# Cut N' Paste
## A Semantic Segmented Style Transfer Tool
### 097200 - Deep Learning Course Final Project

Tzabar Dolev[1] and Yotam Amitai[2]

[1] Technion Israel Institute of Technology, Haifa , ISR
tzabar@campus.technion.ac.il, 301783056
[2] Technion Israel Institute of Technology, Haifa , ISR
yotama@campus.technion.ac.il, 200827384

**Abstract.** Style transfer between a scene image and and an artistic image is a common and well-studied sub-field of computer vision. The scene parsing sub-field is also well-studied and well-used in many applications. The combination between the two methods opens a new approach for transforming art into daily human operations by offering us to create a style-per-object program which enriches and bold different parts of a given scene. The proposed method is, as far as we know, the first automated end-to-end combination between the two computer vision fields.

## 1 Introduction

### 1.1 Convolutional Neural Networks

One of the leading Deep Neural Networks architectures used today is the Convolutional Neural Net(CNN). The CNN has gained significant popularity due to it's ability to process visual information in such a way that has provided great progress in the fields of computer vision such as image recognition, object classification and more. Today it is rare to see any visual computation done without such a network. The Convolutional Neural Networks consist of layers of small computational units that process visual information hierarchically in a feed-forward manner . Each layer of units can be interpreted as a collection of image filters, each of which puts emphasis on a certain feature of the original image. Thus, the output of a given layer is defined as a feature map, meaning differently filtered versions of the input image.

### 1.2 Semantic Segmentation

Image segmentation is one of the key problems in computer vision as for today, and it is the process of partitioning a digital image into multiple segments. The main goal of image segmentation is to simplify the representation of an image into something more meaningful and easier to analyze, and is typically used to locate objects and boundaries in images. Image partitioning means basically labeling every pixel in an image such that pixels which share the same label share certain characteristics. When discussing an image we can relate it to a grid of pixels, when each pixel is connected to it's corresponding pixels of the same segment by nodes. Each node is connected to its neighbors

by a weighted edge, when large weight is correlated with strong connection between the pixels. Grouping is a method for dividing the image to segments by grouping pixels with shared features.

Semantic Segmentation is a very difficult task and an important one, as the field of Autonomous Driving, Vision Aided Robotics and the continuous improvement of scene acquisition sensors. The main idea of semantic segmentation is to help us understand what's in the image in pixel level. It basically relies on the traditional image segmentation but divides the image to multiple segments with multiple meaningful objects, for example: an image with a clear sky, trees, two persons riding a bike and a dog playing catch with it's owner. One can not divide the image into the traditional foreground and background parsing due to the multiple types of objects to detect at the scene, and each object can have multiple parts to be segmented to. Semantic Segmentation, or Scene Parsing can assist a system like an autonomous car, which receives information from it's LiDAR sensor, receive a better understanding of it's surroundings in order to help it's decision-making protocol to successfully maintain a steady and safe driving while avoiding collisions of any kind.

### 1.3 Style Transfer

Style transfer is a popularly studied sub-field in computer vision that aims to transfer the style or texture of a given image to some other image of interest. For instance, using convolutional neural networks, an image can be recreated in the artistic style of a painting while maintaining the key contents of the reference photograph. Formally, from two given images $\overrightarrow{p} \& \overrightarrow{a}$, we wish to generate an new image $x$ using the content from $\overrightarrow{p}$ and the style of $\overrightarrow{a}$. To do so, we first need to extract content and style from the given images, and combine them in the output image. The first to introduce this method was Gatys et al.[2], whose work will be further discussed in the section 2. The applications of style transfer are mostly for recreational and artistic purposes.

## 2 Related Work

Our work seeks to build upon the papers [2] by Gatsy et al and [5] by B. Zhou et al. Throughout this paper we will give an in-depth explanation of neural style transfer and of semantic segmentation as described by these articles and their approach. The two articles were published at a leading conference for Computer Vision and had a major influence on the society in terms of image understanding and scene visualization.

### 2.1 Scene Parsing through ADE20K Dataset

The article, which was published at 2017, received a very high attention due to it's substantial improvements of scene parsing and understanding, and influenced hundreds of works so far (in the short time that passed). The article introduces the ADE20K dataset, spanning diverse annotations of scenes, objects, parts of objects and in some cases parts of parts. It also introduces a novel network design called "Cascade Segmentation Module" which parses a scene into it's basic ingredients.
Nowadays, a robot equipped with a convolutional neural network can understand a

simple scene, but to freely navigate inside a complex territory and manipulate the object inside, the robot needs to have far more complex abilities and to digest significantly greater amounts of data. As part of the article's contribution, a new dataset, called ADE20K, is presented. The images in the dataset are manually segmented in great detail, covering a diverse set of scenes, object and object part categories. The dataset was annotated by a single expert annotator which provided extremely detailed image annotations, with an average of 29 annotated segments per image (for a comparison, workers from Amazon Mechanical Turk label an average of 16 segments per image). The paper illustrates the ADE20K dataset, the collection process and statistics, followed by a generic network design called Cascade Segmentation Module, which enables convolutional neural networks to segment objects, object parts and elements (which are denoted "stuff" in the original article). After reviewing different types of existing datasets, such as object classification and detection, semantic segmentation, datasets with: objects, parts, attributes and semantic segmentation/parsing models, the article presents a new dataset and analyzes it through a variety of informative statistics.

## ADE20K: Fully Annotated Image Dataset

1. Dataset summary The ADE20K dataset is presented in few main steps:
   (a) 20,210 images in training set.
   (b) 2,000 images in validation set.
   (c) 3,000 images in testing set.
   All the images are exhaustively annotated with objects, many of them are annotated with their parts and there is additional information about whether the object is occluded, cropped and other attributes.
2. Annotation consistency To create a benchmark which could become a useful dataset for future works, it is essential to maintain few data annotation consistencies:
   (a) Segmentation quality: one typical source of error arises when segmenting complex objects such as buildings and trees which can be segmented with different degrees of precision.
   (b) Object naming: due to ambiguity or similarity between concepts, a big car could be a 'car' or a 'truck', and a 'palm tree' could be a 'tree'. these ambiguities could be reduced by creating a precise terminology, but becomes hard with a large growing vocabulary.
   (c) Segmentation quantity: there is a large number of objects in each image, some images might be annotated more thoroughly than others, usually with small objects.
3. Comparison with other datasets Table. 1 compares ADE20K with other existing datasets such as COCO, ImageNet, NYU Devpth V2, Cityscapes, SUN etc. Compared to the largest annotated datasets, ADE20K comprises of much more diverse scenes, and is also larger in terms of object instances.

| | Images | Obj. Inst. | Obj. Cls. | Part Inst. | Part Cls. | Obj. Cls. per Img. |
|---|---|---|---|---|---|---|
| COCO | 123,287 | 886,284 | 91 | 0 | 0 | 3.5 |
| ImageNet* | 476,688 | 534,309 | 200 | 0 | 0 | 1.7 |
| NYU Depth V2 | 1,449 | 34,064 | 894 | 0 | 0 | 14.1 |
| Cityscapes | 25,000 | 65,385 | 30 | 0 | 0 | 12.2 |
| SUN | 16,873 | 313,884 | 4,479 | 0 | 0 | 9.8 |
| OpenSurfaces | 22,214 | 71,460 | 160 | 0 | 0 | N/A |
| PascalContext | 10,103 | ∼104,398** | 540 | 181,770 | 40 | 5.1 |
| ADE20K | 22,210 | 434,826 | 2,693 | 175,961 | 476 | 9.9 |

**Fig. 1.**
Comparison of semantic segmentation datasets

**Cascade Segmentation Module** There is a large difference in terms of class frequency between large elements (stuff) and discrete objects. For example: classes like 'wall', 'floor', and sky occupy 40% of all annotated pixels, while small objects like 'cat', 'vase' or a 'microwave' occupy only 0.03%. Due to that, a semantic segmentation network can be easily dominated by the most frequent element classes. On the other hand, there are relations between large elements and specific objects - for example, a drawing on a wall is a part of the wall, and the wheels of a car are a part of the car. The module is a generic network design that can potentially be integrated with and previous semantic segmentation networks. We catagorize semantic classes of the scenes into three macro classes: elements (stuff), foreground objects and object parts. In some scenerios there are object classes that could belong tpo either of few macro classes, like 'building' or 'door'. In this case, the assignment is directed at the most likely macro class. In the scene parsing network, different streams of high-level layers represent different macro classes and the results from each stream are then fuzed to generate the segmentation.

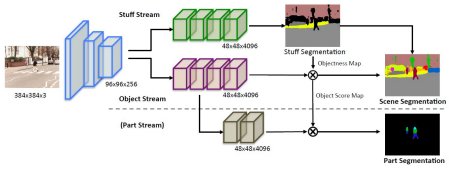


**Fig. 2.**
Cascade Segmentation Module Architecture Visualization

After training, the elements stream generates elements segmentation and an 'Objectness Map' indicating the probability that a pixel belongs to the foreground class. Afer training, the object streamfurther segments each discrete object on the objectness map, and the result is merged with the elements segmentation to generate the scene segmentation. The network with two streams (elements and objects) or three streams (elements, objects and parts) could be trained end-to-end. The loss is calculated by per-pixel cross entropy loss. The objectness map is given as a binary mask, indicates whether a pixel belongs to a class (elements or object parts), used to exclude the penalty for pixels who belong to the elements classes in the training loss for the object stream. The training losses for the two streams and for the three streams are: $\mathcal{L} = \mathcal{L}_{elements} + \mathcal{L}_{object}$ and $\mathcal{L} = \mathcal{L}_{elements} + \mathcal{L}_{object} + \mathcal{L}_{part}$.
The configurations of each layer are based on the base-networks being used. The proposed segmentation module integrates with Segnet and DilatedNet.

## 2.2 A Neural Algorithm of Artistic Style

The key finding of this paper is that the representations of content and style in the Convolutional Neural Network are separable. This significant understanding paves the way to creating manipulations on both representations independently in order to produce new, perceptually meaningful images.
The article shows that by reconstructing the processed image after each convolutional

layer using the layer's feature map, it is possible to visualize the information contained in that layer. In their article Gatys et al. writes *"When Convolutional Neural Networks are trained on object recognition, they develop a representation of the image that makes object information increasingly explicit along the processing hierarchy. 8 Therefore, along the processing hierarchy of the network, the input image is transformed into representations that increasingly care about the actual content of the image compared to its detailed pixel values."* meaning that the further along the way in the CNN processing layers the image is, the more the filters created by the layer pay attention to high-level content while early layers simply reproduce the exact pixel values of the original image. The article proposes to model the *content* of photo as the feature responses from a pre-trained CNN, and further model the *style* of an artwork as the summary feature statistics. In the article a pre-trained VGG 19 model was used. VGG [4] is a Convolutional Neural Network that rivals human performance on a common visual object recognition benchmark tasks. Only it's convolutional layers were used and the classification section section was discarded. The networks architecture is as described in [3]

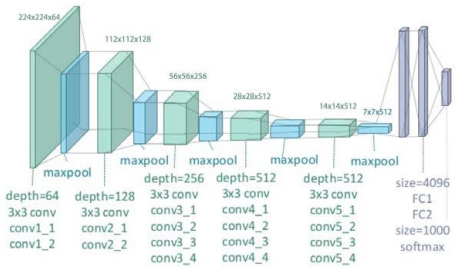


**Fig. 3.**
VGG model illustration

*Note:* After every convolution listed, the *ReLU* activation function is applied. All filters in the network are of size 3x3. Pooling in the network is of size 2x2.
(A description of the layers can be found in Appendix B.)

Now we shall describe the content representation and loss in the content extraction method by utilizing the feature map obtained after each convolutional layer.

**Content extraction** Each convolutional layer defines a group of non-linear filters with increasing complexity as the layers progress through model. Hence a given input image $\vec{x}$ is encoded in each layer of the CNN by the filter responses to that image. We shall define $N_\ell, M_\ell$ to be the number of filters and the size of each filter $(H \times W)$ in the $\ell$-th layer respectively. We can represent this three dimensional layer output as a two dimensional matrix $F^\ell \in \mathbb{R}^{N_\ell \times M_\ell}$ where $F^\ell_{i,j}$ is the activation of the $i$-th filter at position $j$ in layer $\ell$. This matrix will be denoted as the feature representation matrix in layer $l$. To visualize the image information that is encoded at different layers of the hierarchy the article proposes performing gradient descent on an image of white noise in order to find another image that matches the feature responses of the original image.

We will denote $(\overrightarrow{x}, F^\ell)\&(\overrightarrow{p}, P^\ell)$ as the generated image and the original image along with their corresponding feature representation at the $\ell$-th layer respectively. We can now define the Content Loss function as the Mean Squared Error between the two image representations:

$$\mathcal{L}_{content} = \frac{1}{2}\sum_{ij}(F_{ij}^\ell - P_{ij}^\ell)^2 \tag{1}$$

The derivative of this loss with respect to the activations in layer l equals:

$$\frac{\mathcal{L}_{content}}{\delta F_{ij}^\ell} = \begin{cases} (F^\ell - P^\ell)_{ij} & \text{if } F_{ij}^\ell > 0 \\ 0 & \text{if } F_{ij}^\ell < 0 \end{cases} \tag{2}$$

Using this formula, the gradient with respect to the input image $\overrightarrow{x}$ can be computed using standard error back-propagation. Like so we change the initially random input until it it generates the same response as a certain layer does to the original image $\overrightarrow{p}$

**Style extraction** In order to extract style from an image, on top of each layer of the CNN, the correlations between each activation map is computed. These feature correlations are given by the Gram matrix $G^\ell \in \mathbb{R}^{N_\ell \times N_\ell}$, where $G_{ij}^\ell$ is defined as the inner product between the vectorised feature map $i$ and $j$ in layer $\ell$:

$$G_{ij}^\ell = \sum_k F_{ik}^\ell \cdot F_{jk}^\ell \tag{3}$$

Similarly as with content extraction, lower level layers reveal local structures of style whereas higher level layers reveal a more general sense of styling.
As with the content extraction, the article uses gradient descent from a white noise image to find another image that matches the style representation of the original image. This time the Loss is defined as the mean-squared error between the Gram matrices of the input image and the style image $\overrightarrow{x}\&\overrightarrow{a}$ respectively. We can thus compute the contribution of layer $\ell$ to the total loss as:

$$E^\ell = \frac{1}{4N_\ell^2 M_\ell^2}\sum_{ij}(G_{ij}^\ell - A_{ij}^\ell)^2 \tag{4}$$

where $A^\ell \& G^\ell$ are the style representations in layer $\ell$. Thus the total loss is:

$$\mathcal{L}_{style} = \sum_{\ell=0}^{L} w^\ell E^\ell \tag{5}$$

where $w^\ell$ serve as weights representing the contribution of each layer to the total loss. The derivative of $E^\ell$ with respect to the activations in layer $\ell$ can now be computed analytically:

$$\frac{\delta E^\ell}{\delta F_{ij}^\ell} = \begin{cases} \frac{1}{N_\ell^2 M_\ell^2}((F^\ell)^T(G^\ell - A^\ell))_{ij} & \text{if } F_{ij}^\ell > 0 \\ 0 & \text{if } F_{ij}^\ell < 0 \end{cases} \tag{6}$$

Using this formula, the gradients of $E^\ell$ with respect to the activations in lower layers of the network can be computed using standard error back-propagation.

**Putting it all together** To bring these two components together to create an output image that combines both content and style, the article jointly minimizes the distance of the random image $\overrightarrow{x}$ to the styling image $\overrightarrow{a}$ and the content image $\overrightarrow{p}$. Thus, the overall loss to minimize is:

$$\mathcal{L}_{style} = \alpha\mathcal{L}_{content} + \beta\mathcal{L}_{style} \tag{7}$$

where $\alpha\&\beta$ are the weighting factors for content and style reconstruction respectively.

To conclude this section we'll end with with an inspiring quote from the original article *"All in all it is truly fascinating that a neural system, which is trained to perform one of the core computational tasks of biological vision, automatically learns image representations that allow the separation of image content from style."*

## 3 Approach

The goal of this work is to develop an end-to-end tool which is capable to create a scene parsing and style transfer image processing method. This method should receive an image which the user wants to create an artistic style of, parse the scene to different objects and segments, and fit a style for each segment the user wishes to transform. The approach is composed of four main stages described below:

- User interface construction
- Scene parsing module
- Artistic style transfer module
- Final image construction

Combining these stages should enable us to create a user-friendly system, capable of performing multiple tasks at few short steps and produce a multi-layer styled image to the users needs.

These stages are detailed below:

1. **User Interface Construction:** First of all, when we wish to combine the two computer vision modules together (scene parsing and the style transfer), an easy-to-use user interface should be considered, to offer the opportunity to use the entire program as a whole and should not require any prior knowledge about any of these tasks. In order for us to offer this option, the user should be able to perform each step while instructed to, and the required styles and modules should be readily available without the need to create anything by it's own. The user interface should offer the user the options for choosing which segments he wishes to style, and which style he wants to use for each segment which was chosen. Moreover, it should offer the user the opportunity to leave some of the segments unstyled for a better contrast, or to bold the foreground out of the background.

2. **Scene Parsing Module:** The scene parsing module should receive the image which the user wishes to style and to parse the image to different segments. Segmenting each object separately is crucial for a better visualization of the image, and therefore should consist of a fully functioning standalone neural network to parse the scene for different objects and isolate each segment from the rest. The output of this module should consist of two level:

(a) constructing a scene image which is an integration of all of the objects which appear at the scene. Each object should be segmented and painted in different color to offer the user a basic idea about how the image will look after it will be styled: which objects where isolated, where are the objects boundaries and how many segments are there available for style.
(b) isolating each object at a different mask. This type of information gives the user a clue about which specific objects the user wish to style and which should be left untouched.

3. **Artistic Style Transfer Module:** After the user chooses which objects and segments he wish to style, he should be offered with a list of styles available for transfer. The user interface would ask the user to state which style he wishes to transfer for each segment he chose, and the styles would would be automatically downloaded and sorted at the user's computer. The user could also have the option to add style of he's own and use them instead or additionally to those which are already given with the downloaded module.

4. **Final Image Construction:** Each segment which was parsed out of the scene and styled under the user's choices is eventually combined together to re-assemble the image for it's original structure. The styled segments should differ from the ones which weren't styled or styled using a different style image and create a contrasted image with multiple flavours and colors.

## 4 Implementation

The foundation of this work was based on combining and integrating two ready-available tools, one for Semantic Segmentation [1] and the other for Neural Style transfer [3]. The process of this integration was as follows:

### 4.1 First Stage: Input Image, Resize, Semantic Segmentation

We've allowed the system to receive an input image of any kind and dimension. The first step of the process is to resize this image to be $512 \times 512$ and pass it on to the semantic segmentation as a *.jpg* file. The semantic segmentation module will then operate on the image and output a new image that is a concatenation of the input and the semantic segmentation of it, as can be seen in [4].

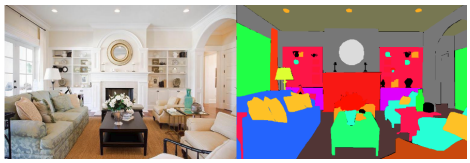

**Fig. 4.**
Semantic Segmentation Output

We have altered the output of the module in order to only receive the semantic segmentation representation of the image. This representation is then passed to the the next stage.

### 4.2 Second Stage: Creating & Choosing Masks & Styles

From the semantic representation created in the previous stage we create multiple black and white submasks, each representing a single color in the semantic representation, which in aspiration represents a different element in the original image.
As can be seen in 4 sometimes there are many small submasks that might not contribute to the overall image if they were to be treated individually. For that reason we have implemented a method where the system will supply the user with the overall number of submasks and request that the user input the number of submasks he wishes there to be, meaning that the user can decide that he'd like there to only be two or three masks (instead of 17 for example). Our method will then unite the smallest submasks to their neighbouring ones until the number of submasks requested is achieved. We will use these submasks to "color" certain areas of the image in a chosen style. The process will now request that the user specify which submask should correlate to which style [9].

### 4.3 Third Stage: Style transfer & Putting It All Together

After the choices have been made, the style transfer module begins. For each chosen style an image is generated using the style transfer module with an input of the original image and the style image. Once these images are generated we cut out of each of them their corresponding submask and stitch all submasks together to create the final output image.

## 5 Experiments and Results

### 5.1 Evaluation

There is currently no metric known to us that is widely used in order to evaluate style transfer, thus we will subjectively evaluate results by their visual appearance.

### 5.2 Hyperparameter Tuning

Several parameters had to be tuned in order to make sure we obtained the optimal results. While this model has numerous hyperparameters, we will mainly focus on the following:

- Set of layers to be used for content
- Set of layers to be used for style
- Various pre-trained networks
- Content and style weights in style transfer module
- Number of epochs for style transfer

### 5.3 Testing Different Content & Loss Layers

While observing the style transfer module we noticed that out of the five convolutional layers in the VGG pre-trained model, only the forth layer was used for style loss while all layers played a role in content loss. When testing the different combinations of the content layers, we saw that adding them resulted in a lower content loss (as to be expected), thus explaining why all layers are used. We experimented with style layers by first observing the change in the image style when only changing the single style loss layer as can be seen in [5]


**Fig. 5.**
A comparison between different Content layers

What we observed is that indeed the forth layer most represented the style of the original style image compared to the other layers. The style varying between the layers, although hard to notice at times, could be seen by observing the brush strokes, color smears and continuity of the style.
The next step was to compare these results when combining multiple layers for style loss. For this stage we observed the best resulting layers from the previous part, layers 3-5. The results can be seen in [6]


**Fig. 6.**
A comparison between different combinations of Content layers

As we were able to see, all combinations resulted in a less detailed style compared to the single layer tests.
We have concluded that layer 4 as a single loss layer returns the best visible result.

### 5.4 Testing Different Models

The style transfer as shown in [3] uses a VGG19 pre-trained network with about 20 Mil parameters. We substituted this network with other pre-trained ones in order to observe

a difference in computation speed and visual output. The models we experimented with are: *VGG13,VGG16, VGG19, VGG19 with batch normalization* The results can be seen in [7]

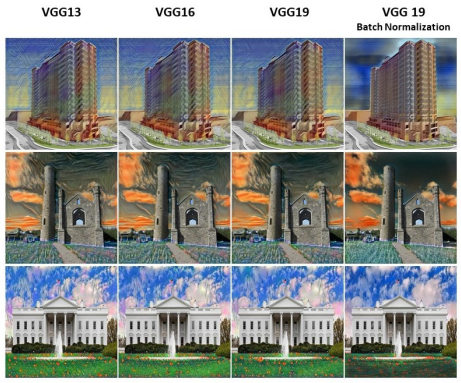


**Fig. 7.**
A comparison between different VGG models

From our experiments we observed that although VGG19 has around 11 Mil more parameters than the VGG13, the output image was similar from both. Another interesting observation was that although this difference in parameters, no big difference was noted in computation time as well. On the other hand it can clearly be seen that using a model with batch normalization raises the style loss.

### 5.5 Adjusting weights

While we modified the different weights given for each of the loss functions (style loss and content loss), we tried to get a better understanding of how much each loss contributes to the given image. It can be seen in Figure [8] that giving the content an intensified value over the style creates a very loyal-to the-original style transfer with no brush strokes and color continuities. On the other hand, an intensified value for the style over the content broke the signals and probably exploded the gradients due and wasn't able to transfer the given style properly.


**Fig. 8.**
Testing different scales of weights

## 6 Additional Contribution

Apart from integrating two highly useful tools into one complete end-to-end process, our work has added further contribution to these processes individually.

**Adding Early Stopping** When running and experimenting with the style transfer module we noticed that occasionally, after gradually decreasing with each epoch, both the Content Loss and the Style Loss would "crash" and jump to extreme numbers and the output we would receive was a mess. in order to deal with this issue we added a method that would keep track of the best Loss achieving image representation throughout the training and would always return that one in the end, meaning we can now run the style transfer module with as many epochs as we wish without fear that any value will be lost. This mechanism prevents the model from being affected from singularities and over-fitting.

**Reducing Style Transfer Parameters** changing architecture from VGG19 to VGG13 outputed similar results while using less parameters. (reduction of about 11 million parameters)

**Adding User Choice** Our work allows users to not only choose which styles they want to use for their image but also choose which part of the image to associate with which style.

## 7 Conclusions & Future Work

In this work, we have implemented both style transfer and semantic segmentation and combined the two to create a unique end-to-end tool for user choice based image augmentations. By segmenting the image we allow it to be sectioned and split as desired by the user and then "colored" as he wishes. We have made the style transfer process more robust to singularity spikes and over-fitting by adding a mechanism for saving the highest value image observed in the process.
Future work on this tool could introduce a method for allowing the user to create his own sub-masks through an interface like Graph-Cut, additionally to an improvement of creating a GUI for the program. Any improvement to the segmentation module would highly benefit this project as well.

## References

1. CSAILVision. Semantic segmentation on mit ade20k dataset in pytorch- tutorial.
2. Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.
3. Alexis Jacq. Neural transfer using pytorch- tutorial.
4. Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
5. Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, page 4. IEEE, 2017.

# APPENDIX

## A Styles

The styles used in the project are presented at the following figure:


**Fig. 9.**
Styles numbering

Using the styles presented above, these are the styles which created each of the styled images presented in this work:

- White house:
  1. House: Original.
  2. Grass: style number 17.
  3. Sky: style number 13.
- Church:
  1. Church: Original
  2. Sky: style number 12.
  3. Grass: style number 13.
- Building:
  1. Building: 5.
  2. Road + garden: Original.
  3. Sky: style number 0.

## B VGG layers

**Convolutional block 1:**
$Conv1_1$: 64 filters, each with depth 3.
$Conv1_2$: 64 filters, each with depth 64.
Max Pooling 1
**Convolutional block 2:**
$Conv2_1$: 128 filters, each with depth 64.
$Conv2_2$: 128 filters, each with depth 128.
Max Pooling 2
**Convolutional block 3:**
$Conv3_1$: 256 filters, each with depth 128.
$Conv3_2$: 256 filters, each with depth 256.
$Conv3_3$: 256 filters, each with depth 256.
$Conv3_4$: 256 filters, each with depth 256.
Max Pooling 3
**Convolutional block 4:**
$Conv4_1$: 512 filters, each with depth 256.
$Conv4_2$: 512 filters, each with depth 512.
$Conv4_3$: 512 filters, each with depth 512.
$Conv4_4$: 512 filters, each with depth 512.
Max Pooling 4
**Convolutional block 5:**
$Conv5_1$: 512 filters, each with depth 512.
$Conv5_2$: 512 filters, each with depth 512.
$Conv5_3$: 512 filters, each with depth 512.
$Conv5_4$: 512 filters, each with depth 512.
Max Pooling 5