Image Sharpening using Autoencoders

Andrew Fausak
Phillip Merritt
Ryan Moye
Theophilus Medeiros

Abstract

Photos can be blurry or pixelated for a variety of reasons, however we seldom want to keep these pixelated images. The goal of image upscaling is to take these photos and create a cleaner version of them. Our UI takes in an image or video as input, then passes it to our backend using Flask. Once our backend has the input, it applies upscaling to the image or to each individual video frame. The upscaled version is then pushed to our front end to be compared to the original input. The applications of this software can improve cameras, surveillance, videos and gaming.

Project Motivation and Goals

- We chose to do this project so that we could learn more about image upscaling.
- We wanted to learn about the following:
  - Autoencoder architecture
  - Generative models
  - Image upscaling
  - Pattern Recognition
- We also wanted to implement our model on a website so that users can input their images and get an upscaled output.

Design

The design of our model is a simple convolutional autoencoder. We take the input frame or image and reduce it down from 256 to 128 to 64 on the decoder side, then decode back to the original input size.

To create our inputs for training, we take the original image and pixelate it to create a degraded version, then we extract patches of size 100x100 pixels from both version, finally we pass a handful of sample patches from the degraded image through the model. The model attempts to sharpen each of the patches and the loss (mse) is calculated by comparing the output patch to the original, non-degraded patch.

Our frontend takes image or video data as input and passes it to the backend for processing, then receives the upscaled image or video and displays it for comparison with the original input.

Milestones

I. Finish Proposal (9/30/2020):

Finalize the proposal and turn it in.

II. Complete Model Research (9/30/2020):

Finish researching the different generative models used in frame interpolation and choose one for the project.

III. Project Update (10/07/2020):

Update of the current progress of the project.

IV. Project Report and Presentation (10/14/2020):

Final project report and presentation.

Data Specifications

- We used the Places dataset: specifically the validation set as the full training set was over 200gb.
- Our dataset consists of 36,500 images.
- We did a random split of 10% to divide these images into a training set of 32,850 training images and 3650 validation images.
- These images consist of people, landscapes, architecture and cityscapes.

Model Architecture

- We chose to use an autoencoder architecture approach for this project.
- The autoencoder takes in the blurred image patch as input, reduces the image to a latent dimensionality, then decodes it back into a sharpened image patch.
- The decoded patch is compared to the original unblurred patch, and the loss (mse) is calculated based on the pixel value differences between the two.
- To work with larger images, we implemented a patching technique that takes 100x100 pixel patches of the image and runs them through the model then the upscaled patches are stitched back together by averaging the overlaps.
- The model returns the upscaled image in less than 10 seconds for most image sizes. While this is acceptable for images, a 30 fps video takes almost 2 minutes per second of the video to upscale. E.G. a 2 minute, 30 fps video will take approximately 228 minutes to return the upscaled video.

Website Setup

- We implemented a frontend that allows the user to input an image and receive an upscaled version as output.
  - CORS issue was resolved with flask-restful API on the backend
  - For the frontend we utilized the following:
    - Vuejs - framework
    - Vuetify - for frontend icons, gui and built in components
    - Nodejs - for frontend server
    - Prettier - for code ‘prettiness’
    - Axios - for REST calls
    - File reader API for image upload
- We used Python Flask to implement our model on the backend of the website.

Results: Patching Issue

When initially training on large images with patches, we noticed that in certain areas the patches were not stitched back together correctly.

This was simply a training issue, and was resolved by using smaller strides and more overlaps when training.

Results: Artifacts

We ran into an issue with these colored artifacts appearing in the sharpened image. The issue was with out of range values wrapping around. E.G. -0.1 would wrap around to 0.9, creating an incorrect color value.

This issue was corrected by clipping values to the range [0, 1] to ensure that the colors remained in the correct range.

Results: Proper Upscaling

Backend Demo

Results

- Our model performs a variety of upscaling methods, e.g. 2x and 4x scaling.
- We see the best results from poor quality images as input.
- When using a high quality image as input, the output image typically only has changes in contrast, and not in pixel quality.

Future Directions for this Project

- Performing hyperparameter tuning on the model can be done to achieve higher quality output images and potential speed increase.
- Further development of the web frontend.

Repository and Archive

- Our Github repository can be found at:
  - https://github.com/PhillipMerritt/frameSharpening