# Neural Networks for Images (67103): Exercise 4

## Style Manipulation and Classifier-Guided Denoising Diffusion

By: Amit Halbreich, ID: 208917393

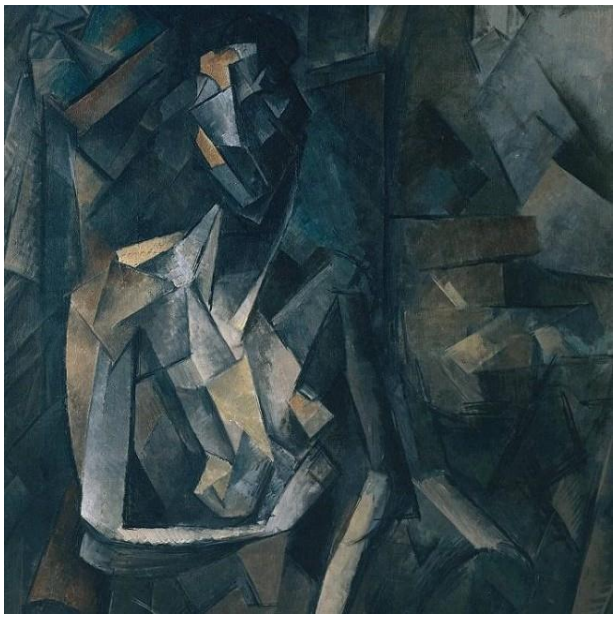
Eitan Stern, ID: 211997275

The goal was to manipulate the style of images by extracting the style from one image and transferring it to another images, all while preserving the content of the target image.

In the process, we carried out various experiments such as Feature Inversion, Texture Synthesis, and implementation of the “MeanVar” style loss method.

We load two types of images - a style image, which we want our output image to mimic in terms of style, and a content image, the content of which we want to preserve in our final output image.

Style Image



Content Image

The content loss is obtained by comparing the feature maps of the content image and the generated image at a particular layer in the network. As thought in class - these feature maps essentially capture the 'content' of the images. The comparison is done using the Mean Squared Error of the two feature maps. The resulting value is taken as the content loss, which reflects the discrepancy in content between the generated image and the content image. During training we tried to minimize this content loss, which would mean the generated image closely matches the content image in terms of high-level features.

The style loss is derived from the style representation of an image, which is computed using the Gram matrix.

The first step in calculating the style loss is to obtain the feature maps of the generated image and the style image at a specific layer in the network. These feature maps are then reshaped into a 2D matrix, where the number of rows corresponds to the number of feature maps, and the number of columns corresponds to the number of elements in each feature map.

The Gram matrix is then calculated by multiplying this reshaped matrix with its transpose. This operation produces a matrix that captures the correlations between the different feature maps, effectively encapsulating the style of the image.

Next, this Gram matrix is normalized by dividing each of its elements by the total number of elements in the matrix. This step ensures that style features from deeper layers (which typically contain larger Gram matrices due to higher dimensional feature maps) have a comparable impact to those from earlier layers during the gradient descent optimization process.

After creating the loss functions, we create the model that we'll use for neural style transfer. This involves setting up a pre-trained Convolutional Neural Network (CNN), adding custom modules to compute the content and style losses, and then modifying the architecture to incorporate these custom modules.

We begin by setting default layers where the content and style losses will be computed. We've selected early, medium and late stages of Convolutional layers, ReLu Activation layers and Max Pooling layers for Feature Inversion and Texture Synthesis processes.

We then define a function that will set up the style transfer model and compute the losses. This function takes as input the pre-trained CNN, mean and standard deviation for normalization, style and content images, layers for computing content and style losses, and the style loss function to use.

Within the function, we first create a normalization module using the provided mean and standard deviation values. Then, for each layer in the CNN, we perform different actions based on the layer type. If it's a convolutional layer, we check if it's one of the layers where we need to compute content or style loss. If it is, we add the appropriate loss module right after that layer in the model.

This setup allows us to compute the content and style losses while the image is passing through the network. This is possible because these loss modules act as transparent layers in the network that compute the losses but allow the forward pass to continue normally.

Finally, we trim any additional layers in the network that come after the last loss computation layer. This ensures that our computations are only performed up to the point where they are required. At the end of the function, we return the modified model along with lists of the style and content losses.

# PART 1

## FEATURE INVERSION

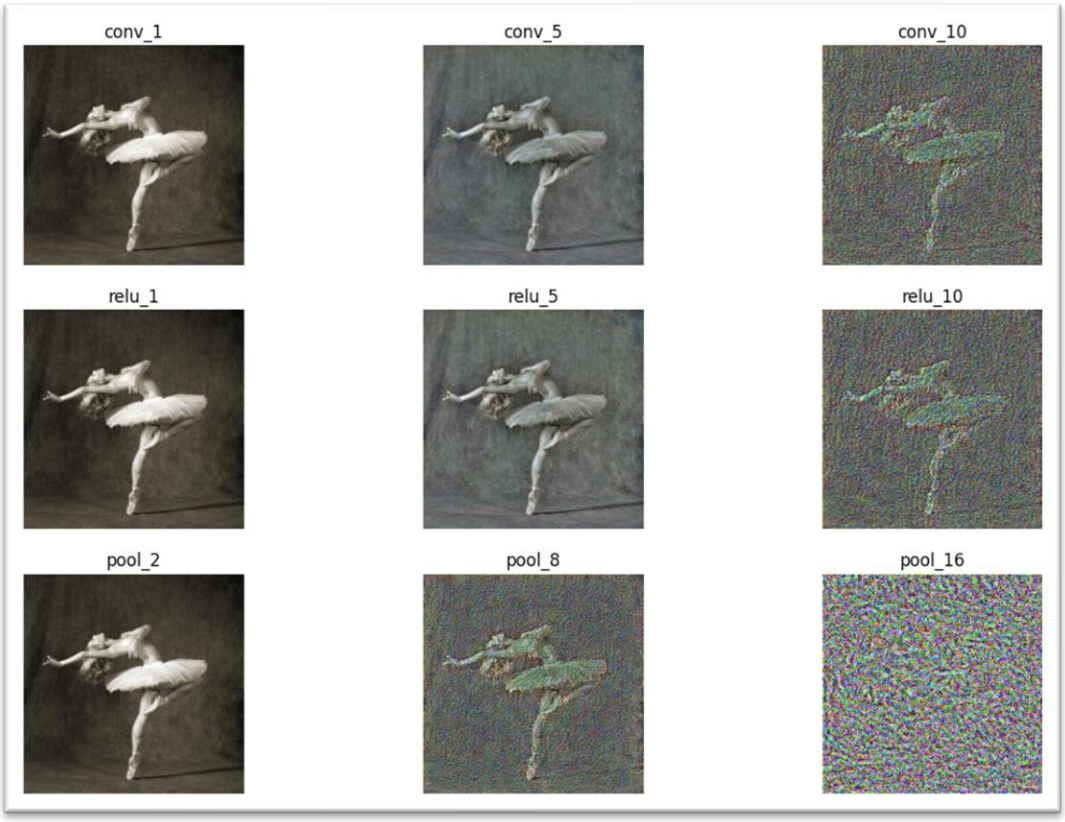
**1.A.** We begin by setting default layers where the content and style losses will be computed. We've selected the following layers, for content loss computation:

- $1^{st}, 5^{th}, 10^{th}$ Convolutional layers ('conv_1', 'conv_5', 'conv_10 accordingly)
- $1^{st}, 5^{th}, 10^{th}$ ReLu activation layers (relu_1', relu_5', relu_10 accordingly)
- $1^{st}, 8^{th}, 16^{th}$ Max Pooling layers (pool_1', pool_8, pool_16 accordingly)

We began the process by generating a random noise image of the same size as the target image. Next, we extracted feature maps from various layers of the VGG-19 network, including convolutional layers, ReLU, and Max Pooling layers from early, mid, and late stages of the network. Our objective was to optimize the input image iteratively until its feature maps closely resembled the corresponding feature maps of the VGG-19 network. To achieve this, we utilized the Mean Squared Error (MSE) loss between the target image and the extracted features.

To accomplish this task, we made modifications to the 'run_style_transfer' function. The modified function now accepts the names of the content layers as parameters, allowing us to specify which layers to target for content preservation. Additionally, we set the 'style_weight' parameter to 0, indicating that the optimization process would prioritize matching the content rather than the style of the images.

Now, let's try to optimize for different layers for the content. We're doing the same thing but let's try to optimize for the content for different layers when we did that, we got these results:



Upon observation, we notice that the variation among different layer types (such as convolution, ReLU, and Max Pool) at a specific depth is minimal. The actual disparity lies in the varying depths of the layers.

The initial layers of the network tend to extract basic, low-level features from the target image. When we optimize an image to match these early layer features, the resulting image showcases these fundamental visual elements. As we progress deeper into the network, the layers start capturing more complex, high-level features. Consequently, images generated to match the features of these mid layers exhibit more noise and exhibit sharper transitions in edges. In the later layers, the resulting image appears noisy and devoid of recognizable patterns. These higher-level features are more abstract in nature and have a broader receptive field, which implies that they contain less precise spatial information compared to the earlier layers.

## TEXTURE SYNTHESIS

**1.B.** We begin by setting the default layers (same as previous section) where the style losses will be computed. We've selected the following layers, for style loss computation:

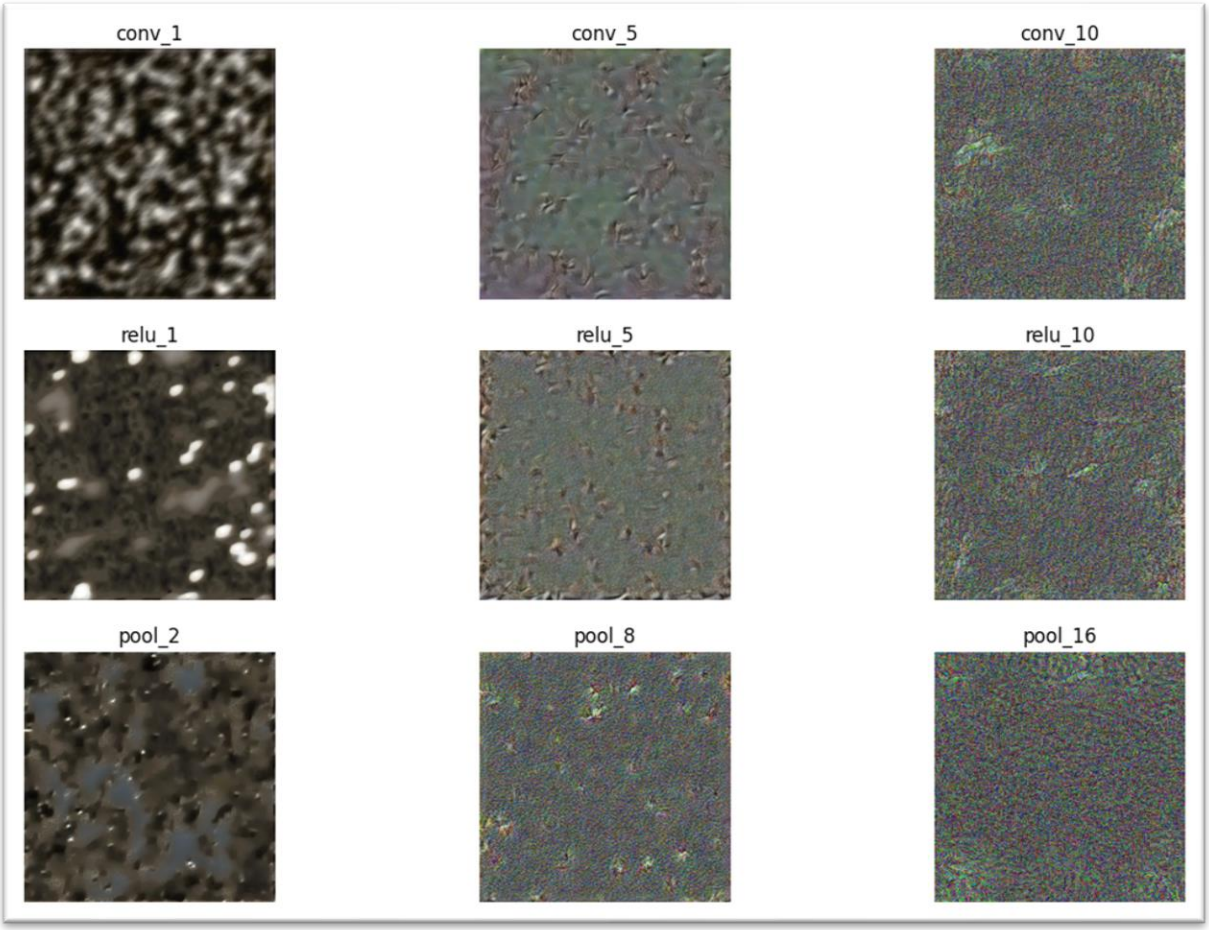
- $1^{st}, 5^{th}, 10^{th}$ Convolutional layers ('conv_1', 'conv_5', 'conv_10 accordingly)
- $1^{st}, 5^{th}, 10^{th}$ ReLu activation layers (relu_1', relu_5', relu_10 accordingly)
- $1^{st}, 8^{th}, 16^{th}$ Max Pooling layers (pool_1', pool_8, pool_16 accordingly)

Now as we looked at the images that were created from trying to optimize for certain layers in the pre-trained VGG-19 network for the style gram metrics from a randomly initialized noise – this is what we got:

We began by generating a random noise image with the same dimensions as the target image. Next, we extracted feature maps from various layers of the VGG-19 network at different stages (early, mid, and late). We then computed the Gram matrix for these feature maps. The goal was to optimize the input image iteratively until its feature map's Gram matrix closely resembled the corresponding Gram matrix of the VGG-19 network's layer feature maps.

To achieve this, we made modifications to the 'run_style_transfer' function. The updated function now accepts the names of the style layers as parameters, enabling us to specify which layers to target for style transfer. Additionally, we set the 'content_weight' parameter to 0, indicating that the optimization process would prioritize matching the style rather than the content of the images.

The results are presented below:




As previously mentioned, the initial layers of the network capture basic, low-level features from the texture image. When we optimize a noise image to match these early layer features, the resulting image highlights these low-level visual characteristics. As we delve deeper into the network, the layers capture more complex, higher-level features, revealing finer details within the texture in the deeper layers. However, as we continue to go even deeper, it becomes increasingly challenging for us to discern the smaller details due to the abstract nature of the features being captured.

As taught in class it seems like the first layers were able to detect the edges and the small details in the original Style picture and as we progress it seemed to capture the more general style of the picture and not the edges.

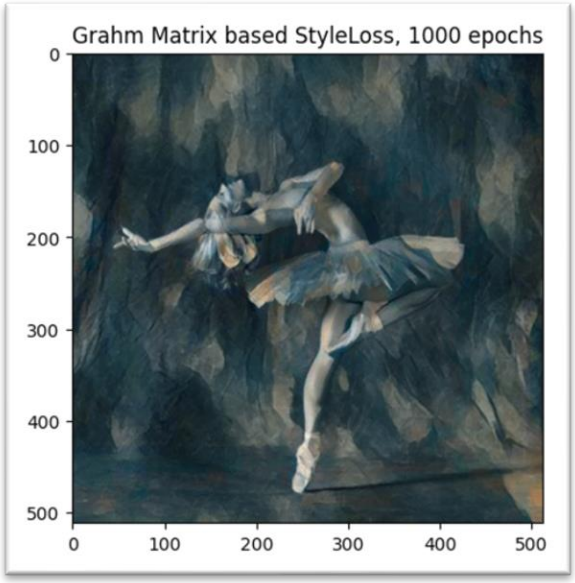
## MIN-VAR STYLE LOSS VS. GRAM BASED STYLE LOSS COMPARISON

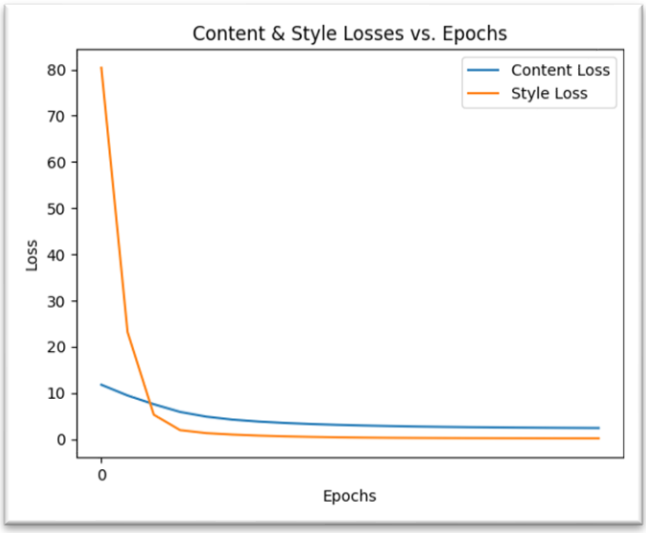
1.C. For this section we defined a new loss function called 'MeanVarStyleLoss'. Within the forward call of this loss function, we calculate the mean and variance of both the input feature map and the target image feature map. We then compute the Mean Squared Error (MSE) loss between these mean and variance sum values and combine the results. This allows us to capture and incorporate both the statistical mean and variance information in the style transfer process.

For Gram Matrix based StyleLoss results: The image that we got at the end is very similar in content to the original image. And varies according to the selected weight. With 300 epochs the sharpness of the content image contour lines is duller.

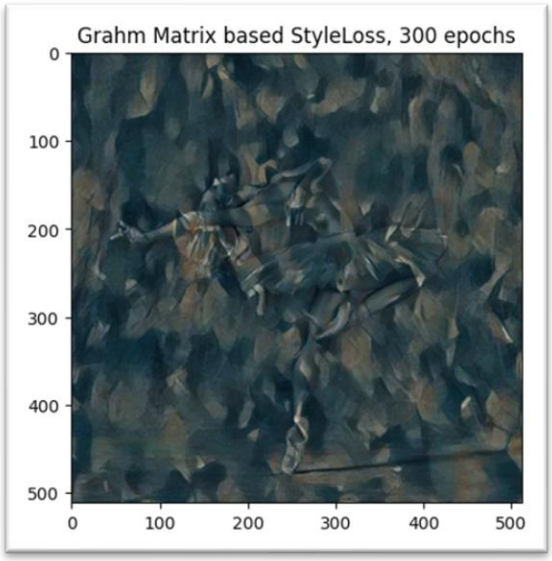
### 1000 EPOCHS, STYLE WEIGHT = 1 MILLION

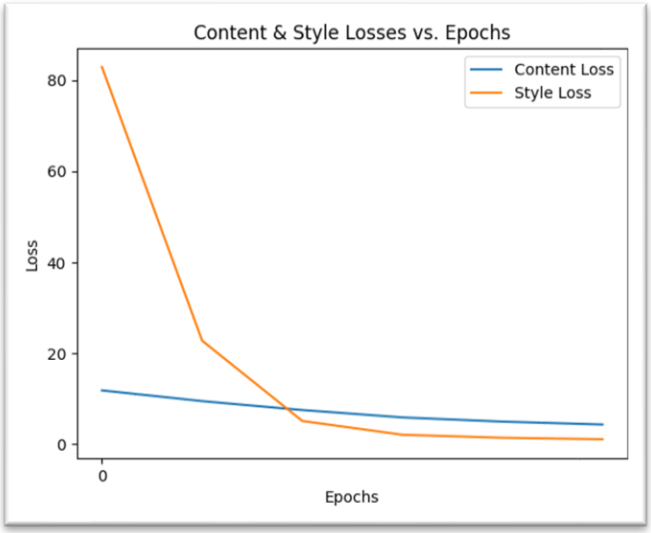
(Gram Matrix has low value of correlations therefore the style weight is increased)




### 300 EPOCHS, STYLE WEIGHT = 1 MILLION



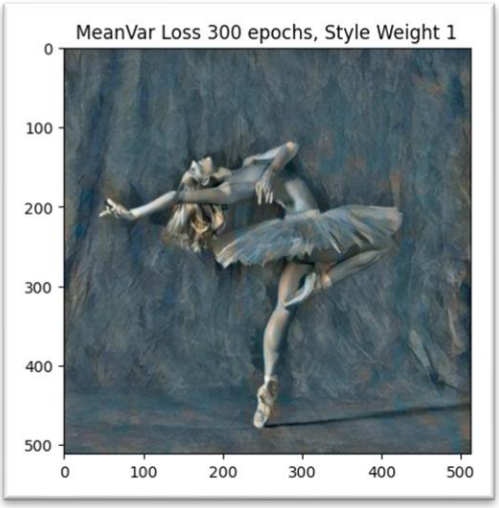



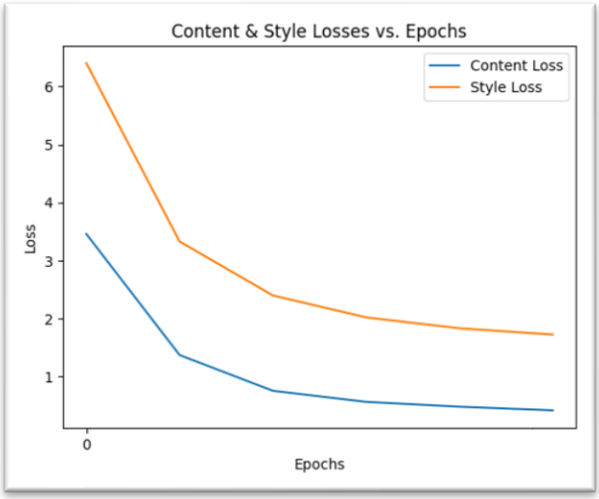

Next, for the MeanVarStyleLoss - which is another way to encapsulate the style of an image, but instead of using the Gram matrix, it makes use of the mean and variance of the feature maps.

To compute the MeanVarStyleLoss, we start by obtaining the feature maps of the style image and the generated image at a certain layer in the network. We then calculate the mean and variance of each of these feature maps across their width and height dimensions. The mean captures the average activation value for each feature map, while the variance captures the spread or diversity of these activation values.

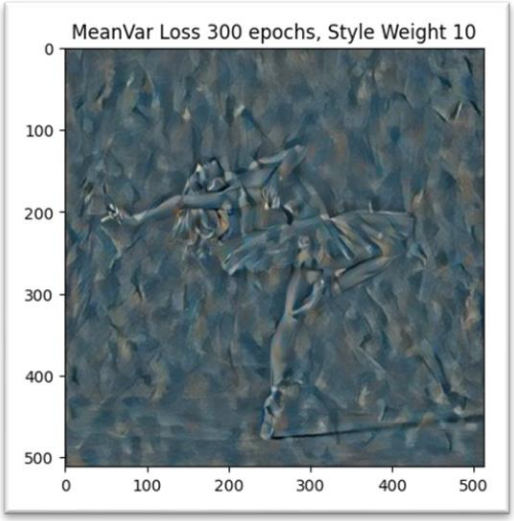
The mean and variance values for the style image act as the targets for the corresponding values of the generated image. The MeanVarStyleLoss is then calculated as the sum of the Mean Squared Errors between the mean values and the variance values of the generated image and the style image, respectively. For trying to optimize the content image to the style image with the Mean-Var Loss - we ended up getting the following results:

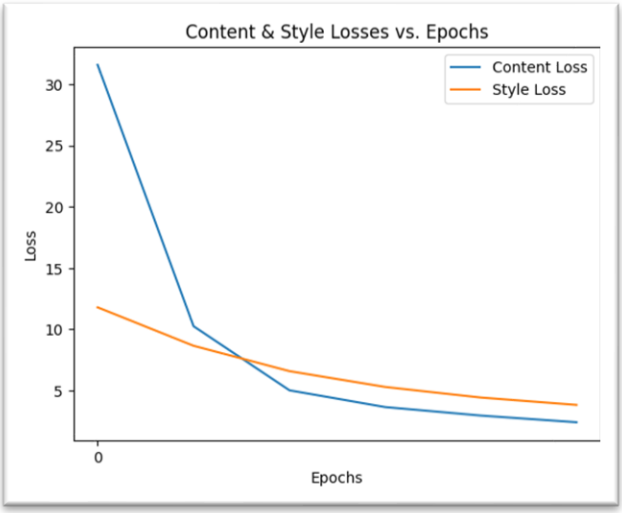
### 300 EPOCHS, STYLE WEIGHT = 1



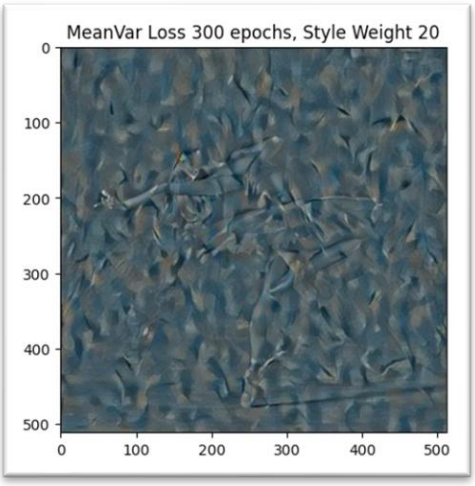



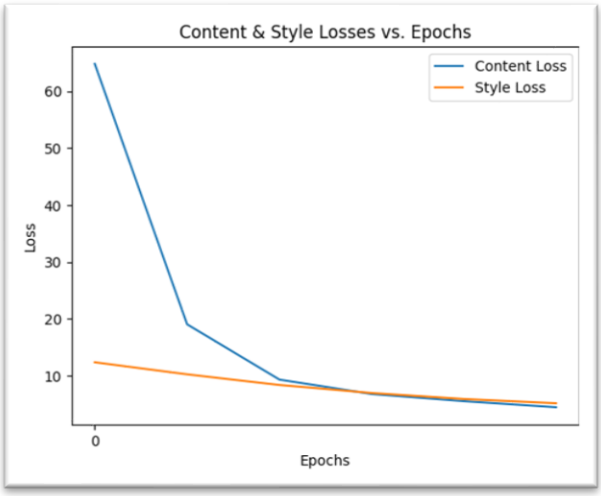

### 300 EPOCHS, STYLE WEIGHT = 10



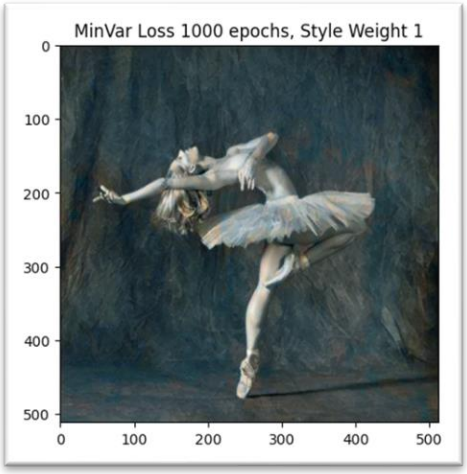



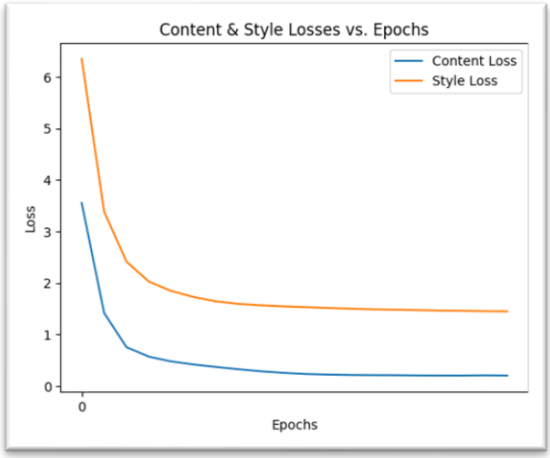
## 300 EPOCHS, STYLE WEIGHT = 20



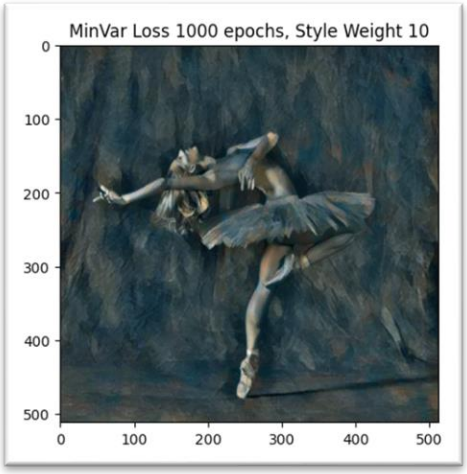



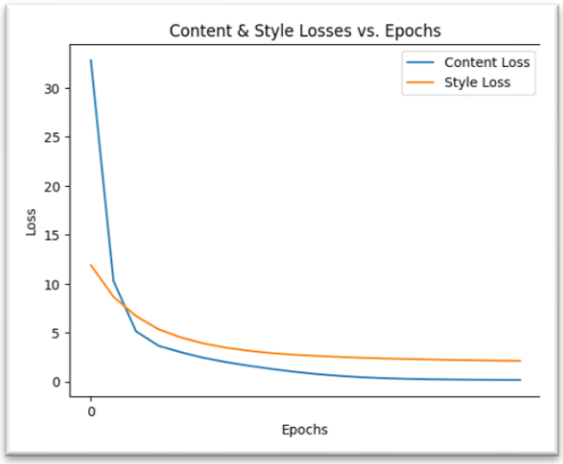

## 1000 EPOCHS, STYLE WEIGHT - **1**



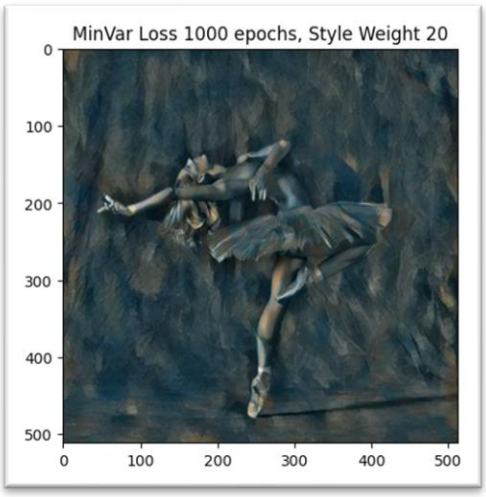



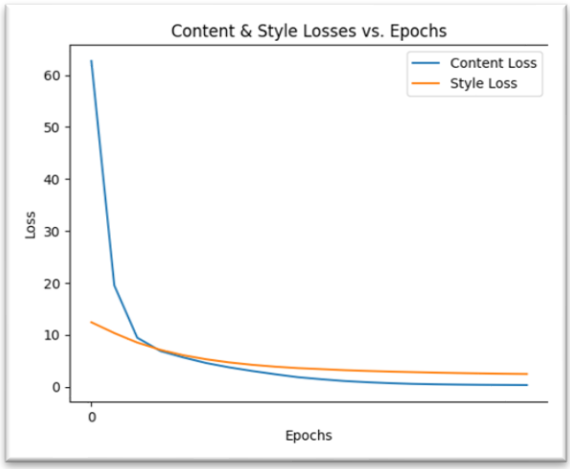

## 1000 EPOCHS, STYLE WEIGHT – **10**



## 1000 EPOCHS, STYLE WEIGHT – 20




The conventional style loss technique relies on the utilization of the Gram matrix, which effectively captures the interrelation among different feature maps and serves as a representation of the image's texture or style. However, with the introduction of the 'MeanVarStyleLoss' class, we take a different approach. This new loss function computes the Mean Squared Error (MSE) between the means and variances of the input and target feature maps.

By incorporating mean and variance statistics, we introduce an alternative method that yields distinct outcomes compared to using the Gram matrix alone. Matching the means of feature maps compels the output image to possess a similar average intensity within each feature map as observed in the style image. Similarly, matching the variances compels the output image to exhibit similar variability within each feature map as observed in the style image.

It is important to note that the correlation between different feature maps plays a vital role in capturing texture or style representation.

# Part Two

For the second part we attempted to steer the result towards some target style by using the gradient of the style loss instead of the gradient of the classifier's logit.

We add the code of this part as a .py file (because we had to use GPUs from Eitan's PC from work, and it didn't work with Google Colab).

First, lets start with a picture the cat without steering the result at all and using the basic model (from the link provided in the ex4 explanation).

The prompt entered - *"A cat"*:

### Generated Image – Classifier Guidance Free Diffusion Model

Now, let's steer the results to the style pic (same style pic as last question) after playing with the hyper parameters the best image we get for 100 steps was *"A cat"*:

Diffusion obtained with Style Loss Guidance using Guidance Factor 0.6 and 0.85:

### Guidance Factor = 0.6

### Guidance Factor = 0.85

And for 150 steps with the same prompt – "*A cat*" We can see resemblance to the style image.

GUIDANCE FACTOR = 0.6

When we tried to put the loss of the style steering with classifier-guidance to be the MeanVar loss this is the result we got for the prompt "A cat": GUIDANCE FACTOR = 0.6

Now, when using the prompt - *"A picture of a cat with the style of Picasso"*, We get the following image:

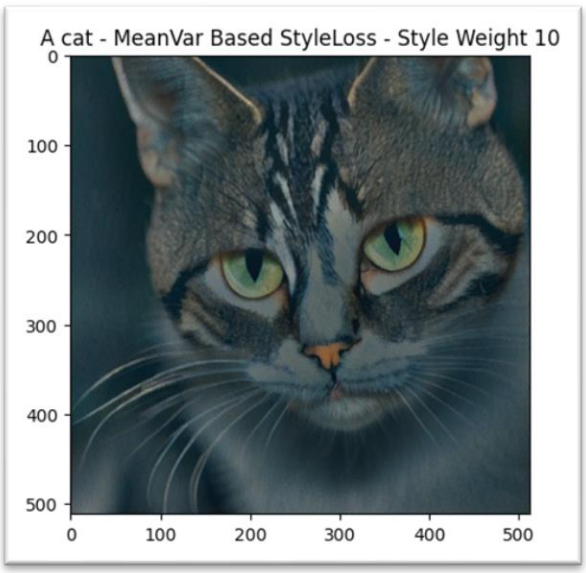
For comparation to the first part of the assignment, when we use the generated guidance-free content image we got from the diffusion model pretrained network and combining it with the style image:

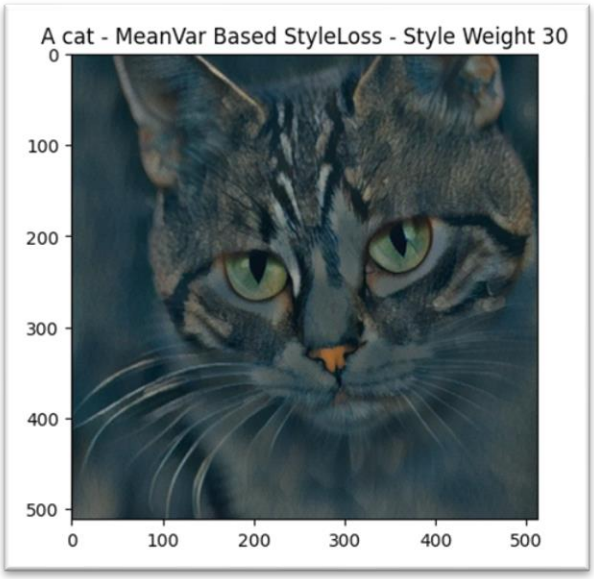
STYLE IMAGE

GENERATED IMAGE

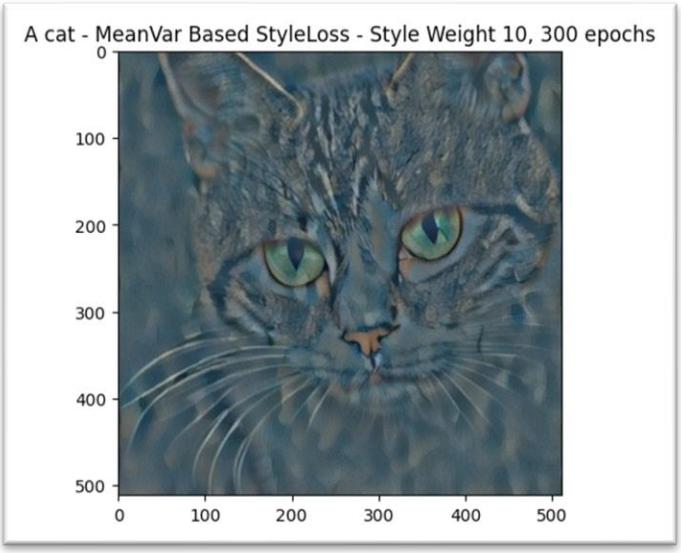
And when we try to optimize for the MeanVar loss we got:

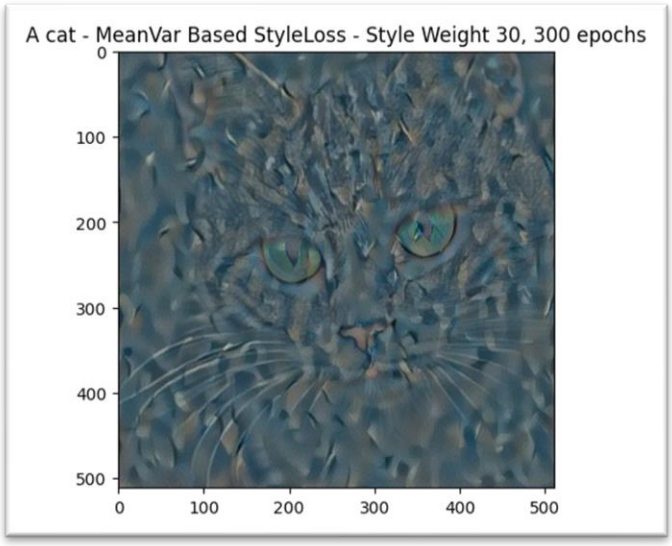
1000 EPOCHS



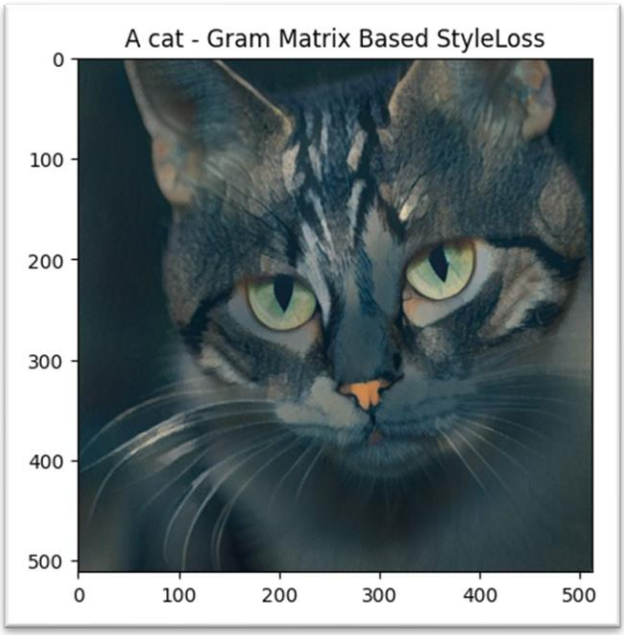
300 EPOCHS



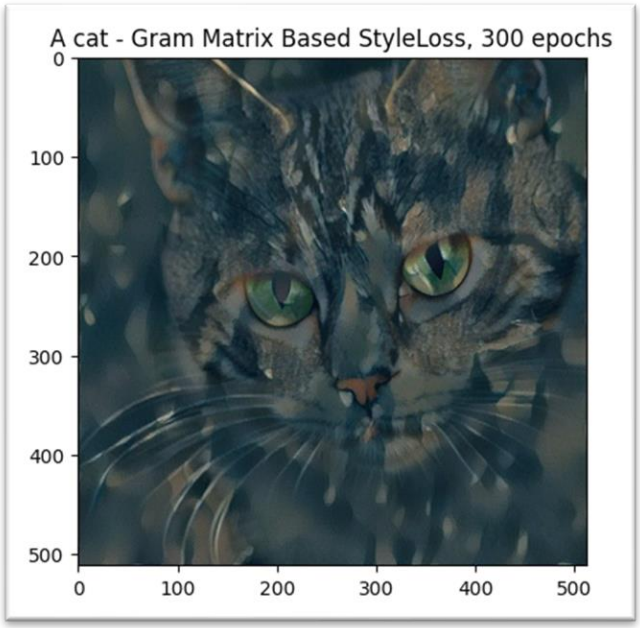




And for Gram Matrix based style loss we got:

1000 EPOCHS ($Style\ Weight\ 10^6$)




300 EPOCHS ($Style\ Weight\ 10^6$)




In optimization-based Neural Style Transfer, the goal is to apply the style of one image onto another image. This is achieved by iteratively modifying the target image to match the style of the reference image. However, in the diffusion process guided by style loss, the generated image evolves gradually over iterations while considering both style and content simultaneously.

This is why the diffusion process guided by style loss produces more natural-looking results. The style is seamlessly blended into the generated image, resulting in a cohesive and harmonious appearance. On the other hand, in traditional neural style transfer, the style is often imposed on the target image as a forced texture, which can make the final result appear less natural and more like a combination of the original content image and the applied style.

Upon observation we can see that using MeanVar based Style Loss with 300 epochs as well as 1000 epochs in comparison to the 300 epochs and 1000 Gram Matrix based Style Loss the results seem a lot better, blended-in and natural using Gram Style and not MeanVar loss. When using MeanVar Loss the Style Image "takes over" and shadows the Content Image.

This particular meanVar loss, was not covered during class lectures. However, it appears that the meanVar loss exhibits limited effectiveness in preserving the intricate details present in the style image. This can be attributed to its distinct approach, as it does not engage in the multiplication of all features with one another, unlike the Gram Matrix multiplication process. Consequently, the meanVar loss primarily focuses on optimizing the color mean and variance, characteristics similar to those found in the style image. However, this loss does not actively strive to retain the original style image's intricate features.

The stable-diffusion model underwent training using a diverse collection of style images from Picasso. By incorporating the style into the text prompt, the model generates images that closely resemble Picasso's unique artistic style. When all guidance in the text prompt is utilized, the image is guided to align with the entirety of the description in a coherent manner.

On the contrary, when guiding the content through text and the style via style classifier guidance, two distinct optimization processes occur. Each process strives to optimize its respective results independently. This is likely the reason why we obtain superior results that closely resemble Picasso's style when we add style guidance to the prompt. The combination of content guidance through text and style guidance results in a more pronounced Picasso-like representation within the generated images.