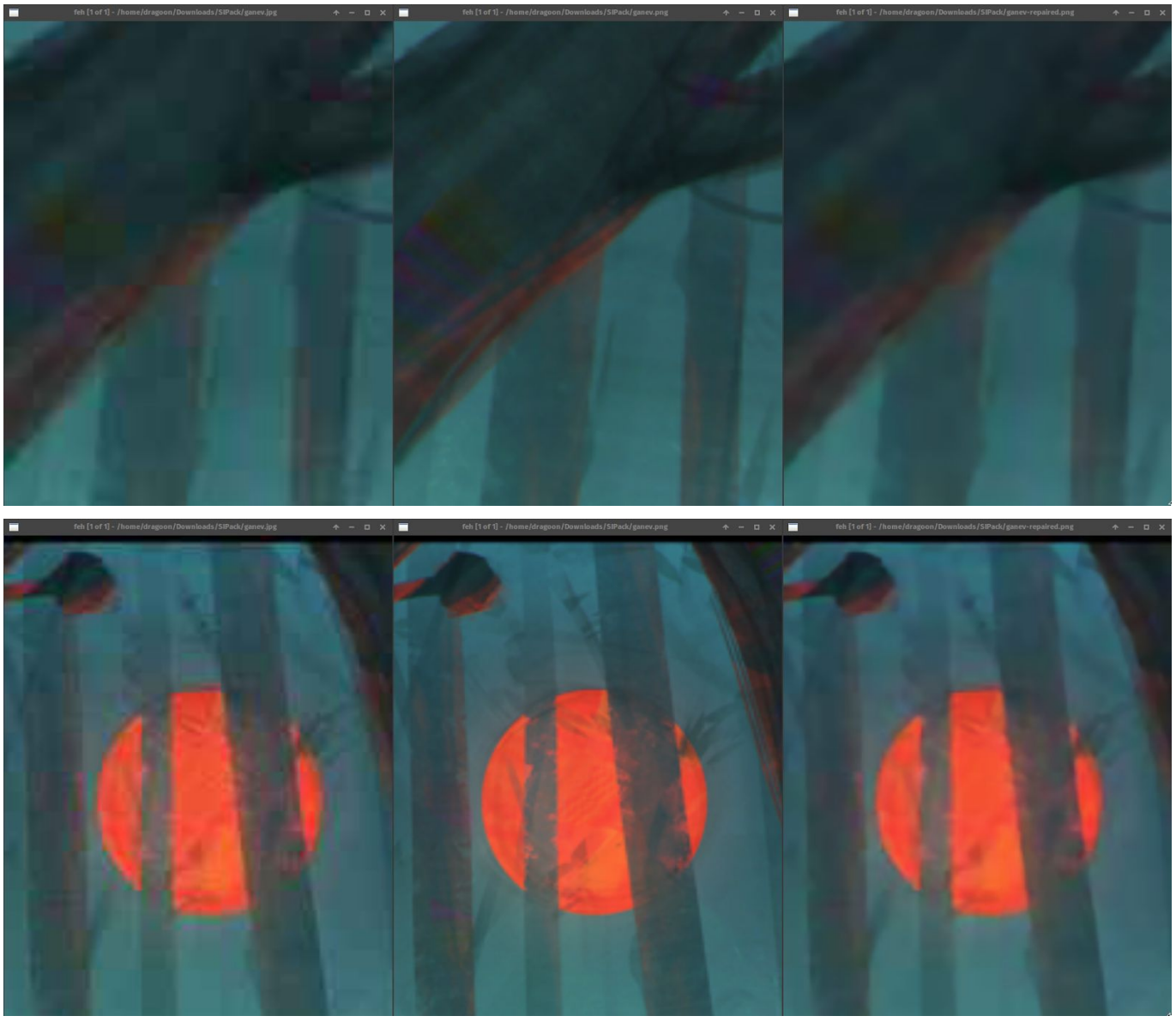
Zestaw uczący został wygenerowany na podstawie 115 rysunków (“concept art”) pochodzących z serwisów dla artystów (deviantART, ArtStation - serwisy te umożliwiają pobranie nieskompresowanych obrazów w formacie PNG jeśli artysta załadował taki format). Każde z tych zdjęć zostało skompresowane do formatu JPEG przy użyciu ImageMagicka z jakością 25. (`convert input.png -quality 25 output.jpg`). Następnie przy użyciu własnego narzędzia (`tiler.py` w paczce z kodem) każde ze zdjęć zostało pocięte na małe fragmenty 32x32 pikseli, które w odpowiadających im parach (X: JPG, Y: PNG) zostało włożone do zestawu uczącego. Zestaw testowy stworzony został przez wyciągnięcie 20% losowych fragmentów z zest. uczącego i usunięcie ich z niego. Efekt końcowy to 70376 par w zest. uczącym, 17594 w testowym. (W źródłach: `tiler.generate_tileset`)

Sieć była uczona standardowymi narzędziami dostępnymi w Kerasie, szczegóły w samej implementacji sieci w `repair.py`. W tym samym pliku dostępne są funkcje `try_model(n)` (umożliwiające szybkie przetestowanie sieci i prezentację wyników na N fragmentach 32x32) oraz `repair_image(filename)` naprawiająca zdjęcie o podanej nazwie i zapisujące je obok z sufiksem “-repaired”.

Przykłady działania algorytmu - **JPG - Naprawiony - PNG** (wymaga przybliżenia):



JPG - PNG - Naprawiony:



Próbowaliśmy też kilku innych sposobów, z czego warte uwagi były:

- Użycie funkcji straty PSNR - Często używana w dziedzinie naprawy zdjęć, próba nauczenia sieci z tą funkcją spowodowała stworzenie inwertera kolorów.
- Użycie prostszej sieci, pomijającej jeden Upsampling -> Conv2D -> MaxPooling. Sieć działała, choć nie dawała oszałamiających efektów.
- Użycie pierw MaxPoolingu, następnie Upsamplingu (często wykorzystywane w denoiserach). Powodowało efekt “wypłaszczania” kolorów (szczegóły albo “chropowate” powierzchnie otrzymywały jeden, stały kolor, znacząco niszcząc obraz).

Dodatkowe zasoby dostępne na https://stuff.dragonic.eu/ETI/SI - tilesety użyte do nauczenia sieci, obrazy źródłowe, pliki projektu itd.