AI Based Color Mapper User Guide

Team Machine Churning

June 1, 2020

Contents

1 Preface:

1.1 README

Authors: Atharva Chalke, Henry Yi, Monica Magallanes, Marcus Chan

1.2 Audience

This user guide is intended for people with experience with Python and Linux. Experience with machine learning is not required to build the project; however, it is needed if you want to extend its capability and understand the ml aspect of it. Understanding of Microscopy is required to understand the use case. We recommend using MacOS if running locally.

1.3 Glossary

VM	Virtual Machine; a full operating system, several of which may berunning on a single physical computer concurrently.
Google Compute Engine	Google Compute Engine is the Infrastructure as a Service component of Google Cloud Platform which is built on the global infrastructure. Google Compute Engine enables users to launch virtual machines on demand.
GCP	Google Cloud Platform. The collection of cloud services Google provides, including Google Compute Engine (virtual machines), Cloud Storage (simple file storage),Networking and several other services.
Google Cloud Storage	A cloud service that allows for simple file uploads and downloads
Cycle GAN H&E	A model architecture that uses Generators and Discriminators to train a model for inference [Cyclic General Adversarial Network] Traditional tissue staining: Hematoxylin and Eosin stain
MUSE	New method to produce an image from a tissue sample without having to stain it: Microscopy with UV Surface Excitation
GPU	Graphic Processing unit
Image segmentation	Segment an image into different categories and assign a label that represents a category to each pixel
Saliency Map	A map that shows the regions that are important and of the interest so that the model could pay more attention toward the important regions.

2 Overview of product

2.1 Background of product

Histology is the study of microscopic structures of tissues, also referred to as microscopic anatomy or micro-anatomy. Dr. Richard Levenson's lab has developed a microscopy application named MUSE (Microscopy with UV Surface Excitation), which generates images of the tissue without having to do the classic HE (Hematoxylin and Eosin) staining. Thus, making the process much more efficient. However, this MUSE image has to be further converted for human viewing or for quantitative analysis which takes a lot of time and current methods to do so are prone to errors.

2.2 Description

Making this process automated is currently an area of interest to the Histology community. It is new, and although there have been various attempts in doing such a task, they have not been successful. Automation of this sort is valuable to this field as it would save time and various resources, increasing productivity and advancement in projects and research. We employ Machine Learning, specifically Cycle General Adversarial Networks to convert MUSE images to H&E images. For people without Machine Learning knowledge, we provide an interface where the user can upload images and get the generated image using a GUI (Graphical User Interface). The process employs multiprocessing, and supports GPU's as well to improve the efficiency of the inference.

2.3 Approach

2.3.1 Pre-processor

Since, the images provided by the user can be large in size, we cannot fit it all within main-memory. Thus, we break the original image down into patches with overlapping regions. We decide to generate overlapping regions since without it the final reassembled image will have tile like structures for different regions. This process supports multiprocessing to improve efficiency.

2.3.2 Inference

Once the overlapping patches have been generated, we feed it through a Generator within the cycle-gan to produce a mapping from MUSE to H&E images. This process supports using GPU's.

2.3.3 Post-processor

Once, the inference is done, we stack the images together to reassemble the original image. Average blending is used in the overlapping regions to smooth the image out. After this process completes, we push the image to a Google Cloud Storage Bucket for the month. Thus, users can access the prediction in the future, and this data can be used for further training in the future.

2.3.4 Frontend

The approach and main goal for this frontend Was to provide user with an easy and interactive method to utilize the Machine Learning inference models. This a web browser based interface. Interface design patterned after other normal upload websites for familiarity of use. In this way, user is now able to use images stored on their respective device for instant and simple access to inference.

2.4 Technical Specifications

2.4.1 Backend: Flask

Flask is a lightweight web frame of Python. It provides the user with libraries, modules and tools to help build Web-Applications

2.4.2 Database: Google Cloud Storage

Google Cloud Storage is a RESTful online file storage web service for storing and accessing data on Google Cloud Platform infrastructure.

2.4.3 Github

This is the best tool for us to collaborate and build our site together, providing us with the ability to code concurrently and efficiently.

2.4.4 PyTorch

PyTorch is an open source machine learning library based on the Torch library, used for applications such as computer vision and natural language processing, primarily developed by Facebook's AI Research lab.

2.4.5 TensorFlow

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural network.

2.4.6 Frontend: HTML, CSS, Bootstrap & Java Script

HTML is the defacto language of the web, CSS its styling component and Javascript(JS), it's interactivity method. The library Bootstrap provided a quick and effective interface toolset to contruct said interface. Specific tech used include JS's features including aynchronous handling of image download browser side as well as rendering thereafter on the website.

3 Installation and Setup

3.1 Prerequisites

1. A machine capable of running python 3.6 and 1 GPU (we recommend using the NVIDA TESLA K80), and the cuda tool kit installed. If hosting on cloud, we require a linux machine with at least 1 GPU.
2. A Google Cloud account (free accounts are fine, but you may end up using your credits fast if running the machine 24/7).
3. Anaconda installed on the machine.
4. The cuda tool kit can be found https://developer.nvidia.com/cuda-downloads. Only needed to speed up the training and inference process.

3.2 Basic Setup

1. Create a directory for the project. We will use MUSE_APP in our example.
2. Then we create and activate our environment using the following:

```
conda create -n mypython3 python=3.6
conda activate mypython3
```

3. Clone the repository using

```
git clone https://github.com/atharvacc/MachineChurning.git
```

4. Navigate inside the folder and install the libraries required using

```
pip install -r requirements.txt
#Run the command below if you get GPU Driver issues
pip install torch==1.5.0+cu101 torchvision==0.6.0+cu101 -f

https://download.pytorch.org/whl/torch_stable.html
```

5. Download the pretrained model from
https://drive.google.com/drive/folders/1FdxMdDVCZHN-V0x_tOvS4wemccTl6XEg?usp=sharing
6. Copy the file to MachineChurning/App
7. Navigate to MachineChurning/App
8. Run the server by running the following:

```
bash run_flask.sh local # To run the interface locally
bash run_flask.sh # To run the interface on the
    port 8888 for cloud
```

3.3 Cloud Setup

1. Create a project on Google Cloud
2. Navigate to VM instances on Compute Engine.

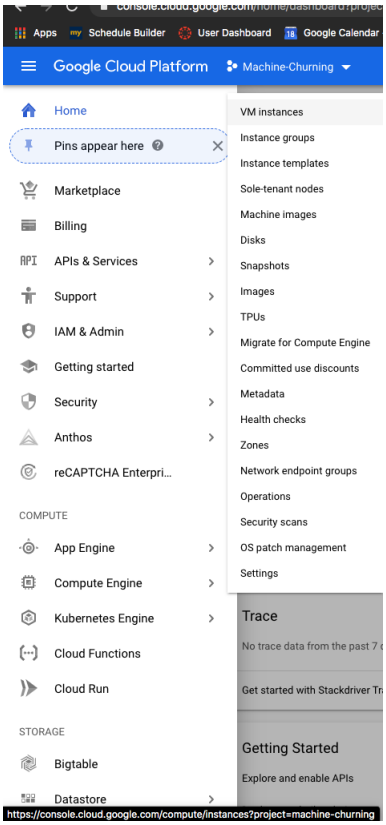

Figure 1: Navigate to VM instances

3. Create a deep learning VM from the marketplace.

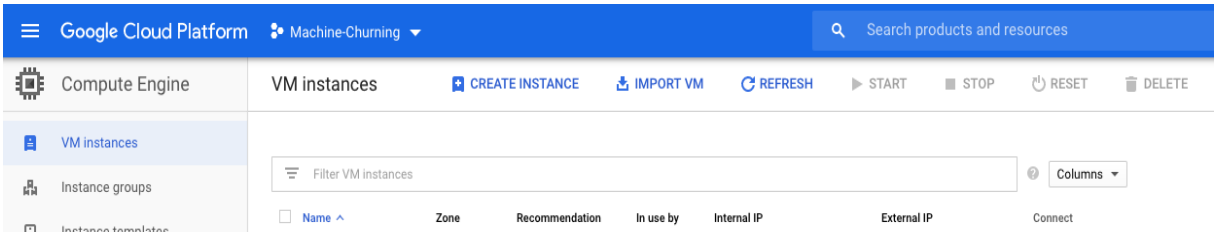

Figure 2: Select Create Instances

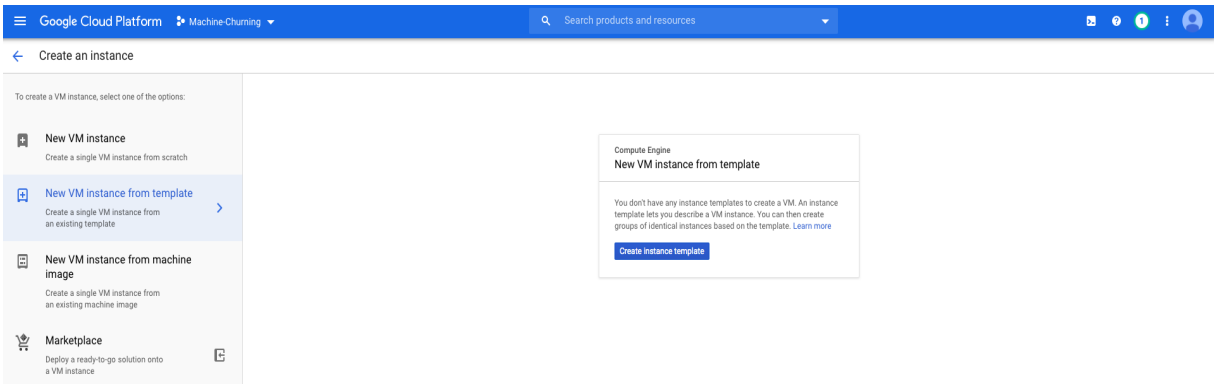

Figure 3: Navigate to Marketplace

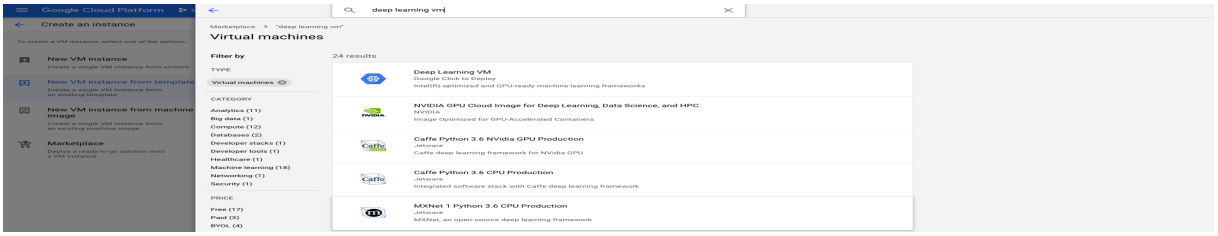

Figure 4: Use the Deep Learning VM tempelate

4. Select the option "Install NVIDIA GPU driver automatically on first startup?" and then deploy the VM.

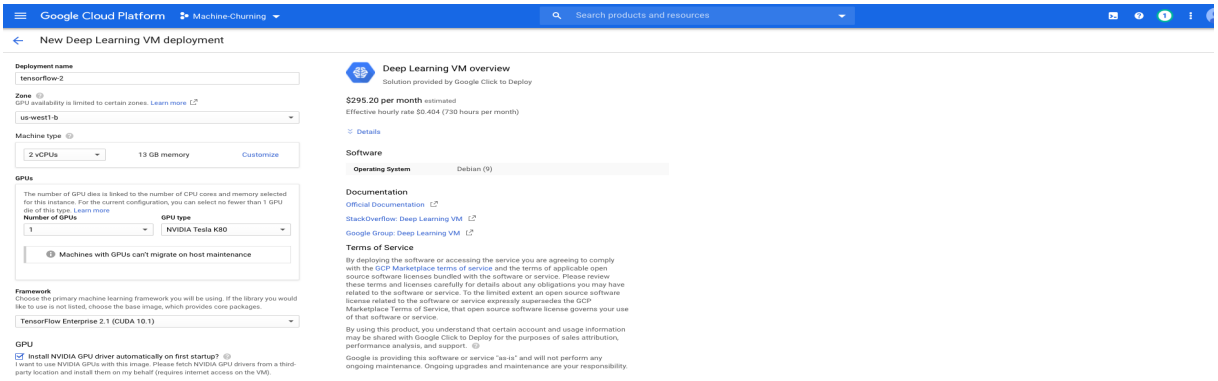

Figure 5: Use the Deep Learning VM tempelate

5. Once deployed, STOP the Virtual Machine in the Compute Engine ->VM instances.
6. Change the firewall settings by editing the virtual machine that was just started and allow access to cloud storage.

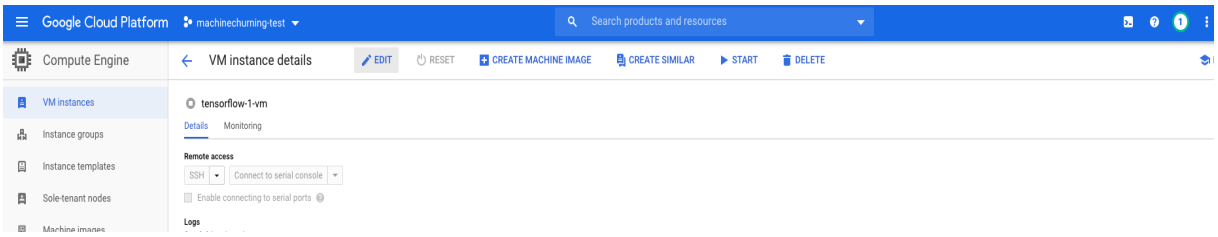

Figure 6: Edit the machine

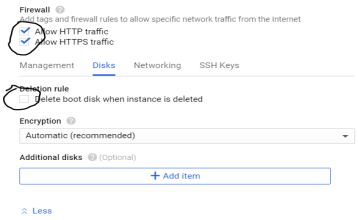

Figure 7: Firewall settings and deletion rule

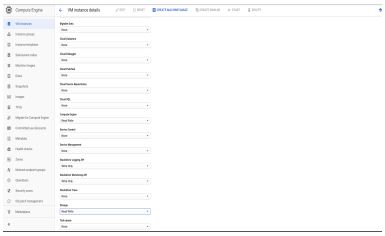

Figure 8: Allow read/write access to storage

7. Note down the External IP located here.

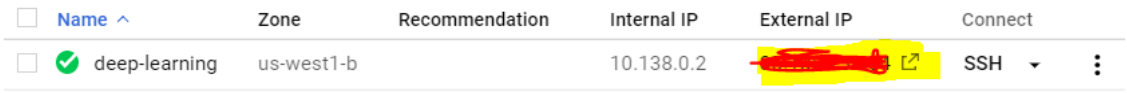

Figure 9: External IP for VM instance

8. Make the external IP address as static

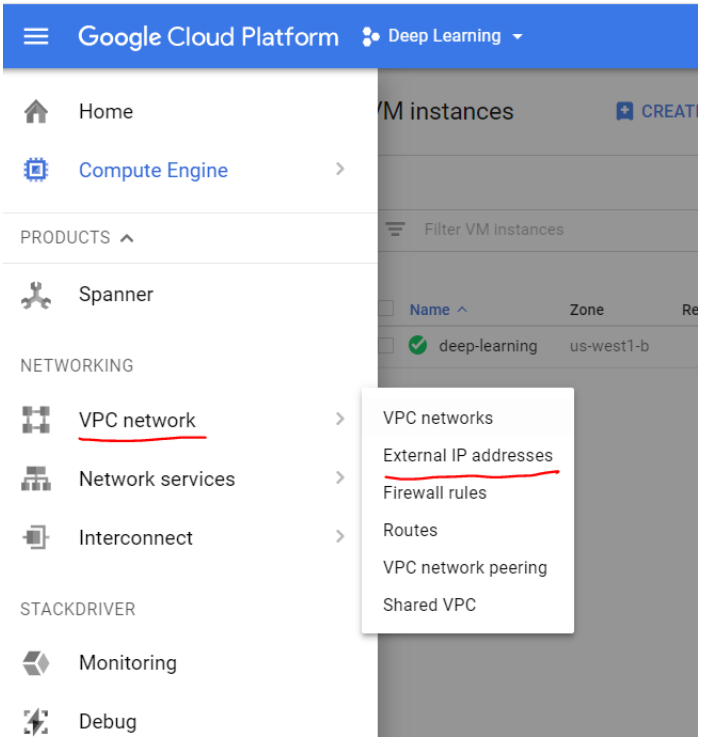

Figure 10: Navigating to firewall rules

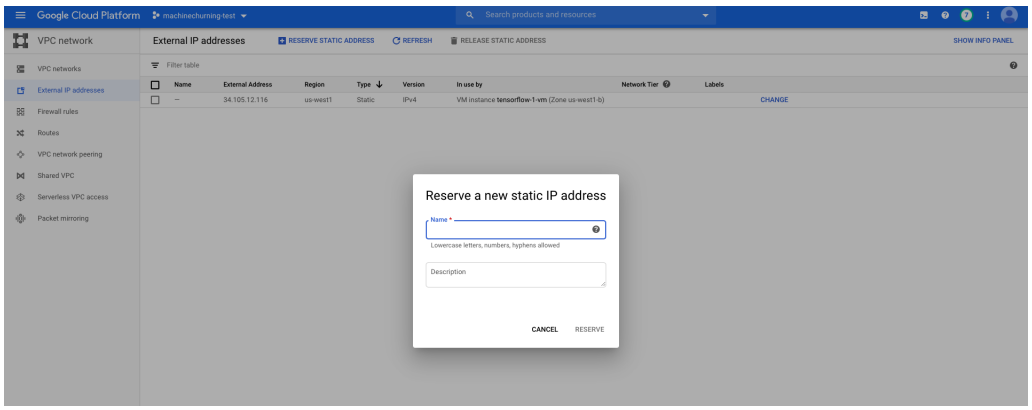

Figure 11: Make external IP static

9. Click on 'Create Firewall Rules' and refer to the below image:

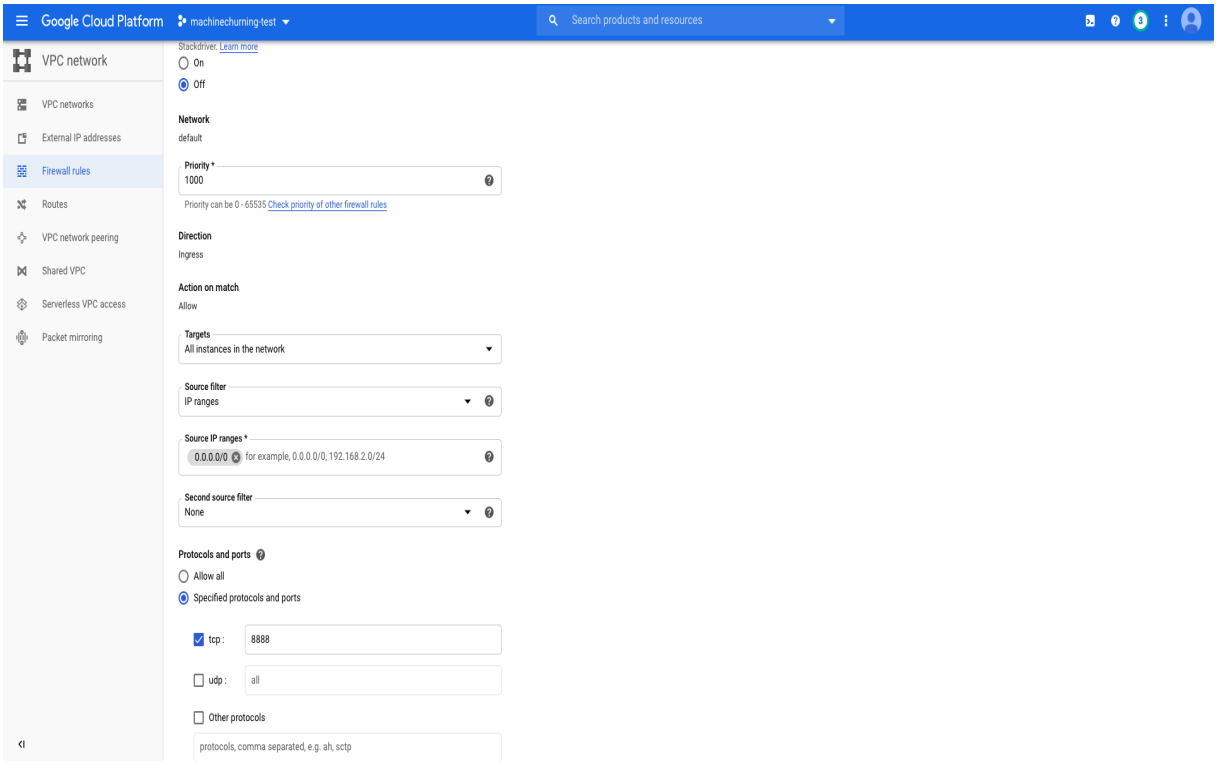

Figure 12: Open TCP 8888 and allow external traffic

10. Create a service account and download the json key related to it.

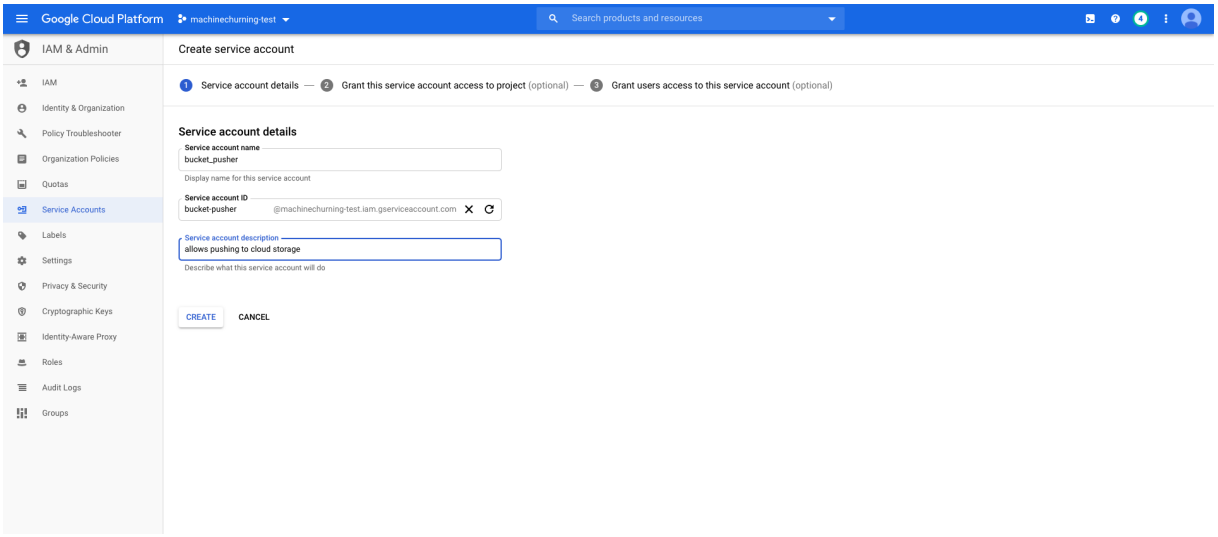

Figure 13: Creating a service account

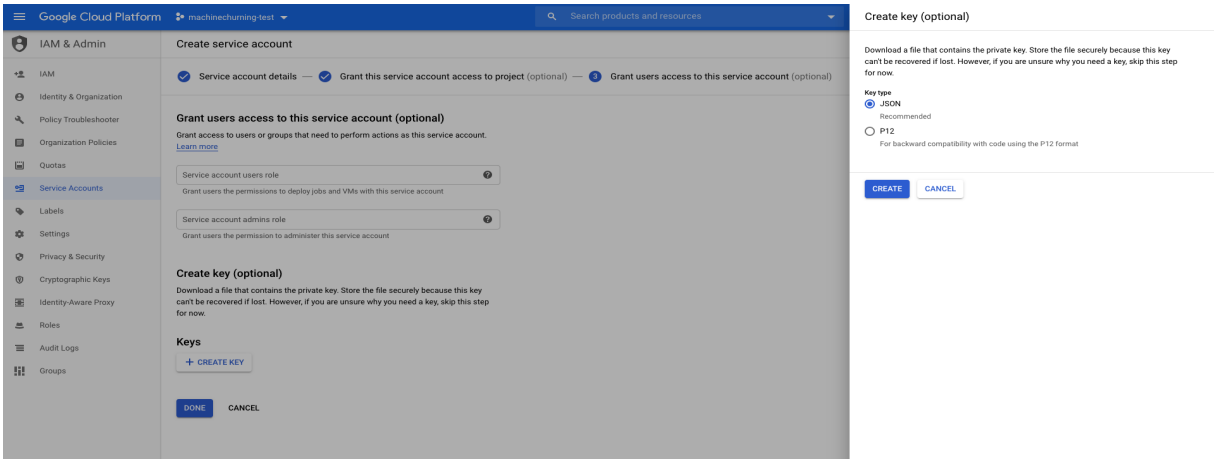

Figure 14: Download the JSON key

11. Create a bucket to store the data

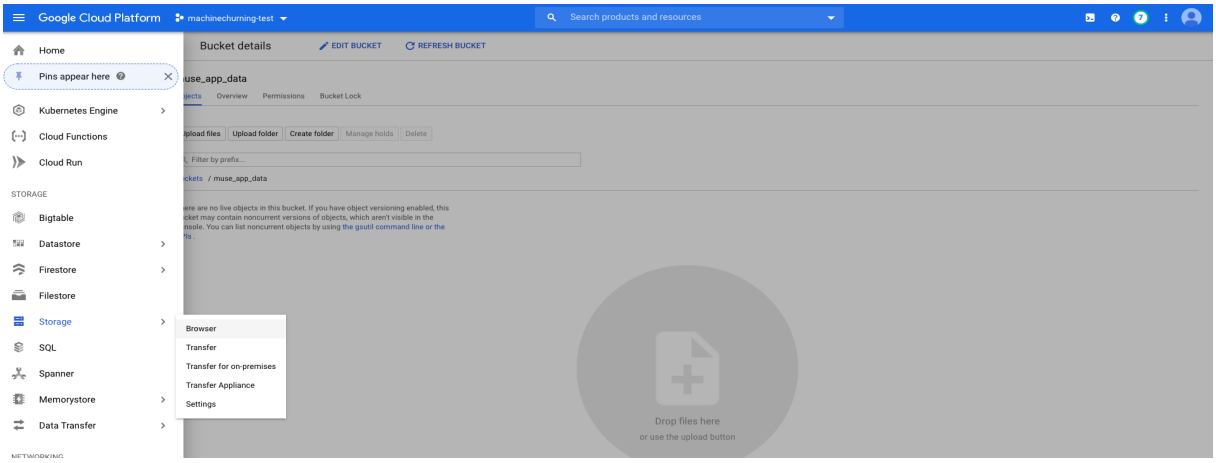

Figure 15: Navigate to storage

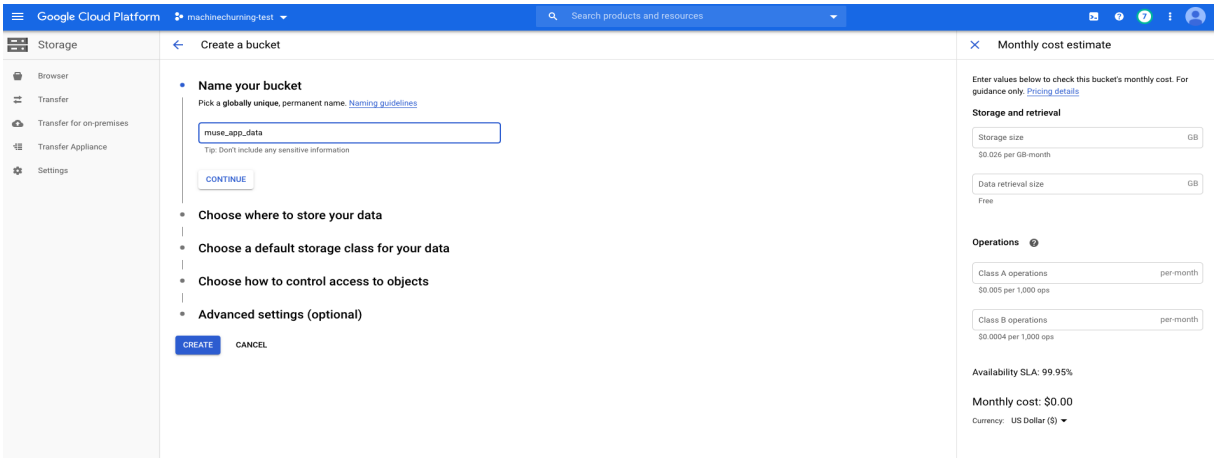

Figure 16: Create a bucket

12. SSH into the instances and copy the json file and export the credentials.

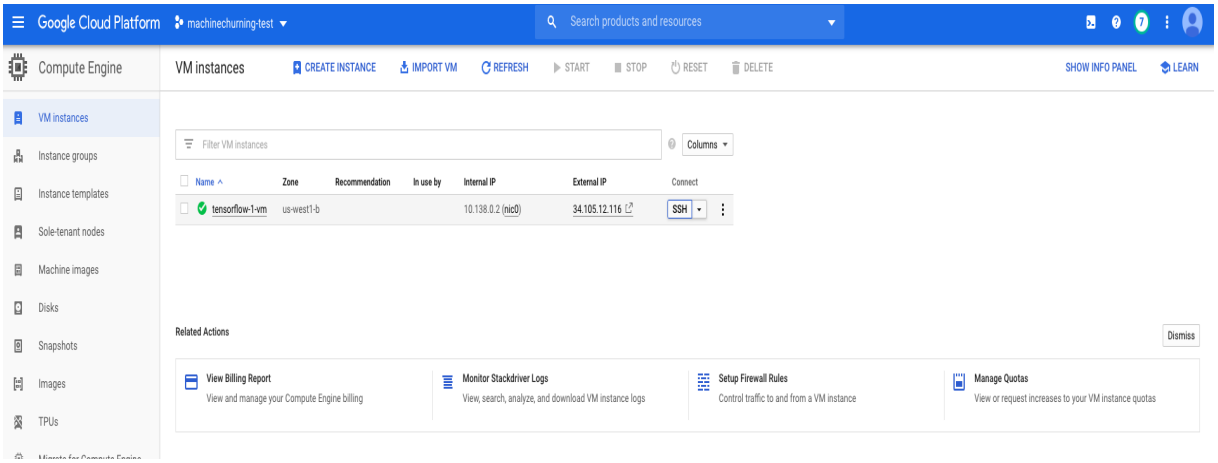

Figure 17: SSH into the instance

Figure 18: Upload the file

13. Export the json credentials using

    EXPORT GOOGLE_APPLICATION_CREDENTIALS=<path to json file>

14. After this, follow the steps for basic usage.

4 Functionality

4.1 Training the model

1. We provide a command line interface to training and inferencing with the model.
2. Use the main parser located at MachineChurning/main_parser.py. The arguments for which are located MachineChurning/docs/parser_documentation.md.
3. While training we generate logs for the training process which can be found at saved_model/test_cyclegan/loss_log.txt.
4. We also do some inferencing at checkpoints and generate a HTML with a few results which can be found at saved_models/test/web/index.html.
5. The saved models are located within the test_cyclegan directory mentioned above.

```
(base) atharvachalke@atharvas-mbp test_cyclegan (master) $ #head loss_log.txt
================ Training Loss (Tue May 26 01:49:29 2020) ================
(epoch: 1, iters: 100, time: 1.239, data: 1.173) D_A: 0.357 G_A: 0.396 cycle_A:
1.082 idt_A: 0.451 D_B: 0.420 G_B: 0.507 cycle_B: 1.036 idt_B: 0.586
================ Training Loss (Tue May 26 01:55:11 2020) ================
(epoch: 1, iters: 100, time: 1.238, data: 0.366) D_A: 0.321 G_A: 0.325 cycle_A:
4.924 idt_A: 0.610 D_B: 0.403 G_B: 0.411 cycle_B: 1.273 idt_B: 2.483
(epoch: 1, iters: 200, time: 1.235, data: 0.002) D_A: 0.324 G_A: 0.272 cycle_A:
3.073 idt_A: 0.571 D_B: 0.259 G_B: 0.458 cycle_B: 1.184 idt_B: 1.540
(epoch: 1, iters: 300, time: 1.238, data: 0.002) D_A: 0.248 G_A: 0.335 cycle_A:
1.453 idt_A: 0.472 D_B: 0.263 G_B: 0.483 cycle_B: 1.000 idt_B: 0.740
(epoch: 1, iters: 400, time: 2.616, data: 0.002) D_A: 0.341 G_A: 0.562 cycle_A:
5.717 idt_A: 0.537 D_B: 0.145 G_B: 0.695 cycle_B: 1.204 idt_B: 2.789
(epoch: 2, iters: 100, time: 1.236, data: 0.286) D_A: 0.267 G_A: 0.296 cycle_A:
5.608 idt_A: 0.860 D_B: 0.195 G_B: 0.417 cycle_B: 1.862 idt_B: 2.857
(epoch: 2, iters: 200, time: 1.237, data: 0.002) D_A: 0.361 G_A: 0.288 cycle_A:
7.284 idt_A: 0.744 D_B: 0.170 G_B: 0.346 cycle_B: 1.439 idt_B: 3.693
(epoch: 2, iters: 300, time: 1.238, data: 0.002) D_A: 0.223 G_A: 0.375 cycle_A:
5.874 idt_A: 0.510 D_B: 0.077 G_B: 0.456 cycle_B: 0.900 idt_B: 2.908
```

Figure 19: Loss_log

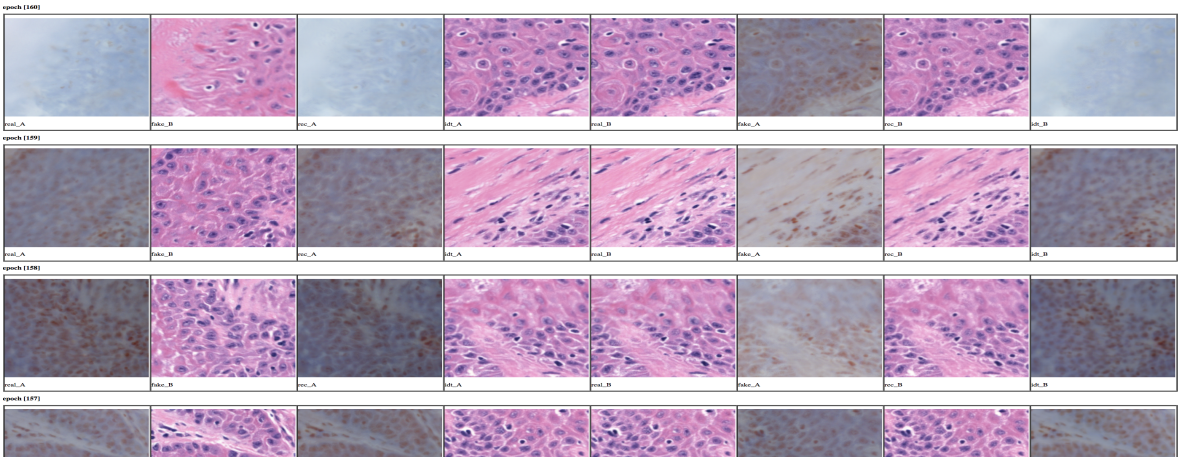

Figure 20: Html file generated while training

4.2 Using the model for local inference

1. We recommend using the interface after you have a copy of the saved models for inferencing since it automates the whole process efficiently.
2. If you manually desire to do so, you can use the main_parser.py to do so.
3. It required a directory structure as mentioned in the github documentation repository.
4. The stored image from the inference using the interface can be found in the bucket created.

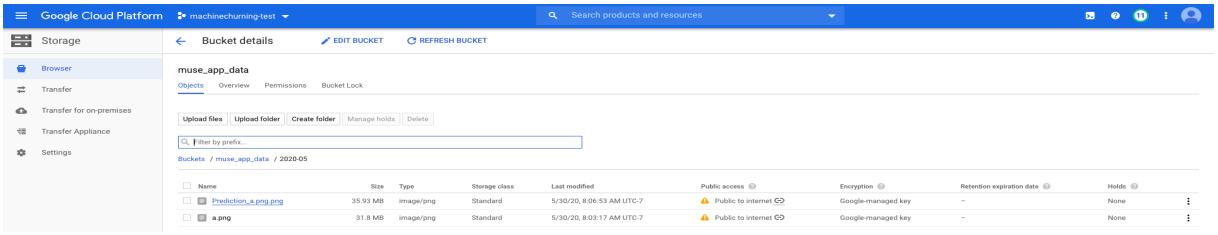

Figure 21: Original and Predicted Image

4.3 Hosting the server using Flask

1. The interface can be run using the following after navigating to MachineChurning/App:

```
bash run_flask.sh #Run on port 8888
bash run_flask.sh local #Run locally
```

2. The usage should be quite intuitive as you just upload the image to the portal and it handles the rest of the process for you.

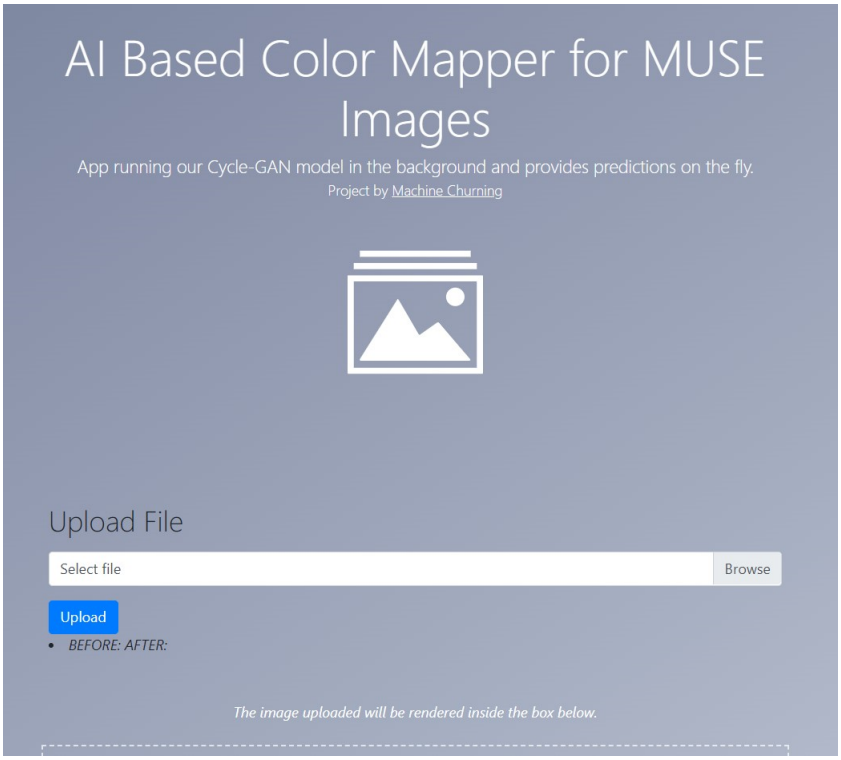

Figure 22: Interface

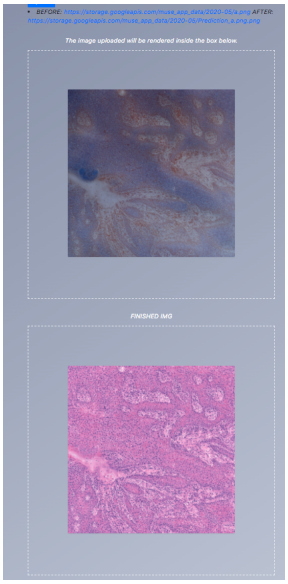

Figure 23: Results displayed with links

4.4 Retraining the model

1. We can download the predicted images to look at the predictions generated to look at the performance of our model. These images after manual review can be used to retrain the model to improve performance. Please do not retrain using the entire dataset since the model will be biased to its current usage.

5 Troubleshooting

- You should try to refresh the web page if it doesn't show up on the browser try refreshing the page or restarting the flask server.
- If GPU drivers are not installed, make sure you have the requisite cuda toolkit and gpu drivers.
- If the issue persists, please contact with us.

6 Frequently Asked Questions

1. **Question:** How should we do if the uploading process was stuck?

   Answer: Please, try to reload the web page and upload it again in a stable internet connection.

2. **Question:** I cannot open the links to the image objects in buckets. What is going wrong ?

   Answer: Make sure the access is set to public for the entire bucket, and not just fine grained objects.

3. **Question:** Help ! The external IP for my VM is not loading.

   Answer: Make sure you're using a HTTP connection and not a HTTPS connection. i.e http:// <your external ip >:8888 . Also make sure you have done all the steps done in setting up the firewall in the Cloud setup section.

7 Contact Information

- Atharva Chalke
  Email: anchalke@ucdavis.edu
  Phone: (530) 760-9170

- Henry Yi
  Email: henyi@ucdavis.edu
  Phone: (530) 761-3756

- Marcus Chan
  Email: mtchan@ucdavis.edu
  Phone: (408) 660-7757

- Monica Magallanes
  Email: mgmagallanes@ucdavis.edu
  Phone: (831) 998-6733

8 Appendix: Design Document

8.1 Technology Survey

8.1.1 Machine Learning Models

- Image Generation Network - CycleGAN
  - Background
    It is a GAN network that can do training of image-to-image translation models without paired examples. The models are trained in an unsupervised manner using a collection of images from the source and target domain that do not need to be related in any way.
  - Modifications
    * We added skip connections to our model as proposed in the ganilla paper [1]
    * We also added a saliency loss function as proposed in [2] to improve our content by using a K-means classifier to segment the images based on color
    * We made a manual segmentation model by manually selecting nuclei and connective tissue in the images and training a neural network to generate a saliency map based on color

Pros:
1. Works well with authentic image translation without corresponding ground truth images.
2. Improves resolution of MUSE images.

Cons:
1. Generates artifacts in blurry regions sometimes. The saliency loss functions helps improve the process.
2. Highly reliant on the training data for inference.
3. No large dataset available for MUSE images.

- Gray Scale Model
  - Background
    It converts every image to gray scale, and uses the colored channels

as the output for H&E images. While training, we use predict the H&E channels using its corresponding gray-scale image. During inference, we use the gray-scale MUSE images to predict the H&E images. Thus, the problem is converted to a supervised settings.

Pros:
1. Ensures that the content is always the same, while changing the color.
2. Large H&E datasets are available which can be used for training.
3. Much faster than Cycle-GAN since the model is smaller and thus faster.
Cons:
1. No improvement in resolution.
2. Mapping gray scale objects in MUSE to H&E will not always results in the same values for various biological entities since the gray-scale values for MUSE and H&E for corresponding objects is not guaranteed to be the same.

Conclusion: We will use the cycle-gan model since it improves resolution and the artifacts are usually not generated.

8.1.2 Backend/Framework

- Django
  Pros:
  1. Uses Python, which is a very powerful and well-supported language with simple syntax.
  2. Plenty of documentation, and mature community
  Cons:
  1. No experience with Django.
  2. Bulkier due to the number of features that we may not end up using.

- Flask
  Pros:
  1. Simpler to learn than Django.
  2. Slightly faster than Django due to its simplicity.
  3. Easier integration with Google App Engine.
  Cons:
  1. Not as much functionality as Django.

Conclusion: We used Flask, since we had experience with it. Most of our work was on the modelling section, therefore, we just needed a simpler user-interface. It has an easier integration with Google App Engine as well which can be used in the future for deployment.

8.1.3 Frontend

- Ant Design
  Pros:
  1. Very nice, simple design.
  2. Uses React, which makes under-the-hood design simpler to manage at scale
  Cons:
  1. A lot of the community is Chinese, harder to find support in English
- Bootstrap
  Pros:
  1. Many of us already have experience with it
  2. Plenty of documentation
  Cons:
  1. Not as advanced as other frontend options

Conclusion: We will be using bootstrap, because it is simple and fits the purpose of our site very well.

8.1.4 Database

- Google Cloud storage
  Pros:
  1. Fine grained object storage which is convenient for images
  2. Allows us to generate public url.
  3. Easier integration with codebase on Google Cloud Platform.
  Cons:
  1. No Experience with using the Python API for interface
- AWS S3
  Pros:

1. Same as Cloud storage but with cheaper pricing.

Cons:

1. Not as easy integration with Google Cloud Services.

Conclusion: We will be using Cloud storage since it integrated well with out code base.

8.1.5 Real World Constraints

- It is currently constrained to a 5120x5120 image. It currently supports two types of tissues; however, on retraining it can support multiple regions.

8.2 System Architecture Overview

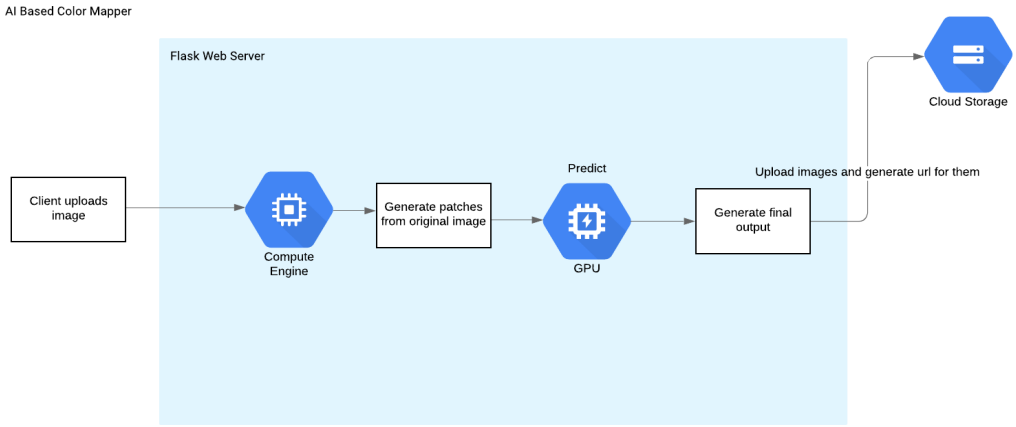

Figure 24: System Architecture

9 Requirements

9.1 User Stories

1. As a researcher, I want to train the model easily using a different set of images to do Al-based color matching and conversion on a different type of image.

2. As a data collector, I want to upload various formats of images fast.

3. As a data collector, I want to do image processing in a batch and quickly.

4. As a pathologist, I want to pass the microscopic images into the system and get the HE images in real-time for pathologists to analyze.

5. As a researcher, I'd like the images to be converted correctly regardless of image distortion or noise.

6. As a researcher, the mechanism should not add any new structures as GAN does.

7. As a pathologist, we want the structures to be translated independently of the color intensity so that structures are not interchanged.

10 Github Repository

Our Github Repository can be found in the following link:
https://github.com/atharvacc/MachineChurning

11 Technologies Employed

- Cloud Services
  - Google Cloud Engine
  - Google Cloud Storage
- Programming Languages
  - Python
  - HTML
  - JavaScript
  - MatLab
- Libraries
  - Bootstrap
  - CSS
  - TensorFlow
  - PyTorch
  - Skimage
  - Numpy

12 Social/Legal Aspect of Product

This project is done in conjuction with the University of California, Davis under the supervision of Professor Xin Liu. Our client is Dr. Richard Levenson. The intellectual property is reserved.

13 Moving Foward

- Extend our software to make it compatible with gigabyte sized images
- Continue to improve the accuracy of the conversion especially in the areas where the shape is not clear, or the color is not consistent
- Improve the speed of the model as it is the key to industrial use
- Beautify interface

14 References

1. Samet Hicsonmeza, Nermin Samet, Emre Akba, Pinar Duygulu, GANILLA: Generative Adversarial Networks for Image to Illustration Translation
2. Xinyang Li1, Guoxun Zhang, Jiamin Wu, Hao Xie, Xing Lin, Hui Qiao, Haoqian Wang,Qionghai Dai, Unsupervised content-preserving image transformation for optical microscopy