Giorgio Mendoza
CS549-E24-E01
Dr. Alexandros Lioulemes

Week 9 Lab

Part B_1:

Objective:

The objective of this task is to explore the use of autoencoders for image compression and denoising by following the examples provided in the TensorFlow tutorial. Specifically, the task involves creating two Python scripts: one for compressing and reconstructing images using a basic autoencoder, and another for removing noise from images using a denoising autoencoder.

The goal is to understand how autoencoders can learn efficient representations of input data in a lower-dimensional latent space and how they can be used to reconstruct clean images from noisy input.

Code Description (basic_autoencoder.py):

In this section, a basic autoencoder was implemented using TensorFlow and Keras. The Fashion MNIST dataset is used as the input data, consisting of grayscale images of size 28x28 pixels. The autoencoder is constructed with a simple encoder-decoder architecture:

The encoder compresses the input images into a latent representation of size 64 using a flatten layer followed by a dense layer with ReLU activation.

On the other hand, the decoder reconstructs the original images from the latent representation using a dense layer with sigmoid activation followed by a reshape layer to return to the original image size.

The model is trained for 10 epochs, using the mean squared error as the loss function. After training, the script displays the original and reconstructed images side by side, allowing visual assessment of the autoencoder's performance.

Code Description (image_denoising.py):

The goal of this part was to implement a denoising autoencoder to remove noise from images. The same Fashion MNIST dataset is used, but with added Gaussian noise to simulate corrupted data. The denoising autoencoder is composed of the following layers:

In this case, the encoder uses two convolutional layers to compress the noisy images into a lower-dimensional representation, with ReLU activation and strides of 2 to downsample the input.

The decoder consists of two transposed convolutional layers that upsample the encoded representation to reconstruct the original clean images. The final convolutional layer uses sigmoid activation to output pixel values in the range [0, 1].

The model is trained for 10 epochs using mean squared error as the loss function. After training, the script displays the noisy input images and the corresponding denoised output images, demonstrating the effectiveness of the autoencoder in filtering out noise while preserving important visual details.

Results:

The results from the basic autoencoder demonstrate its ability to compress and reconstruct the Fashion MNIST images. As shown in the comparison, the reconstructed images (bottom row) capture the general structure and features of the original images (top row), although with some loss in detail and sharpness. The reconstructed images retain the overall shape and outline of the clothing items, illustrating that the autoencoder successfully learned a compact representation of the input data in its latent space. This result shows the potential of autoencoders for tasks like dimensionality reduction and feature extraction.

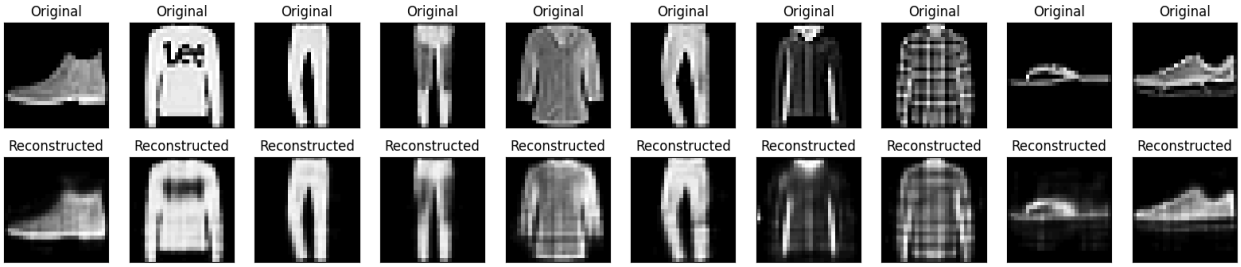

Figure 1. Basic_autoencoder.py results

The results from the image denoising autoencoder illustrate its ability to remove noise from corrupted Fashion MNIST images. The top row shows the noisy input images, which have been purposely corrupted with random noise. The bottom row shows the output from the denoising autoencoder, where lots of the noise has been successfully removed, restoring the original appearance of each item.

The denoised images retain the key features of the original images, such as shapes and outlines, while significantly reducing the noise.

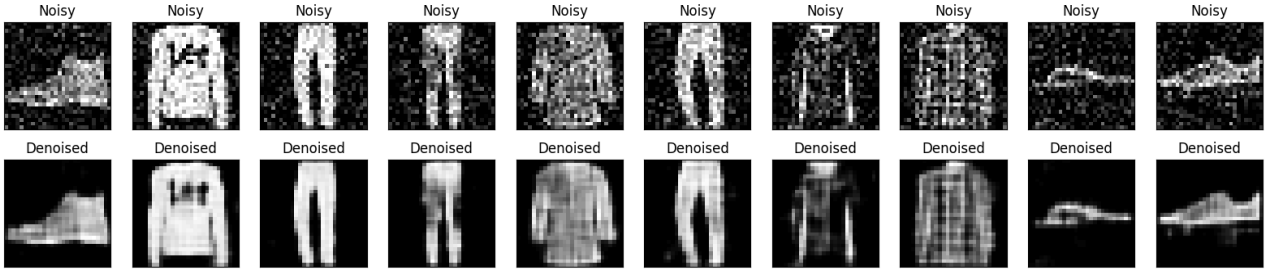

Figure 2. Image_denoising.py results

Part B_2:

Objective:

The objective of this task is to implement a Variational Autoencoder (VAE) using TensorFlow and Keras to learn and generate images from the MNIST dataset. The VAE is trained to compress images into a lower-dimensional latent space and then reconstruct them, allowing for the generation of new images based on this learned latent space. The goal is to understand the principles of generative models and apply them to generate and visualize new digit images.

Code Description:

This implementation starts by defining the architecture of a Convolutional Variational Autoencoder, consisting of an encoder and a decoder. The encoder compresses the input images into a latent space, represented by a mean and log-variance, which are then used to sample a latent vector.

The decoder reconstructs the images from the sampled latent vectors. The loss function combines the reconstruction loss and the Kullback-Leibler divergence, encouraging the learned distribution to be close to a normal distribution. The model is trained on the MNIST dataset over multiple epochs, and during each epoch, the model's performance is evaluated using the ELBO.

Lastly, the generated images are saved and displayed at each epoch, providing a visual representation of the model's learning progress as shown below. This part was also implemented in Google Collab to its GPU.

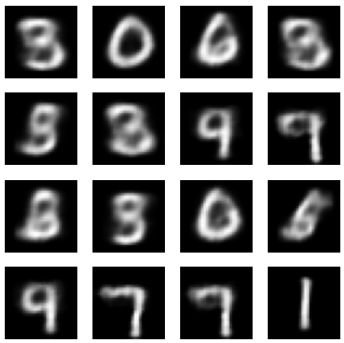

Figure 3. generation.py results

Part B_3:

Objective: The objective of this task is to apply Neural Artistic Style Transfer to a real-time camera feed using a pre-trained VGG19 model. The goal is to blend the content of the camera input with the style of a chosen reference image, creating an artistic effect that is displayed in real-time.

Code Description: The implementation begins by loading the VGG19 model, pre-trained on the ImageNet dataset, which is used to extract intermediate layer outputs crucial for style transfer. The StyleContentModel class is designed to compute the style and content representations of the input image using these intermediate layers. The style of an image (Van Gogh) is captured using a Gram matrix, which reflects the correlations between different feature maps.

The style transfer process is executed within a real-time loop that captures frames from the user's camera. Each frame is treated as the content image, while a pre-loaded style image is used to calculate the style targets. The content and style representations are then blended by minimizing a loss function that combines content loss, style loss, and a total variation loss, which helps in reducing high-frequency noise in the output. The resulting image, which merges the camera frame with the style of the reference image, is displayed continuously until the user decides to exit.

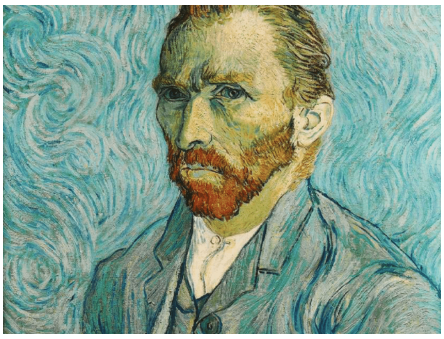

Figure 3. Reference Van Gogh image

Results: As shown in the results, the Van Gogh style was successfully applied to the camera images, giving them a characteristic swirl and color pattern similar to his work. Despite the computational challenges of running style transfer in real-time, the approach was effective in showcasing the capabilities of neural networks in artistic applications. The resulting images reflect the unique texture and colors of the Van Gogh style, applied directly to real-world objects captured through the camera.

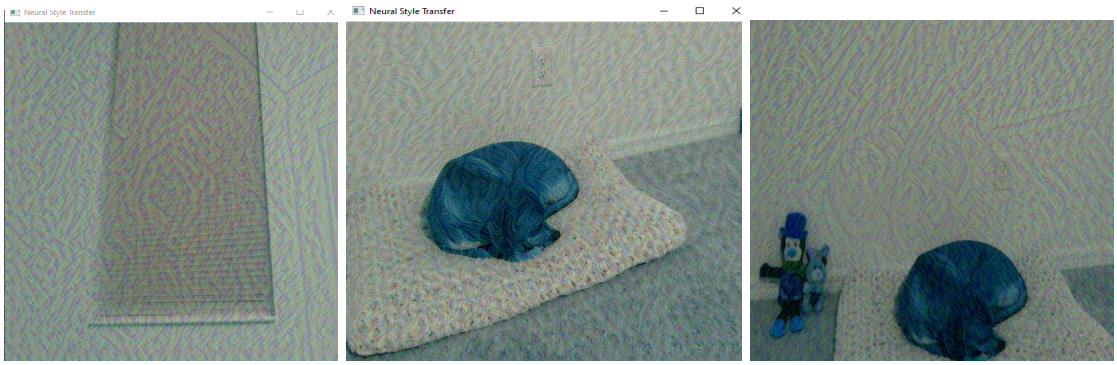

Figure 3. camera_nst.py results

However, the real-time camera feed was significantly slow when using the optimized parameters that produced the best visual effects, as shown above. This slowdown occurred because my PC was unable to

utilize the GPU due to an issue with NVIDIA CUDA, resulting in Python being unable to access GPU acceleration.

Num GPUs Available: 0
PS C:\Users\Gio\Desktop\tinyml\archive>