112 TP1 Design Proposal

Ethan Lu, AndrewID: ethanlu

November 14, 2018

1 Project Overview

1.1 Project Description

1.2 Competitive Analysis

1.3 Structural Plan

1.4 Algorithmic Plan

Style Transfer: Cool subset of Machine Learning that seeks to quantify the "style" or an image or painting, then apply it to other images. Examples:

Doing Style Transfer in Real Time: Issue with generalized style transfer algorithms for real time video: Very computationally expensive. For arbitrary video streams, based on the algorithm used, single frames can take anywhere from seconds to minutes to process, even on (very) good hardware. Doesn't preserve temporal coherence, frequently resulting in "flickering" between frames.

My project aims to implement real-time style transfer on a very narrow subset of "arbitrary video streams:" video games.

The output from video games is algorithmically generated, computationally using a combination of in-game models and textures to specify color. Thus, we can "style transfer" onto the assets/textures used by the game, effectively getting similar results. Analogy: doing style transfer at "compile-time" instead of "run-time."

Algorithms: Terms: Style Image: the image from which the "Style" of the desired output image is extracted. Input/Content Image: the image that dictates the "content" and overall geometry of the desired output image

Convolutional Neural Network + Gradient Descent: Using a pretrained neural network (traditionally, vgg16/vgg19), we can quantify "style" and "content" differences between two image by looking at various layers of the convolution. To do this, we simply run each image through the neural network, examine the desired layer, and take the RMS difference between each component in the tensor. This gives us two different notions of "distance": style distance and content distance. Our goal is to minimize these.

To do this, we simply compute the gradients of each entry of the "image tensor," and perform gradient descent on the entire tensor.

After the desired number of iterations, we return the output image and save it to a file.

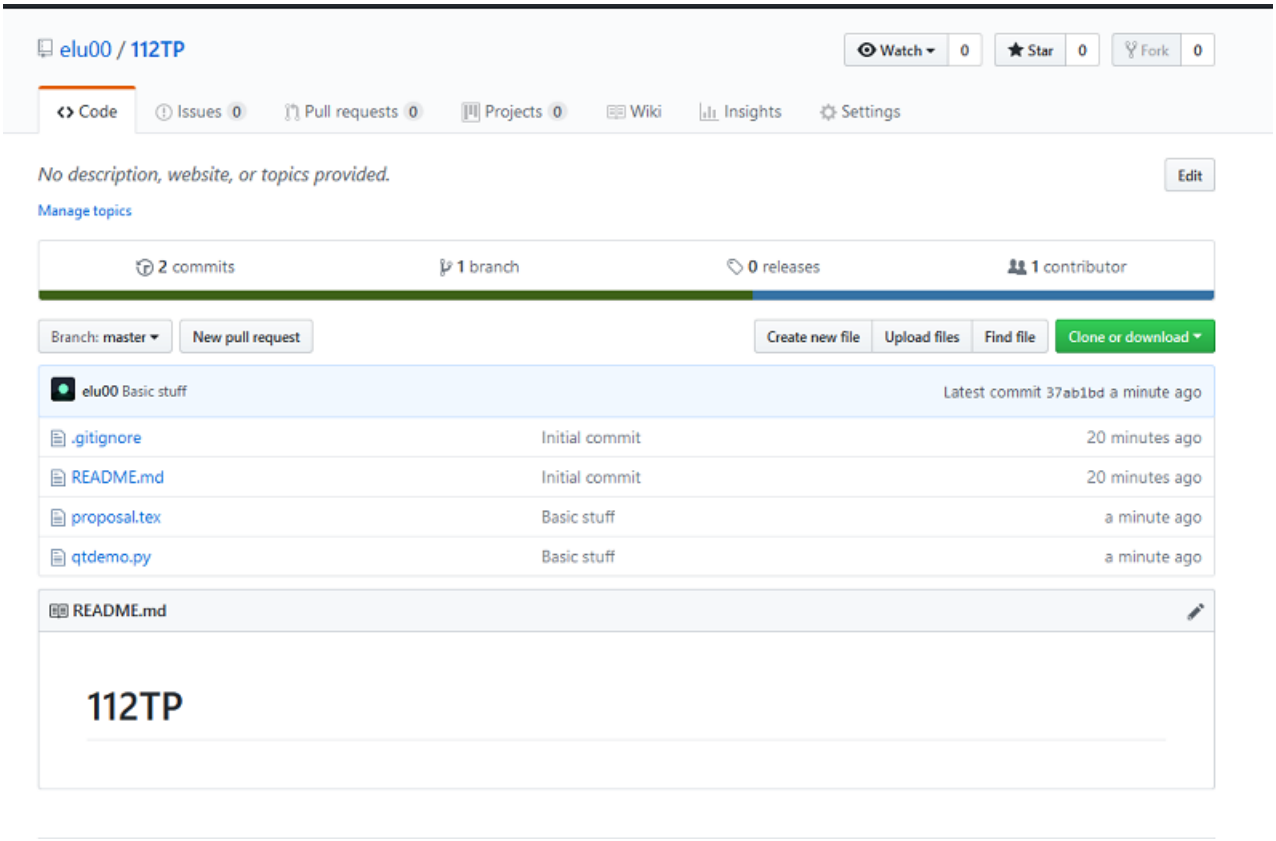
Cycle GANs: Two primary components: Generator Discriminator First, we write a generator that naively does some basic operations on the input/style images. Most importantly, this generator is very fast, and can produce multiple images from a given pair of inputs very quickly.

Then, we write a discriminator that can pick the "best" image out a set of candidate images, and run the generator again on this "best" image. As part of the "cycle" aspect, we also frequently regularize the outputs of this as to not get stuck in local maxima. Complete toolchain: Main technologies: Pytorch, QT, Dolphin Emulator (for actually running the games) Workflow: (Optional/computationally demanding) Extract textures from the game using Dolphin Emulator, facilitated by a QT GUI Parse the extracted textures from dolphin with Python builtin file operations Run the style transfer algorithms on above images with custom implementations in Pytorch Replace the original textures in dolphin Play the game again, using a button somewhere in the GUI.

1.5 Timeline Plan

1.6 Version Control Plan

I'll be using GitHub to backup/version control my code. The repo I'll be using can be found at https://github.com/elu00/112TP.



1.7 Module list

1. Pytorch
2. PyQT

2 Storyboard