

Optimizing Classification Efficacy of Image Classifiers Through the Usage of Neural Style Transfer in Image Preprocessing

Kevin Gauld

Abstract

Currently, machine learning models are becoming incredibly prevalent in the industry due to their usefulness in providing quick access to classifications and calculations, with one common implementation being in the classification of tumors as benign or malignant. However, such neural image classifiers require a large amount of data, which is often difficult to obtain. Thus, image data augmentation has been used to artificially increase the data sets used in training image classifiers. Also utilizing machine learning, Neural Style Transfer allows for the recreation of an image in a different style, commonly used in recreating images in the styles of famous artists. The purpose of this study was to assess the efficacy of image classifiers when data was augmented using varying intensities of Neural Style Transfer, compared to common methods of static image augmentation. Images of scorpions, crabs, and butterflies were accessed from the Caltech-101 Image Classification Dataset. Style transfer was performed on 30 images of each class, transferring style from Wassily Kandinsky's *Composition VII*. Classifiers were trained on 10 images along with their augmented counterparts, and tested on the unused images within each used class in the Caltech dataset. Style transfer was tested at content-to-style ratios of 10^9 , 10^8 , and 10^7 , in order of increasing intensity. The control classifier was trained only on the original images without any augmentation. The precision and recall values were analyzed for each class within the networks, and were used to show that the neural style transfer process increases the classification accuracy of image classifiers.

Introduction

Within the last few years, numerous advancements in fields such as autonomous driving, game theory, and astronomical research have all benefited from major developments in deep learning applications. However, many of these applications require a large data set in order to generalize patterns, and make valid predictions. If there is not enough data present, classification networks can fall prey to overfitting, which is when the classifier becomes too strict in its decision making. Datasets which are large enough to combat overfitting are commonly very difficult to form. (Simonyan, 2015)

To alleviate the issue of collecting a large amount of training and testing data, neural style transfer can be used as a possible solution for nets which require images to train on (Jackson, 2019). This process focuses on taking stylistic features from a given “style image”, and recreating a given “content image” using the extracted style (Gatys, 2015). This process can be implemented using neural networks used to classify images which implement hidden layers to store the feature space (Jackson, 2019).

Neural networks are computational structures which attempt to model the functioning of a human brain to assist computers in pattern recognition and decision making. Neural nets are structured as a layered system of neurons (nodes with numerical values) with weights (multipliers) altering the connections between neurons in the subsequent layer, and biases altering the neurons themselves. Layers in between the input and output layers are called hidden layers. Weights indicate how often a neuron fires, while biases impact the outcomes of the firing of the neurons. These weights and biases are the values which are modified in the learning process. (Krizhevsky, 2012)

Initially defined randomly, these weights and biases are enhanced to make the final output of the neural network “optimal”. An optimal neural network has the lowest loss, which is defined by a function which defines how close the output is to the intended state. (Karras, 2019) To minimize loss, the neural network undergoes gradient descent, where the error of the output is found and used to minimize the gradient of the error with respect to the weights of the net. A gradient is calculated using partial derivatives as shown in Equation 1, which defines the gradient ∇f at coordinates (x,y) . This equation can be expanded as the dimensions of the function increase, allowing for extra parameters and partial derivatives as necessary. (Ruder, 2017) Through gradient descent, a backpropagation algorithm alters weights and biases such that the error moves down the gradient of the loss function, leading the error value to a local or global minimum where the output state is determined to be optimal (Zeiler, 2012).

$$\nabla f(x, y) = \left(\frac{\partial f}{\partial x}(x, y), \frac{\partial f}{\partial y}(x, y) \right) \quad (\text{Equation 1})$$

While gradient descent is consistent as an idea, there are many different methods by which the loss propagates down the gradient. One of the simpler forms of gradient descent is in Mini-Batch Gradient Descent. This algorithm combines Batch Gradient Descent (the gradient of the loss function as a

whole is computed) and Stochastic Gradient Descent (the gradient of each training example is computed) to batch data into a batch size n and to perform Batch Gradient Descent on each of these small batches, combining the batches stochastically after the gradients are computed. (Ruder, 2017)

Momentum is one strategy used to increase the efficiency of gradient descent. As shown in Figure 1, a Stochastic Gradient Descent (SGD) model will reach a minimum much faster when using momentum, which accelerates the descent along the negative gradient, decreasing the time and computations needed to reach a minimum. (Ruder, 2017) Another acceleration algorithm is the Adadelta algorithm. This algorithm updates the learning rate as the process takes place, allowing for accelerated learning as the network descends down the gradient, allowing for better predictions of when the gradient will change and better estimations for where the minimum value is (Zeiler, 2012). An algorithm that combines these two, known as Adaptive Movement Estimation (Adam) allows for more efficient gradient descent. Adam will descend down the gradient relatively quickly while also updating the learning rates. Due to this combination, Adam is widely considered one of the fastest and most effective gradient descent algorithms. (Ruder, 2017)

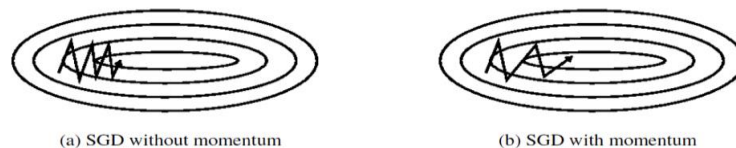


Figure 1: A Depiction of a SGD Algorithm (a) With and (b) Without Momentum (Ruder, 2017)

Neurons in neural networks are structured into hidden layers within the network. These hidden layers are 3-dimensional tensor structures, data structures with dimensions corresponding to width, height, and depth. The image used as an input is fed into the neural network as a 3-dimensional tensor with depth corresponding to 3 color channels and height and width corresponding to the height and width of the input image. (Krizhevsky, 2012) As the network progresses towards the output, the depth of the layer tensors increase, while the values for the height and width decrease. A value of 1 for height or width indicates that the feature space has been exhausted, as the height and width have been minimized, maximizing the depth of the layer. (Simonyan, 2015) As the layer depth increases, the tensor can store more specific information. A higher level (lower depth) layer stores general structural information, while a lower level (higher depth) layer stores detailed information about the feature space.

Convolutional Neural Networks (CNNs) are networks in which there are many hidden layers which can process an image spatially by retaining the original relative positioning of the image pixels. CNNs are vastly more accurate than Feed-Forward Neural Networks, which flatten the image into a single dimensional array of pixels. This spatial awareness of CNNs make them a very popular option for networks which have to work with processing and/or classifying images. (Ding, 2016) The VGG19

Network is a well-known CNN which has been trained to classify images using the ImageNet dataset, which is a very large dataset of images tagged with classes. Figure 2 shows the VGG19 network architecture, with higher level layers on the right and lower level layers to the left. (Yosinski, 2015)

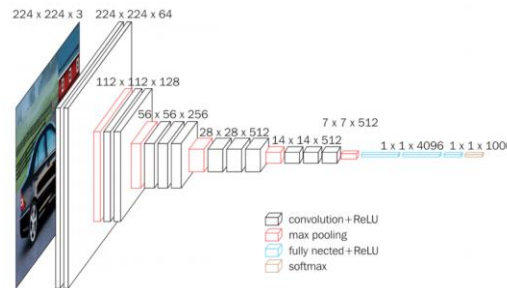


Figure 2: VGG19 Network Architecture (Simonyan, 2015)

In 2015, Gatys et al. used a Convolutional Neural Network with VGG architecture in the extraction of features from images. In manipulating the network, he trained the neural network on a content image and a style image, then extracted lower level layers corresponding to the style image as a whole, along with higher level layers corresponding to the textural features of the style image. As a style image, Gatys decided to use *Composition VII* by Wassily Kandinsky, while choosing an original photograph of the Neckarfront in Tübingen, Germany for the content image, showing that the weighting of the content and style can be weighted to alter the output to more closely mimic either the content or style.

Nikulin et al (2016) expanded on the work of Gatys in the field of neural style transfer. Nikulin was able to show that the GoogLeNet architecture can also be used for style extraction procedures, just like the VGG19 architecture used by Gatys. The GoogLeNet architecture is similar to the architecture used in the Inception network, a deep network containing many layers which “pool” into a single layer, whereas the VGG architecture simply links hidden layers to each other. (Szegedy, 2015) In expanding the usage of style transfer to create false images, Nikulin explores season and illumination transfer, showing that, although Gatys’s method of style transfer only causes the style to get “painted over” the content, if the transfer process is made “content aware”, style transfer can be used to modify the season or illumination patterns within a picture with moderate success.

Ding et al. (2016) worked with Convolutional Neural Networks (CNNs), outlining how such networks learn and defining methods of data augmentation to improve the classification efficacy for Synthetic Aperture Radar (SAR) target recognition. Ding applied noise to the images and rotated or translated them, showing that the modifications improved the classifications of the CNN on the testing data. This shows that image augmentation can improve the efficacy of a CNN in image identification.

Purpose & Hypothesis

The purpose of this project is to investigate the feasibility of using VGG19-based Neural Style Transfer as an augmentation method to produce more training data for Neural Image Classifiers. This new image data augmentation mechanism will be compared to more standard methods, assessing the efficacy of such a method. The alternate hypothesis is that convolutional neural networks will be able to classify images better when given extra test cases with either original or stylistically augmented data, and that a higher content to style weight ratios will allow for greater prediction accuracy, as it allows for essential structural patterns to remain present in the modified image.

Methodology

This project tested if the preprocessing of images with Neural Style Transfer improves the classification efficacy of an Image Classifier CNN. In testing this, images of Scorpions, Crabs, and Butterflies were processed with content to style ratios of 10^9 , 10^8 , and 10^7 to allow for effective style transfer while maintaining the general structure of the content image. These images were then used to train convolutional neural networks, with training sets of 10 original images per category and 80 original images for testing. These images were taken from the Caltech-101 image classification dataset. Eight neural nets were trained. As a control, the 10 images were used to train a network on their own. For the three experimental groups, the training sets were modified to include the repainted images, with each experimental group having a different content to style ratio. These pairings were all trained both with and without translational and rotational augmentation. Three different subsets of 10 images were selected from the 90 images for different training networks, generating three data points for each individual training setup.

I. General Setup

All of the files for the neural style process were written in Python and run through Google CoLaboratory (CoLab) with Graphics Processing Unit (GPU) acceleration. The CoLab environment was chosen as it has the TensorFlow and Keras libraries built in, which are essential to the creation and usage of neural networks. CoLab also has free GPU acceleration, which allows for the neural networks to be trained quickly at no cost. In each file, TensorFlow and Keras will both be imported, along with Matplotlib and the Python Imaging Library (PIL) for visualizing image data.

In training the CNNs with and without style transfer augmentation, Apple's CreateML software was chosen to build the models. Released alongside macOS Catalina in early October 2019, CreateML provides an interface which can not only intuitively train Image Classifier CNNs, but can also augment data automatically, performing popular augmentations which provide larger training datasets. In the implementation of the CNNs within this project, the translational and rotational augmentations were enabled for half of the nets, with the other half being trained without augmentation.

II. Implementation of Style Transfer

A. Setup

The VGG19 image classifier is used in transferring stylistic elements onto the content image. This classifier was trained to recognize images on the ImageNet dataset, which is a large dataset of tagged images, and was then published within the TensorFlow Keras library to be used for other projects. In this implementation, VGG19 is trained using the style and content images, allowing the stylistic elements to be removed from the network and applied to the content image.

To allow for the network to run properly, Eager Execution was enabled in TensorFlow. This allowed for the commands to be processed as they were called rather than stored in the memory of the program, enabling the algorithms to run faster, utilizing the GPU processor provided by CoLab.

B. Preparation of Images

30 Images are loaded from the Crab, Butterfly, and Scorpion classes contained in the Caltech-101 dataset as content images for augmentation, while Wassily Kandinsky's *Composition VII* is loaded as the style image, which is shown in Figure 3 to the left. All of the images are extended to include a "batch" dimension, which serves to store more specific information as the information passes through the neural network. Also, to view the results, the images must be processed after passing through the neural network, removing the mean weights applied by the classification, which correspond to the training dataset. The mean weights for the ImageNet dataset are [103.939,116.779,123.68], with values corresponding to the three color channels of the image. By removing the ImageNet weights, the style transfer takes on a form which is meaningful to the observer rather than to the neural network.



Figure 3: *Composition VII*
By Wassily Kandinsky

C. Building the Model

The VGG19 model has to be built within the program before it can be used within Keras. The model is loaded in with pretrained ImageNet weights, with the softmax "top" being excluded, as the images are not passing through to the end of the network and are instead being inspected within the hidden layers. The network is declared as untrainable to prevent the ImageNet weights from being altered by the feeding of the content and style images into the network.

To apply the style transfer, layers of the VGG19 network had to be extracted which corresponded to the content and style feature spaces. In choosing a content layer, one was chosen which takes more of the structural features present in the content, which was layer 'block5_conv2'. In choosing layers for the style representation, higher level layers were chosen to represent the texture of the style image. These layers were 'block1_conv1', 'block2_conv1', 'block3_conv1', 'block4_conv1', and 'block5_conv1'. The

style and content layers are all extracted during the style transfer process, and are applied to the content image to create the stylistically modified final image.

D. Loss Computation

Loss functions must be created for the algorithm to define how close the style and content of the image are to the optimal state. Calculating the content loss is rather straightforward, requiring only the Mean-Square Error (MSE) between the outputs of intermediate layers. Given a pre-trained convolutional neural network (in this case VGG19) denoted as C_{nn} and an image X , then $C_{nn}(X)$ is the network fed by image X . Further, let $F_{ij}^l \in C_{nn}(x)$, and $P_{ij}^l \in C_{nn}(p)$ define the intermediate feature representations of the neural network with inputs x and p at layer l . Given this information, we can define the content as shown in Equation 2.

$$L_{content}^l(p, x) = \sum_{i,j} (F_{ij}^l(x) - P_{ij}^l(p))^2 \quad (\text{Equation 2})$$

The style loss serves a similar purpose to the content loss, but is calculated very differently. The stylistic elements are taken out of the network using Gram Matrices. Gram Matrix G_{ij}^l denotes the correlation between feature maps i and j . To calculate the total style loss, first the style loss for a single layer must be defined. Given that G_{ij}^l and A_{ij}^l are the respective style representations in layer l of x and a . Given that N_l describes the number of feature maps of size $M_l = \text{height} * \text{width}$, the single-layer style loss can be defined as shown in Equation 3.

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - A_{ij}^l)^2 \quad (\text{Equation 3})$$

Once the loss is defined for individual layers, it is somewhat trivial to calculate the network loss, as it is the sum of all layers within the network. This is shown in Equation 4 below. The factor w_l regulates the contribution of the layer's loss to the style loss as a whole. In this case, all layers have an equal weighting, where $w_l = \frac{1}{|L|}$.

$$L_{style}(a, x) = \sum_{l \in L} w_l E_l \quad (\text{Equation 4})$$

To find the function loss, the style and content losses are summed together, with content weight α and style weight β as shown in Equation 5.

$$L_{total}(p, a, x) = \alpha L_{content}(p, x) + \beta L_{style}(a, x) \quad (\text{Equation 5})$$

This total loss is what is minimized in gradient descent. As the α to β (content to style weight) ratio increases, the content will become more represented in the final stylized image, and vice versa.

E. Loss Minimization

In transferring the style onto the content image, both the style and content losses must be minimized. This was done through performing gradient descent with the Adam optimizer. The TensorFlow GradientTape function was used to automatically calculate the gradients for the Adam optimizer, allowing for gradient descent to take place more efficiently. For each image, this gradient descent method was run for 1000 cycles, bringing the image closer to the desired content to style ratio with each cycle. Once the loss was minimized, the modified image was processed and saved to a folder for usage in image classifier training.

III. Creation of Image Classifiers

A. CreateML

Apple's CreateML software was used to train image classifiers which used the modified images in training. Three projects were created, each containing 8 individual neural networks. These networks corresponded to no style transfer, medium style transfer (content : style ratio = 10^8), heavy style transfer (content : style ratio = 10^7), and light style transfer (content : style ratio = 10^9). For each of these groups, one network was trained without any other augmentation, while another was trained with static augmentation alongside the pre-existing neural style augmentation. These networks each contained 10 images from the Caltech-101 dataset alongside their 10 stylized counterparts (except groups without style transfer, which only contained the 10 images from the Caltech dataset) for each class. The other 70 images contained in the dataset for each class were used in the testing of the neural networks.

B. Image Augmentation

Image augmentation was relatively simple to implement within CoreML. When additional augmentation was applied, it was applied as both a translational shift and a rotation. This simplifies the process of adding additional processing, as it can be done within CreateML, allowing augmentation to be performed within the model as it builds, rather than within a separate Python script.

IV. Data Collection

Data was collected within the CoreML system. Once the testing dataset ran, the software provided 6 numerical values, describing the precision and recall of the network with respect to each of the 3 classes. The precision is a measure of what proportion of predicted positives were true positives, while the recall is a measure of what proportion of the actual positives were identified correctly. Equations 6 and 7 respectively show the formula to calculate the precision and recall, where TP stands for True Positives, FN stands for False Negatives, and FP stands for False Positives.

$$\text{Precision} = \frac{TP}{TP + FP} \quad (\text{Equation 6})$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad (\text{Equation 7})$$

A higher recall value typically signifies that the network provides fewer false negatives, while a higher precision shows that the classification is unlikely to give false positives. In most cases, a higher recall value is more desirable than a higher precision, since a prediction made with higher recall is unlikely to miss true positive cases. As an example, in medical diagnoses, a false positive is significantly better than a false negative, which can result in a disease being left untreated, and further incentivises a higher recall value

V. Data Analysis

Data was analyzed qualitatively to show trends in the precision and recall values of the neural network. As the training sets used were different, means cannot be compared, leaving qualitative analysis as the best option for interpreting the data.

Results

Table 1 shows the precision values for each class within each of the neural networks. Denoted by 1, 2, and 3 next to the class name are the project identifiers, which show which data points were bundled together. Each neural network gives out three precision values at once, as one value is given to each class. All classes with project ID 1 get their values together, and the same is true for IDs of 2 and 3.

Table 3 shows the recall values for each network in the same format as the precision values in Table 1. The recall data was given alongside the precision data for each network, meaning that the networks labeled in Table 3 are the exact same networks labeled in Table 1.

Table 2 shows the overall accuracy of the groups as a whole. This is based on both the recall and precision values. The higher the recall and precision, the higher the corresponding overall accuracy is. This accuracy does not give information about the individual classes themselves, but can show how the network is functioning as a whole, including data from all classes.

Class	No Styling		Medium Styling		Heavy Styling		Less Styling	
Precision	Without Augmentation	With Augmentation	Without Augmentation	With Augmentation	Without Augmentation	With Augmentation	Without Augmentation	With Augmentation
Butterfly 1	99%	96%	95%	92%	95%	88%	99%	95%
Butterfly 2	96%	95%	100%	92%	99%	91%	98%	93%
Butterfly 3	98%	100%	95%	99%	100%	95%	98%	100%
Crab 1	83%	88%	91%	89%	88%	71%	88%	92%
Crab 2	98%	98%	90%	98%	92%	90%	95%	98%
Crab 3	100%	94%	100%	100%	93%	100%	100%	98%
Scorpion 1	97%	97%	80%	100%	85%	89%	86%	97%
Scorpion 2	80%	96%	91%	96%	97%	97%	95%	96%
Scorpion 3	80%	96%	87%	95%	77%	79%	92%	89%

Table 1: Precision Percentages for Three Neural Networks Each Trained on Three Classes With All Eight Different Augmentation Combinations (Table Created by Competition Entrant)

Class	No Styling		Medium Styling		Heavy Styling		Less Styling	
Overall Accuracy	Without Augmentation	With Augmentation	Without Augmentation	With Augmentation	Without Augmentation	With Augmentation	Without Augmentation	With Augmentation
Group 1	93%	94%	88%	94%	89%	83%	91%	95%
Group 2	90%	96%	94%	95%	96%	93%	96%	95%
Group 3	91%	97%	93%	98%	88%	89%	96%	95%

Table 2: Overall Accuracy Percentages for Each Trained Convolutional Neural Network (Table Created by Competition Entrant)

Class	No Styling		Medium Styling		Heavy Styling		Less Styling	
	Without Augmentation	With Augmentation	Without Augmentation	With Augmentation	Without Augmentation	With Augmentation	Without Augmentation	With Augmentation
Butterfly 1	89%	99%	95%	99%	94%	100%	91%	100%
Butterfly 2	98%	100%	92%	99%	94%	98%	99%	100%
Butterfly 3	99%	95%	100%	100%	92%	100%	100%	99%
Crab 1	97%	92%	65%	90%	73%	73%	81%	90%
Crab 2	68%	90%	89%	84%	95%	85%	90%	87%
Crab 3	69%	95%	76%	92%	69%	61%	87%	85%
Scorpion 1	95%	91%	100%	91%	99%	74%	99%	93%
Scorpion 2	100%	97%	100%	100%	100%	93%	97%	97%
Scorpion 3	100%	100%	100%	100%	100%	100%	100%	100%

Table 3: Recall Percentages for Three Neural Networks Each Trained on Three Classes With All Eight Different Augmentation Combinations (Table Created by Competition Entrant)

Discussion

The purpose of this project was to test the impact of stylizing training data with neural style transfer on the efficacy on a CNN trained to classify images. The recall and precision were both assessed in determining classification efficacy. Overall, the efficacy did increase as the images were augmented, supporting the alternate hypothesis, however the efficacy actually seemed to be highest with a more moderate stylization, rather than with the initially predicted heavier stylization.

Overall, the precision is relatively high for all classes, with most values being 85% and above. A major exception to this is Scorpion 3 with Heavy Styling, which could be attributed to structural features being blurred by the heavy augmentation, making the network more unsure of what a “Scorpion” is and thus more likely to return a false positive, lowering the precision. Also, in almost all cases, the precision increases from no styling to medium styling, indicating that the precision is boosted by the application of a more moderate content to style ratio. However, heavy styling almost always decreased precision, which could be attributed to the network becoming more uncertain, and thus more likely to provide false

positives. Scorpions 2 and 3 are interesting, as their lowest precision is without the augmentation, at around 80%, but once style transfer is added, it rises to around 95% depending on the intensity of the transfer. For these samples, in each case the static augmentation increased the percentage after style transfer was applied, showcasing that these modifications can be overlaid on the original images to produce a much better training set for CNNs.

The recall values are much harder to interpret than the precision values. While the precision values were typically in the mid 80% to high 90% range, the recall values have a much larger variance, with many values either dropping to around 60/70% or capping off at 100%. When recall value for all the networks gets extremely close to 100% for any one class, the changes in recall become negligible, and thus the data is indeterminate, because in this case augmentation would not have been used to raise the recall in the first place. This happens for Butterfly 3, Scorpion 2, and Scorpion 3, all obtaining a recall of 100% for at least half of the networks which were trained using their datasets. However, Crab 2 and Crab 3 had dramatic boosts in recall when style transfer was applied, increasing from about 70% without the transfer to about 90% for Crab 3, and to around 80% for Crab 2. However, for both of these classes, the static augmentation without any neural style transfer did as well as or better than most of the stylistically augmented data. This suggests that recall is not necessarily benefited by the neural style transfer process.

The precision value being higher than the recall values is significant in that it shows how the network's decisions are impacted by altered images. As the precision increases, the number of false positives will decrease, while a mostly constant recall indicates the number of false negatives remains close to the same value. This is useful for applications in which false positives are avoided, while the static augmentation seems to be better for preventing false negatives.

Due to the recall falling or staying constant as the precision rises, the overall accuracy of the networks mostly increased when stylized, but not by much. The only network whose efficiency did not rise with stylization was Group 1. This could be due to a sudden drop in the recall for Crab 1 from 97% without augmentation to 65% when stylized moderately, and 73% when heavily stylized. This sudden drop could shift the overall accuracy downwards so much that the network appears to be doing much worse, when in actuality only the crab dataset is doing badly. Thus, overall, it seemed that the neural style transfer process did increase the classification efficacy, even if only slightly for some image sets.

All of the images included in the dataset were stylized using *Composition VII* by Wassily Kandinsky, as many studies, including Gatys et al (2015), used this painting in stylization. Table 4 shows one image from each dataset stylized using *Composition VII*.













	Butterfly	Crab	Scorpion
Original			
Light Stylization			
Moderate Stylization			
Heavy Stylization			

Table 4: The Application of the Style of *Composition VII* at Different Content to Style Weight Ratios to Different Images within the Caltech-101 Dataset. (Figure Created by Competition Entrant)

Although *Composition VII* is used by Gatys and various others, there are other famous images, such as *Starry Night* by Vincent Van Gogh, which are just as, if not more famous, and also have a distinct style that can be transferred onto other pictures. To illustrate this, Figure 4 shows the application of *Starry Night* to an image of a Corgi with a snapshot taken after every 100 cycles of gradient descent. For comparison, Figure 5 outlines the same process, but with *Composition VII* as the style image. In each of these images, there is a similar focusing of the image around the dog rather than on top of it as an overlay, allowing the content of the dog itself to be preserved. Table 5 shows the initial content and style images, along with the final result of the style transfer for both of these images.



Figure 4: Applying Style Transfer from *Starry Night* to an Image of a Corgi, With Snapshots Taken After Every 100 Cycles of Gradient Descent (Figure Created by Competition Entrant)

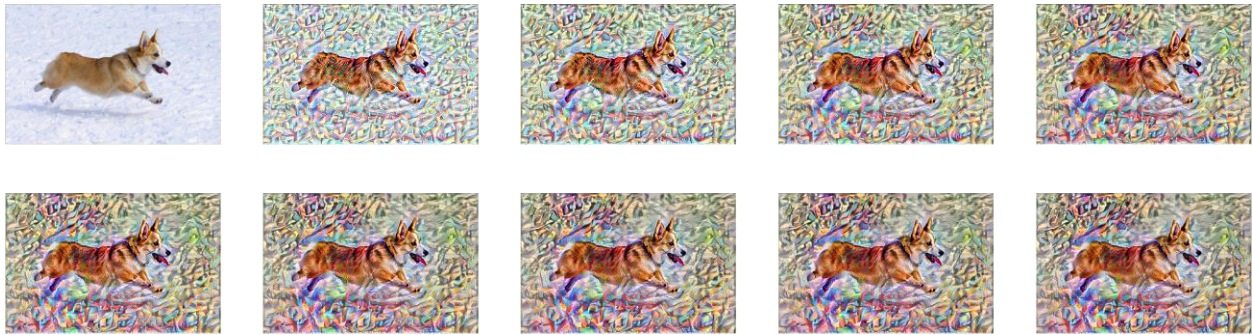


Figure 5: Applying Style Transfer from *Composition VII* to an Image of a Corgi, With Snapshots Taken After Every 100 Cycles of Gradient Descent (Figure Created by Competition Entrant)






	Corgi	Starry Night	Composition VII
Original			
Stylized			

Table 5: Original Corgi Content Image Along with Style Images and Stylized Corgi Images (Table Created by Competition Entrant)

Looking at the style images alongside the stylized images, there seems to be a very large difference in appearance, possibly indicating a different method by which the style was applied. However, to see this conclusively, a map of the vertical and horizontal deltas must be developed. This map will show the texture of the image, and where there are areas of changes in texture while moving vertically and while moving horizontally. As shown in Table 6, when the deltas are taken for the original picture and the two stylized images, there is a residue of the content image in both style images, whose delta plots look exactly the same. This similarity is explained by the process of gradient descent. Due to the Adam optimizer and a constant content to style ratio being used for both *Starry Night* and *Composition VII*, the loss is defined as the same function, and is thus minimized using the same traversal. This traversal is represented in the deltas, so there is effectively no difference in the texture application between two different style images when applied to the same image at a constant content to style ratio.


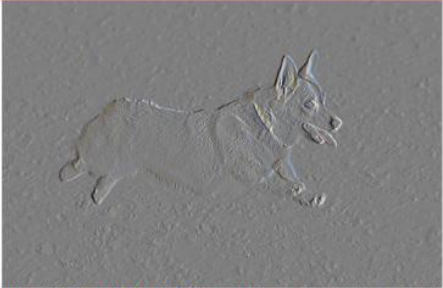
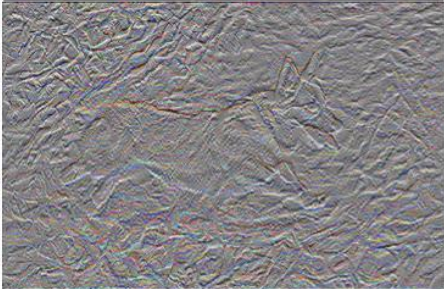
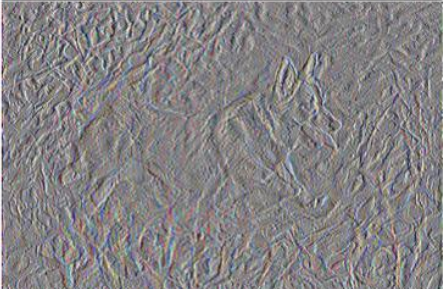
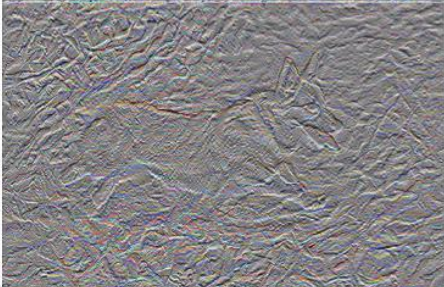
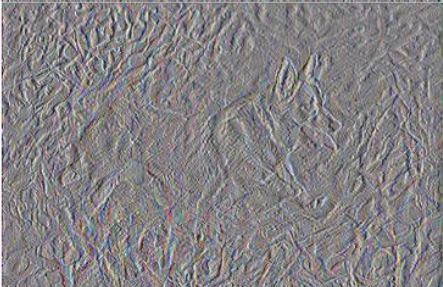
	Horizontal Deltas	Vertical Deltas
Original		
Composition VII		
Starry Night		

Table 6: The Horizontal and Vertical Deltas of a Corgi When Unstyled, Stylized With *Composition VII*, and When Stylized With *Starry Night* (Table Created by Competition Entrant)

Because the deltas are consistent based on loss function rather than on the content image, the style transfer process extends to other content images, as long as the content to style ratio is kept consistent and the Adam optimizer is used in gradient descent. Thus, any style image can be applied to content images regardless of the content itself or the actual content of the style image.

One major limitation in this project was the shortage of images used in classification models. While the data used was sufficient, the changes in recall and precision would have been more pronounced had there been a larger sample of data. A larger image set would have allowed for more accurate data, and fewer results in which the recall or precision were either too close to 100% or too erratic to be meaningful. However, the current sample size of 10 images and test set of 80 images did provide a large amount of meaningful and significant data within the context of this study.

Conclusion

As shown in the collected precision and recall data, the process of neural style transfer can be used as a form of data augmentation to enlarge data sets and combat overfitting in classification networks. This has implementations in medical data sets, where certain types of cells must be identified relatively accurately, but there is a shortage of tagged images with which to train a classifier network on, resulting in unreliable classifiers. Data augmentation akin to the processes outlined in this paper can be applied in scenarios such as these to boost the efficacy of classifier networks.

Future Studies

A possible future project could be to test how the deltas themselves impact the classification efficacy. As this project only focused on changing the content image, there was not much focus on the stylistic portion of the process. A focus on the stylistic aspects of style transfer could characterize the roles of the horizontal and vertical deltas in the process of classification, highlighting which features the neural classifier picks up on when trained.

Another possibility for a future study could be to use different CNNs in classification, or a different network architecture altogether. The Inception network, described by Szegedy et al (2015), is a possible CNN that could be implemented, while a CycleGAN could be trained on larger image datasets, as done by Zhu et al (2018). The comparison of these network architectures could display how the different architectures function as they all work on a common task.

Bibliography

- Asokan, Raghu. "Neural Networks Intuitions: 2. Dot Product, Gram Matrix and Neural Style Transfer." Medium, Towards Data Science, 21 June 2019, towardsdatascience.com/neural-networks-intuitions-2-dot-product-gram-matrix-and-neural-style-transfer-5d39653e7916.
- Ding, Jun, et al. "Convolutional Neural Network With Data Augmentation for SAR Target Recognition." IEEE Geoscience and Remote Sensing Letters, 26 Jan. 2016, pp. 1–5., doi:10.1109/lgrs.2015.2513754.
- Fei-Fei, Li, et al. "Learning Generative Visual Models from Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories." 2004 Conference on Computer Vision and Pattern Recognition Workshop, 2004, doi:10.1109/cvpr.2004.383.
- Fischer, Michael, et al. "ImagineNet: Style Transfer from Fine Art to Graphical User Interfaces." 2018.
- Gatys, Leon, et al. "A Neural Algorithm of Artistic Style." 2 Sept. 2015.
- Jackson, Philip T., et al. "Style Augmentation: Data Augmentation via Style Randomization." 12 Apr. 2019.
- Karras, Tero, et al. "A Style Based Generator Architecture for Generative Adversarial Networks." 29 Mar. 2019.
- Krizhevsky, Alex, et al. "ImageNet Classification with Deep Convolutional Neural Networks." Communications of the ACM, vol. 60, no. 6, 2017, pp. 84–90., doi:10.1145/3065386.
- Nikulin, Yaroslav, and Roman Novak. "Exploring the Neural Algorithm of Artistic Style." 13 Mar. 2016.
- Perez, Luis, and Jason Wang. "The Effectiveness of Data Augmentation in Image Classification Using Deep Learning." 13 Dec. 2017.
- Ruder, Sebastian. "An Overview Of Gradient Descent Optimization Algorithms." 15 June 2017.
- Simonyan, Karen, and Andrew Zisserman. "Very Deep Convolutional Networks For Large-Scale Image Recognition." 10 Apr. 2015.
- Szegedy, Christian, et al. "Going Deeper with Convolutions." 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, doi:10.1109/cvpr.2015.7298594.
- Ulyanov, Dmitry, et al. "Improved Texture Networks: Maximizing Quality and Diversity in Feed-Forward Stylization and Texture Synthesis." 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 10 Mar. 2017, doi:10.1109/cvpr.2017.437.
- Yosinski, Jason, et al. "Understanding Neural Networks Through Deep Visualization." 22 June 2015.
- Zeiler, Matthew D. "Adadelta: An Adaptive Learning Rate Method." 22 Dec. 2012.
- Zhu, Jun-Yan, et al. "Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks." 15 Nov. 2018.

“Classification: Precision and Recall & Machine Learning Crash Course.” Google, Google, developers.google.com/machine-learning/crash-course/classification/precision-and-recall.