Using a pre-trained neural network (VGG16) to solve the image style transfer problem.

ABSTRACT: In the operation of any classification algorithm or face detection, two logical blocks should be distinguished: a feature extractor and a classification mechanism. The action of the extractor is based on the selection of information useful for the classifier from a huge input data stream. In this work, we will consider how to adapt a pre-trained model to solve the problem of classifying and transferring an image. Our main contribution is the proposal of a new method for processing and transferring image style based on the pre-trained VGG16 model.

Keywords: face recognition, image recognition, convolutional neural networks, Regional Convolutional Neural Networks (R-CNN) model, bounding box, anchor.

RESUME: Dans le fonctionnement de tout algorithme de classification ou de détection de visages se compose de deux blocs logiques : un extracteur de caractéristiques et un moteur de classification. L'extracteur extrait les informations utiles au classificateur à partir d'un flux massif de données d'entrée. Dans ce travail, nous étudions comment adapter un modèle pré-entraîné pour résoudre le problème de la classification d'images et du transfert de style d'images. Notre principale contribution est la proposition d'une nouvelle méthode de traitement d'images et de transfert de style d'image basée sur un modèle VGG16 pré-entraîné.

Mots-clés : reconnaissance d'images, transfert de style, modèle pré-entraîné, réseaux de neurones de convolution (R-CNN), cartes de caractéristiques, couches.

UNDER PEER REVIEW IN IJAR

INTRODUCTION

VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman of Oxford University in their article [1], [9]. This model achieves 92.7% accuracy, ranking among the top 5, in ImageNet object recognition tests. This dataset comprises over 14 million images distributed across 1,000 classes. VGG16 is one of the best-known models to have participated in the ILSVRC-2014 competition. It is an improved version of AlexNet that replaces the large filters (sizes 11 and 5 respectively in the first and second convolutional layers) with several consecutive 3x3 filters. The VGG16 network was trained for several weeks on NVIDIA Titan Black graphics cards [8].

This article presents the results of a new image style processing and transfer method based on the pre-trained VGG16 model.
In the first part, we will examine in detail the ready-to-use pre-trained models in PyTorch.
In the second part, we will see an example of using pre-trained models for project implementation and fine-tuning.
In the third part of this work, we will implement a network project using a new image style processing and transfer method based on the pre-trained VGG16 model.

PRE-TRAINED MODELS IN THE PYTORCH LIBRARY

PyTorch is an open-source machine learning framework for Python, based on Torch. It is used to solve various problems, including computer vision and natural language processing [7]. It is primarily developed by Facebook's artificial intelligence research team. It is written in Python, C++, and CUDA.

2.1 DATASETS

ImageNet is a dataset consisting of over 15 million high-quality, labeled images, divided into 22,000 categories. The images were retrieved from the internet and manually annotated by human operators via the Amazon Mechanical Turk crowdsourcing platform. In 2010, as part of the Pascal Visual Object Challenge, the annual ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) was launched. The ILSVRC uses a subset of

ImageNet, containing 1,000 images for each of the 1,000 categories. Thus, the training set comprised approximately 1.2 million images, the validation set 50,000 images, and the test set 150,000 images.

2.2 VGG16 ARCHITECTURE

The architecture of VGG16 is shown in the diagram (Figure 1) below. Images are processed through a stack of convolutional layers using filters with a very small 3x3 receiving field (the smallest size that allows for capturing the concepts of left/right, top/bottom, and center). In one configuration, a 1x1 convolutional filter is used, which can be represented as a linear transformation of the input channels followed by nonlinearity. The convolution stride is set to 1 pixel. The spatial padding of the convolutional layer input is chosen to preserve spatial resolution after convolution; that is, it is equal to 1 for the 3x3 convolutional layers. Spatial pooling is performed using five max-pooling layers that follow certain convolutional layers (not all convolutional layers are followed by max-pooling). The max-pooling operation is performed on a 2x2 pixel window with a step size of 2. After the stack of convolutional layers, there are three fully connected layers: the first two have 4,096 channels each, and the third has 1,000 (because the ILSVRC competition requires classifying objects into 1,000 categories; therefore, one channel corresponds to one class). The configuration of the fully connected layers is the same in all neural networks of this family [10].

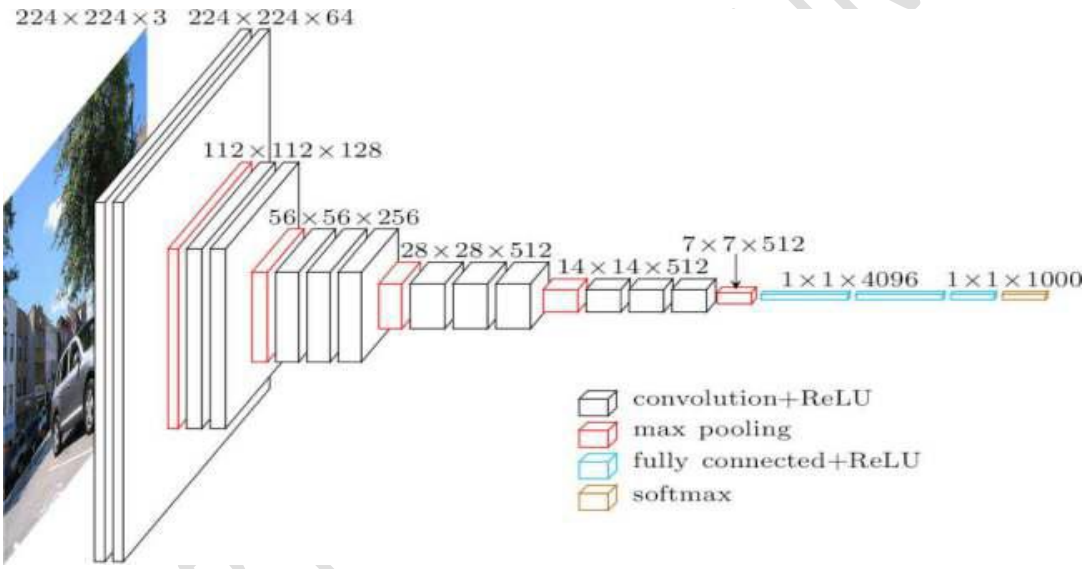


Fig.1. VGG16 Architecture

2.3 CONVOLTAGE NETWORK CONFIGURATIONS

The configurations of the convolutional neural networks are shown in Figure 2. Each network corresponds to a specific configuration (A-E). All configurations share a common structure and differ only in depth: from 11 weighted layers in network A (8 convolutional layers and 3 fully connected layers) to 19 (16 convolutional layers and 3 fully connected layers). The width of the convolutional layers (number of channels) is relatively small: from 64 in the first layer to 512 in the last, with the number of channels doubling after each max-pooling layer. Next comes a single-layer block – Adaptive Average Pooling – followed by the third block, the "classifier," composed of a sequence of fully connected layers of 4096 neurons and a final layer of 1000 neurons, trained on the ImageNet dataset (a set of 1000 classes). Let's load the trained model and try using it. `torchvision.models.vgg16(pretrained=True).eval()`. The models are located in the `models` module. We load `vgg16(pretrained=True)` with the trained weights for evaluation. `eval()`. `class_idx = json.load(open("imagenet_class_index.json"))`.
`imagenet_class_index.json` contains information about the classes in the format "identifier": "class name". It contains the total number of classes.

transforms.ToTensor(), transforms.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)).For each image, we apply a set of transformations: we first convert it into a tensor, then we normalize it with the parameters best suited to its processing. The architecture used for rendering, that is, everything we have drawn, is VGG16 [9].

The AdaptiveAvgPool2d layer transforms any input into a 7x7 matrix. For a 21x14 input matrix, the convolution window will be 2x3 and the output 7x7. A 2x2 matrix will also produce a 7x7 matrix. Thus, any tensor entering this layer generates a 7x7 tensor as output. This system is universal: any input image will produce a 7x7 matrix as output, allowing the addition of custom layers.

pk = transform(pk).

Applying the specified set of transformations yields a tensor of size 3 × 600 × 600.

`vgg16.avgpool(pk.unsqueeze(0))`.

If accessed directly, the output will be 1 × 3 × 7 × 7. In other words, regardless of the size of the input image, we will always obtain a vector of dimensions 3 × 7 × 7. `tns = np.random.random(size=(1,3,2,2))`. Even when trying a different dimension, we always get the same result. `nn.Softmax()(vgg16(im.unsqueeze(0))).view(-1)`. `im.unsqueeze(0)` adds a dimension before passing it to the model.

Next, we call `nn.` Softmax and .view(1)` to flatten it into a single vector. `torch.argsort(probas, descending=True)` allows sorting the array by probabilities and displays the 5 maximum probabilities obtained: `idx2label[sortedOutputs[i]], probas[sortedOutputs[i]])`.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Fig. 2. Convolutional neural network configurations

USE OF PRE-TRAINED MODELS FOR PROJECT IMPLEMENTATION AND FINE TUNING

3.1 DESCRIPTION OF THE IMAGE PROCESSING AND STYLE TRANSFER METHOD BASED ON THE PRE-TRAINED VGG16 MODEL.

We are interested in the problem of identifying dog breeds from images, among 60 breeds. In a standard dataset, if there are several breeds, it is preferable to have 3 or 4, but most often only the class "dog" is present. We could use a pre-trained weightless model and train it, but this is resource-intensive and requires a very large database: 10,000 images per breed and unlimited resources. Fine Tuning allows us to use an existing model, keep the convolutional layers, and implement a custom classifier. We will remove the 1,000-class classifier and replace it with the 60-neuron Dense layer classifier. We will freeze the convolutional layers; they can already recognize features, but we need to build a model with our own classes [5]. Another possibility would be to train only the classifier; this is the first option.

The second option is to train the entire network, but with a small step: level 1e-5 - 1e-7, so that the trained weights do not lose their values and adapt to our task.

A symbiotic approach is also possible: we first train only the classifier with a normal step size, then we reduce our step size and retrain the entire model to modify the weights and adapt it to our image.

We remove the standard classifier and, to recognize images from the CIFAR10 database, we must add a classifier: a fully connected layer with 10 neurons.

The input is therefore 3 × 32 × 32. The first layer is 64 × 32 × 32, then after MaxPooling, it will be 128 × 16 × 16, MaxPooling 256 × 8 × 8, MaxPooling 512 × 4 × 4, MaxPooling 512 × 2 × 2, and then another MaxPooling layer of 512 × 1 × 1. The output of AdaptiveAvgPool2d is 512 × 7 × 7, which will always be the same. A fully connected layer of 10 neurons must be added.

This gives us a ready-to-use model for solving our problems, complete with its own classifier. However, the convolutional neural network is too powerful for small models, which explains its limited accuracy of 61%. We can increase the size of the input image or reduce the number of layers to improve this accuracy.

IMPLEMENTATION OF A NETWORK PROJECT WITH IMAGE STYLE TRANSFER.

4.1 STYLE TRANSFER

If, for example, we draw a parrot in the style of the second drawing, what will the resulting image be? In this specific case, we don't have a comprehensive database on which to perform traditional training. We only have two images: the original and the stylized drawing. Simply superimposing the two images isn't enough; each element—the wing, the beak, the black patch on the back—must also be colored according to the color palette of the stylized drawing. In other words, the color of the beak must match the color of the sky in the stylized drawing. The error is calculated from two parts: the image error and the style error (Figures 3, 4, and 5).

To preserve the context of the original image during the convolutional layers, it is necessary to extract its contours. The first convolutional layer returns the final model: 64 feature maps, corresponding to the data extracted from the original image. Since clear images are paramount, contouring is performed using a VGG16 or VGG19 network, with each block receiving an additional convolutional layer.

This method receives an image and extracts a list of features from each layer. If an index is present in the list of indexes, the corresponding features are added to the list for all layers—five elements from each layer.



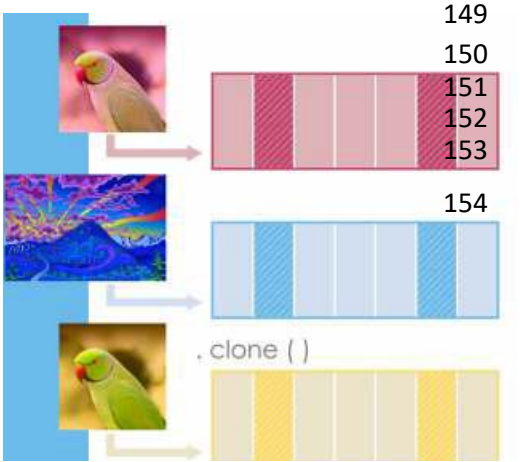


Fig. 3. Diagrams of the problem formulation

ORIGINAL

UNDER PEER REVIEW IN IJAR

STYLE

GENERATED

Fig. 4. Diagrams of the problem formulation

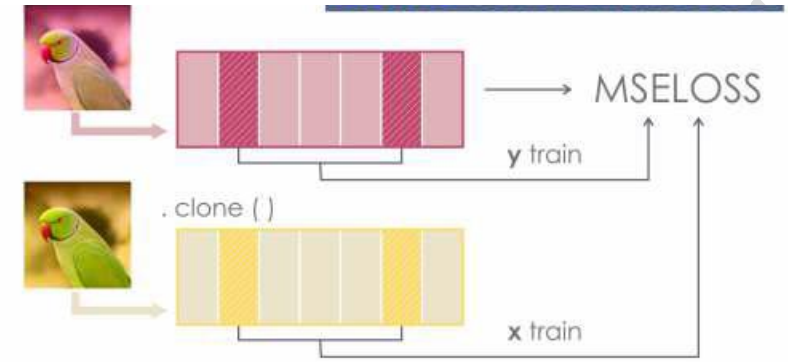


Fig. 5. Diagrams of the problem formulation

4.2. DESCRIPTION AND RESULT OF THE IMAGE STYLE TRANSFER METHOD

We consider the case of a source image depicting a parrot. We will identify the parrot if we can distinguish its silhouette, head, beak, eyes, wings, and certain characteristic shapes. Furthermore, even if the parrot is not green, but red, for example, this should not prevent us from identifying it. In addition, we must determine the style (lighting, image texture, processing method). Our goal is to make the parrot's features visible in the final image.

For the original image (what we want to preserve), the neural network extracts features, several of which are related to the image's contours—in this case, those of the parrot. These features can be extracted from different layers of the network and compared. Let's take the original image and compare the features learned by the network for the original image and the target image. We will use the mean squared error (MSE) to calculate the difference. Thus, if a feature is missing, the difference will be significant. If the contours are preserved, the error will be smaller. We will calculate the MSE for all the feature maps provided to the network.

For content error, we will process the original image through 19 convolutional layers and collect a set of feature maps from specific layers. The random assignment of these maps is irrelevant; we will collect these examples for our original image. We proceed in the same way for our target image. All that remains is to compare these sets pairwise and calculate the standard error, which is essential for preserving outlines. We will facilitate the work of our network by copying the original image as the output. For the output image, we initialize the process by copying the original image to help the network maintain shapes [1],[3].

Style error: We will extract the same information as for image error, but the value will be different. By combining a parrot and mountains, we need to extract color information using the Gram matrix (correlation matrix). The first convolutional layer produces a set of 64 feature maps. Next, a zero filter (parrot images) is applied. We multiply the image features element by element and sum them. Black is assigned a value of 0, and

UNDER PEER REVIEW IJAR

if the features do not intersect, the value will be close to zero. The more features intersect, the higher the correlation value. This process is repeated for all possible variations; the output is a 64 × 64 matrix.

Let's define the correlation matrix and the Gram matrix:

To determine stylistic similarity, it is advisable to use the Gram matrix, calculated using the following formula:

$$G_l(x) = \frac{1}{H_l W_l} F_l(x)^T F_l(x) \in R^{cl \times c_l}$$

Integrated as an n-dimensional matrix, the Gram matrix is equal to the uncentered covariance matrix. It thus contains information about the interdependencies between the channels of the feature map.

The intersection corresponds to the correlation values calculated for the reference image and the target image. We calculate a correlation matrix for each of the five feature maps, as well as for the target image. These five feature maps are then used to apply the mean squared error (MSE) to the pairwise correlation matrices. This allows us to overlay the images onto the matrix. The method receives an image and extracts a list of features from each layer. If an index matches the index list, the features are added to that list. Each layer contains five elements.

RESULT:

We optimize the weightings so that the similarity per pixel decreases and is replaced by style, without being displaced.

Loss=content loss + style loss



Fig. 6. Transfert style image

We reset the gradients and calculate the weights. We then obtain and modify the values. We train the model: the content error decreases. The changes are minimal, but the objects become increasingly distinct. Contrast appears.

MULTI-STAGE TRAINING:

By performing the training in different steps, we process the images through the model to extract their features. In the first step, the features of the generated image and the initial image are identical. We resize the tensors to calculate the correlation matrices and apply coefficients to adjust the emphasis of style over content using the following formula:

$$styleLoss+ = contentCriteria(f_1, f_2) \times kf_2$$

Next, we reset the gradients and calculate the weights. Then, we obtain and modify the values. We train the model: the content error decreases. The changes are minimal, but the objects become increasingly distinct. Contrast appears.

UNDER PEER REVIEW IN IJAR

The first 100 eras

```
Шаг [10/600], Ошибка для оригинала: 108.6528, Ошибка для стиля: 1.5729523187009978e+19
Шаг [20/600], Ошибка для оригинала: 142.9272, Ошибка для стиля: 6.421507996269937e+18
Шаг [30/600], Ошибка для оригинала: 143.3061, Ошибка для стиля: 3.7223600570740244e+18
Шаг [40/600], Ошибка для оригинала: 145.3548, Ошибка для стиля: 2.437901502535172e+18
Шаг [50/600], Ошибка для оригинала: 144.7937, Ошибка для стиля: 1.7428130663214612e+18
Шаг [60/600], Ошибка для оригинала: 145.2337, Ошибка для стиля: 1.3452978314385818e+18
Шаг [70/600], Ошибка для оригинала: 145.3654, Ошибка для стиля: 1.0957929420119081e+18
Шаг [80/600], Ошибка для оригинала: 145.6941, Ошибка для стиля: 9.267002857268183e+17
Шаг [90/600], Ошибка для оригинала: 145.9705, Ошибка для стиля: 8.040705169303798e+17
Шаг [100/600], Ошибка для оригинала: 146.1997, Ошибка для стиля: 7.1115731761732e+17
```

The first 200 eras

```
Шаг [110/600], Ошибка для оригинала: 146.4362, Ошибка для стиля: 6.375147088484762e+17
Шаг [120/600], Ошибка для оригинала: 146.6672, Ошибка для стиля: 5.77454435937026e+17
Шаг [130/600], Ошибка для оригинала: 146.8909, Ошибка для стиля: 5.272496355011461e+17
Шаг [140/600], Ошибка для оригинала: 147.0867, Ошибка для стиля: 4.8464995083616256e+17
Шаг [150/600], Ошибка для оригинала: 147.2752, Ошибка для стиля: 4.4800906943791104e+17
Шаг [160/600], Ошибка для оригинала: 147.4387, Ошибка для стиля: 4.160059968050954e+17
Шаг [170/600], Ошибка для оригинала: 147.6069, Ошибка для стиля: 3.8782702561368474e+17
Шаг [180/600], Ошибка для оригинала: 147.7597, Ошибка для стиля: 3.6287312818497126e+17
Шаг [190/600], Ошибка для оригинала: 147.9002, Ошибка для стиля: 3.406313136251208e+17
Шаг [200/600], Ошибка для оригинала: 148.0392, Ошибка для стиля: 3.2068366443636326e+17
```

The first 300 eras

```
Шаг [210/600], Ошибка для оригинала: 148.1763, Ошибка для стиля: 3.026618442233938e+17
Шаг [220/600], Ошибка для оригинала: 148.3067, Ошибка для стиля: 2.8632337631254938e+1
Шаг [230/600], Ошибка для оригинала: 148.4333, Ошибка для стиля: 2.715033339596636e+17
Шаг [240/600], Ошибка для оригинала: 148.5494, Ошибка для стиля: 2.5803309674869555e+1
Шаг [250/600], Ошибка для оригинала: 148.6617, Ошибка для стиля: 2.4568991049580544e+1
Шаг [260/600], Ошибка для оригинала: 148.7633, Ошибка для стиля: 2.3434530413923533e+1
Шаг [270/600], Ошибка для оригинала: 148.8704, Ошибка для стиля: 2.23934080075431 94e+1
Шаг [280/600], Ошибка для оригинала: 148.9707, Ошибка для стиля: 2.14345479687766e+17
Шаг [290/600], Ошибка для оригинала: 149.0706, Ошибка для стиля: 2.0550446130764186e+1
Шаг [300/600], Ошибка для оригинала: 149.1621, Ошибка для стиля: 1.9732995313238016e+1
```

The first 400 eras

```
Шаг [310/600], Ошибка для оригинала: 149.2527, Ошибка для стиля: 1.897239611881554e+17
Шаг [320/600], Ошибка для оригинала: 149.3340, Ошибка для стиля: 1.8264147421770547e+17
Шаг [330/600], Ошибка для оригинала: 149.4171, Ошибка для стиля: 1.7602521453717094e+17
Шаг [340/600], Ошибка для оригинала: 149.4974, Ошибка для стиля: 1.698578132988068e+17
Шаг [350/600], Ошибка для оригинала: 149.5741, Ошибка для стиля: 1.6408767935545344e+17
Шаг [360/600], Ошибка для оригинала: 149.6497, Ошибка для стиля: 1.5867287664643277e+17
Шаг [370/600], Ошибка для оригинала: 149.7261, Ошибка для стиля: 1.5358370117792563e+17
Шаг [380/600], Ошибка для оригинала: 149.7966, Ошибка для стиля: 1.4878751119848243e+17
Шаг [390/600], Ошибка для оригинала: 149.8646, Ошибка для стиля: 1.442617151801262e+17
Шаг [400/600], Ошибка для оригинала: 149.9358, Ошибка для стиля: 1.3998248464429875e+17
```

The first 500 eras

```
Шаг [410/600], Ошибка для оригинала: 150.0058, Ошибка для стиля: 1.3593201265659085e+17
Шаг [420/600], Ошибка для оригинала: 150.0763, Ошибка для стиля: 1.3209060249698304e+17
Шаг [430/600], Ошибка для оригинала: 150.1931, Ошибка для стиля: 1.284894270381097e+17
Шаг [440/600], Ошибка для оригинала: 150.3253, Ошибка для стиля: 1.2526184501451162e+17
Шаг [450/600], Ошибка для оригинала: 150.4017, Ошибка для стиля: 1.2203682279142195e+17
Шаг [460/600], Ошибка для оригинала: 150.5404, Ошибка для стиля: 1.1938272198105498e+17
Шаг [470/600], Ошибка для оригинала: 150.7326, Ошибка для стиля: 1.1732161012537754e+17
Шаг [480/600], Ошибка для оригинала: 151.0229, Ошибка для стиля: 1.181253445353472e+17
Шаг [490/600], Ошибка для оригинала: 151.4545, Ошибка для стиля: 1.2359501974654157e+17
Шаг [500/600], Ошибка для оригинала: 151.2945, Ошибка для стиля: 1.1938928469108326e+17
```

The first 600 eras

```
Шаг [510/600], Ошибка для оригинала: 151.3776, Ошибка для стиля: 1.1698025471462605e+17
Шаг [520/600], Ошибка для оригинала: 149.9437, Ошибка для стиля: 1.1054100781622886e+17
Шаг [530/600], Ошибка для оригинала: 150.4061, Ошибка для стиля: 1.1142254985373286e+17
Шаг [540/600], Ошибка для оригинала: 154.3599, Ошибка для стиля: 3.144260688847831e+17
Шаг [550/600], Ошибка для оригинала: 149.1942, Ошибка для стиля: 2.93860356722262e+17
Шаг [560/600], Ошибка для оригинала: 153.1192, Ошибка для стиля: 2.5488148410662912e+17
Шаг [570/600], Ошибка для оригинала: 151.1968, Ошибка для стиля: 1.6494201707010458e+17
Шаг [580/600], Ошибка для оригинала: 150.9484, Ошибка для стиля: 1.3711699413355725e+17
Шаг [590/600], Ошибка для оригинала: 151.3110, Ошибка для стиля: 1.222953282830336e+17
Шаг [600/600], Ошибка для оригинала: 151.0451, Ошибка для стиля: 1.0953642355562906e+17
```

Fig. 7. Results of the variation of the loss function as a function of different.

Image with transferred style

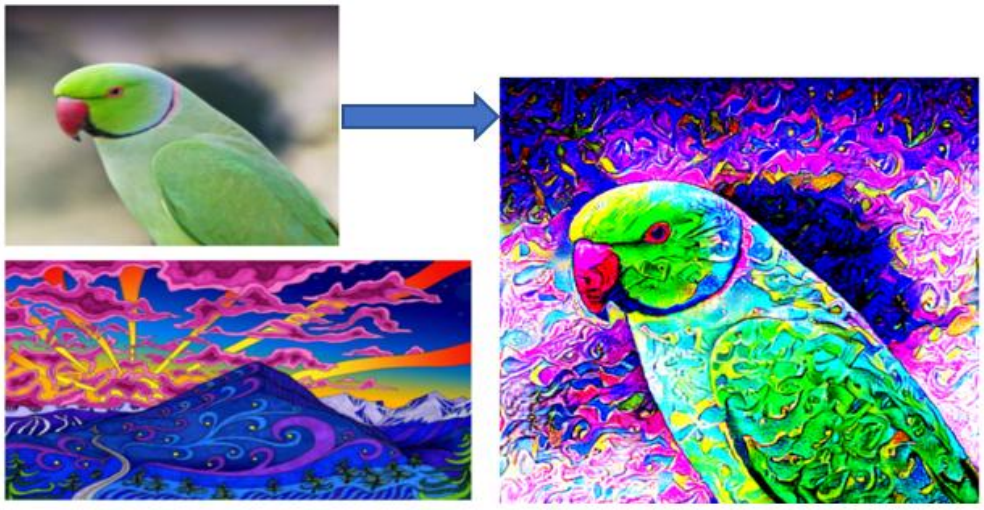


Fig. 8. The parrot is colored according to the original image.

CONCLUSION

In this article, we propose an image processing and style transfer method based on a pre-trained VGG16 model. This method extracts a list of features from each layer of an image. If an index matches a feature in the list, that feature is added to it. Each layer contains five elements. The method was applied to a parrot image for style transfer. After 600 training epochs, the following errors were obtained: error for the original image = 151.0451 and error for the transposed image = 1.0953. This satisfactory result demonstrates the effectiveness of the proposed method. The resulting parrot was then colored according to the original image.

BIBLIOGRAPHICAL REFERENCES

1. SPIRIDONOV I. N. Technologies biométriques d'identification de la personnalité et sécurité de l'État. – Infoforum. – 2011. – T. 57. – C. 25–30
2. R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In CVPR, 2014
3. H a n d b o o k of face recognition / Eds. S.Z. Li, A.K. Jain. – Springer Science+ Business Media, Inc., 2005. – 395 p.
4. D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov. Scalable object detection using deep neural networks. In Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on, pages 2155–2162. IEEE, 2014.
5. K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In ECCV, 2014.
6. J. Deng, O. Russakovsky, J. Krause, M. Bernstein, A. Berg, L. Fei-Fei. **Scalable multi-label annotation**. *ACM conference on human factors in computing (CHI)*, 2014. pdf | bibtex | slides
7. O. Russakovsky, J. Deng, Z. Huang, A. Berg and L. Fei-Fei, **Detecting avocados to zucchinis: what have we done, and where are we going?**, *Proceedings of the International Conference of Computer Vision (ICCV). 2013.* pdf | supplement | website | bibtex | slides | video
8. H. Su, J. Deng, L. Fei-Fei. **Crowdsourcing Annotations for Visual Object Detection.** *AAAI Human Computation Workshop*, 2012. pdf
9. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, **ImageNet: A Large-Scale Hierarchical Image Database.** *IEEE Computer Vision and Pattern Recognition (CVPR), 2009.* pdf | BibTex.
10. URL : https://neurohive.io/ru/vidy-nejrosetej/vgg16-model/ , Accessed on 5 June 2026

UNDER REVIEW IN IJAR