COMP 4211: Machine Learning

Programming Assignment 2

Authors

HUANG, I-wei - 20824074

March 26, 2024

Contents

1 Datasets and Data Loaders **2**
1.1 [Q1] ImageDataset . . . 2
1.2 [Q2] ClassificationDataset . . . 2

2 Style Transfer **3**
2.1 [Q3] . . . 4
2.2 [Q4] . . . 4
2.3 [Q5] . . . 4
2.4 [Q6] . . . 4
2.5 [Q7] . . . 5
2.6 [Q8] . . . 5

3 Classification **6**
3.1 [Q9] . . . 6
3.2 [Q10] . . . 6
3.3 [Q11] . . . 6
3.4 [Q12] . . . 7
3.5 [Q13] . . . 7
3.6 [Q14] . . . 7
3.7 Data Augmentation by Style Transfer . . . 8
3.8 [Q15] . . . 8
3.9 [Q16] . . . 8
3.10 [Q17] . . . 9
3.11 [Q18] . . . 9

1 Datasets and Data Loaders

For this programming assignment, we will focus on two major tasks: style transfer and image classification. The COCO (Common Objects in Context) and WikiArt datasets are utilized for style transfer, and the PACS (Photo-Art-Cartoon-Sketch) dataset is used for image classification. Each task requires different datasets, hence different data loaders are required to manage the data efficiently during training. Two Dataloader classes are implemented for the tasks: `ImageDataset` and `ClassificationDataset`.

1.1 [Q1] ImageDataset

`ImageDataset` is a data loader that, given a directory path and a specified file extension, finds all files in the directory with the matching file extension and loads them on demand during training. These files are not labeled and are used for the Style Transfer task. Table 1 displays the dataset sizes of COCO and WikiArt based on the implemented `ImageDataset`.

Dataset	Size
COCO dataset	3557
WikiArt dataset	7492

Table 1: Dataset size of COCO and WikiArt

1.2 [Q2] ClassificationDataset

`ClassificationDataset` is a data loader that, when given a path to a directory and a `.tsv` file, loads the files listed in the TSV file, and provides their perspective labels during training. Table 2 showcases the results of calling the `__len__` function on the PACS training dataset and PACS testing dataset that are loaded using the implemented `ClassificationDataset`.

Dataset	Size
PACS training dataset	1641
PACS test dataset	2723

Table 2: Dataset size of PACS

2 Style Transfer

The style transfer implements the AdaIN (Adaptive Instance Normalization) style transfer model to combine content and style from two distinct images. The model will accept a content image, a style image, and an alpha blending parameter as inputs. These inputs pass through an encoder to produce feature maps, which are then altered by the AdaIN layer to match the mean and variance of the style feature maps to the content feature maps. The modified content feature map is then passed through a decoder to generate the stylized image. Figure 1 provides a conceptual overview of the AdaIN style transfer model.

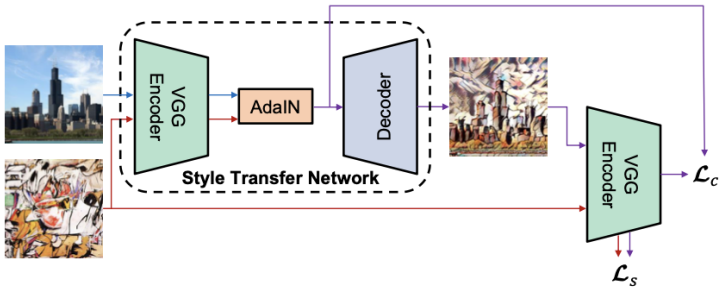

Figure 1: An overview of the AdaIN style transfer algorithm

Encoder:

The encoder employs a partial VGG19 architecture. Table 3 illustrates the detailed structure of the model layers. The encoder is loaded with pre-trained weights trained on a large dataset, providing a strong feature extraction capability for the style transfer task.

Layer Type	Configuration	Output Size	Name
Convolution + ReLU	64 filters, 3x3, same padding	Input Size	`conv1_1`
Convolution + ReLU	64 filters, 3x3, same padding	Input Size	`conv1_2`
Max Pooling	2x2, stride 2	Input Size / 2	`pool1`
Convolution + ReLU	128 filters, 3x3, same padding	Input Size / 2	`conv2_1`
Convolution + ReLU	128 filters, 3x3, same padding	Input Size / 2	`conv2_2`
Max Pooling	2x2, stride 2	Input Size / 4	`pool2`
Convolution + ReLU	256 filters, 3x3, same padding	Input Size / 4	`conv3_1`
Convolution + ReLU	256 filters, 3x3, same padding	Input Size / 4	`conv3_2`
Convolution + ReLU	256 filters, 3x3, same padding	Input Size / 4	`conv3_3`
Convolution + ReLU	256 filters, 3x3, same padding	Input Size / 4	`conv3_4`
Max Pooling	2x2, stride 2	Input Size / 8	`pool3`
Convolution + ReLU	512 filters, 3x3, same padding	Input Size / 8	`conv4_1`

Table 3: Structure of the encoder

AdaIN Layer:

Given content input x and style input y, the AdaIN operation is defined as follows:

$$\text{AdaIN}(x, y) = \sigma(y)\left(\frac{x-\mu(x)}{\sigma(x)}\right) + \mu(y)$$

where $\mu(x)$ and $\sigma(x)$ represent the mean and standard deviation of the content features, and $\mu(y)$ and $\sigma(y)$ represent those of the style features.

Decoder:

The decoder is an essential component of the style transfer model that reconstructs an image from the encoded features, allowing the transferred style to be visualized in the output image. Table 4 shows the architecture of the decoder model.

Layer Type	**Configuration**	**Output Size**
Convolution + ReLU	256 filters, 3×3, same padding	Input Size
UpSampling	2×2	2×Input Size
Convolution + ReLU	256 filters, 3×3, same padding	2×Input Size
Convolution + ReLU	256 filters, 3×3, same padding	2×Input Size
Convolution + ReLU	256 filters, 3×3, same padding	2×Input Size
Convolution + ReLU	128 filters, 3×3, same padding	2×Input Size
UpSampling	2×2	4×Input Size
Convolution + ReLU	128 filters, 3×3, same padding	4×Input Size
Convolution + ReLU	64 filters, 3×3, same padding	4×Input Size
UpSampling	2×2	8×Input Size
Convolution + ReLU	64 filters, 3×3, same padding	8×Input Size
Convolution	3 filters, 3×3, same padding	8×Input Size

Table 4: Structure of the decoder

2.1 [Q3]

In the encoder, the input image undergoes a series of convolutional and pooling layers. Convolutional layers help extract feature maps, increasing their depth to capture more abstract features of the input, and pooling layers reduce the spatial dimensions of these feature maps while preserving their depth. The decoder performs the reverse operation: the upsampling layers increase the spatial resolution of feature maps to recover the spatial information lost during downsampling operations, while the convolutional layers decrease their depth, ultimately converting them back to the standard three channels (red, green, blue) of an output image.

2.2 [Q4]

Pooling and upsampling layers do not have trainable layers. The formula for calculating trainable parameters in a convolutional layer is:

$$\text{Parameters} = (\text{Kernel Width} \times \text{Kernel Height} \times \text{Input Channels} + 1) \times \text{Output Channels}$$

Using the formula the number of trainable parameters in the decoder is 3505219. Since the encoder uses pre-trained weights, it does not have trainable parameters.

2.3 [Q5]

Both architectures of the encoder and decoder utilize convolutional layers extensively, but their purposes differ, as stated in Section 2.1. In the encoder, convolutional and pooling layers reduce spatial dimensions while increasing depth, focusing on feature abstraction, whereas, in the decoder, convolutional and upsampling layers increase spatial dimensions and reduce depth, aiming to reconstruct or generate detailed images from abstracted representations.

2.4 [Q6]

The training of a style transfer model has two fundamental objectives: preserving the content of the input image and adopting the features of a style reference image. The combination of style loss and content loss is crucial because it provides a balanced mechanism to effectively guide the model toward achieving these dual objectives. By using a combination of style loss and content loss, we can achieve controlled style transfer that maintains both the content's integrity and the aesthetic of the style. This allows the creation of images that seamlessly blend content and style, preserving the key features of the original image while applying the style in a visually appealing way.

2.5 [Q7]

The initial AdaIN style transfer model was trained with a batch size of 16 for 15 epochs. Figure 2 shows the loss over the epochs. The overall total loss is 17747.963. A second model was trained with a batch size of 8 for 15 epochs, and the final total loss was 8093.251. Figure 3 depicts the loss across epochs. The reason for this could be that with a lower batch size, the model is exposed to a broader range of instances in each iteration. This greater exploration can help the model identify and learn from diverse patterns and variances in the data, resulting in better generalization and lower loss.

Figure 2: Training loss of AdaIN model (batch size 16)

Figure 3: Training loss of AdaIN model (batch size 8)

2.6 [Q8]

Figure 4 demonstrates how the model efficiently combines the content and style of photos. Two samples, a photo of myself and a shot of HKUST, are changed into five various styles: Monet, Picasso, Chinese, Van Gogh, and Ukiyo-e.

Figure 4: Examples of style transfer

3 Classification

This classification task implements a Convolutional Neural Network (CNN) model to solve a multiclass classification problem using the PACS dataset, which contains images of various objects and styles. The goal of this task is to predict the object labels given the images.

3.1 [Q9]

Figures 5 and 6 show a comparison of image styles and labels in the training and test datasets. It is revealed that, while the testing dataset has evenly distributed images for each style and label, the training images available have a non-ideal distribution across styles and labels.

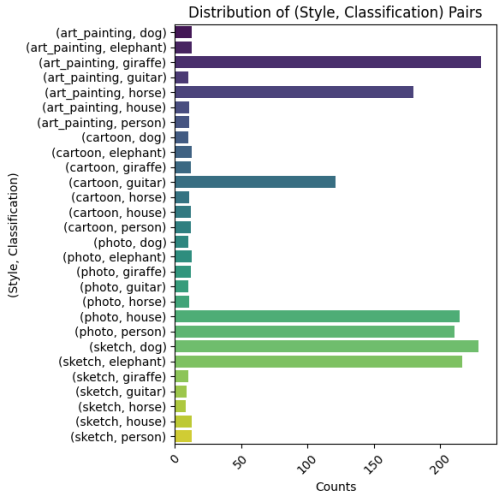

Figure 5: PACS training dataset distribution

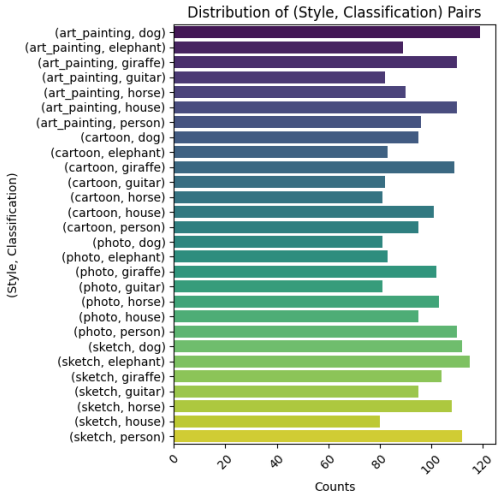

Figure 6: PACS testing dataset distribution

3.2 [Q10]

The distribution disparity may lead to biased predictions and poor generalization in test-time performance. The model may be biased toward labels or styles that are overrepresented in the training data. As a result, it may have difficulty accurately classifying images from underrepresented labels or styles during the testing phase.

3.3 [Q11]

The classification model is a simple extension of the VGG19 model in Table 3. In particular, the following layers in Table 5 are added.

Layer Type	**Output Size**	**Name**
Global Average Pooling	512 dimension	`global_pool`
Dense + ReLU	1024 dimension	`dense1`
Dense + ReLU	1024 dimension	`dense2`
Dense + ReLU	512 dimension	`dense3`
Dense + Softmax	Number of classes	`dense4`

Table 5: Additional Layers for the VGG19-based classifier model

The formula for calculating trainable parameters in a Dense (Fully Connected) Layer is:

$$\text{Parameters} = (\text{Number of Input Units}+1) \times \text{Number of Output Units}$$

Using the above formula and the formula for convolutional layers mentioned in Section 2.2, the number of trainable parameters in the classification model is 5609031.

3.4 [Q12]

The model is trained with the training dataset for 100 epochs. Figure 7 records the loss obtained throughout training, the final loss is 0.102.

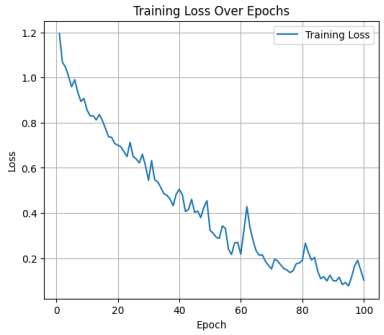

Figure 7: Training loss of Classification model

3.5 [Q13]

When tested against the training and test datasets, the training accuracy is 0.945, and the test accuracy is 0.467. Figure 8 shows the confusion matrices.

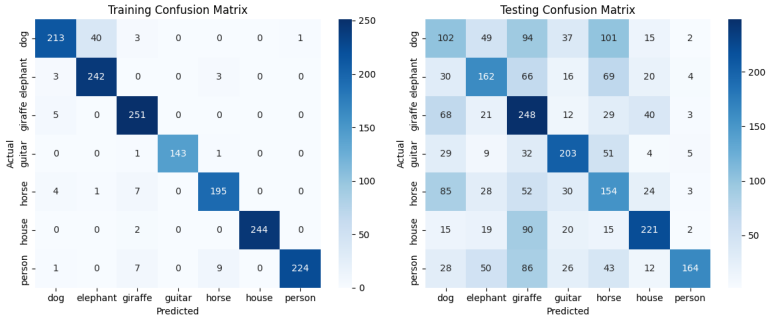

Figure 8: Confusion matrices of the model when tested against the training and test datasets

3.6 [Q14]

Figure 9 and Figure 10 showcase some mislabeled images with their true and reported samples when tested against the training and test datasets, respectively. Figures 8 and 9 show that dogs are frequently mislabeled as elephants, particularly in sketch styles. It is possible that the sketches of dogs and elephants are just too simila. On the other hand, Figure 10 does not indicate any particular style and label being mislabeled the most, which is consistent with the confusion matrix and that the model's performance is simply not that excellent.

Figure 9: Training dataset mislabeled images with their true and reported labels

Figure 10: Testing dataset mislabeled images with their true and reported labels

3.7 Data Augmentation by Style Transfer

Since the training images for the classification task are in a non-ideal distribution, data augmentation using the AdaIN style transfer model in Section 2 is implemented to generate new samples using the available training data. The idea is to pick an image A with label l, and another B with style s to create a new image C with A as content image and B as style image. Figure 11 illustrates the distribution of the training dataset with the addition of the newly generated images, each style and label are expanded with 100 new images.

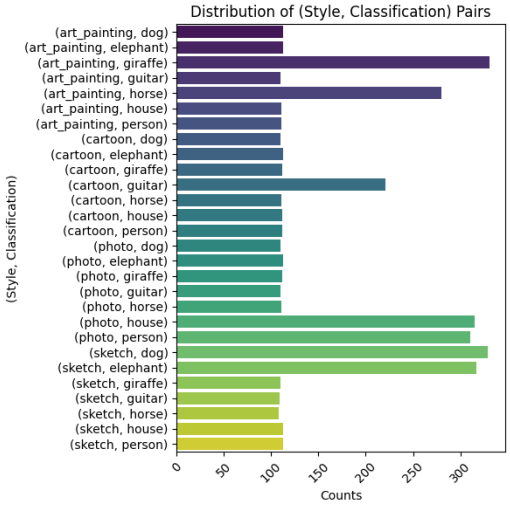

Figure 11: Distribution of the dataset with newly generated images

3.8 [Q15]

Figure 12 displays some samples from the generated dataset, with all seven object labels, and all four styles.

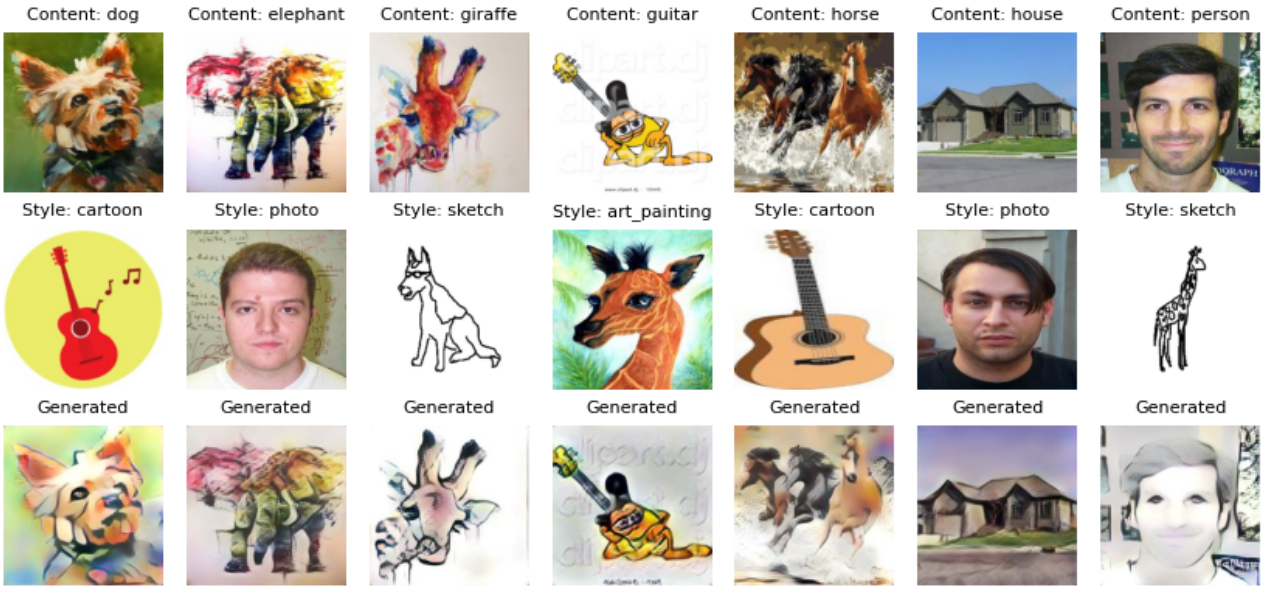

Figure 12: Samples from the generated dataset

3.9 [Q16]

A new model is trained using a dataset combining both the given and generated training datasets for 100 epochs. Figure 13 records the loss obtained throughout training, the final loss is 0.153.

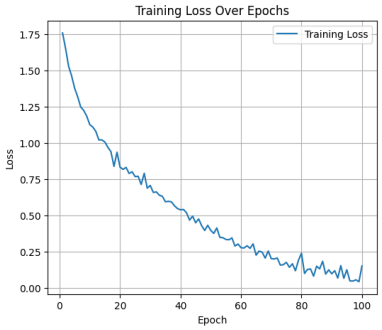

Figure 13: Training loss of Classification model

3.10 [Q17]

When tested against the un-augmented training and test datasets, the training accuracy is 0.934, and the test accuracy is 0.492. Figure 8 shows the confusion matrices.

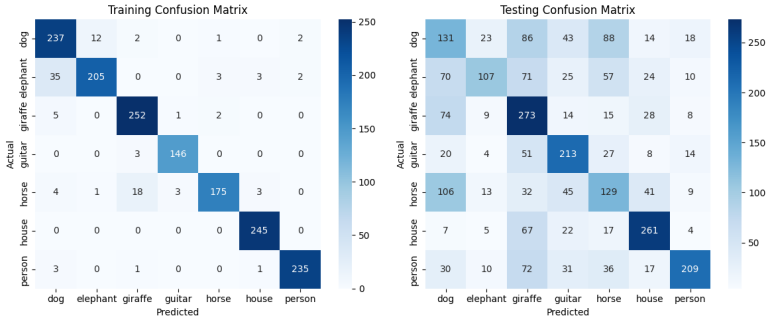

Figure 14: Confusion matrices of the model when tested against the training and test datasets

3.11 [Q18]

According to the test-time performance, the model trained with the augmented dataset has a slightly better accuracy. This might be due to the model training with more samples from each label and style, resulting in a better generalization. However, the difference is insignificant, indicating that further strategies will be required to improve the model's performance.