Introduction to Artificial Neural Networks with Tensorflow – WS 2021/2022

Artistic Style Exploration with Artificial Neural Networks

Janina Klarmann, Laura Kühl

***Abstract*—In this project the artistic style transfer, performed by CNNs like the VGG19 and the ResNet50 is explored and different angles regarding for example performance and aesthetics are compared. We look at the theoretical basics of a style transfer and how its feature extraction works. Later we try to compare the networks' results by using an art-style-classifying ResNet and our subjective opinion. Additionally, we will have a look at how the ratio of the weights, different learning rates layer selection and the input image influences the style transfer.**

***Index Terms*—Artistic Style Exploration, Art, Convolutional Neural Networks**

I. Introduction

Mastery over the fine arts as well as a sense of aesthetics are sometimes argued to be uniquely human traits that serves to distinguish our species among the animal kingdom. We can create and appreciate art. We can stand in complete awe in front of a painting in which the artist has managed to portrait an object from our shared reality in a way that evokes emotions unlike an encounter with the object itself would do. If we consider that art and aesthetics tend to be perceived as intrinsically human traits that serve to support a claim to mankind's cognitive domain being qualitatively distinct and elevated -even more so when compared to kinds of "artificial cognition" such as algorithms built for analysis of data- ,the concept of utilizing algorithms to instead create art and thus emulate parts of the behaviour our species defines it's identity by, is naturally a topic worth exploring.

Conversely, understanding how not only content but style and texture are captured in neural networks increases our understanding of the mechanisms and dynamics of the processing performed on images. Some existing work relating to this approach has explored the topic of the topic of synthesizing texture from images [4]. Expanding our fundamental understanding of the mechanisms within the "black box" models that ANNs are, is also of practical interest for the industries that aim to broadly employ models for applications like autonomous driving, cashier-less grocery shopping and other technology building on reliable and predictable computer vision. Besides the economical interest, cultural values are considerably as important. There is also some scientific work out there about fusing art and neural networks. Like the paper by Gatys et. al (2015) [5] in which they use a Convolutional Neural Network (CNN), precisely a VGG19 to transfer an art style onto a photo. The VGG19 is a deep Convolutional Neural Network that is specialised in image recognition [11]. Inspired by their work the question arises, how other deep CNN architectures would perform on the Art-Style-Transfer task and if the concept can be transferred. A ResNet was chosen to be compared to the VGG19.

As CNNs are the best performing networks for image feature extraction, sticking to that architecture appears reasonable for further explorations. Earlier layers in CNNs represent low level features like specific textures or horizontal and vertical edges. Therefore, the feature maps of these layers relate more closely to the style than the content of an image and are used for the art-style extraction of an image. Later layers represent higher level and more complex features, the actual content of an image. Therefore, the last convolutional layer is used as our content layer, which extracts the content of an image [5]. To perform the style transfer the different layers responses to a white noise image and the original style or content image are used respectively. A more detailed explanation is given in the Methods.

There were several possible deep CNNs to chose, like a DenseNet or a MobileNet, but the ResNet with its shortcut connections can perform well and counteract vanishing gradients despite of being very deep and having significantly more layers than the VGG19. The ResNet50 has with 50 layers a significantly higher number of layers compared to the VGG with 19 layers. Additionally, the ResNet also outperforms the VGG in terms of processing power and speed, and it appears reasonable to take a closer look at this fact to see if it was reflected in our work [8].

There are also other approaches to perform a style transfer, which perform well, like by Chen et. al. (2016) [3], where they trained an inverse network architecture and a patch-based transfer. Even though this is an interesting topic for artistic exploration, it would go beyond the scope of this lab report.

II. Methods

To perform the style transfer the representations of an image in a certain layer are used and the loss between a white noise image and a content and style image are minimised. Each convolutional layer in the network has a number of filters that are equal to the number of feature maps created. An input image evokes a specific respond in that layer's

feature maps. This response encodes the representation of the image in that layer. Usually with increasing depth of the network, the complexity of these feature maps increases as well. These responses are used for our style transfer. The Content visualization is straight forward: To visualize the networks response to an image on a layer, a white noise image is fed into the network and try to evoke the same response, as our original image does. Therefore the squared-error loss between the image-representations is calculated and the error signal is propagate back to change the white-noise-input image respectively. The Squared-Error loss would then be:

$$Loss_{content}(\vec{o}, \vec{w}, l) = \frac{1}{2} \sum_{i,j} \left(W^l_{i,j} - O^l_{i,j}\right)$$

Where $\vec{o}$ stands for the vector of the original image and $\vec{w}$ for the one of the white noise image. l denotes the layer. i, j are the layers i-th filter activation on position j. So $W^l_{i,j}$ is the white noise images activation in layer l for the i-th filter on a position j. This indexing convention equivalently applies to $O^l_{i,j}$ on the original image.
The results of the back propagated error signal are visualised in Figure 2, where the reconstructed response in the first convolutional layers of each block (a-e) of the VGG19 is shown. It can be observed that when recreating the content response, the texture of the image becomes decreasingly important and that the images texture becomes noisier (Figure 2, Content Image, a-f). The increase of noise is most clearly visible in the last convolutional layers. Consequentially, 'conv4_1' was later chosen for the content representation layer and subjectively, the results appeared less noisy than with later layers.
As the style is more than the mere representation of features a gram matrix is used to make it possible to capture the image-style, speaking the texture-relations. The feature maps of a layer represent certain low level features, like lines or colors. A style is a combination of several low level features, like which colors appear with which shape. In our example (Figure 2, Style Image) yellow appears with circles and black with straight lines. Hence to capture it the correlation between the low-level features must be computed. This is done by calculating the correlations between the feature matrices. This is done by calculating the dot product of the matrices respectively to each other. For this the feature matrices are transformed into vectors which are brought into a matrix shape. This is then multiplied with its transposed and outputs the correlation of the features and hence the style. The visualisations of this are shown in Figure 1. (Figure 2, Style Image, a-e). It is visible that with increasing layer-depth the correlations become more prominent. As in the first convolutional layer of the first block (Figure 2, Style Image, a) the style representation is rather colorful patches. Whereas in e. bigger style concepts, like whole circles with yellow color, are represented. To calculate the overall loss the squared error loss between the gram matrices of the style and the input image of a layer is calculated first. Let E_l be that calculated loss for the layer l and w_l its respective weight. The total Error loss is then:

$$Loss_{style}(\vec{s}, \vec{w}, l) = \sum_{l=0}^{L} (w_l E_l)$$

The details of deriving this formula for $Loss_{style}$ and it's mathematical background are beautifully explained in Yanghao Li et. al (2017) [10] and the initial work by Gatys et. al. (2015) [5]. In the latter paper an alternative approach to artistic style transfer using Maximum Mean Discrepancy was explored as well.

After calculating the loss of the content and the style, it can be put together for the total loss:

$$\alpha Loss_{content}(\vec{o}, \vec{w}) + \beta Loss_{style}(\vec{s}, \vec{w})$$

α and β are the different weightings of the content or style representation. The ratio between these weights is represented by α/β as λ. Some ratio-explorations are captured in Figure 1. and further discussed later.

III. Experiments

A. Style-transfer

1) Using a VGG19 oriented model: Since the main paper [5] and code [13] rely on VGG models to transfer the art style of one image to another, we first looked more deeply into the topic of the VGG and later the VGG19 specifically. A VGG model is a deep Convolutional Neural Network with 19 convolutional layers in case of the VGG19 and 16 in case of the VGG16. One important feature of VGGs are the multiple 3x3 kernel-sized filters that succeed each other instead of larger kernel-sized filters. A second characteristic of a VGG is the "pyramid" shape. The convolutional layers stacked on top of each other increase in depth and the max pooling reduces the volume size of the layers. At the end, two fully linked layers with 4096 nodes each follow and the last layer contains a softmax classifier. [9]. The default input shape of the model is 224x224x3 and it was trained on the ImageNet [12] dataset which containes roughly 14 million images, 1000 categories and is organized according to the WordNet hierarchy.

In the tensorflow tutorial we followed to lay our groundwork for this project, the pretrained keras VGG19 model was used [13]. We recreated some of their code, adjusted it to our needs and then tried to implement and train our own VGG19 model.
In this process, we faced multiple difficulties. The first conspicuous feature was the input size of 224x224. The images that were used to train the VGG model were noticeably larger than the images we handled during the course "Implementing ANNs with TensorFlow". This raised several problems. We knew that we would need the larger image size for adequate artistic expression and feature

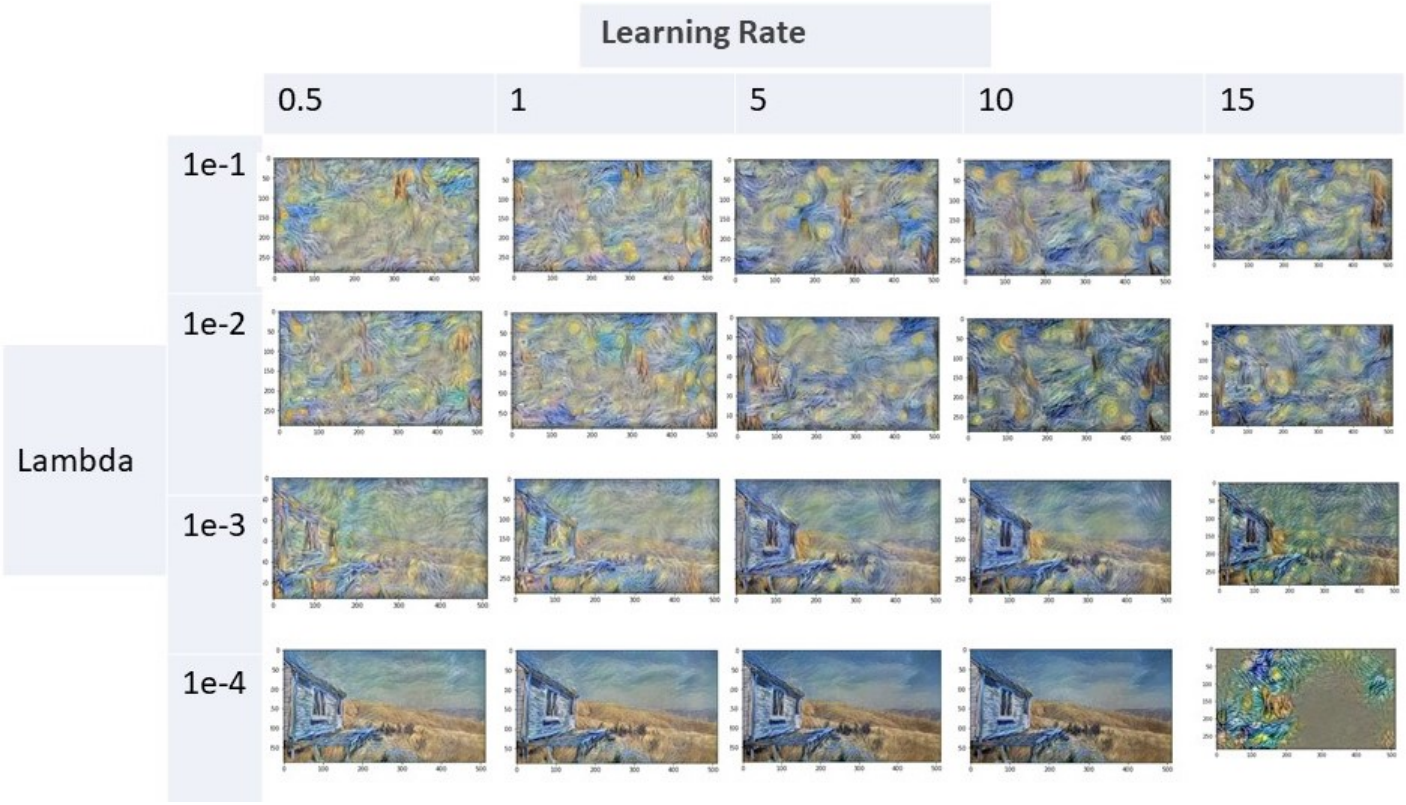

Fig. 1. Different ratios for the weighting (λ) with different learning rates. 1000 training steps. The noise image has been used to perform gradient descent on.

extraction, but also that higher computational power was needed to load the datasets and train such a network on our own. Additionally, it was not that easy to search for and find a suitable dataset that we were allowed to download and that met our and the models requirements. Because we knew that we could not use the original ImageNet dataset, we first tried datasets that we already used in our course like the Cifar10 and Cifar100. The goal was to extract features from an image and assign them to another one. But in order to do that, we needed a model that was trained on a wide range of images belonging to different classes, to get a sufficient feature extraction. We chose to create a Tensorflow Keras sequential model and add the different layers that belong to a VGG19 model. We did not use the usually used class, as we got repeatedly errors when trying to access and pull the layers for the content and style representation.

```
# Creating a VGG19 Model
class Our_VGG19_Model(tf.keras.Sequential):
  def __init__(self):
    super(Our_Own_VGG19_Model, self).__init__()
    self.add(tf.keras.layers.Conv2D(32, (3, 3), padding="same", activation='relu',
          name='convlayer11', input_shape=(224,224,3)))
    self.add(tf.keras.layers.Conv2D(32, (3, 3), padding="same", activation='relu',
          name='convlayer12'))
    self.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2))
    self.add(tf.keras.layers.Conv2D(64, (3, 3), padding="same", activation='relu',
          name='convlayer21'))
    self.add(tf.keras.layers.Conv2D(64, (3, 3), padding="same", activation='relu',
          name='convlayer22'))
    self.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
    self.add(tf.keras.layers.Conv2D(128, (3, 3), padding="same", activation='relu',
          name='convlayer31'))
    self.add(tf.keras.layers.Conv2D(128, (3, 3), padding="same", activation='relu',
          name='convlayer32'))
    self.add(tf.keras.layers.Conv2D(128, (3, 3), padding="same", activation='relu',
          name='convlayer33'))
    self.add(tf.keras.layers.Conv2D(128, (3, 3), padding="same", activation='relu',
          name='convlayer34'))
    self.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
    self.add(tf.keras.layers.Conv2D(265, (3, 3), padding="same", activation='relu',
          name='convlayer41'))
    self.add(tf.keras.layers.Conv2D(265, (3, 3), padding="same", activation='relu',
          name='convlayer42'))
    self.add(tf.keras.layers.Conv2D(265, (3, 3), padding="same", activation='relu',
          name='convlayer43'))
    self.add(tf.keras.layers.Conv2D(265, (3, 3), padding="same", activation='relu',
          name='convlayer44'))
    self.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
    self.add(tf.keras.layers.Conv2D(265, (3, 3), padding="same", activation='relu',
          name='convlayer51'))
    self.add(tf.keras.layers.Conv2D(265, (3, 3), padding="same", activation='relu',
          name='convlayer52'))
    self.add(tf.keras.layers.Conv2D(265, (3, 3), padding="same", activation='relu',
          name='convlayer53'))
    self.add(tf.keras.layers.Conv2D(265, (3, 3), padding="same", activation='relu',
          name='convlayer54'))
    self.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
    self.add(tf.keras.layers.Flatten())
    self.add(tf.keras.layers.Dense(4096, activation=tf.nn.relu))
    self.add(tf.keras.layers.Dense(4096, activation=tf.nn.relu))
    self.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))
    self.compile(optimizer= tf.keras.optimizers.Adam(0.00001),
    loss='categorical_crossentropy', metrics=['accuracy'])
```

Based on Gatys et. al. (2015) [5] describing a perceived improvement of results when substituting the MaxPooling layer with AveragePooling, we eventually made the same adjustment, although the observed effect on our model appeared to be minor in comparison. Because the Cifar10 and later the Imagenette/Full-Size-V2 datasets both contained ten classes, the last dense layer of our model had an output space of ten dimensions instead of 1000 like the VGG19 that was pre-trained on the Imagenet dataset. After training our model for 30 epochs on the cifar10 dataset with a learning rate of 0.00001, an Adam optimizer and the categorical crossentropy loss, an accuracy of only 60 percent was reached. Although

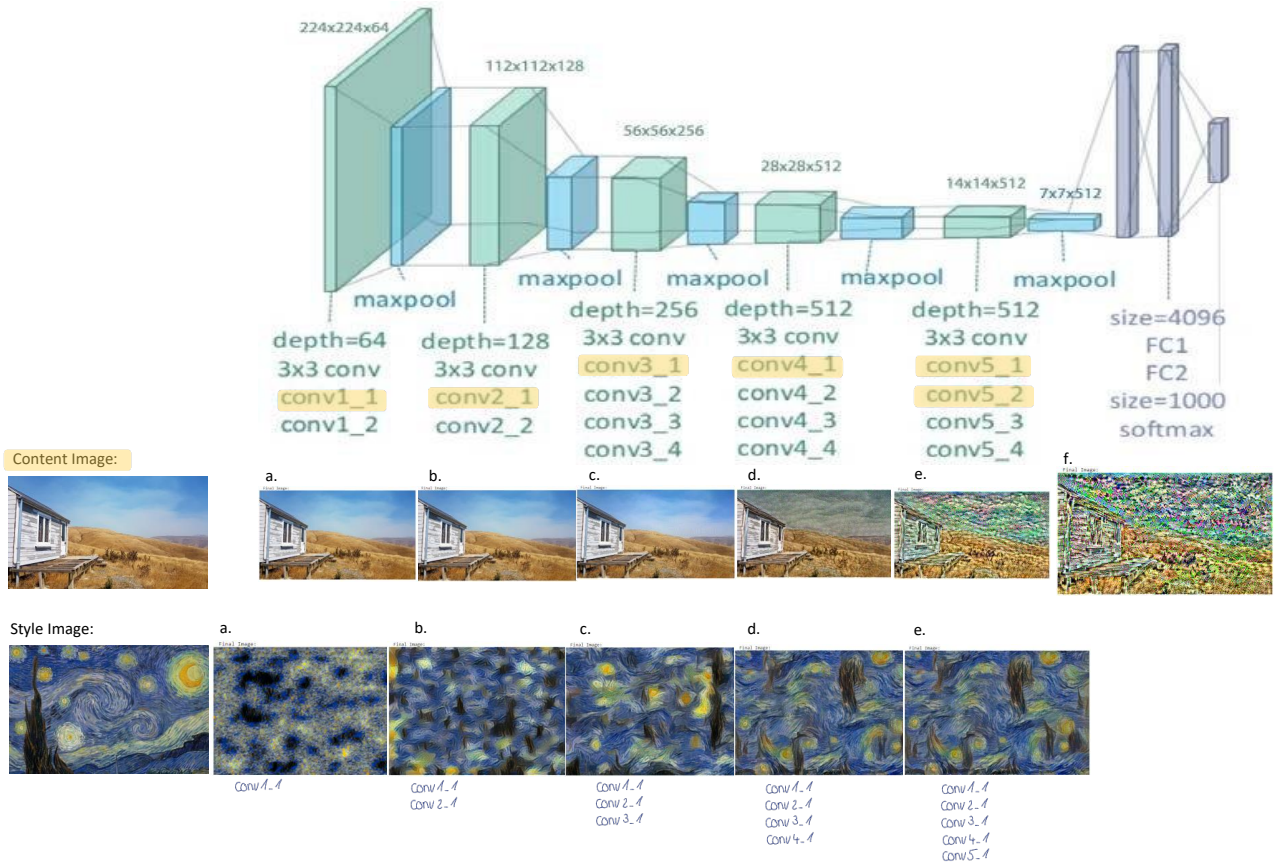

Fig. 2. VGG19 architecture and the representations of the images in the different layers. The content image representation is displayed from the first convolutional layer in the first block, 'conv1_1' (a), the first convolutional layer in the second block, 'conv2_1' (b), 'conv3_1' (c), 'conv4_1' (d), 'conv5_1' (e), 'conv5_2' (f). The representation of the style image, as calculated with the gram-matrix for layer 'conv1_1' (a), 'conv1_1' and 'conv2_1' (b),'conv1_1', 'conv2_1' and 'conv3_1' (c), 'conv1_1', 'conv2_1', 'conv3_1' and 'conv4_1' (d), 'conv1_1', 'conv2_1', 'conv3_1', 'conv4_1' and 'conv5_1' (e)

this number is low for dataset with as little complexity and dimensionality as cifar10, it was decided to move forward using this model performance, as optimizing this preliminary step was not the focus of this exploration and was not deemed likely to be a source of failure going forward. Thus, capacity was concentrated on the focus of the style transfer topic instead.

We proceeded to insert our VGG19 model into the adapted code used to transfer the style of images. The first Step was to make intermediate layers of our model accessible by making use of several Keras functionalities like **get_layer()** and **.output**.

```
# Get output layers corresponding to style and content layers
outputs = [vgg.get_layer(name).output for name in layer_names]
```

Those layers where then used inside the Style Extraction and Style Training Models we created in order to transfer the style of one image to another.

The Style Extraction Model returns the style and content tensors and uses the gram matrix to calculate the style.

```
def gram_matrix(self, input_tensor):
    result = tf.linalg.einsum('bijc,bijd->bcd', input_tensor, input_tensor)
    input_shape = tf.shape(input_tensor)
    num_locations = tf.cast(input_shape[1]*input_shape[2], tf.float32)
    return result/(num_locations)
```

The Style Training Model then performs gradient decent by calculating the style and content losses of the extracted layers from the Style Extraction Model, using the Adam optimizer and updating the image based on the gradients.

```
@tf.function()
def train_step(self, img):
    with tf.GradientTape() as tape:
      outputs = self.ExtractionModel(img)
      loss = self.style_content_loss(outputs)

    grad = tape.gradient(loss, img)
    self.opt.apply_gradients([(grad, img)])
    image.assign(self.clip_0_1(img))
```

Afterwards, we chose layers from our model that represent the style and the content for the transfer and were similar to the layers that were chosen in our reference paper and code [5]: convolutional layers at the beginning of each block to represent the style. We then pre-processed our input images and inserted them into our Style Training Model. Because the

Tensorflow tutorial [13] we used as an orientation took the content image and not a white noise image like the paper [5], we decided to first imitate their approach in order to make sure that the groundwork of our code worked. Later on, we switched to using a white-noise image as input.

After 100 trainingsteps our output image looked pixelated but no disinctive style could be seen:

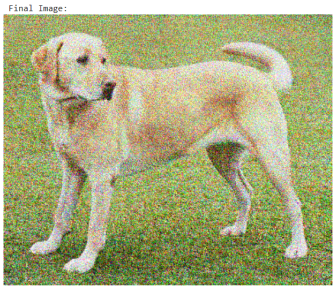

Fig. 3. 100 trainingsteps with our self-trained VGG19. Content image: Labrador, style image: Kandinsky.

After seeing no success in our own training, we decided to try out to take the weights of a (with the imagenet dataset) pretrained VGG19 and load them into our own model. We wanted to test if the problem was the way we structured our model or the training process. Because our model had to fit the weights exactly, we had to adapt our last dense layer and choose a dimensionality of 1000 because of the 1000 classes the imagenet dataset contains. We loaded the weights:

```
# Loads the weights
ourmodel.load_weights("vgg19_weights_tf_dim_ordering_tf_kernels.h5")
```

And then we followed the same steps for the style transfer process as before. The output showed a clear difference (Figure 4).

Fig. 4. 100 trainingsteps with our VGG19 model and the weights of a pretrained model. Content image: Labrador, style image: Kandinsky.

As you can see in Figure 4, the style of the Kandinsky image emerges and changes the appearance of the content image significantly. Nevertheless, you can also see that our result has some faults like vertical stripes across the image. This seems to be related to our model structure. To compare the VGG19 and the ResNet50 model and have a look at different outputs, we decided to use the pretrained VGG19.

2) Using a ResNet: A Residual Network makes it possible to train very deep neural neworks with a lot of layers and still achieve a good performance. The feature that sets it apart from the VGG is the use of shortcut connections between blocks that perform identity mapping. That means that there are fewer multiplications in the backpropagation and less problems occur regarding exploding or vanishing gradients. The shortcut identity mapping also does not add any parameters and therefore the computational time is kept in check [8]. When comparing a "plain", flatter model to a ResNet, the results show, that for the ResNet a deeper model with 34 layers performs better than a flatter one with 18 and has lower error rates. For the plain model, more layers lead to higher error rates and even the worse error rate for the ResNet-18 is better than the better one for the simple-18 [6].

In our implementation of the Style Transfer Model with a ResNet50, we tried to have a similar layout of the code to the VGG19 model. That means we again split the code in three main sections: a function that allows us to access intermediate layers of the ResNet50, a Style Extraction and a Style Training Model. With the ResNet we used the Adam optimizer with a learning rate of 20 and style and content weights of 1e7 and 1 respectively. We also decided to use a similar layer selection as the author of the Kaggle article [2] on which our code for the ResNet50 is based, as he stated that he selected each layer by hand and made sure that the result would be appealing. The layers were mainly Batch Normalization layers in the middle of the convolutional blocks. Later, we will discuss the effects of different style layer selections on the output (see section III-A3). With the mentioned structure and settings, we achieved the result in Figure 5.

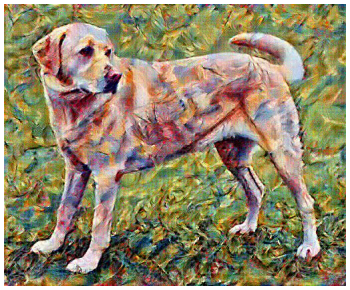

Fig. 5. 1000 trainingsteps with the pretrained ResNet50 model. Content image: Labrador, style image: Kandinsky.

3) Comparing Layer Selection - VGG19 and ResNet50: While trying to find the right settings for comparing the VGG19 and the ResNet50, we dealt with the different structures and layers of the models and had to decide which layers would give us a good style and content representation.

In our experiment to differentiate between feature extraction in varying layers, we selected an image of a beach house and a cubistic painting.

The ResNet50 consists of four bigger convolutional blocks that consist of three to six smaller blocks with three different convolutional layers. This gives us the opportunity to choose from a variety of layers to select as our style representation. We decided to try out nine different versions of combinations when it comes to choosing the "sub-block" and the position of the convolutional layer inside of it. As an example: the first

Fig. 6. Content and Style image to analyse outputs of different layer selection.

column of Figure 7 shows the output of the style transfer when every chosen style layer is located in the first sub-block of the four bigger blocks. And then we distinguish between the three different convolutional layers inside the selected block. This results in a range from the positions of the style layers at the very beginning to the very end of the four ”big” convolution blocks.

Fig. 7. Visualization of outputs when different layers of the ResNet50 are chosen to transfer style.

Results show that both the position of the style layers inside a block as well as a sub-block greatly effect the output of the style transfer. It seems as if the later a convolutional layer (”Conv Layer 3”) inside a sub-block (”Block 1”, ”Block 2”, ”Block 3”) the less the style is transmitted. When the difference between the sub-blocks is inspected, it appears that in the later sub-block (”Block 3”) the color stands out and in the earlier sub-block (”Block 1”) the shape is more pronounced.

When inspecting the layers of the VGG19, it is noticeable that the VGG19 has significantly less layers than the ResNet50. That means that the VGG does not consist of blocks that can be divided in smaller blocks and compared to each other. But it can be seen effects when different content layers are chosen.

As seen in Figure 8, there are no significant differences when the first layers of each block are chosen for the style representation in comparison to the last layers of each block. When looking closely, the colors seem to be brighter in the later style layers. However, when examining the different

Fig. 8. Visualization of outputs when different layers of the VGG19 are chosen to transfer style and content.

content layers, we can see that when the last convolutional layer of a VGG19 is chosen to represent the content, the colors in particular are closer to the image that contains the style (in this case, a cubist painting) that we want to apply to the content image than when the first convolutional layer is chosen.

4) Comparing Computational Performance - VGG and ResNet: After comparing the two models regarding layer selection, we will briefly discuss the difference in structure and resulting performance differences.

VGGs use the same number of filters for the same output feature map size and if the feature map size is halved, the number of filters is doubled to preserve time complexity per layer. A Residual Network has the shortcut connections that turn it into its counterpart residual version [6]. There are three different ways of performing shortcuts. If the input and output have the same dimensions, the identity shortcuts can be directly applied. However, if the dimensions increase, identity shortcuts can be performed with extra zero padding where there are no additional parameters needed, or projection shortcuts are used: The dimension is matched by utilizing 1x1 convolutions.

When looking at the structure differences of the models, the question of performance arises. In the VGG19 the first two layers use convolution on the full 224x224 input image. That is quite expensive and ”costs” the first layer around 170 million floating point operations per minute (FLOPs) because of the fact that it has to take the 224x224x3 input and turn it into a 224x224x64 output. With this, the FLOPs are close to 170 million * (64/3) which are almost 3.7 billion FLOPS and roughly as much as the whole ResNet34 (Residual Network with 34 layers) which has 3.6 billion FLOPs. A ResNet avoids this problem in the first layer by reducing the number of rows and columns by a factor of 2 and uses only 240 million FLOPs. The next max pooling layer again reduces by a factor of 2. When comparing this with the VGG, it becomes clear that the VGG needs roughly 10 billion FLOPs in the first four convolutional layers. The ResNet builds up the convolutional filters slowly by using less kernels but having more of them

stacked up (alternating between convolutional layers and non-linear activation functions). **The ResNet model makes use of the concept of thinner but deeper models.** In different versions of the ResNet like the ResNet34 and ResNet50, you can also notice that the ResNet50 reduces number of operations even more but also applies networks with larger number of filters in the convolutional layers. Additionally, 1x1 convolutional layers are used in order to reduce channel depth before performing 3x3 convolution and upscaling [7]. The differences between the VGG and ResNet result in the VGG19 needing 19.6 billion and the ResNet50 3.8 billion FLOPs, making the ResNet computationally exceeding.

IV. Further Exploration

A. Comparison of Performance of the Art-Transfer with a Neural Network and a Subjective Human Opinion

Now that both Networks were able to create a more or less aesthetically pleasing style transfer it would be interesting how they perform on a more objective measure. To obtain a more objective evaluation measure, another Network on the discrimination of different art styles was trained.

1) Training a Neural Network to discriminate Art Styles: The 'art movements' data-set from Kaggle was used, which contained a training and a testing set with three subcategories and an overall of 960 files. The subcategories were cubism, expressionism and romanticism. As the data-set is rather small, image augmentation was used to create 500 batches of 64 images each. The performance was higher, when only augmentation that not excessively distorted the original image was used. This limited further augmentation. This might be caused by the fact that art style heavily relies on its arrangement and thus distortions could lead to an adulteration of the style. Due to an insufficient number of samples, a validation data set was omitted. The first try to train an own Residual Network did not produce satisfactory results, as a training-accuracy of 70% could not have been exceeded . Additionally, the network was rather unstable while training. To get somewhat valuable, comparative results out of the classification, different pre-trained networks performances were compared. Different Network architectures were used for transfer-learning inter alia the InceptionV3, the VGG19, the ResNet101V2. Probably due to the small data-set all networks over-fitted on the training data-set. All networks were expanded with a GlobalAveragePooling-layer and two fully connected layers, with a dimensionality of 256 and 3, because of the number of output-classes, namely cubism, expressionism, romanticism. The ResNet101V2 showed with Adam as an optimizer and a learning rate of 25e-6 after training the best performance on unseen data with an accuracy of 85%. Therefore it was chosen for the objective comparison of the created images. For the training the best results have been reached when disabling training for the first 128 layers, which capture low level features. There was a total of 43,151,875 total parameters of which 37,005,315 were trainable (Figure 9). The trained model can be accessed for usage here.

Layer (type)	Output Shape	Param #
resnet101v2 (Functional)	(None, None, None, 2048)	42626560
global_average_pooling2d (Gl	(None, 2048)	0
dense (Dense)	(None, 256)	524544
dense_1 (Dense)	(None, 3)	771

Total params: 43,151,875
Trainable params: 37,005,315
Non-trainable params: 6,146,560

Fig. 9. Convolutional Network for Art-Style labeling

To decrease the issue of an insufficient amount of data further gathering of more art style examples from the internet to achieve even better results is suggested. Additionally, the dataset can be criticized, as it is not completely transparent, how the images were categorised and if it is done correctly. Even art historians are not always agreeable on an art works categorization, as art epochs fluently merge into each other and are probably more of a continuous categorisation than an assigned class.

Being aware of the shortcomings of this approach, the accuracy (acc) for the different style-transfer-networks was calculated and is displayed in Figure 10. To simplify transferring a huge amount of styles a loop was used and the overall default parameter of the Style Transfer Model were a learning rate of 10, beta of 0.99, epsilon of 1e-1, using Adam. The Style weight was set to 10000 and the content weight to 1e6. It did not influence the networks performance when the content instead of the noise image was used as the input for the style transfer with the VGG19 (overall acc = 0.56). It is visible, that the performance for discriminating expressionism was poorly in all cases (acc = 0.0). The ResNets style transferred images performed slightly better but still bad (acc = 0.2). When looking at the stylistic elements from cubism and expressionism, which are both abstract epochs, which also seem to overlap in their chronology. With cubism being labeled from around 1907 and 1914, and therefore merely a sub-categorical current of expressionism, which is stated around 1905 to 1933 [1]. It is noticeable that they are stylistically closer than to naturalistic epochs. This could have influenced the performance on the image categorisation heavily.

2) Subjective Human Opinion: While exploring the two networks and trying different layers for the content representation, we figured that the subjective performance of the style transfer, depends not only on the chosen hyper-parameter but also on the style-image. For an aesthetically most pleasing result there has to be fine-tuning of the weights performed. Additionally, we sometimes got better results when using the content image to calculate the loss for the gradient descent instead of a white noise image (Figure 11). It can be hypothesized that by this the content image will have a mathematically heavier influence on the loss, as the vector

	VGG19		ResNet50
Accuracy:	White Noise Input	Content as Input	Content as Input
Overall	56%	56%	73%
Cubism	90%	90%	100%
Expressionism	0%	0%	20%
Romanticism	81%	81%	100%

Fig. 10. Accuracy from the Classification Network for the style transferred images for the VGG19 with the noise image as an input, the VGG19 with the content image as an input, the ResNet50 with the content image as an input. Shown is the overall accuracy on the whole data set and the accuracy for the single labels

of the content image and not a random noise image is taken for the loss calculation. Therefore the resulting image will be more similar to it and does seem like a better transfer to us. It also seems as if this method is not as sensitive to the lambda or the learning rate. As the content image is adjusted, the in Figure 13. shown noisy patches will be more similar to the content image and therefore not as noticeable.

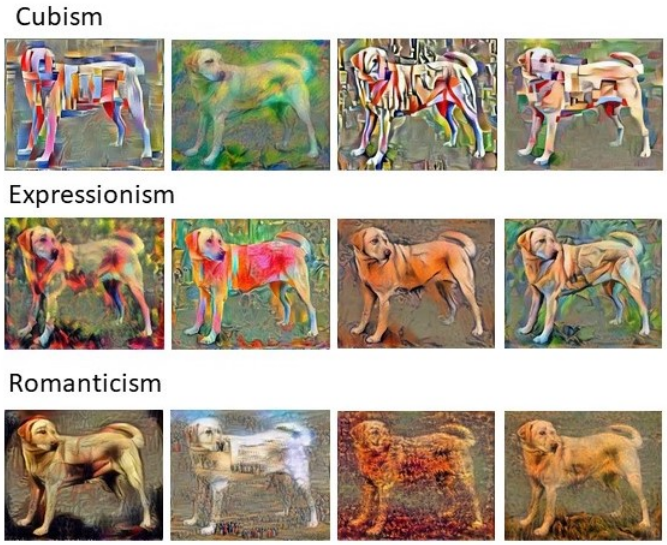

Fig. 11. Style Transfer with the VGG when using the content image as a reference and not the noise image. 1000 Training steps were performed.

Overall, even when either network was not always able to capture the style elements completely without further fine-tuning, it was noticeable, that the prevailing atmosphere was usually captured pretty well. We captured some images that we considered aesthetically pleasing in Figure 12. and some that we did not in Figure 13. The not well transferred style images have some patches that lack content and style. It could be, because another weighting of the loss is required. It was not explored further in the scope of this work, but it could be suggest that the images may need smaller and more training steps and that the parameter may jump over the global optima while training, because the phenomena was more often observed with an increasing learning rate with uniform training steps (Figure 1. All in all both networks were able to created aesthetically pleasing images.

Fig. 12. Style Transfer with the VGG when using the noise image as input from cubism (a), expressionism (b), romanticism (c)

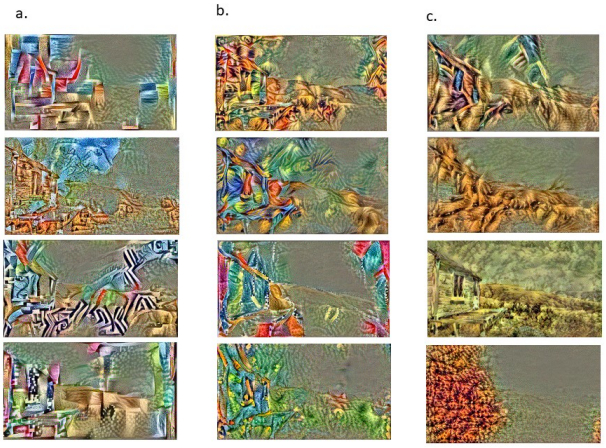

Fig. 13. Style Transfer with the VGG when using the noise image as input from cubism (a), expressionism (b), romanticism (c)

B. Style representations

When comparing the earlier layed out VGGs style representations to the one of the layers from our final ResNet selection, it is noticeable that in the VGG with adding later layers the stylistic concepts grew in size. Whereas in the ResNet there was not that much of a stylistic difference between the the earlier layer or the same gradual increase in pattern complexity visible. Especially when comparing 'conv2_block3_3_conv', 'conv3_block3_3_conv' (Figure 14. (b)) and 'conv2_block3_3_conv', 'conv3_block3_3_conv', 'conv4_block3_3_conv' (Figure 14. (c)). These layers look comparitatively similar. As both layers were taken from the same residual block, the feature representation similarity can

be explained. Like in the VGG, when adding the last layer of the ResNet higher level, more complex and larger stylistic concepts are represented (Fig. 14, (d)). One can again see yellow with circle structures or black with straight longer lines. The first representational layer shows smaller and more vague patterns than the other layers, even though it differs from the first VGG layer by already showing larger style features, like rounded lines and paint-brushed-like texture (Fig. 14, (a)).

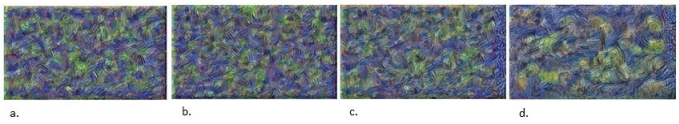

Fig. 14. ResNet style representations of the images in the different layers. The representation of the style image, as calculated with the gram-matrix for layer 'conv2_block3_3_conv' (a),'conv2_block3_3_conv' and 'conv3_block3_3_conv' (b), 'conv2_block3_3_conv', 'conv3_block3_3_conv' and 'conv4_block3_3_conv' (c), 'conv2_block3_3_conv', 'conv3_block3_3_conv', 'conv4_block3_3_conv' and 'conv5_block3_3_conv' (d)

C. Changing Lambda

When adjusting the weighting for the content and style representation it became visible that with an increasing learning rate and a decreasing lambda the content became more prevalent. Furthermore, an increase in the style-element-size with an increasing learning rate was observed. They were less fine grained and appeared to be more saturated. The transfer, understandably also appeared to need more training steps with a decreasing learning rate. Due to computational accessibility it was refrained from increasing the number of training-steps for the quantitative style transfer. More exploration of the lambda combinations and the learning rate is uploaded on GitHub. The results are uploaded on GitHub as well. It can also be seen that with a decreasing lambda the content became more prominent. This is opposite to the finding described in the paper by Gatys et. al. (2015) [5]. They code probably differed. With the coding decisions made from us it is explainable that when taking a lower number for the style weights, and thereby decreasing the lambda that the style becomes less prominent. For example when the content weight is 1e6, and a lambda of 1e-3 is chosen, the matching style weights are 1000. When the lambda is decreased to 1e-4 the style weights decrease to 100 and the style becomes less influential.

D. Ablation Studies

As mentioned before, there were some difficulties while executing the experiments and carrying out this project. Some elements that effected the success of our studies were for example the limited computational power and the datasets. Working with Google Colab gave us the opportunity to use a GPU. That was very useful, even though we switched to Kaggle from time to time, due to the limited time per day and the elongated processing when rendering a bigger batch of style transferred images.

Furthermore, while trying to train our own VGG19, the problem of choosing a dataset arose. To train such a model, a dataset with a great number of different images, that had a sufficient enough size for reasonable feature extraction for the style and content is needed. With our computational limitations it was the better choice to use a pre-trained network for maximal satisfying results. Still, the subjective style transfers performance did rely on the chosen layers for the content representation. Through the exploration of different weight ratios and learning rates some vivid style-transfers may have been created.

V. Conclusion

Training an own implementation of the VGG19 could not be done sufficiently because of the lack of an adequately large dataset and ample computing power. But the structure of the VGG19 could be recreated and, with the weights of a previously trained VGG19, an output image with a recognizable art style transformation could be created.

A ResNet was also able to transfer the art style from one image to another. But it could be noticed that more trainingsteps were needed in order to achieve a good style representation. The success of the style transfer was also dependent on the intermediate layer selection. When analysing the different layers, it was seen that the location of a layer in a convolutional block reflects the feature extraction of low- and high-level features of an image. In the VGG19, a content layer has to be chosen that is located at the end of the model. In the ResNet, coosing a content layer from the beginning of the model seems to be effective. But the style layer should also be selected from the beginning of each convolutional block in order to transfer color as well as shape from the style to the content image.

Computationally, the ResNet50 performs better than the VGG19. After analysing the floating point operations per second (FLOPs) each of the models uses, and taking into consideration that the ResNet50 is significantly deeper than the VGG19, is was expected that the ResNet would have a higher accuracy and lower error rate and therefore perform better in transferring style. his is reflected in the experiments of classifying the art style of transformed images, where the ResNet50 outperformed the VGG19 regarding accuracy.

When using an algorithm to determine which style transfer was better, the ResNet outperformed the VGG, even though the difference was minor. When looking at the miss-classified samples, another problem with the objective classification of art arose. Epochs are usually fluent and a style of an epoch develops and even fluctuates over time. Therefore it is challenging to classify art correctly, not only for artificial-algorithms but also for humans. It would be interesting to see how a continuous classification would categorise art pieces in general. Subjectively, both networks performed a good style transfer and it is up to the viewer to decide which art style

transfer method they prefer. Because great art is something truly subjective and beauty lies in the eye of the beholder.

REFERENCES

[1] "Expressionism movements overview." [Online]. Available: https://www.theartstory.org/movement/expressionism/

[2] L. Barbosa. Neural style transfer using resnet50 with tf.keras. [Online]. Available: https://www.kaggle.com/code/lbarbosa/neural-style-transfer-using-resnet50-with-tf-keras/notebook#Closing-Remarks

[3] T. Q. Chen and M. Schmidt, "Fast patch-based style transfer of arbitrary style," *arXiv preprint arXiv:1612.04337*, 2016.

[4] L. Gatys, A. S. Ecker, and M. Bethge, "Texture synthesis using convolutional neural networks," *Advances in neural information processing systems*, vol. 28, 2015.

[5] L. A. Gatys, A. S. Ecker, and M. Bethge, "A neural algorithm of artistic style," *arXiv preprint arXiv:1508.06576*, 2015.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[7] itdxer. Why is resnet faster than vgg. [Online]. Available: https://stats.stackexchange.com/questions/280179/why-is-resnet-faster-than-vgg

[8] A. Kaushik. Understanding resnet50 architecture. [Online]. Available: https://iq.opengenus.org/resnet50-architecture/

[9] ——. Understanding the vgg19 architecture. [Online]. Available: https://iq.opengenus.org/vgg19-architecture/

[10] Y. Li, N. Wang, J. Liu, and X. Hou, "Demystifying neural style transfer," *arXiv preprint arXiv:1701.01036*, 2017.

[11] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[12] P. U. Stanford Vision Lab, Stanford University. Imagenet. [Online]. Available: https://www.image-net.org/

[13] Tensorflow. Neural style transfer. [Online]. Available: https://www.tensorflow.org/tutorials/generative/style_transfer#setup