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

Research Plan

Kevin Gauld

Category: Robotics and Intelligent Machines

A. Rationale

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)

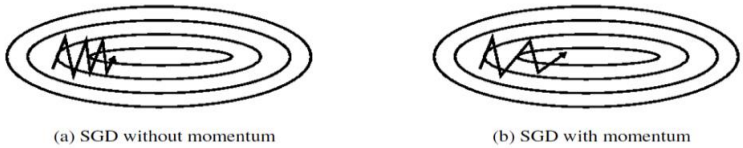

Figure 1: A Depiction of a SGD Algorithm (a)With and (b)Without Momentum (Ruder, 2017)

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 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)

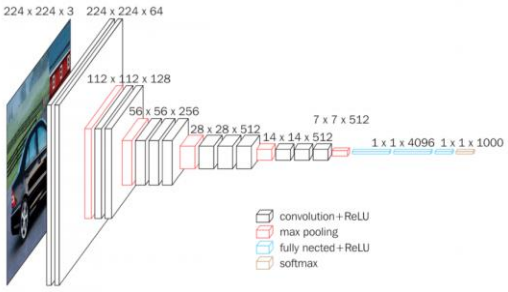

Figure 2: VGG19 Network Architecture (Simonyan, 2015)

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.

B. Research Questions, Hypothesis(es), Engineering Goals, Expected Outcomes

a. Research Question

Can Neural Style Transfer be used to improve the classification efficacy of Convolutional Image Classifier Networks?

b. Hypothesis

Alternate Hypothesis: The classification efficacy of the Convolutional Neural Networks will be improved when style transfer is applied, showing that Neural Style Transfer is an effective form of image augmentation.

Null Hypothesis: The classification efficacy of the Convolutional Neural Networks will not improve when style transfer is applied

c. Engineering Goals *Not Applicable*

d. Expected Outcomes

Based on the work of Ding et al (2016) in increasing the efficacy of Image Classifiers, the Neural Style Transfer process should work as a method by which the efficacy of a Convolutional Image Classifier can be increased. This is because, in creating more unique data for training, random variance is minimized as a deterministic factor in the model creation, minimizing overfitting and improving the classification efficacy.

C. Procedures

This project will test 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 will be 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 will then be used to train convolutional neural networks, with training sets of 10 original images per category and 80 original images for testing. These images will be taken from the Caltech-101 image classification dataset. Eight neural nets were trained. As a control, the 10 images will be used to train a network on their own. For the three experimental groups, the training sets will be modified to include the repainted images, with each experimental group having a different content to style ratio. These pairings will all be trained both with and without translational and rotational augmentation. Three different subsets of 10 images will be 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 will be written in Python and run through Google CoLaboratory (CoLab) with Graphics Processing Unit (GPU) acceleration. The CoLab environment has been 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.

A Python program will be used to train the CNNs both with and without style transfer. Along with style transfer, this program will be made such that it can produce static translational and rotational augmentations using the Python Imaging Library. This program will create a model of a classifier which can then be analyzed for classification efficacy.

II. Implementation of Style Transfer

A. Setup

The VGG19 image classifier will be used in transferring stylistic elements onto the content image. This classifier was initially 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 will be 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 will be enabled in TensorFlow. This will allow 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

Figure 3: *Composition VII* By Wassily Kandinsky

30 Images will be 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* will be loaded as the style image, which is shown in Figure 3 to the left. All of the images will be 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 will 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.

C. Building the Model

The VGG19 model has to be built within the program before it can be used within Keras. The model will be loaded with pretrained ImageNet weights, with the softmax "top" being excluded, as the images will not be passing through to the end of the network and are instead being inspected within the

hidden layers. The network will be 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 which corresponded to the content and style feature spaces will be extracted. In choosing a content layer, one will be chosen which takes more of the structural features present in the content, which will be layer 'block5_conv2'. In choosing layers for the style representation, higher level layers will be chosen to represent the texture of the style image. These layers will be 'block1_conv1', 'block2_conv1', 'block3_conv1', 'block4_conv1', and 'block5_conv1'. The style and content layers will all be extracted during the style transfer process, and will be applied to the content image to create the stylistically modified final image.

D. Loss Computation

Loss functions will 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 will require 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 $\square^{\square}_{\square\square} \in \square_{\square\square}(\square)$, and $\square^{\square}_{\square\square} \in \square_{\square\square}(\square)$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.

$$L^{l}_{content}(p, x) = \sum_{i,j}(F^{l}_{ij}(x) - P^{l}_{ij}(p))^2$$

(Equation 2)

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

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j}(G^{l}_{ij} - A^{l}_{ij})^2$$

(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 sloss to the style loss as a whole. In this case, all layers have an equal weighting, where $\square_{\square} = \frac{1}{|\square|}$.

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

(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.

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

(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 will be minimized. This will be done through performing gradient descent with the Adam optimizer. The TensorFlow GradientTape function will be 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 will be run for 1000 cycles, bringing the image closer to the desired content to style ratio with each cycle. Once the loss is minimized, the modified image will be processed and saved to a folder for usage in image classifier training.

III. Creation of Image Classifiers

A. Image Classifiers

A Python program will be written to train image classifiers which will use the modified images in training. Three programs will be 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 will be trained without any other augmentation, while another will be trained with static augmentation alongside the pre-existing neural style augmentation. These networks will each contained 10 images from the Caltech-101 dataset alongside their 10 stylized counterparts (except groups without style transfer, which will only contain the 10 images from the Caltech dataset) for each class. The other 70 images contained in the dataset for each class will be used in the testing of the neural networks.

B. Image Augmentation

Python code will be written to apply image augmentation to the images that will be used to train the classifier. When additional augmentation will be applied, it will be applied as both a translational shift and a rotation onto the image used to initially train the network. This allows for a standard augmentation method to be applied in all trials, allowing for data to remain consistent between test cases.

IV. Data Collection

Data will be collected from the Python image classifier training output. Once the testing dataset is run, the software will provide 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.

- **Risk and Safety-**

1. Human participants research: *Not applicable*
2. Vertebrate Animal research: *Not applicable*
3. Potentially Hazardous Biological Agents: *Not applicable*
4. Hazardous Chemicals, Activities and Devices: *Not applicable*

1.

- **Data Analysis**

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

D. Bibliography

Asokan, Raghul. “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.

Project Summary:

C. III. Creation of Image Classifiers

A. Image Classifiers

~~A Python program will be written to train image classifiers which will use the modified images in training. Three programs will be created, each containing 8 individual neural networks.~~ ***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, light style transfer (content : style ratio = 10^9), medium style transfer (content : style ratio = 10^8), and heavy style transfer (content : style ratio = 10^7). For each of these groups, one network will be trained without any other augmentation, while another will be trained with static augmentation alongside the pre-existing neural style augmentation. These networks will each contained 10 images from the Caltech-101 dataset alongside their 10 stylized counterparts (except groups without style transfer, which will only contain the 10 images from the Caltech dataset) for each class. The other 70 images contained in the dataset for each class will be used in the testing of the neural networks.

B. Image Augmentation

~~Python code will be written to apply image augmentation to the images that will be used to train the classifier. When additional augmentation will be applied, it will be applied as both a translational shift and a rotation onto the image used to initially train the network. This allows for a standard augmentation method to be applied in all trials, allowing for data to remain consistent between test cases.~~

Image augmentation was implemented 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.