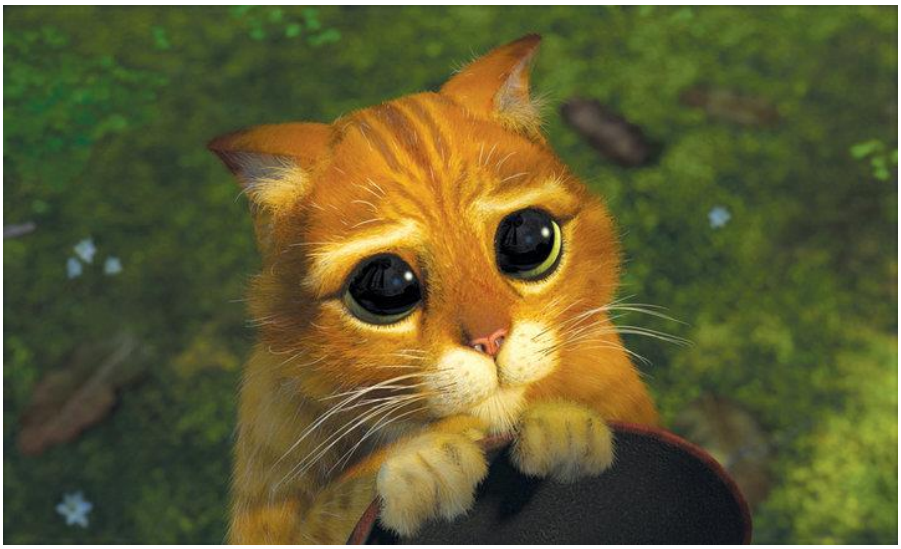
Image Style Changer

Kamil Pudlewski

1. Zadanie realizowane przez aplikację

Aplikacja którą stworzyłem umożliwia wczytanie dowolnego zdjęcia i przerobienie go w oparciu o inne zdjęcie z którego są pobierane charakterystyczne cechy obrazu, na podstawie których następuje rekonstrukcja pliku wejściowego. Całość realizowana jest poprzez sieć głębokiego uczenia. Umożliwia to uzyskanie efektu np. malunków w stylu Witkacego, czy Jana Matejki. Załóżmy że mamy rysunek kota ze słynnej bajki Shrek, który jest przedstawiony na rysunku 1.1 widocznym poniżej.



Rysunek 1.1 – Przykładowy rysunek kota

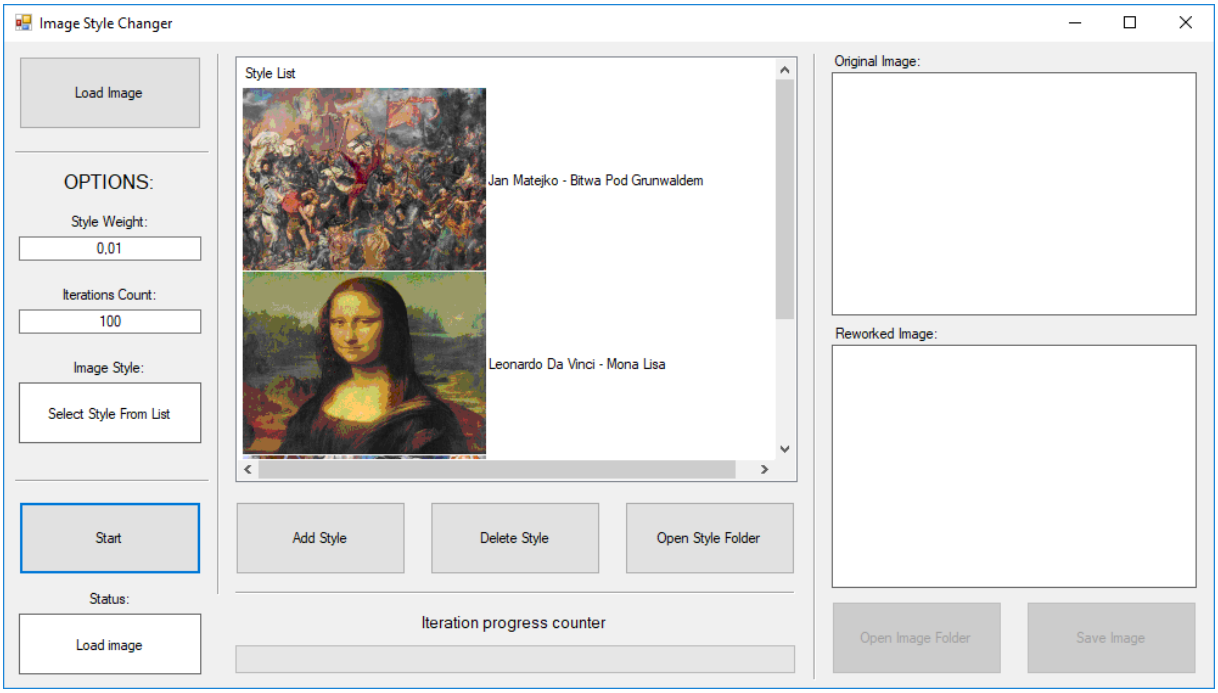
Czy ktoś kiedyś zastanawiał się jakby mógł wyglądać podobny obrazek wychodzący z rąk na przykład Witkacego? Dzięki głębokim sieciom neuronowym jest to możliwe do zasymulowania. Jako przykład dla stylu weźmiemy jeden z jego najpopularniejszych dzieł, czyli obraz zatytułowany jako Kuszenie Świętego Apostoła, widoczny na rysunku 1.2.

Rysunek 1.2 – Obraz stylu Witkacy - Kuszenie Świętego Apostoła

Odpowiedź na zadane wyżej pytanie mogłaby wyglądać mniej więcej tak jak na rysunku 1.3 umieszczonym poniżej. Taki efekt możemy uzyskać dzięki opisywanemu programowi. Końcowy wygląd będzie zależał od parametrów jakie zostaną ustawione w trakcie transferu stylu zdjęcia. Zastosowana sieć neuronowa do zmiany stylu jest techniką optymalizacji łączącą zdjęcie wejściowe oraz cechy pobrane z obrazu stylu, a następnie wymieszania ich razem, tak aby finalny efekt przypominał na przykład namalowany projekt znanego artysty.

Rysunek 1.3 – Zrekonstruowany obraz kota w stylu Witkacego

Program podzielony jest na dwie części. Pierwsza z nich jest interfejsem użytkownika. To tutaj można załadować dowolne zdjęcie do przetworzenia, wybrać odpowiedni styl i opcje transferu zdjęcia. Druga natomiast obsługuje skrypt uczenia i transformacji obrazu. Interfejs graficzny przedstawiony jest na rysunku 1.4 umieszczonym poniżej.



Rysunek 1.4 – Interfejs użytkownika aplikacji

Przycisk „Load Image” znajdujący się w lewym górnym rogu służy do wybrania dowolnego zdjęcia z dysku, które ma zostać przerobione na inny styl. Gdy to zrobimy podgląd wybranego zdjęcia pojawi się w okienku oznaczonym jako „Original Image”. W sekcjo oznaczonej jako „OPTIONS” dostępne są opcje które można dowolnie zmieniać. Pierwszą z nich jest waga stylu, oznacza ona poziom mocy odwzorowywania według którego będzie uruchomiony transfer zdjęcia. Możemy zmienić też liczbę cykli przetwarzania, która jest odpowiedzialna za nałożony efekt przerabianego zdjęcia. W środkowej sekcji programu umieszczona jest przewijana lista z obrazami, według których będzie następowało zmienianie stylu. Z listy można wybrać jedno zdjęcie które będzie przetwarzane podczas uczenia sieci. Styli można dodać więcej za pomocą przycisku „Add Style”, możliwe jest także usunięcie ich z listy poprzez wybranie opcji „Delete Style”. Niezalecane jest używanie w nazwach styli polskich znaków, gdyż mogą one powodować problemy w działaniu programu. Jeżeli jakieś zdjęcie posiada niestandardowe znaki, należy wówczas nacisnąć przycisk

odpowiadający za otworzenie folderu ze stylami i zmienić nazwę pliku ze zdjęciem. Aby odświeżyć naniesione zmiany należy uruchomić aplikację ponownie. Jeżeli wszystkie wspominane powyżej parametry zostały prawidłowo ustawione można wystartować proces przerabiania obrazu poprzez przycisk „Start”. Rozpoczęcie zostanie zasygnalizowane poprzez wyświetlenie odpowiedniego komunikatu w okienku statusu. Na środku, w dolnej sekcji programu umieszczony jest pasek progresu wraz z licznikiem który sygnalizuje podczas której iteracji znajduje się program. Po ukończeniu zmiany stylu zdjęcia, finalny obraz zostanie wyświetlony w sekcji „Reworked Image”. Wówczas można odtworzyć jego lokalizację lub zapisać na dysku. Jeżeli zapomni się zapisania pliku zawsze można go pobrać z folderu „Reworked Image” znajdującego się w głównym katalogu aplikacji, gdzie znajduje się historia przerobionych zdjęć. Pomysł i początkowy wzór aplikacji został zaczerpnięty z wpisu udostępnionego na forum Tensorflow dostępnego pod adresem: https://www.tensorflow.org/beta/tutorials/generative/style_transfer

2. Wymagania i dodatkowe biblioteki

Aplikacja jest podzielona na dwie części interfejs użytkownika wykonany w technologii C# z użyciem Windows Forms, oraz skrypt w języku Python. Zdecydowałem się na takie podzielenie ze względu na ulepszenie funkcjonalności i łatwości obsługi programu. Dzięki interfejsowi graficznemu możliwe jest łatwe wczytanie zdjęcia do przerobienia, a także obrazu z którego pobrany będzie styl służący do zamiany. Oba projekty zostały tworzone w środowisku Visual Studio.

Aby program mógł poprawnie uruchomić skrypt napisany w języku Python należy edytować plik o nazwie python.cfg znajdujący się w głównym katalogu projektu, i zmienić podaną ścieżkę na ścieżkę z zainstalowanym Pytonem. W moim przypadku użyłem ścieżki z której korzysta również środowisko Visual Studio:

C:\ProgramFiles (x86)\Microsoft VisualStudio\Shared\Python37_64\python.exe

Jeżeli aplikacja nie będzie mogła zlokalizować wskazanej ścieżki Pythona, pojawi się stosowny komunikat z błędem w okienku statusu, znajdującym się w menu graficznym programu. Dodatkowo wskazana instancja Pytona musi mieć zainstalowane biblioteki: *numpy*, *PIL*, *requests*, *tensorflow*, *os*, *time*, *datetime*. Jeżeli zawarta w pliku konfiguracyjnym instancja języka Python nie będzie zainstalowana

poprawnie, program nie wykona transferu stylu obrazu. Jeżeli po wystartowaniu zmiany stylu zdjęcia pasek progresu długo pozostaje na początku informując że nie wykonał żadnej iteracji, może to sygnalizować właśnie brak jakiejś biblioteki. Wersja biblioteki *numpy* powinna być mniejsza bądź równa 1.16.4, gdyż przy nowszych odsłonach *tensorflow* zgłasza ostrzeżenie.

Plik konfiguracyjny config.cfg generowany jest automatycznie po rozpoczęciu procesu przerabiania zdjęcia, zawiera on informacje na temat wybranych opcji takich jak zdjęcie stylu, współczynnik wagi stylu oraz liczbę iteracji.

3. Opis aplikacji

Program pobiera zdjęcie wejściowe które zostało wybrane w graficznym interfejsie, a następnie wczytuje obraz ze stylem który zostanie zaznaczony. *Po* rozpoczęciu przetwarzania aplikacja pobiera statystyki treści i stylu obrazu, dzięki zastosowaniu sieci splotowej. W celu przyspieszenia operacji wykonywanych na bardzo dużych obrazach, są one zmniejszane do rozmiaru 512x512 pikseli.

Kilka pierwszych aktywacji wejściowej sieci odpowiada za analizę zdjęć i wykrywanie krawędzi i tekstur. Ostatnie warstwy sieci odpowiadają natomiast za kształty takie jak koła czy oczy. Za wspomnianą analizę odpowiada architektura sieci VGG19 z której skorzystałem. Jest to przygotowana sieć do klasyfikacji obrazów. Ten krok jest niezbędny aby dobrze odwzorować reprezentacje stylu który został podany do transformacji.

Aby sieć mogła przeprowadzić klasyfikację obrazu musi go zrozumieć. Wymaga to potraktowania wczytanego obrazu jako pikseli wejściowych, na podstawie których budowana jest reprezentacja, która przekształca piksele, w elementy obecne na obrazie. Sieć z której korzystam używa jednej warstwy opisującej wczytane zdjęcie, natomiast warstwy stylu składają się z pięciu bloków. Warstwy pośrednie są niezbędne do zidentyfikowania reprezentacji zawartości oraz stylów z obrazów. Reprezentacja ta przedstawiona jest na rysunku 2.1.

```
content_layers = ['block5_conv2']

style_layers = ['block1_conv1',
                'block2_conv1',
                'block3_conv1',
                'block4_conv1',
                'block5_conv1']

num_content_layers = len(content_layers)
num_style_layers = len(style_layers)
```

Rysunek 2.1 – Warstwy obrazu i stylu

Dzięki zastosowaniu poszczególnych warstw splotowa sieć neuronowa jest w stanie wychwycić cechy na obrazie, czy nawet rozpoznawać obiekty. Aby poprawnie sklasyfikować obiekt potrzebny jest model, który pozwala na odpowiednie opisywanie stylu obrazu. Model w aplikacji budowany jest za pomocą funkcji *get_model(),* umieszczonego w skrypcie KerasTensorflowImage.py. Funkcja ta jest widoczna na rysunku 2.2, widocznym poniżej.

```
def get_model():
    vgg = tf.keras.applications.vgg19.VGG19(include_top=False, weights='imagenet')
    vgg.trainable = False
    style_outputs = [vgg.get_layer(name).output for name in style_layers]
    content_outputs = [vgg.get_layer(name).output for name in content_layers]
    model_outputs = style_outputs + content_outputs
    model = models.Model(vgg.input, model_outputs)
    for layer in model.layers:
        layer.trainable = False
    return model
```

Rysunek 2.2 – Funkcja tworząca model

Gdy zostanie stworzony model, program przygotowuje wszystkie dane do użycia, przygotowując odpowiednio zarówno zdjęcie przeznaczone do transferu stylu jak i obraz oznaczony jako styl. Funkcja która jest za to odpowiedzialna to get_feature_representations(model, content_image, style_image), widoczna na rysunku 2.3 umieszczonym poniżej.

```
def get_feature_representations(model, content_image, style_image):
    content_image = process_image(content_image)
    style_image = process_image(style_image)
    style_outputs = model(style_image)
    content_outputs = model(content_image)
    style_features = [style_layer[0] for style_layer in style_outputs[:num_style_layers]]
    content_features = [content_layer[0] for content_layer in content_outputs[num_style_layers:]]

    return style_features, content_features
```

Rysunek 2.3 – Funkcja przygotowująca dane

Styl obrazu opisywany jest jako korelacja różnych map obiektów odczytanych z obrazu. Te informacje można uzyskać obliczając macierz Grama, która jest liczona dla konkretnej warstwy. Funkcja odpowiadająca za jej liczenie została przedstawiona na rysunku 2.4, umieszczonym poniżej.

```
def gram_matrix(input_tensor):
    channels = int(input_tensor.shape[-1])
    a = tf.reshape(input_tensor, [-1, channels])
    n = tf.shape(a)[0]
    gram = tf.matmul(a, a, transpose_a=True)
    return gram / tf.cast(n, tf.float32)
```

Rysunek 2.4 – Algorytm obliczania macierzy Grama

Gdy macierz Grama zostanie obliczona program przechodzi do właściwego algorytmu transferu stylu. Obliczany jest średnie błąd kwadratowy wyjścia obrazu względem każdego celu, a następnie brana jest suma ważona strat. Do transferu stylu wykorzystywany jest algorytm spadku gradientu, oraz optymalizator Adam, z ustawioną stałą uczenia równą 5, beta1 na poziomie 0.99 i epsilonem 0.1. Komenda ta zwizualizowana jest na rysunku 2.5, widocznym poniżej.

```
opt = tf.compat.v1.train.AdamOptimizer(learning_rate=5, beta1=0.99, epsilon=1e-1)
```

Rysunek 2.5 – Optymalizator Adam

Algorytm liczy również tak zwany content_loss, czyli utratę treści obrazu wejściowego. Do zachowania oryginalnej treści obrazu, minimalizuje się odległość między obrazem wejściowym a wyjściowym. Kod do obliczenia starty został umieszczony na rysunku 2.6, umieszczonym poniżej.

```
def get_content_loss(base_content, target):
    return tf.reduce_mean(tf.square(base_content - target))
```

Rysunek 2.6 – Algorytm obliczania starty treści obrazu wejściowego

Podobnie jak utrata zawartości wczytanego obrazu, utrata treści stylu jest również liczona, jako odległość między dwoma obrazami. Tym razem jednak aby zastosować nowy styl, utrata ta jest definiowana jako odległość między obrazem stylu a obrazem wyjściowym, co pokazane jest na rysunku 2.7.

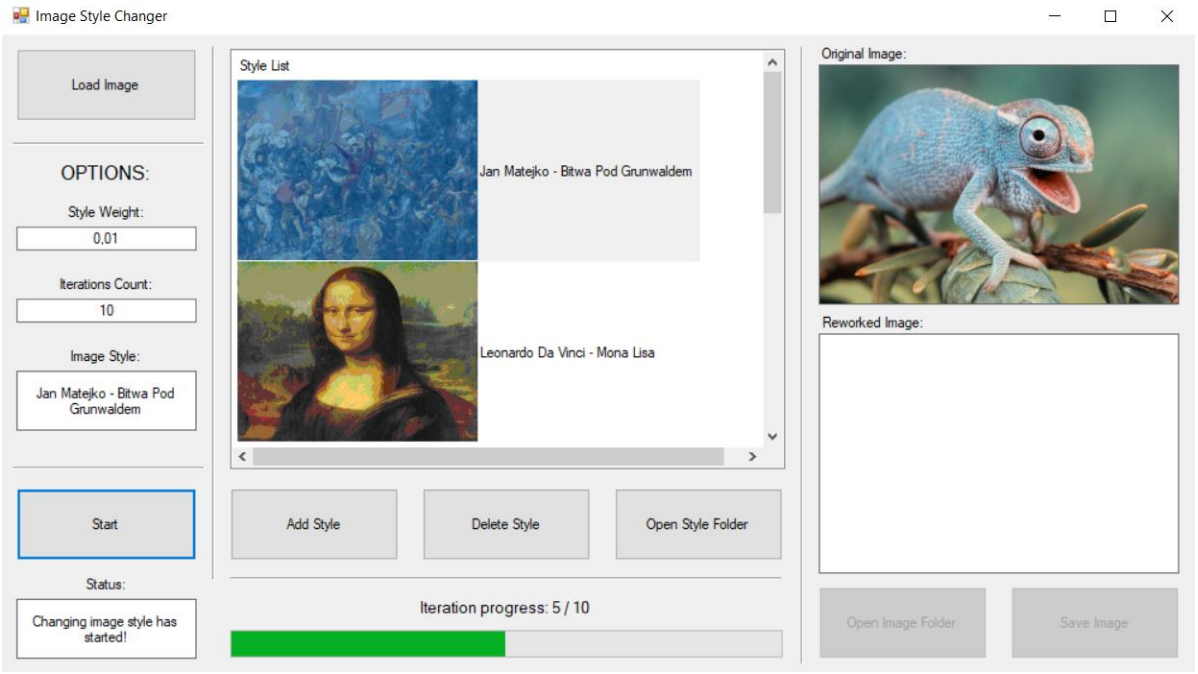
```
def get_style_loss(base_style, gram_target):
    height, width, channels = base_style.get_shape().as_list()
    gram_style = gram_matrix(base_style)
    return tf.reduce_mean(tf.square(gram_style - gram_target))
```

Rysunek 2.7 – Algorytm obliczania starty treści stylu

Na końcu ustalane są gradienty, algorytm wówczas powtarzany jest tak długo, na ile została ustawiona liczba iteracji. Program dąży do osiągnięcia jak najmniejszej starty. Dlatego po każdej iteracji jest ona sprawdzana, a ostatecznym wynikiem, czyli przerobionym zdjęciem ze zmienionym stylem jest to dla którego starta była najmniejsza.

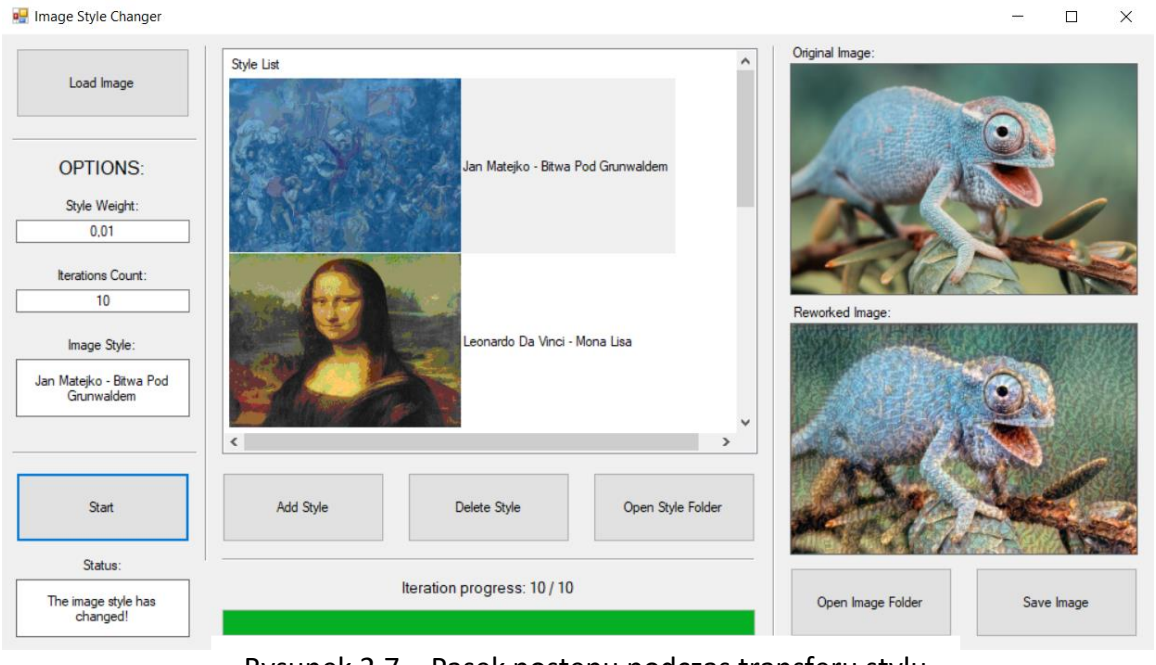
Ostatnim elementem jest zapisanie obrazu w odpowiednim katalogu, tak aby program główny razem z menu graficznym wiedział że praca sieci została zakończona. Więcej o systemie wymiany informacji opisuję poniżej.

Gdy użytkownik wczyta zdjęcie do przerobienia i wybierze odpowiednie opcje w graficznym interfejsie wraz z obrazem stylu, uruchamiany jest skrypt który rozpoczyna analizowanie podanych plików wejściowych, następnie tworzony jest model i macierz Grama. Po czym rozpoczynany jest proces transferu stylu do zdjęcia które użytkownik wybrał na początku. Podczas gdy przetwarzane są kolejne iteracje pasek progresu wzrasta. Tą sytuację można zaobserwować na rysunku 2.6, umieszczonym poniżej. Sprawdza to osobny wątek który działa w tle.



Rysunek 2.6 – Pasek postępu podczas transferu stylu

Gdy pasek postępu dojdzie do końca, należy chwilkę odczekać aż przerabiany obraz zostanie zapisany. Gdy wszystko się skończy miniaturka wyjściowego obrazu pojawi się w okienku „Reworked Image”. Wtedy też odblokują się dwie opcje otworzenia folderu ze zdjęciem gdzie możemy go otworzyć w pełnej okazałości, lub zapisać obraz w dowolnym miejscu na dysku, widoczne na rysunku 2.7.



Rysunek 2.7 – Pasek postępu podczas transferu stylu

Za odczytywanie informacji o statusie przetwarzanego zdjęcia odpowiedzialny jest „background worker” działający na osobnym wątku. To ten proces sprawdza co chwilę czy udało się zakończyć kolejną iteracje, aktualizując przy tym pasek postępu. Z tym wiązał się kolejny kłopot do rozwiązania, gdyż pracując na osobnym wątku nie mogłem bezpośrednio zaktualizować progres bar. Udało mi się rozwiązać tą sytuację stosując Invoke, który wywołuje anonimową metodę zdefiniowaną przez delegata, co zostało przedstawione na rysunku 2.8 poniżej. Podobną technikę zastosowałem do odświeżenia miniaturki przerobionego obrazu, w momencie gdy sieć zakończy pracę.

```
if (tmpProgressNumber != getProgressCount())
{
    this.Invoke((MethodInvoker)delegate ()
    {
        incrementProgressBar();
        tmpProgressNumber = getProgressCount();
    });
}
```

Rysunek 2.8 – Aktualizacja paska postępu przez background workera

Gotowy przetworzony plik trafia do folderu „Reworked Image” który znajduje się w głównym katalogu projektu. Jego nazwa będzie oznaczana liczbą, im jest ona większa tym plik jest nowszy. Dzieje się to tak dlatego aby w jednym folderze umieszczane były wszystkie przerobione przez sieć zdjęcia. Oczywiście istnieje możliwość zapisania pliku w dowolny folder poprzez naciśnięcie klawisza „Save Image”. Klawisz „Open Image Folder” otwiera roboczą ścieżkę gdzie będzie się znajdował tylko jeden aktualny obraz przez co nie dochodzi do pomyłki przy otwieraniu przerobionego zdjęcia.

4. Przykładowe wyniki

Do przetestowania działania programy użyłem znalezione w sieci zdjęcie kameleona które w oryginale zostało przedstawione na rysunku 3.1 który znajduje się poniżej.

Rysunek 3.1 – Obraz kameleona użyty do testów

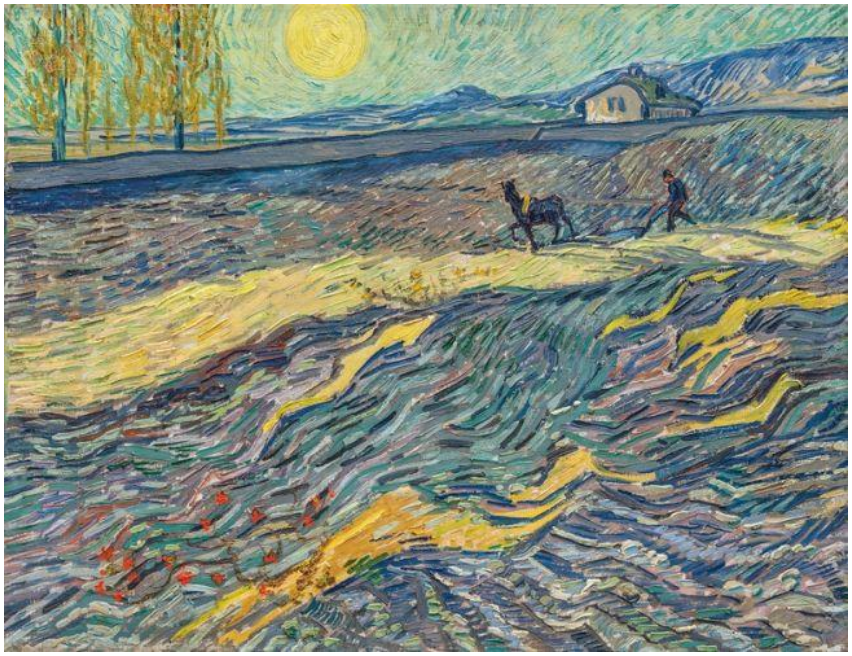
Na początek sprawdziłem trzy róże style dla tej samej liczby iteracji oraz wagi stylu. Ustawiłem ilość powtórzeń na sto, oraz wagę stylu wynoszącą 0.01. Wybrałem jako styl obraz Witkacego Kuszenie Świętego Apostoła, przedstawiony na rysunku 3.2.

Rysunek 3.2 – Styl Witkacy - Kuszenie Św Apostoła

Obraz który był wynikiem algorytmu został przedstawiony poniżej, na rysunku 3.3. Jak można zauważyć zdjęcie które otrzymałem znacząco się różni od oryginalnego, i posiada cechy stylu z wybranego obrazu Witkacego. Warto wyszczególnić bardzo dobre odwzorowanie kształtu koła co jest zasługą działania architektury sieci VGG19.

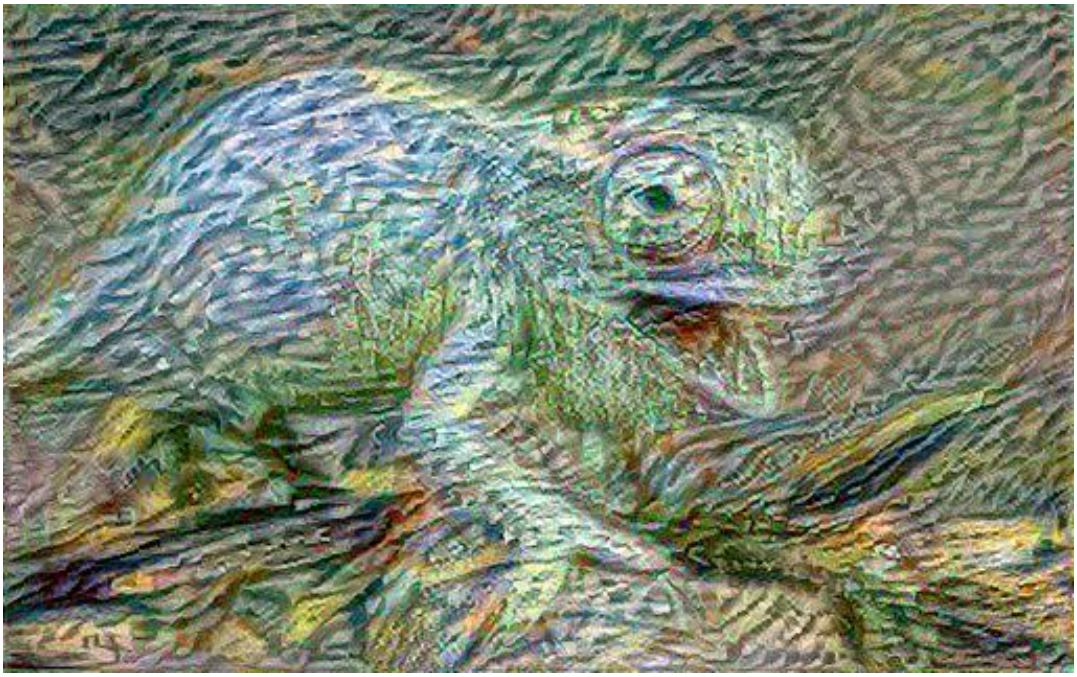
Rysunek 3.3 – Obraz kameleona ze stylem Witkacego

Kolejnym stylem dla tych samych parametrów był słynny obraz Vincenta Van Gogha zatytułowany jako Rolnik W Polu, przedstawiony na rysunku 3.4.



Rysunek 3.4 – Styl Vincent Van Gogh - Rolnik W Polu

Podobnie jak wcześniej obraz wyjściowy który otrzymałem znacznie różni się od poprzednika. Wynik został przedstawiony na rysunku 3.5, umieszczonym poniżej. W tym przypadku zdjęcie wyszło bardzo podobne do wskazanego wcześniej obrazu, a efekt końcowy bardzo pozytywnie mnie zaskoczył.



Rysunek 3.5 – Obraz kameleona ze stylem Vincenta Van Gogha

Trzecim obrazem który posłużył mi za styl upiększenia kameleona było dzieło Jana Matejki pod tytułem Bitwa Pod Grunwaldem, który został zaprezentowany na rysunku 3.6.

Rysunek 3.6 – Styl Jan Matejko - Bitwa Pod Grunwaldem

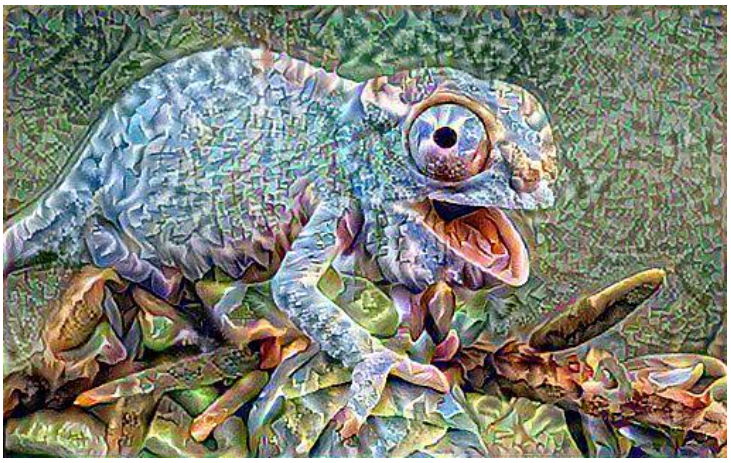
Wynik tego eksperymentu wygląda inaczej niż poprzednie i został pokazany na rysunku 3.7 poniżej. Tym razem obraz wyszedł dziwnie rozmazany, jest to skutkiem wielu elementów znajdujących się na stylu z obrazem które nie tworzą razem spójnej struktury.

Rysunek 3.5 – Obraz kameleona ze stylem Jana Matejki

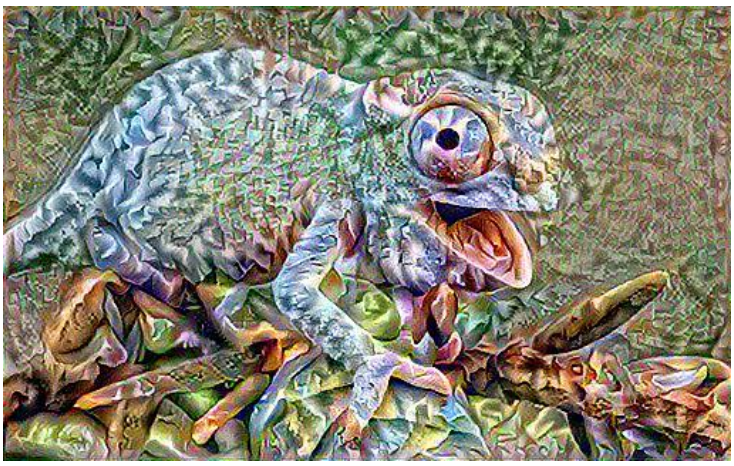
Jak można zauważyć zdjęcie z stylu skutkuje sporymi zmianami które otrzymujemy. Jednak w poprzednio zaprezentowanych przypadkach nie zmieniałem parametrów liczby iteracji, oraz wagi stylu. Stylem który najbardziej mi się spodobał jest obraz Witkacego i z niego będę korzystał w reszcie porównań. Ustawiłem wagę stylu na poziomie 0.01 i ustawiłem liczbę iteracji odpowiednio na 1, 10, 50 i 100. Wyniki zaprezentowane są w zestawieniu poniżej.

Iteracje: 1

Iteracje: 10



Iteracje: 50



Iteracje: 100

Wynik tej operacji pokazuje, że jedna iteracja lekko rozmyła tło, po dziesięciu cyklach widać wyraźne akcenty wynikające z połączenia stylu. Pomiędzy pięćdziesięcioma a stoma iteracjami różnica jest już znacznie mniejsza gdyż na obu zdjęciach obraz jest bardzo zmodyfikowany.

Kolejnym testem jaki przeprowadziłem było ustawienie liczby iteracji na 10 oraz wybranie tego samego stylu z obrazu Witkacego. Tym razem zmianie poddałem wagę stylu, wartości zmieniałem kolejno na 0.1, 0.01, 0.001, 0.0001. Wyniki tej operacji zostały przedstawione poniżej.

Waga stylu: 0.1

Waga stylu: 0.01

Waga stylu: 0,001 Waga stylu: 0,0001

Zmiana wagi stylu nie przynosi już tak spektakularnego efektu jak ilość iteracji. Istnieje różnica w nałożonym na zdjęcie efekcie, przy bardzo małych wagach jest mniej zagęszczony.

5. Podsumowanie

Aplikacją służąca do zmiany stylu zdjęcia pokazuje duże możliwości zastosowania głębokich sztucznych sieci neuronowych do przetwarzania zdjęć. Program jest w stanie zinterpretować, kolory, krawędzie a także kształty widoczne na obrazach. Po testach mogę stwierdzić że szczególnie dobrze wychodzą przerobione kształty oczu, czy kół, gdyż wtedy sieć która nakłada efekt na zdjęcie mocno je akcentuje. Efekt ten dobrze jest widoczny chociażby na oku testowanego kameleona. Bardzo dużym plusem jest to, że praktycznie z każdego obrazu sieć jest w stanie odczytać cechy i na ich podstawie stworzyć macierz Grama, co jest podstawą do efektownego łączenia obrazów. Ciekawostką jest to że sieć nawet z pustego obrazu ustawionego jako styl potrafi pobrać dane i nałożyć na dowolne zdjęcie. Jedynym minusem jaki zauważyłem to dość długi czas oczekiwania na przerobienie zdjęcia, jeśli ustawi się liczbę iteracji na 100 lub więcej to trzeba trochę poczekać. Podsumowując możliwości głębokich sieci neuronowych przeznaczonych do analizy obrazu mnie zaskoczyły, nie myślałem że odwzorowanie stylu może być wykonane przez komputer tak dobrze.