Customized Fast Style Transfer & Website

Eric Layer, Ying Wang

Figure 1. Style Transfer picture of Chicago.

Abstract—This project consists of incorporating an existing machine learning model and algorithm, namely fast style transfer, and creating a website which allows users to upload any image of their choice and convert that image into a brand new image with a specific style mixed in. The algorithm used for this project is based on Convolutional Neural Networks, which is a class of Deep Neural Networks that work very well for image processing tasks. The components of the project include python files, which contains the code for the neural network algorithm used here to process images (and of course the programming language used). It also required TensorFlow, which is an open source machine learning library we used for our neural network application. We then used Flask as our web framework for our website implementation, payment system, and SQL database. Throughout this paper, we will discuss the in detail the algorithms and methods, including deep neural networks, convolutional networks, and how these work to make fast style transfer possible. We then will describe and illustrate our experimentation with fast style transfer and the challenges we faced along the way. Finally, we will conclude with our results and what we learned throughout the process of this project. We discovered just how efficient this algorithm is and how it could be applied in the setting we constructed (our website). We hope that after reading this paper, a general understanding behind convolutional neural networks and style transfers can be achieved, and how we were able to take this machine learning model and create an easy-to-use website for users to quickly style any image they wish.

1 Introduction

To begin, we will discuss the background behind content and style of images that led to the rise of how visual perception can be learned and trained through a machine algorithm. Throughout time, humans have honed the skill to create unique visual experiences through complex artwork consisting of content and style of images. In areas of object and face recognition, near-human performance has been demonstrated by a class of biologically inspired vision models called Deep Neural Networks. The artificial system introduced is based on a Deep Neural Network that has the ability to create high-quality artistic images. The system uses "*neural representations to separate and recombine content and style of arbitrary images*"[1]. The focus of this work has offered an effective way towards an algorithmic understanding of how humans create and perceive artistic imagery.

Within Style Transfer, many classic problems can be labelled as image transformation tasks, where a system receives an input image and transforms it into a new output image. An approach for solving these image transformation tasks is to train and model a feed-forward convolutional neural network in a "*supervised manner, using a per-pixel loss function to measure the difference between the output and ground-truth images*"[2].

The class of Deep Neural Networks that we focused on through this project and will focus on primarily in this paper are Convolutional Neural Networks. This class is the most powerful in image processing tasks. These networks consist of layers of small computational units that can process visual information in a feed-forward manner hierarchically. Each layer of units can be understood as a collection of image filters,where each filter extracts certain features from input

images. Therefore, the output of a given layer consists of what are called feature maps, which are differently filtered versions of the input image.

We applied this algorithmic approach in our project from a pre-trained neural network model acquired from GitHub. We then learned from some examples of Flask implementations to create our website which we used for style transformations of images. We essentially are creating a way for users to sign-up into our website, submit a payment to transfer any image they input, and provide the user with a style-transferred image for which they can download to their PC. We feel this is a new avenue of being able to re-create anything from artistic images to family photographs, and transform the picture into something new and unique. We also think that this is a service that is not readily available on the internet or for the public, and with the speed and efficiency with which this algorithm creates new images, this could be a successful way of implementing this algorithm for, as an example, a start-up business.

2 Related/Prior Work Analysis (Neural Algorithm)

We will first discuss the paper of the Neural Algorithm of Artistic Style, and then discuss our source pertaining to the Perceptual Losses for Real-Time Style Transfer and Super-Resolution.

To begin and refer back to the point of Convolutional Neural Networks, these networks are trained on object recognition, and they can develop a representation of the image that makes object recognition increasingly explicit following the process hierarchy. Along this processing hierarchy of the network, the input image is transformed into representations that increasingly care about the content of the image compared to its detailed pixel values.

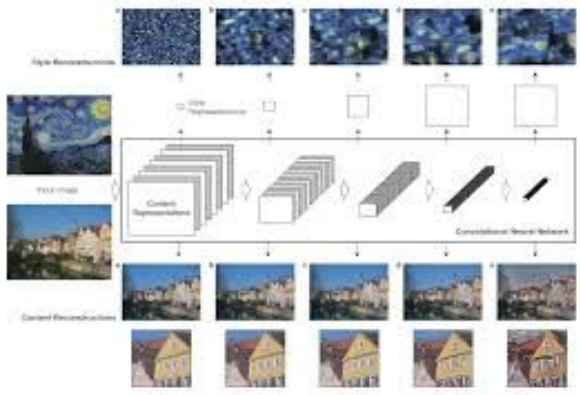


Figure 2. Process hierarchy of the Convolutional Neural Network.

The information that each layer contains can be directly visualized about the input image by reconstructing the image only from the feature maps within that particular layer. In reference to figure 2 above, the higher layers in the network capture the high-level content in terms of the objects and their arrangement in the input image, but do not constrain to exact pixel values when reconstructing. As illustrated, the last two images of content reconstruction have not retained precise pixel representation. This is in contrast to the first three images, where reconstruction from the lower layers simply reproduce the exact same pixel values of the original image. Judging by these terms, the feature responses in higher layers of the network are deemed as the content representation.

In order to obtain a representation of the style of an input image, a feature space is used that is originally designed to capture texture information. This feature space is built on top of filter responses within each of the layers of the network. It consists of "*the correlations between the different filter responses over the spatial extent of the feature maps*"[1]. Thus, by including the feature correlations of multiple layers, we can obtain a stationary and multi-scale representation of the input image, which captures its texture information but not global arrangement.

Referring now to style reconstruction, we can visualize the information captured by the style feature spaces built on different layers of the network by constructing an image that matches the style representation of any given image. Reconstructions from style features produce texturized versions of input images that capture their general appearance in terms of color and localized structures. In addition, size and complexity of local image structures from the input image increases along the hierarchy, which is a result explained by the increasing receptive field sizes and feature complexity. This multi-scale representation can therefore be referred to as style representation. In figure 2, it can be illustrated how along the processing hierarchy, the style is maintained but the contents are transformed, resulting in a new image generated at the end of the hierarchy as seen in the last image at the top right of figure 2.

The primary finding here though is that representations of content and style in Convolutional Neural Networks are separable. Both representations of content and style can be manipulated independently to produce brand-new and meaningful images.

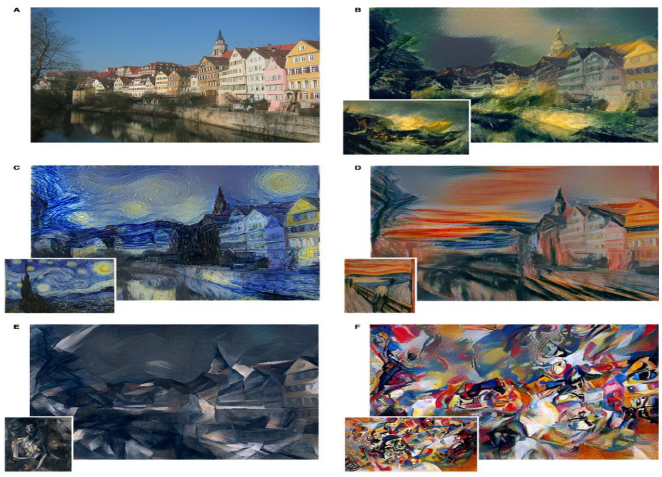


FIgure 3. Image of the "Neckarfront" across different styles.

In figure 3 on the previous page, a photograph of the "Neckarfront" in Germany was taken and transferred across different styles. In the figure, images were generated that mix the content and style representation of the image. The style representations originate from different periods of time. The images are synthesized by finding an image which simultaneously matches the content of the image and style of the artwork used. This efficiently renders the photograph in the style of the artwork, with the global arrangement of the original piece preserved.

Content and style aren't completely separated though. The loss function that is minimized during image synthesis contains both terms for content and style respectively which are well separated.

In this source, it was observed that prior work that consisted of separating content from style was evaluated on sensory inputs of much lesser complexity. The source's demonstration rendered a given image in the style of a range of well-known artworks. By using Deep Neural Networks trained on object recognition, manipulations can be carried out in feature spaces that explicitly represent the high level content of an image.

The methods used within this work were generated on the basis of the VGG-Network, which is a convolutional neural network rivaling human performance of common visual object recognition. Sixteen convolutional and five pooling layers were used in the feature space of the nineteen layer VGG-Network. Each layer in the network defines a non-linear filter bank whose complexity increases with the position of the layer within the network.

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} \left(F_{ij}^{l} - P_{ij}^{l}\right)^2 .$$

Figure 4. Content loss function.

The above equation defines the squared-error loss between the two feature representations of *p* and *x*, with P*l* and F*l* being their respective feature representations in layer *l*. The derivative of this loss function with respect to the activations is shown below in figure 4.

$$\frac{\partial \mathcal{L}_{content}}{\partial F_{ij}^{l}} = \begin{cases} \left(F^{l} - P^{l}\right)_{ij} & \text{if } F_{ij}^{l} > 0 \\ 0 & \text{if } F_{ij}^{l} < 0 . \end{cases}$$

Figure 5. Derivative of the loss function with activations.

The gradient with respect to the image *x* can be computed using standard error back-propagation. On top of the CNN responses in each layer of the network is a style representation built that computes the correlations between the different filter responses. There feature correlations are given by the Gram matrix. Gradient descent is used from a white noise image to find another image that matches the style representation of the original image. This is completed by minimizing the mean-squared distance between the entries in the Gram matrix from the original image and the Gram matrix of the generated image. The total loss is formulated below.

$$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l$$

Figure 6. Total style loss function.

The w*l* are the weighting factors of the contribution of each layer to the total loss.

In order to generate images with a mix of the content from one photograph and the style of a particular painting, we jointly minimize the distance of a white noise image from the content representation of the photograph in one layer of the network and the style representation of the painting in multiple layers of the CNN. The total loss function with both the content and style loss terms combined is formulated below.

$$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{style}(\vec{a}, \vec{x})$$

Figure 7. Total loss function of content and style combination.

Here, the alpha and beta terms represent the weighting factors for content and style respectively, and the *p* and *a* terms represent the photograph and artwork respectively.

In our fast style transfer implementation, we aimed to generate images that mixed in style with content without distorting the content of the original image, which is what happened in a few of the images within this source. This could be attributed to smaller alpha/beta ratios within the above loss function in figure 7.

2.1 Related/Prior Work Analysis (Perceptual Loss)

In our other source we used to learn more about style transfer and the approaches and algorithms behind the model, we explore here the introduction of high-quality images that can be generated by "*defining and optimizing perceptual loss functions based on high-level features extracted from pretrained networks*"[2]. The benefits of this approach are combined with the benefits of training feed-forward Convolutional Neural Networks using a per-pixel loss between the output and ground-truth images. This new approach proposed gives identical results but of three orders of magnitude faster.

In the perceptual loss function approach, high-quality images can be generated based on differences between high-level image feature representations extracted from pretrained Convolutional Neural Networks.

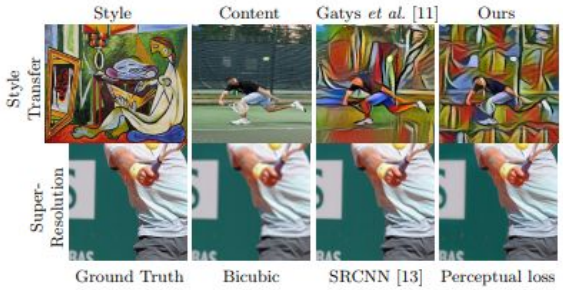


Figure 8. Results of Style Transfer and Super-Resolution.

The figure on the previous page depicts results of style transfer and super-resolution. Style transfer achieves similar results as Gatys *et al*, which is a source that paper references and who proposed the optimization problem being faced here. The super-resolution image trained with a perceptual loss is better at reconstructing fine details compared to the per-pixel loss approach.

In combining the benefits of the two approaches, image transformation tasks have trained feed-forward transformation networks. Rather than using per-pixel loss functions depending on low-level pixel information, the networks are trained using perceptual loss functions that depend on high-level features from a pretrained loss network. During training, the perceptual losses measure image similarities more robustly than per-pixel losses, and during testing, the transformation networks run in real-time.

The two tasks experimented on were style transfer and super-resolution. The success of either task requires semantic reasoning about the input image. In style transfer, the output must be semantically similar to the input despite drastic changes in color and texture. For super-resolution, fine details must be inferred from visually ambiguous, low-resolution inputs.

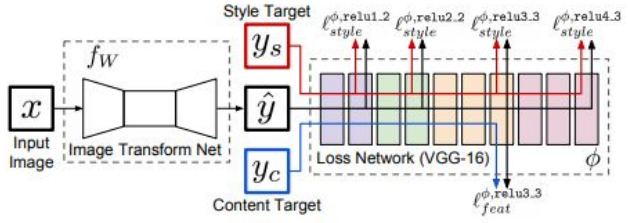


Figure 9. System overview of the image transformation network.

In the above figure, the image transformation network is trained to transform input images into output images. A loss network is used that is pretrained for image classification to define the perceptual loss functions that will measure perceptual differences in content and style between images. During the training process, this loss network remains fixed. The loss network is used to define several loss functions.

The image transformation network is a deep residual Convolutional Neural Network parameterized by weights W, as it transforms input images *x* into output images *y(hat)* via the mapping of *y(hat) = fW(x)*. The image transformation network is trained using stochastic gradient descent to minimize a weighted combination of loss functions in the math formulation given below.

$$W^* = \arg\min_W \mathbf{E}_{x,\{y_i\}} \left[\sum_{i=1} \lambda_i \ell_i(f_W(x), y_i) \right]$$

Figure 10. Formulation of stochastic gradient descent to compute and minimize the weighted combination of loss functions.

The loss network is used to define a feature reconstruction loss and a style reconstruction loss, which measures the differences in content and style between images.

There are two defined perceptual loss functions that measure high-level perceptual and semantic differences among images. These make sure of the loss network pretrained for image classification, which means that these perceptual loss functions themselves are deep Convolutional Neural Networks. In this source's experiments, the loss network was a sixteen layer VGG network.

The image transformation network was trained per style target for several hand-picked style targets, and the results of the experiments performed were measured with against the baseline approach of Gatys *et al* below.

Figure 11. Resulting 512x512 images generated after applying style transfer models trained 256x256 images.

In the experiment's training, the style transfer networks were trained on the MS-COCO dataset. Each of the 80k training images were resized to 256x256 and trained with a batch size of 4 for 40k iterations, which gave about two epochs over the training data. Though the models were trained with 256x256 images, they could be applied to images of any size (the 512x512 images) when test time is reached.

The results were qualitatively similar to the baseline approach, but in some cases, the method this source used produced images with more repetitive patterns. This effective became more obvious in images of higher resolutions, as seen in figure 11. The content image achieved a very high loss, and the method used achieved "*a loss comparable to 50 to 100 iterations of explicit optimization*"[2]. Again, although the networks are trained to minimize the objective function for 256x256 images, they also succeed at minimizing the objective function when being applied to larger images.

The primary results of this source was that the method used for style transfer achieved comparable performance to its related work but drastically improved

speed compared to existing methods. The same goes for single-image super-resolution where training with a perceptual loss allows the model to better reconstruct finer details and edges. Therefore, this work found a way to generate a fast style transfer, but more could be done in order to cut down on the repetitive patterns illustrated in generated images (if that were to be a goal for future work). In our implementation, we accomplish a fast style transfer without any intended repeated patterns.

3 Fast Style Transfer Implementation & Approach

Firstly, we want to appreciate the source code for the fast style transfer[3], their implementation uses TensorFlow to train a fast style transfer network. They used roughly the same transformation network as described in Johnson, except that batch normalization is replaced with Ulyanov's instance normalization, and the scaling/offset of the output tanh layer is slightly different. They used a loss function close to the one described in Gatys *et al*, using VGG19 instead of VGG16 and typically using "shallower" layers than in Johnson's implementation (e.g. we use relu1_1 rather than relu1_2). Empirically, this results in larger scale style features in transformations.

Secondly, since they already provide the trained style transfer networks, we don't have to rent some GPU to train around a week. Then we evaluated style transfer network by this following commands:

python evaluate.py
--checkpoint pretrained-models/scream.ckpt \
--in-path examples/content-images \
--out-path examples/results1

let's look at the six pre-trained weights: they are la_muse, rain_princess, the_scream, the_shipwreck_of_the_minotaur, udnie, and wave.

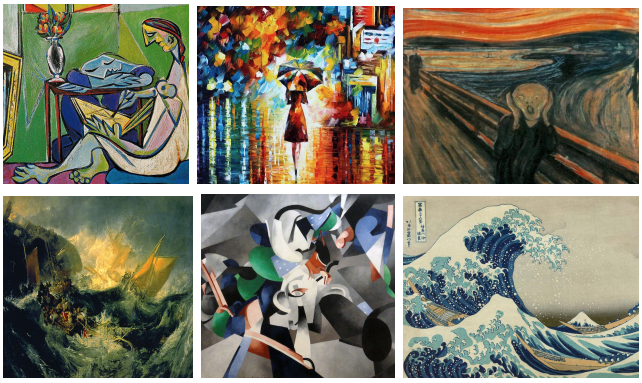


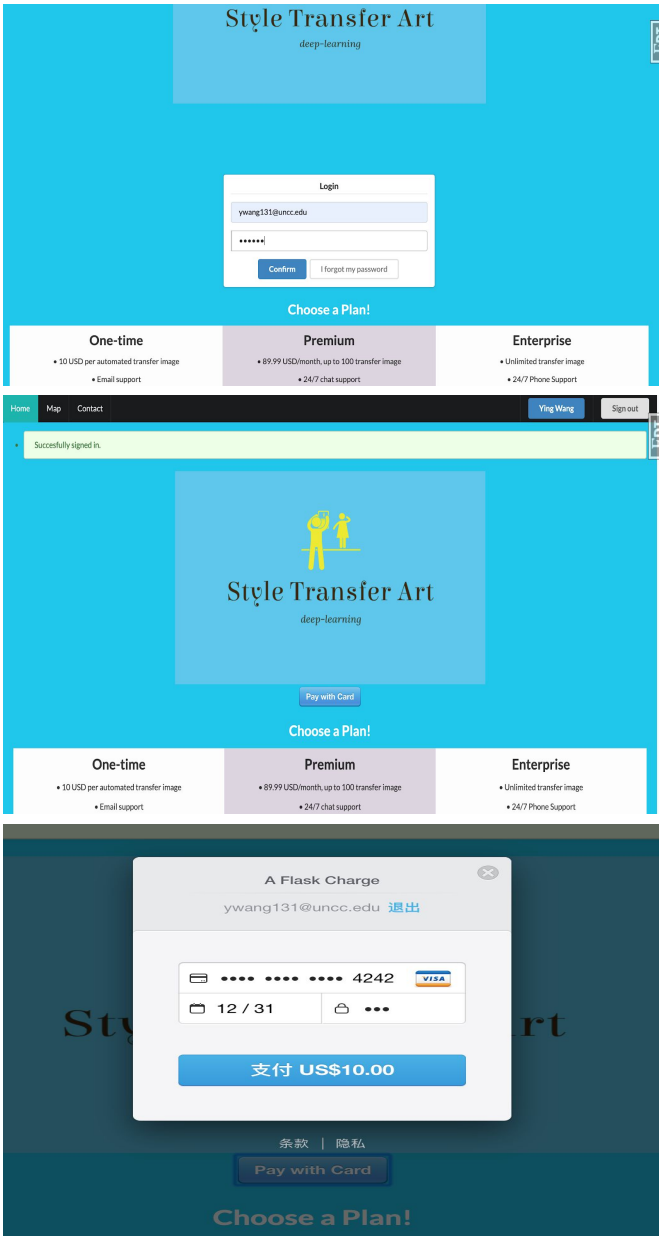
Figure 12. Pre-trained styles to combine with content images.

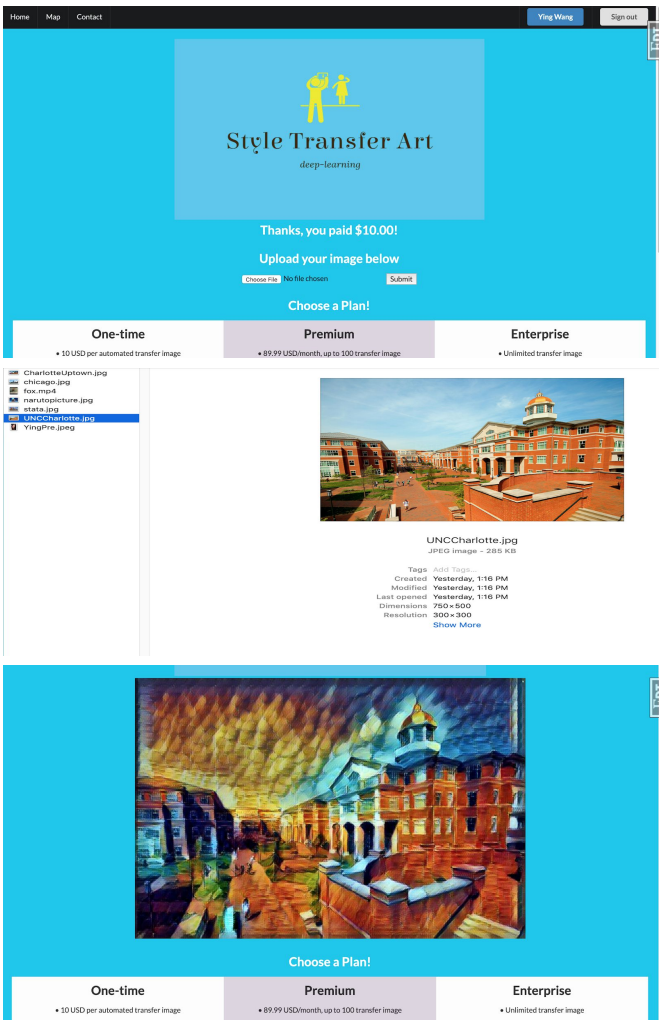
Thirdly, the requirements for the fast style transfer are TensorFlow 0.11.0, Python 2.7.9, Pillow 3.4.2, scipy 0.18.1, numpy 1.11.2. If you want to train (and don't want to wait for 4 months): A decent GPU, All the required NVIDIA software to run TF on a GPU (cuda, etc).

According our project proposal, we want to build up a small AI-Startup called "Style-Transfer-Art" so that we applied the FLASK which is a microframework for Python to build our company website. 1) create our website logo; 2) Design Pipeline, used Database (SQL), keep the user information and let the user to download their arts; 3) Deploy, connect to the tensorflow server and send back the transferred images.

4 Results

Below are a few screenshots of our project. These essentially replicate our demonstration.





In the above screenshots, we have demonstrated the user logging in and processing a payment in order to receive a picture with a style transfer. The user then can upload an image from their computer and submit the image for styling. The website will then display the generated image to the user that has gone through style transfer of any of the selected styles within the Tensorflow system. The user also then has the ability to download the generated image to their computer.

5 Conclusion

Inspired from the papers in the links below, we basically built an AI-Startup-Style-Transfer-Art website which user can login and start doing their own style transfer pictures. We successfully utilized the technique of fast neural style transfer to multiple styles, which allows users to transfer contents of input images into multiple styles. We only have six multiples styles we mentioned before. We experienced what is research based on this project and we also learned bacial knowledge of tensorflow, VGG19, and flask. I think this is a great opportunity to combine someone's work and our ideas to actually develop a product. However, if we were to continue working on this project, we would need to work on how to let the user automatically choose the style instead of manually change the style codes. This project would definitely be intriguing to continue with. We certainly learned more about these different techniques of applying neural network algorithms to something that we thought was very interesting. When reading the articles, we were able to understand them more after learning about neural networks in class. We feel this material and the class have prepared us to study deep neural networks in the future. We may even be able to apply our new knowledge of perceptual loss, VGG networks, deep neural networks, and convolutional neural networks towards a whole new image processing task that could be used for good.

Contributions & Acknowledgements

Ying focused on: read the papers, implementation with running the style transfer code, and report (equal contribution).
Eric focused on: read the papers, presentation, collaborated to set up website for users and helped with implementation, and report
(equal contribution).

The authors wish to that Professor Bhattacharjee and the TA, Ishan, for their help all semester long. We certainly learned a lot over the course of the semester.

References

[1] Gatys, L., Ecker, A. and Bethge, M. (2015). *A Neural Algorithm of Artistic Style*. [online] arXiv.org. Available at: https://arxiv.org/abs/1508.06576 [Accessed 27 Apr. 2019].

[2] Johnson, J., Alahi, A. and Fei-Fei, L. (2016). *Perceptual Losses for Real-Time Style Transfer and Super-Resolution*. [online] arXiv.org. Available at: https://arxiv.org/abs/1603.08155 [Accessed 27 Apr. 2019].

[3] Fast-Style Transfer Github (lengstrom), *fast-style-transfer* (2018). https://github.com/lengstrom/fast-style-transfer

[4] Flask Basic Example, *flask* (2019). https://github.com/pallets/flask

[5] Flask Example for payment and SQL database, *flask-boilerplate* (2018). https://github.com/MaxHalford/flask-boilerplate