COMP4211 Programming Assignment 2 Report

Name: CHAN, Chun Hin

Student ID: 20853893

Email: chchanec@connect.ust.hk

Table of Contents

[Q1]

Implement ImageDataset according to the above description, and report the result of __len__ when loading the COCO and WikiArt dataset respectively.

For loading the WikiArt dataset, the result of __len__ is 7492.

For loading the COCO dataset, the result of __len__ is 3557.

[Q2]

Implement ClassificationDataset according to the above description, and report the result of __len__ when loading the PACS training and test datasets respectively.

For loading the PACS training dataset, the result of __len__ is 1641.

For loading the PACS testing dataset, the result of __len__ is 2723.

[Q3]

What is the usage of upsampling layers?

The usage of upsampling layers is to enlarge the size of the image by repeating the rows and columns of pixels multiple times. It is aim at bring back the higher resolution of the picture to the resolution in one pooling layer before. Please note that upsampling layerscannot bring back any lost information when we are doing maxpooling to the image.

[Q4]

Implement the decoder according to Table 2. Report the number of trainable parameters in the decoder.

The number of trainable parameters in the decoder is 3505219.

[Q5]

Compare the architectures of the encoder and decoder. Discuss the usage of the decoder architecture.

For the encoder, when we are going deeper of the model, the deeper convolutional layer uses more filters to extract more finite detail features of the image as the image size becomes smaller due to maxpooling. It is obvious to see that a maxpooling layer comes after a few convolutional layers to shrink the size of the image.

For the decoder, when we are going deeper of the model, the deeper convolution layer uses fewer filters to extract less coarse features of the image as the images size becomes larger due to upsampling. It is obvious to see that a upsampling layer comes after a few convolutional layers to enlarge the size of the image.

The usage of decoder architecture is to transform the encoded image information from the encoder into the original image after decoding. This means decoder helps to reconstruct the original image based on the encoded image features extracted from the encoder.

[Q6]

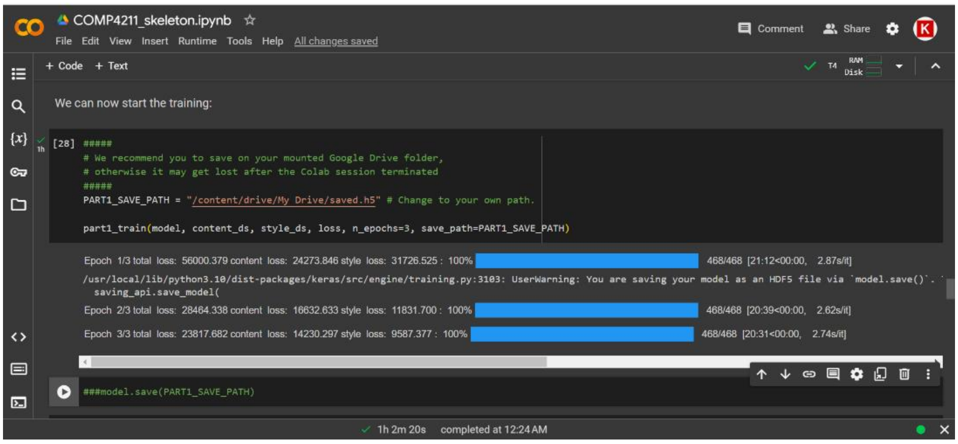
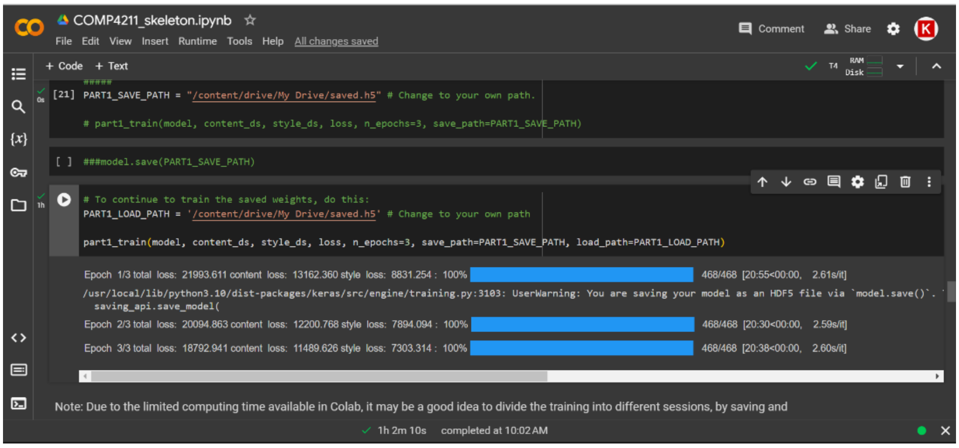
In the training of a style transfer model, why is it necessary to use a combination of style loss and content loss?

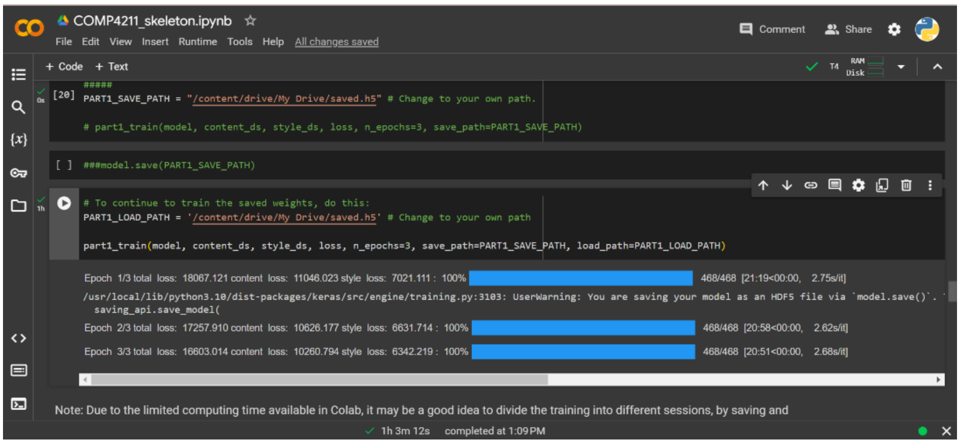
Recall that the content loss measures the differences between the generated image from the AdaIN and the original content image, while the style loss measures the differences between the generated image from the AdaIN and the original style image.

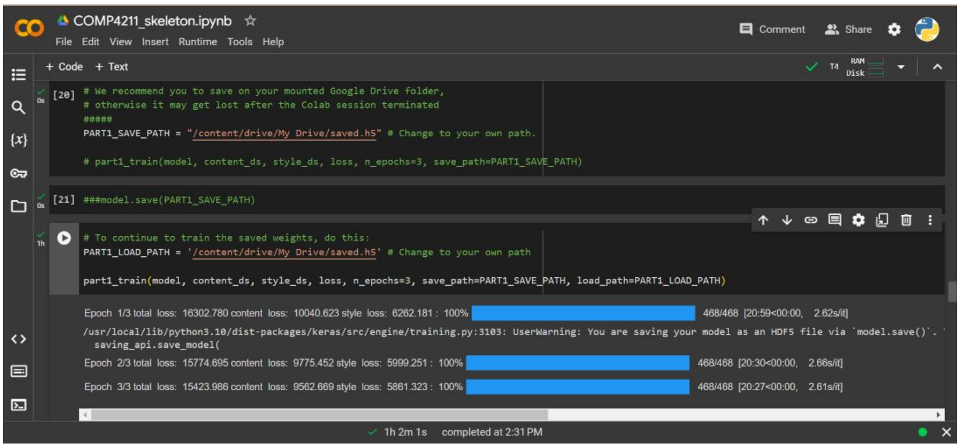
Be using a combination of style loss and content loss, this allows the model to generate an image that can take a balance on the original content information and the style information. This means that this allows the model to control the level of style being used and the level of the preserved content in a flexible manner. This makes both the content and the style blend together in a good portion and produce a good image that contains all the key features from the style image and the content image.

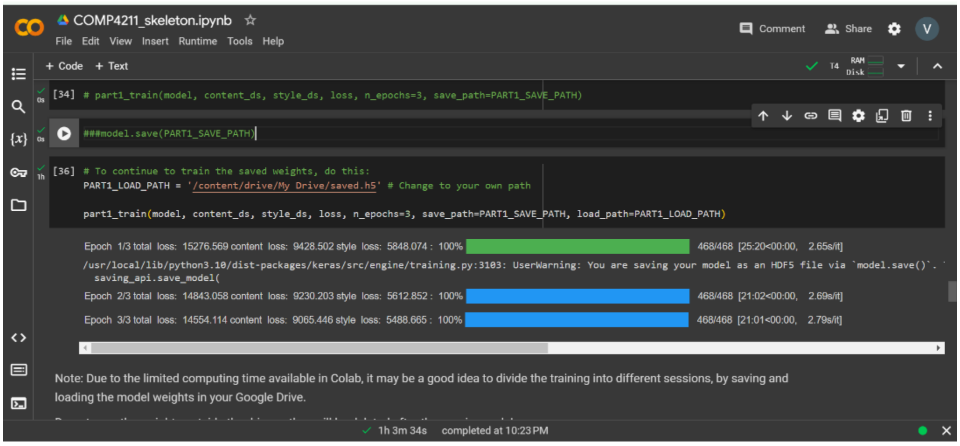
[Q7]

Train your model for at least 15 epochs aiming for a total loss of less than 18000. Report the loss obtained at the end of training.

<table>
<tr><th>Epoch</th><th>Screenshots</th></tr>
<tr><td>1 – 3</td><td>
</td></tr>
<tr><td>4 – 6</td><td>
</td></tr>
</table>

7 – 9	COMP4211_skeleton.ipynb
	File Edit View Insert Runtime Tools Help All changes saved
	Comment Share
	+ Code + Text
	[20] PART1_SAVE_PATH = "/content/drive/My Drive/saved.h5" # Change to your own path.
	# part1_train(model, content_ds, style_ds, loss, n_epochs=3, save_path=PART1_SAVE_PATH)
	[ ] ###model.save(PART1_SAVE_PATH)
	# To continue to train the saved weights, do this:
	PART1_LOAD_PATH = '/content/drive/My Drive/saved.h5' # Change to your own path
	part1_train(model, content_ds, style_ds, loss, n_epochs=3, save_path=PART1_SAVE_PATH, load_path=PART1_LOAD_PATH)
	Epoch 1/3 total loss: 18067.121 content loss: 11046.023 style loss: 7021.111 : 100% 468/468 [21:19<00:00, 2.75s/it]
	/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`.
	saving_api.save_model(
	Epoch 2/3 total loss: 17257.910 content loss: 10626.177 style loss: 6631.714 : 100% 468/468 [20:58<00:00, 2.62s/it]
	Epoch 3/3 total loss: 16603.014 content loss: 10260.794 style loss: 6342.219 : 100% 468/468 [20:51<00:00, 2.68s/it]
	Note: Due to the limited computing time available in Colab, it may be a good idea to divide the training into different sessions, by saving and
	1h 3m 12s completed at 1:09 PM

10 – 12	COMP4211_skeleton.ipynb
	File Edit View Insert Runtime Tools Help
	Comment Share
	+ Code + Text
	[20] # We recommend you to save on your mounted Google Drive folder,
	# otherwise it may get lost after the Colab session terminated
	PART1_SAVE_PATH = "/content/drive/My Drive/saved.h5" # Change to your own path.
	# part1_train(model, content_ds, style_ds, loss, n_epochs=3, save_path=PART1_SAVE_PATH)
	[21] ###model.save(PART1_SAVE_PATH)
	# To continue to train the saved weights, do this:
	PART1_LOAD_PATH = '/content/drive/My Drive/saved.h5' # Change to your own path
	part1_train(model, content_ds, style_ds, loss, n_epochs=3, save_path=PART1_SAVE_PATH, load_path=PART1_LOAD_PATH)
	Epoch 1/3 total loss: 16302.780 content loss: 10040.623 style loss: 6262.181 : 100% 468/468 [20:59<00:00, 2.62s/it]
	/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`.
	saving_api.save_model(
	Epoch 2/3 total loss: 15774.695 content loss: 9775.452 style loss: 5999.251 : 100% 468/468 [20:30<00:00, 2.66s/it]
	Epoch 3/3 total loss: 15423.986 content loss: 9562.669 style loss: 5861.323 : 100% 468/468 [20:27<00:00, 2.61s/it]
	1h 2m 1s completed at 2:31 PM

13 - 15	COMP4211_skeleton.ipynb
	File Edit View Insert Runtime Tools Help
	Comment Share
	+ Code + Text
	[34] # part1_train(model, content_ds, style_ds, loss, n_epochs=3, save_path=PART1_SAVE_PATH)
	###model.save(PART1_SAVE_PATH)
	[36] # To continue to train the saved weights, do this:
	PART1_LOAD_PATH = '/content/drive/My Drive/saved.h5' # Change to your own path
	part1_train(model, content_ds, style_ds, loss, n_epochs=3, save_path=PART1_SAVE_PATH, load_path=PART1_LOAD_PATH)
	Epoch 1/3 total loss: 15276.569 content loss: 9428.502 style loss: 5848.074 : 100% 468/468 [25:20<00:00, 2.65s/it]
	/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`.
	saving_api.save_model(
	Epoch 2/3 total loss: 14843.058 content loss: 9230.203 style loss: 5612.852 : 100% 468/468 [21:02<00:00, 2.69s/it]
	Epoch 3/3 total loss: 14554.114 content loss: 9065.446 style loss: 5488.665 : 100% 468/468 [21:01<00:00, 2.79s/it]
	Note: Due to the limited computing time available in Colab, it may be a good idea to divide the training into different sessions, by saving and
	loading the model weights in your Google Drive.
	1h 3m 34s completed at 10:23 PM

Epoch	Total loss	Content loss	Style loss
1	56000.379	24273.846	31726.525
2	28464.338	16632.633	11831.700
3	23817.682	14230.297	9587.377
4	21993.611	13162.360	8831.254
5	20094.863	12200.768	7894.094
6	18792.941	11489.626	7303.314
7	18067.121	11046.023	7021.111
8	17257.910	10626.177	6631.714
9	16603.014	10260.794	6342.219
10	16302.780	10040.623	6262.181
11	15774.695	9775.452	5999.251
12	15423.986	9562.669	5861.323
13	15276.569	9428.502	5848.074
14	14843.058	9230.203	5612.852
15	14554.114	9065.446	5488.665

I have trained my model for 15 epochs.

At the end of the training:

Total loss = 14554.114

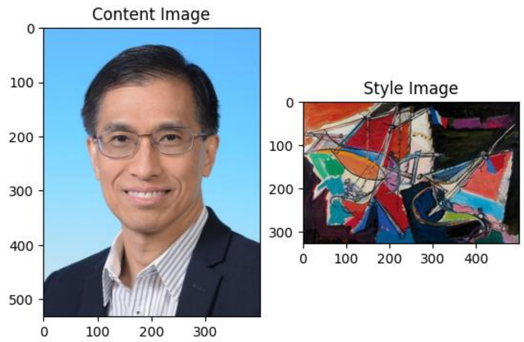
Content loss = 9065.446

Style loss = 5488.665

[Q8]

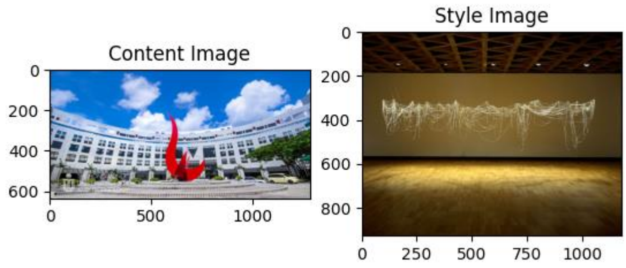
Demonstrate the model's ability to combine the content and style of images effectively, showcasing several examples in your report.

<u>Sample 1</u>

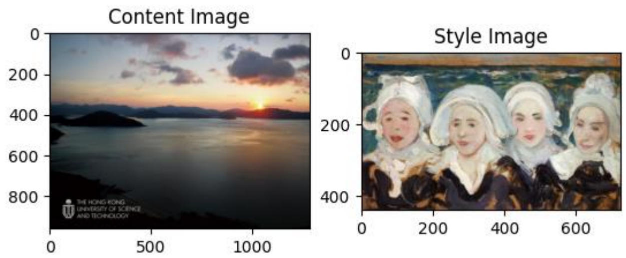
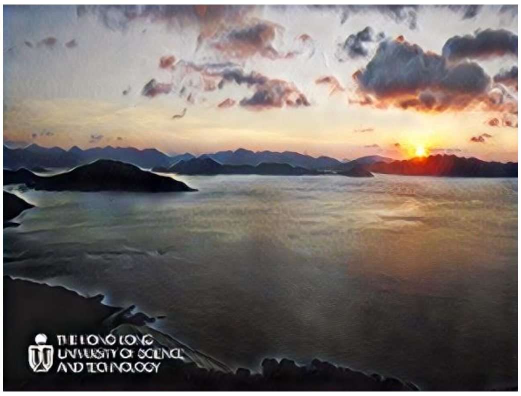
Description	Image
Content Image & Style Image	
When alpha = 0.2	
When alpha = 0.5	

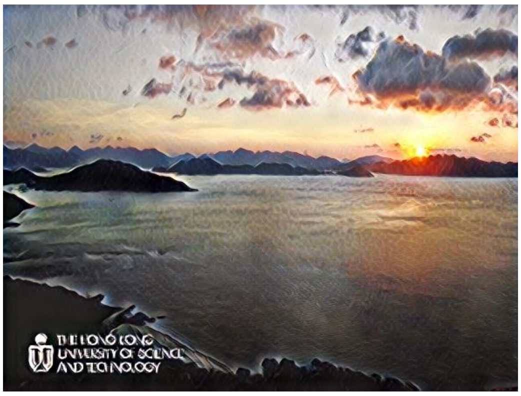
When alpha = 0.65	
When alpha = 0.9	

Sample 2

Description	Image
Content Image & Style Image	
When alpha = 0.2	
When alpha = 0.5	
When alpha = 0.65	
When alpha = 0.9	

<u>Sample 3</u>

Description	Image
Content Image & Style Image	
When alpha = 0.2	
When alpha = 0.5	
When alpha = 0.65	

When alpha = 0.9	

[Q9]

Compare the distribution of image styles and labels between the training and test datasets. What is the difference between the two datasets?

The first obvious difference is that the number of images in test dataset is greater than that in the training dataset.

The second difference lies in the difference of distribution of different labels.
For the training dataset, for each style, the image distribution of different labels is very uneven. Some image belongs to a few certain labels have much more number of images than those belongs to other labels. For example, in the style of cartoon, over 100 images are of “giraffe” label, while for rest of the labels, each contains around 10 images only.
However, the testing dataset, the image distribution of different labels is quite even (more even than that in training dataset). For each style number of images belong to each labels is evenly distributed.

```
Tallies of each pair of (style, label):


When the style is  sketch , the corresponding tally of each label is:
label_dog         229
label_elephant    217
label_giraffe      10
label_guitar        9
label_horse         8
label_house        13
label_person       13
dtype: int64


When the style is  art_painting , the corresponding tally of each label is:
label_dog          13
label_elephant     13
label_giraffe     231
label_guitar       10
label_horse       180
label_house        11
label_person       11
dtype: int64
```

Tallies of each pair of (style, label) for training dataset (Part 1)

```
When the style is  photo , the corresponding tally of each label is:
label_dog           10
label_elephant      13
label_giraffe       12
label_guitar        10
label_horse         11
label_house        215
label_person       211
dtype: int64


When the style is  cartoon , the corresponding tally of each label is:
label_dog           10
label_elephant      13
label_giraffe       12
label_guitar       121
label_horse         11
label_house         12
label_person        12
dtype: int64
```

Tallies of each pair of (style, label) for training dataset (Part 2)

```
Tallies of each pair of (style, label):


When the style is  art_painting , the corresponding tally of each label is:
label_dog          119
label_elephant      89
label_giraffe      110
label_guitar        82
label_horse         90
label_house        110
label_person        96
dtype: int64


When the style is  sketch , the corresponding tally of each label is:
label_dog          112
label_elephant     115
label_giraffe      104
label_guitar        95
label_horse        108
label_house         80
label_person       112
dtype: int64
```

Tallies of each pair of (style, label) for testing dataset (Part 1)

```
When the style is  photo , the corresponding tally of each label is:
label_dog           81
label_elephant      83
label_giraffe      102
label_guitar        81
label_horse        103
label_house         95
label_person       110
dtype: int64


When the style is  cartoon , the corresponding tally of each label is:
label_dog           95
label_elephant      83
label_giraffe      109
label_guitar        82
label_horse         81
label_house        101
label_person        95
dtype: int64
```

Tallies of each pair of (style, label) for testing dataset (Part 2)

[Q10]

Write down your prediction on how the above difference would affect test-time performance.

Due to the skewed distribution of images belonging to different labels, the model would learn a lot of feature information to the images of the top most frequently occurred labels, while it learn too little feature information from the images of the least frequently occurred labels.

This will eventually make the model too sensitive to the images with the most frequently occurring labels, leading to bias in label classification during the testing. This may also lead to low testing accuracy as well.

[Q11]

Implement a function to construct the classifier model. Report the number of trainable parameters in the model.

The number of trainable parameters in the model is 5609031.

[Q12]

Train the model with the given training dataset for at least 100 epochs. Record the loss obtained at the end of training (or better, throughout training).

Epoch	Screenshot
1 – 10	Epoch 1/100 cross-entropy loss: 1.693 : 100% 25/25 [01:24<00:00, 1.71s/it] /usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. saving_api.save_model(Epoch 2/100 cross-entropy loss: 1.388 : 100% 25/25 [00:47<00:00, 1.63s/it] Epoch 3/100 cross-entropy loss: 1.227 : 100% 25/25 [00:41<00:00, 1.66s/it] Epoch 4/100 cross-entropy loss: 1.092 : 100% 25/25 [00:42<00:00, 1.63s/it] Epoch 5/100 cross-entropy loss: 1.152 : 100% 25/25 [00:41<00:00, 1.63s/it] Epoch 6/100 cross-entropy loss: 1.098 : 100% 25/25 [00:41<00:00, 1.63s/it] Epoch 7/100 cross-entropy loss: 1.081 : 100% 25/25 [00:41<00:00, 1.64s/it] Epoch 8/100 cross-entropy loss: 1.028 : 100% 25/25 [00:42<00:00, 1.65s/it] Epoch 9/100 cross-entropy loss: 0.983 : 100% 25/25 [00:41<00:00, 1.64s/it] Epoch 10/100 cross-entropy loss: 1.018 : 100% 25/25 [00:41<00:00, 1.63s/it]
11 – 20	Epoch 11/100 cross-entropy loss: 0.974 : 100% 25/25 [00:41<00:00, 1.65s/it] Epoch 12/100 cross-entropy loss: 0.954 : 100% 25/25 [00:41<00:00, 1.67s/it] Epoch 13/100 cross-entropy loss: 0.928 : 100% 25/25 [00:40<00:00, 1.65s/it] Epoch 14/100 cross-entropy loss: 0.936 : 100% 25/25 [00:41<00:00, 1.64s/it] Epoch 15/100 cross-entropy loss: 0.906 : 100% 25/25 [00:41<00:00, 1.61s/it] Epoch 16/100 cross-entropy loss: 0.819 : 100% 25/25 [00:41<00:00, 1.61s/it] Epoch 17/100 cross-entropy loss: 0.860 : 100% 25/25 [00:40<00:00, 1.62s/it] Epoch 18/100 cross-entropy loss: 0.803 : 100% 25/25 [00:42<00:00, 1.68s/it] Epoch 19/100 cross-entropy loss: 0.762 : 100% 25/25 [00:41<00:00, 1.63s/it] Epoch 20/100 cross-entropy loss: 0.789 : 100% 25/25 [00:41<00:00, 1.64s/it]
21 – 30	Epoch 21/100 cross-entropy loss: 0.788 : 100% 25/25 [04:10<00:00, 1.64s/it] Epoch 22/100 cross-entropy loss: 0.829 : 100% 25/25 [00:41<00:00, 1.65s/it] Epoch 23/100 cross-entropy loss: 0.763 : 100% 25/25 [02:46<00:00, 1.67s/it] Epoch 24/100 cross-entropy loss: 0.678 : 100% 25/25 [00:41<00:00, 1.65s/it] Epoch 25/100 cross-entropy loss: 0.834 : 100% 25/25 [01:23<00:00, 1.63s/it] Epoch 26/100 cross-entropy loss: 0.770 : 100% 25/25 [00:41<00:00, 1.65s/it] Epoch 27/100 cross-entropy loss: 0.759 : 100% 25/25 [00:40<00:00, 1.61s/it] Epoch 28/100 cross-entropy loss: 0.710 : 100% 25/25 [00:43<00:00, 1.62s/it] Epoch 29/100 cross-entropy loss: 0.762 : 100% 25/25 [00:40<00:00, 1.61s/it] Epoch 30/100 cross-entropy loss: 0.735 : 100% 25/25 [00:42<00:00, 1.62s/it]
31 – 40	Epoch 31/100 cross-entropy loss: 0.650 : 100% 25/25 [00:41<00:00, 1.63s/it] Epoch 32/100 cross-entropy loss: 0.730 : 100% 25/25 [00:41<00:00, 1.63s/it] Epoch 33/100 cross-entropy loss: 0.675 : 100% 25/25 [00:40<00:00, 1.64s/it] Epoch 34/100 cross-entropy loss: 0.636 : 100% 25/25 [00:41<00:00, 1.64s/it] Epoch 35/100 cross-entropy loss: 0.617 : 100% 25/25 [00:40<00:00, 1.67s/it] Epoch 36/100 cross-entropy loss: 0.594 : 100% 25/25 [00:41<00:00, 1.65s/it] Epoch 37/100 cross-entropy loss: 0.586 : 100% 25/25 [00:40<00:00, 1.63s/it] Epoch 38/100 cross-entropy loss: 0.590 : 100% 25/25 [00:41<00:00, 1.61s/it] Epoch 39/100 cross-entropy loss: 0.580 : 100% 25/25 [00:40<00:00, 1.61s/it] Epoch 40/100 cross-entropy loss: 0.525 : 100% 25/25 [00:41<00:00, 1.62s/it]

41 – 50	Epoch 41/100 cross-entropy loss: 0.498 : 100% 25/25 [06:56<00:00, 1.63s/it] Epoch 42/100 cross-entropy loss: 0.464 : 100% 25/25 [00:41<00:00, 1.63s/it] Epoch 43/100 cross-entropy loss: 0.579 : 100% 25/25 [05:33<00:00, 1.65s/it] Epoch 44/100 cross-entropy loss: 0.555 : 100% 25/25 [00:41<00:00, 1.65s/it] Epoch 45/100 cross-entropy loss: 0.486 : 100% 25/25 [04:09<00:00, 1.67s/it] Epoch 46/100 cross-entropy loss: 0.472 : 100% 25/25 [00:41<00:00, 1.65s/it] Epoch 47/100 cross-entropy loss: 0.427 : 100% 25/25 [02:46<00:00, 1.64s/it] Epoch 48/100 cross-entropy loss: 0.468 : 100% 25/25 [00:41<00:00, 1.62s/it] Epoch 49/100 cross-entropy loss: 0.458 : 100% 25/25 [01:23<00:00, 1.62s/it] Epoch 50/100 cross-entropy loss: 0.458 : 100% 25/25 [00:41<00:00, 1.62s/it]
51 – 60	Epoch 51/100 cross-entropy loss: 0.458 : 100% 25/25 [00:41<00:00, 1.63s/it] Epoch 52/100 cross-entropy loss: 0.417 : 100% 25/25 [00:41<00:00, 1.62s/it] Epoch 53/100 cross-entropy loss: 0.347 : 100% 25/25 [00:41<00:00, 1.62s/it] Epoch 54/100 cross-entropy loss: 0.458 : 100% 25/25 [00:41<00:00, 1.64s/it] Epoch 55/100 cross-entropy loss: 0.434 : 100% 25/25 [00:40<00:00, 1.65s/it] Epoch 56/100 cross-entropy loss: 0.336 : 100% 25/25 [00:41<00:00, 1.67s/it] Epoch 57/100 cross-entropy loss: 0.386 : 100% 25/25 [00:40<00:00, 1.63s/it] Epoch 58/100 cross-entropy loss: 0.375 : 100% 25/25 [00:41<00:00, 1.64s/it] Epoch 59/100 cross-entropy loss: 0.400 : 100% 25/25 [00:40<00:00, 1.62s/it] Epoch 60/100 cross-entropy loss: 0.319 : 100% 25/25 [00:41<00:00, 1.62s/it]
61 – 70	Epoch 61/100 cross-entropy loss: 0.278 : 100% 25/25 [00:41<00:00, 1.63s/it] Epoch 62/100 cross-entropy loss: 0.306 : 100% 25/25 [00:41<00:00, 1.62s/it] Epoch 63/100 cross-entropy loss: 0.298 : 100% 25/25 [00:41<00:00, 1.65s/it] Epoch 64/100 cross-entropy loss: 0.350 : 100% 25/25 [00:41<00:00, 1.65s/it] Epoch 65/100 cross-entropy loss: 0.358 : 100% 25/25 [00:40<00:00, 1.66s/it] Epoch 66/100 cross-entropy loss: 0.305 : 100% 25/25 [00:41<00:00, 1.65s/it] Epoch 67/100 cross-entropy loss: 0.233 : 100% 25/25 [00:40<00:00, 1.64s/it] Epoch 68/100 cross-entropy loss: 0.254 : 100% 25/25 [00:41<00:00, 1.61s/it] Epoch 69/100 cross-entropy loss: 0.245 : 100% 25/25 [00:40<00:00, 1.62s/it] Epoch 70/100 cross-entropy loss: 0.325 : 100% 25/25 [00:41<00:00, 1.61s/it]
71 – 80	Epoch 71/100 cross-entropy loss: 0.254 : 100% 25/25 [02:46<00:00, 1.63s/it] Epoch 72/100 cross-entropy loss: 0.256 : 100% 25/25 [00:41<00:00, 1.62s/it] Epoch 73/100 cross-entropy loss: 0.264 : 100% 25/25 [01:23<00:00, 1.62s/it] Epoch 74/100 cross-entropy loss: 0.202 : 100% 25/25 [00:41<00:00, 1.64s/it] Epoch 75/100 cross-entropy loss: 0.192 : 100% 25/25 [01:23<00:00, 1.66s/it] Epoch 76/100 cross-entropy loss: 0.199 : 100% 25/25 [00:42<00:00, 1.68s/it] Epoch 77/100 cross-entropy loss: 0.230 : 100% 25/25 [00:41<00:00, 1.65s/it] Epoch 78/100 cross-entropy loss: 0.175 : 100% 25/25 [00:41<00:00, 1.63s/it] Epoch 79/100 cross-entropy loss: 0.185 : 100% 25/25 [00:41<00:00, 1.62s/it] Epoch 80/100 cross-entropy loss: 0.184 : 100% 25/25 [00:40<00:00, 1.61s/it]
81 – 90	Epoch 81/100 cross-entropy loss: 0.490 : 100% 25/25 [00:41<00:00, 1.61s/it] Epoch 82/100 cross-entropy loss: 0.402 : 100% 25/25 [00:40<00:00, 1.61s/it] Epoch 83/100 cross-entropy loss: 0.283 : 100% 25/25 [00:41<00:00, 1.62s/it] Epoch 84/100 cross-entropy loss: 0.195 : 100% 25/25 [00:40<00:00, 1.63s/it] Epoch 85/100 cross-entropy loss: 0.191 : 100% 25/25 [00:41<00:00, 1.65s/it] Epoch 86/100 cross-entropy loss: 0.156 : 100% 25/25 [00:40<00:00, 1.67s/it] Epoch 87/100 cross-entropy loss: 0.235 : 100% 25/25 [00:41<00:00, 1.63s/it] Epoch 88/100 cross-entropy loss: 0.402 : 100% 25/25 [00:40<00:00, 1.61s/it] Epoch 89/100 cross-entropy loss: 0.206 : 100% 25/25 [00:41<00:00, 1.61s/it] Epoch 90/100 cross-entropy loss: 0.196 : 100% 25/25 [00:40<00:00, 1.62s/it]
91 – 100	Epoch 91/100 cross-entropy loss: 0.150 : 100% 25/25 [00:41<00:00, 1.62s/it] Epoch 92/100 cross-entropy loss: 0.146 : 100% 25/25 [00:40<00:00, 1.63s/it] Epoch 93/100 cross-entropy loss: 0.122 : 100% 25/25 [00:41<00:00, 1.64s/it] Epoch 94/100 cross-entropy loss: 0.094 : 100% 25/25 [00:40<00:00, 1.64s/it] Epoch 95/100 cross-entropy loss: 0.115 : 100% 25/25 [00:41<00:00, 1.65s/it] Epoch 96/100 cross-entropy loss: 0.105 : 100% 25/25 [00:40<00:00, 1.64s/it] Epoch 97/100 cross-entropy loss: 0.110 : 100% 25/25 [00:41<00:00, 1.61s/it] Epoch 98/100 cross-entropy loss: 0.172 : 100% 25/25 [00:40<00:00, 1.62s/it] Epoch 99/100 cross-entropy loss: 0.164 : 100% 25/25 [00:41<00:00, 1.61s/it] Epoch 100/100 cross-entropy loss: 0.158 : 100% 25/25 [00:40<00:00, 1.61s/it]

101 – 110	Epoch 1/10 cross-entropy loss: 0.237 : 100% 25/25 [00:43<00:00, 1.63s/it] Epoch 2/10 cross-entropy loss: 0.164 : 100% 25/25 [00:41<00:00, 1.61s/it] Epoch 3/10 cross-entropy loss: 0.145 : 100% 25/25 [00:40<00:00, 1.64s/it] Epoch 4/10 cross-entropy loss: 0.091 : 100% 25/25 [00:41<00:00, 1.64s/it] Epoch 5/10 cross-entropy loss: 0.083 : 100% 25/25 [00:40<00:00, 1.66s/it] Epoch 6/10 cross-entropy loss: 0.086 : 100% 25/25 [00:41<00:00, 1.63s/it] Epoch 7/10 cross-entropy loss: 0.118 : 100% 25/25 [00:40<00:00, 1.65s/it] Epoch 8/10 cross-entropy loss: 0.115 : 100% 25/25 [00:41<00:00, 1.62s/it] Epoch 9/10 cross-entropy loss: 0.166 : 100% 25/25 [00:40<00:00, 1.61s/it] Epoch 10/10 cross-entropy loss: 0.173 : 100% 25/25 [00:40<00:00, 1.60s/it]
111 – 115	Epoch 1/5 cross-entropy loss: 0.169 : 100% 25/25 [00:43<00:00, 1.68s/it] Epoch 2/5 cross-entropy loss: 0.123 : 100% 25/25 [00:41<00:00, 1.64s/it] Epoch 3/5 cross-entropy loss: 0.136 : 100% 25/25 [00:40<00:00, 1.63s/it] Epoch 4/5 cross-entropy loss: 0.115 : 100% 25/25 [00:41<00:00, 1.59s/it] Epoch 5/5 cross-entropy loss: 0.116 : 100% 25/25 [00:40<00:00, 1.60s/it]

I have trained my model for 115 epochs.

At the end of the training:

Cross-entropy loss = 0.116

[Q13]

Report the accuracy and confusion matrix of the model trained in the previous section, when tested against the training and test datasets respectively.

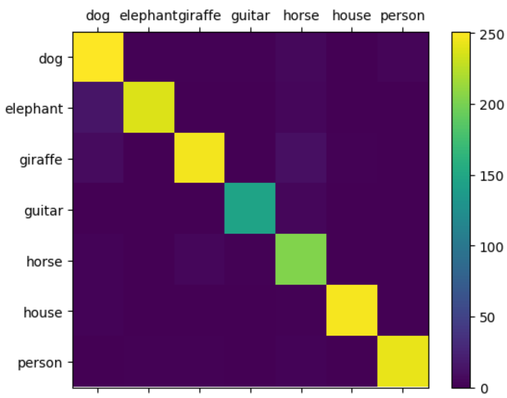
<u>For training dataset:</u>

Accuracy = 0.960390031337738

Confusion matrix

```
The confusion matrix is:
[[251   1   1   1   5   0   3]
 [ 14 238   0   0   4   0   0]
 [  7   0 247   0  10   1   0]
 [  0   0   0 146   4   0   0]
 [  2   0   4   1 203   0   0]
 [  2   0   0   0   1 248   0]
 [  0   1   1   0   2   0 243]]
```

Visualized confusion matrix



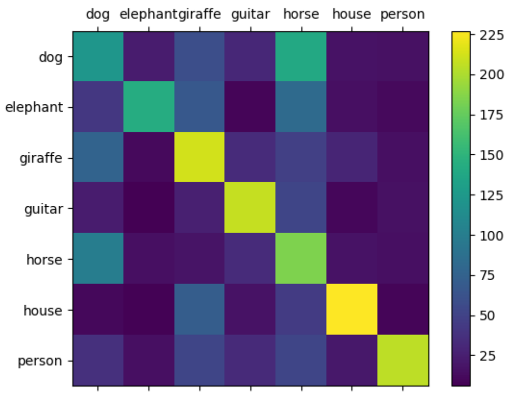
For testing dataset:

Accuracy = 0.47778186202049255

Confusion matrix

```
The confusion matrix is:
[[122  23  59  31 139  17  16]
 [ 42 143  67   9  82  15  12]
 [ 76  12 212  33  48  29  15]
 [ 23   6  25 208  52  10  16]
 [100  15  18  33 184  17  15]
 [ 11   7  71  17  44 227   9]
 [ 37  15  52  32  52  20 205]]
```

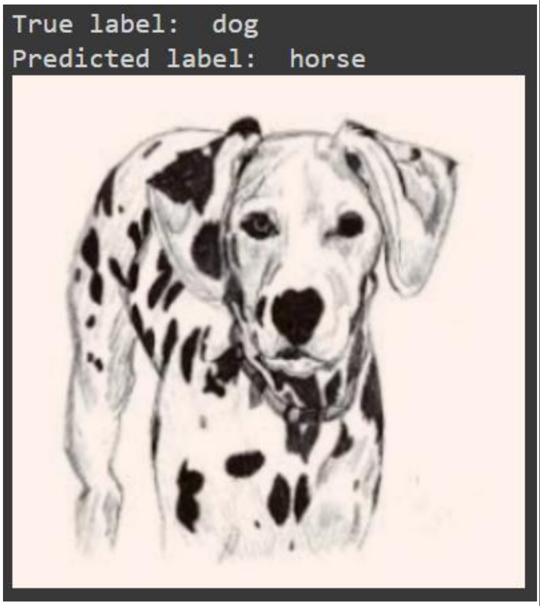
Visualized confusion matrix

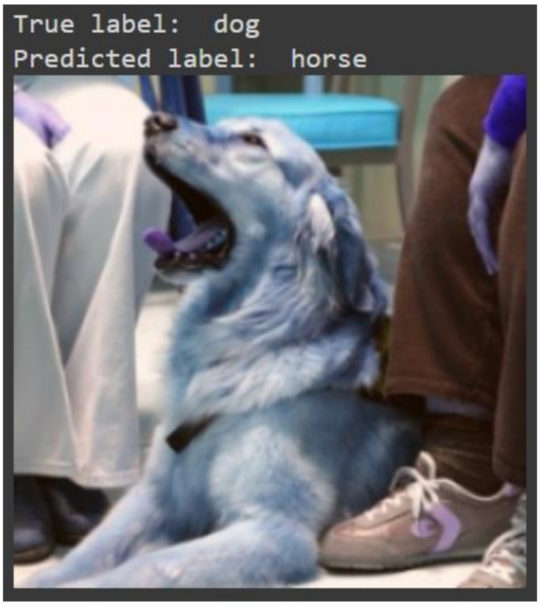


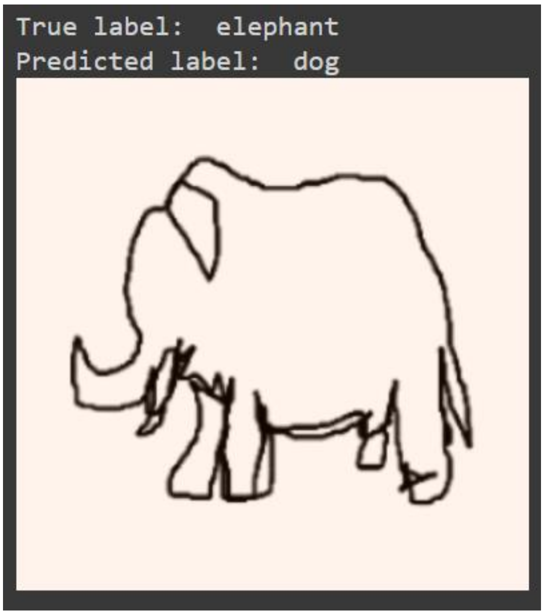
[Q14]

Showcase some of the mislabeled sample images (with their true and reported samples). Discuss your observations (and possible explanation) on some of these failure cases.

<u>Some samples of the mislabeled images:</u>







Observations and explanations on some of these failure cases:

By combining what I have answered in C8, Q9, and Q10, the reason for these failure cases is quite obvious, which is due to the uneven distribution of each label pictures in different style of images.

Recall that due to the fact that the distribution of images (the one in training dataset) with each different labels for different style is highly biased, the model would learn much more features from those images belonging to the most frequently occurred labels (say label A, label B) in each corresponding style (say style X), while learn very little features from the images that belong to those least frequently occurred labels (say label C, label D) in corresponding style(say style Z). This eventually results in higher chance in wrong classification on the testing dataset image with true label as label C and style as style Z. Sometimes, it may classify these image as label A or label B in some cases if the model training make the model highly biased.

Let us take the dog image with art painting style (on top left corner), and the giraffe image with photo style (on bottom left corner) as two examples.

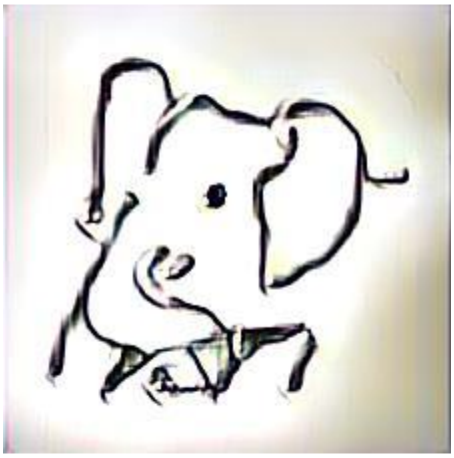
For the dog image with art painting style, by referring to the result from C8, in the training dataset with art_painting style, there are only 13 images of them is labeled as dog. While for the images label as horse is 180 images, which is the second top most frequently occurred label in art_painting style training images (the first top most frequently occurred label is giraffe). This clearly explains why this dog image is misclassified as horse as the model is biased and sensitive to the features of horse in art painting style, and learn very little about the features of dog in art_painting style.

For the giraffe image with photo style, again by referring to the result from C8, in the training dataset with photo style, there are only 12 images of them is labeled as giraffe. While the top two most frequently occurred image labels are house (215 images) and person (211 images). Since the features of giraffes is not similar to neither house and person, so it misclassify the giraffe as a horse (which has only 11 images in photo style in training dataset) because horse shares the most similar features with the giraffe based on my educated guest and common sense.

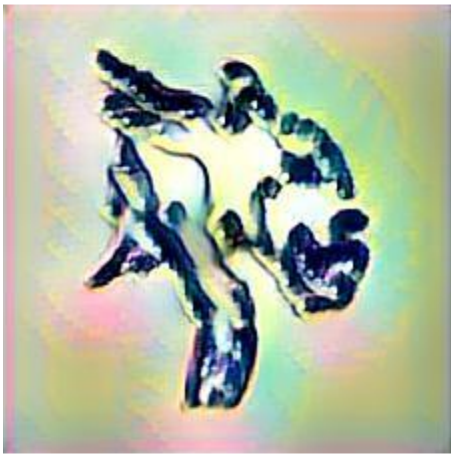
[Q15]

Showcase some samples from the generated dataset, with at least one per object label, and one per style.

<u>7 object labels:</u>

Label	Image
Elephant (mix with cartoon style)	
Guitar (mix with art painting style)	

Giraffe (mix with photo style)	
Horse (mix with art painting style)	
Person (mix with sketch style)	

House (mix with sketch style)	
Dog (mix with art painting style)	

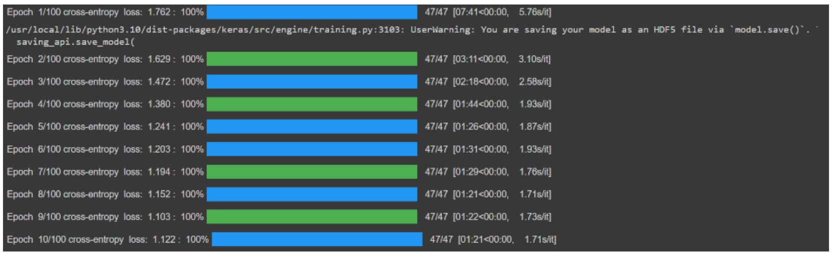
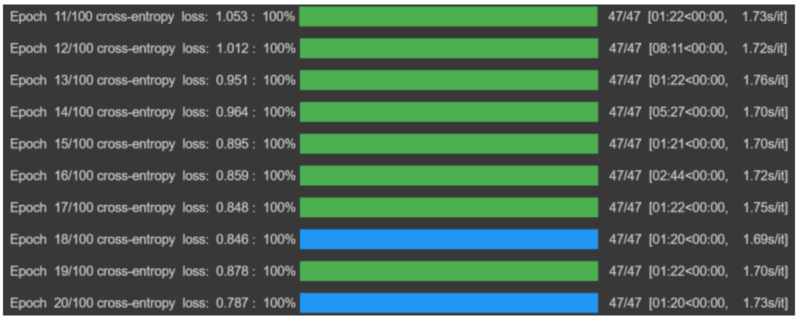
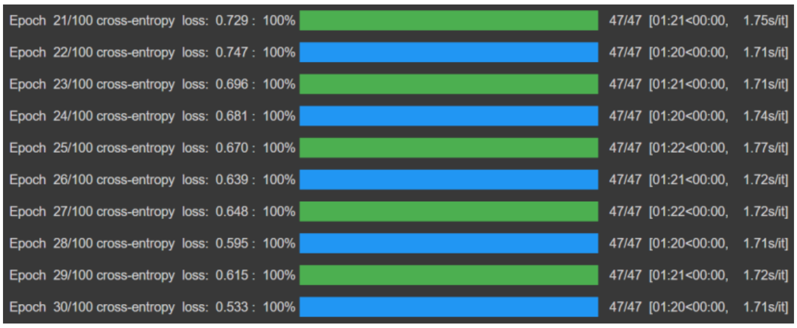
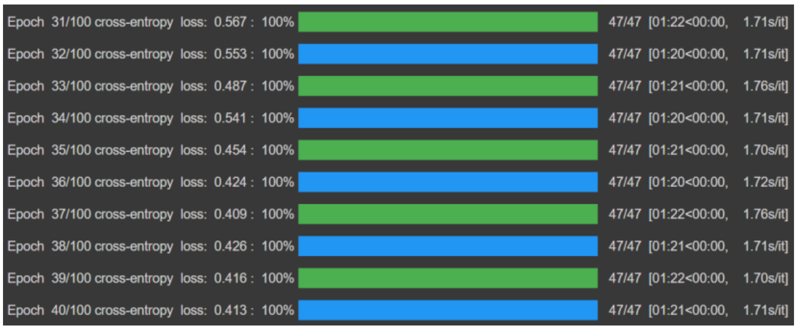
<u>4 styles:</u>

Style	Image
Cartoon (label is house)	

Sketch (label is person)	
Art Painting (label is dog)	
Photo (label is giraffe)	

[Q16]

Train a new model using a dataset combining both the given and generated training datasets for at least 100 epochs, and report the loss at the end of training.

Epoch	Screenshot
1 – 10	 Epoch 1/100 cross-entropy loss: 1.762 : 100% 47/47 [07:41<00:00, 5.76s/it] /usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. ` saving_api.save_model(Epoch 2/100 cross-entropy loss: 1.629 : 100% 47/47 [03:11<00:00, 3.10s/it] Epoch 3/100 cross-entropy loss: 1.472 : 100% 47/47 [02:18<00:00, 2.58s/it] Epoch 4/100 cross-entropy loss: 1.380 : 100% 47/47 [01:44<00:00, 1.93s/it] Epoch 5/100 cross-entropy loss: 1.241 : 100% 47/47 [01:26<00:00, 1.87s/it] Epoch 6/100 cross-entropy loss: 1.203 : 100% 47/47 [01:31<00:00, 1.93s/it] Epoch 7/100 cross-entropy loss: 1.194 : 100% 47/47 [01:29<00:00, 1.76s/it] Epoch 8/100 cross-entropy loss: 1.152 : 100% 47/47 [01:21<00:00, 1.71s/it] Epoch 9/100 cross-entropy loss: 1.103 : 100% 47/47 [01:22<00:00, 1.73s/it] Epoch 10/100 cross-entropy loss: 1.122 : 100% 47/47 [01:21<00:00, 1.71s/it]
11 – 20	 Epoch 11/100 cross-entropy loss: 1.053 : 100% 47/47 [01:22<00:00, 1.73s/it] Epoch 12/100 cross-entropy loss: 1.012 : 100% 47/47 [08:11<00:00, 1.72s/it] Epoch 13/100 cross-entropy loss: 0.951 : 100% 47/47 [01:22<00:00, 1.76s/it] Epoch 14/100 cross-entropy loss: 0.964 : 100% 47/47 [05:27<00:00, 1.70s/it] Epoch 15/100 cross-entropy loss: 0.895 : 100% 47/47 [01:21<00:00, 1.70s/it] Epoch 16/100 cross-entropy loss: 0.859 : 100% 47/47 [02:44<00:00, 1.72s/it] Epoch 17/100 cross-entropy loss: 0.848 : 100% 47/47 [01:22<00:00, 1.75s/it] Epoch 18/100 cross-entropy loss: 0.846 : 100% 47/47 [01:20<00:00, 1.69s/it] Epoch 19/100 cross-entropy loss: 0.878 : 100% 47/47 [01:22<00:00, 1.70s/it] Epoch 20/100 cross-entropy loss: 0.787 : 100% 47/47 [01:20<00:00, 1.73s/it]
21 – 30	 Epoch 21/100 cross-entropy loss: 0.729 : 100% 47/47 [01:21<00:00, 1.75s/it] Epoch 22/100 cross-entropy loss: 0.747 : 100% 47/47 [01:20<00:00, 1.71s/it] Epoch 23/100 cross-entropy loss: 0.696 : 100% 47/47 [01:21<00:00, 1.71s/it] Epoch 24/100 cross-entropy loss: 0.681 : 100% 47/47 [01:20<00:00, 1.74s/it] Epoch 25/100 cross-entropy loss: 0.670 : 100% 47/47 [01:22<00:00, 1.77s/it] Epoch 26/100 cross-entropy loss: 0.639 : 100% 47/47 [01:21<00:00, 1.72s/it] Epoch 27/100 cross-entropy loss: 0.648 : 100% 47/47 [01:22<00:00, 1.72s/it] Epoch 28/100 cross-entropy loss: 0.595 : 100% 47/47 [01:20<00:00, 1.71s/it] Epoch 29/100 cross-entropy loss: 0.615 : 100% 47/47 [01:21<00:00, 1.72s/it] Epoch 30/100 cross-entropy loss: 0.533 : 100% 47/47 [01:20<00:00, 1.71s/it]
31 – 40	 Epoch 31/100 cross-entropy loss: 0.567 : 100% 47/47 [01:22<00:00, 1.71s/it] Epoch 32/100 cross-entropy loss: 0.553 : 100% 47/47 [01:20<00:00, 1.71s/it] Epoch 33/100 cross-entropy loss: 0.487 : 100% 47/47 [01:21<00:00, 1.76s/it] Epoch 34/100 cross-entropy loss: 0.541 : 100% 47/47 [01:20<00:00, 1.71s/it] Epoch 35/100 cross-entropy loss: 0.454 : 100% 47/47 [01:21<00:00, 1.70s/it] Epoch 36/100 cross-entropy loss: 0.424 : 100% 47/47 [01:20<00:00, 1.72s/it] Epoch 37/100 cross-entropy loss: 0.409 : 100% 47/47 [01:22<00:00, 1.76s/it] Epoch 38/100 cross-entropy loss: 0.426 : 100% 47/47 [01:21<00:00, 1.71s/it] Epoch 39/100 cross-entropy loss: 0.416 : 100% 47/47 [01:22<00:00, 1.70s/it] Epoch 40/100 cross-entropy loss: 0.413 : 100% 47/47 [01:21<00:00, 1.71s/it]

41 – 50	Epoch 41/100 cross-entropy loss: 0.418 : 100% 47/47 [01:22<00:00, 1.74s/it] Epoch 42/100 cross-entropy loss: 0.358 : 100% 47/47 [02:43<00:00, 1.72s/it] Epoch 43/100 cross-entropy loss: 0.354 : 100% 47/47 [01:22<00:00, 1.68s/it] Epoch 44/100 cross-entropy loss: 0.371 : 100% 47/47 [01:21<00:00, 1.72s/it] Epoch 45/100 cross-entropy loss: 0.315 : 100% 47/47 [01:21<00:00, 1.75s/it] Epoch 46/100 cross-entropy loss: 0.341 : 100% 47/47 [01:22<00:00, 1.75s/it] Epoch 47/100 cross-entropy loss: 0.320 : 100% 47/47 [01:20<00:00, 1.69s/it] Epoch 48/100 cross-entropy loss: 0.264 : 100% 47/47 [01:22<00:00, 1.70s/it] Epoch 49/100 cross-entropy loss: 0.290 : 100% 47/47 [01:21<00:00, 1.71s/it] Epoch 50/100 cross-entropy loss: 0.261 : 100% 47/47 [01:22<00:00, 1.73s/it]
51 – 60	Epoch 51/100 cross-entropy loss: 0.291 : 100% 47/47 [01:20<00:00, 1.70s/it] Epoch 52/100 cross-entropy loss: 0.206 : 100% 47/47 [01:22<00:00, 1.69s/it] Epoch 53/100 cross-entropy loss: 0.200 : 100% 47/47 [01:21<00:00, 1.71s/it] Epoch 54/100 cross-entropy loss: 0.228 : 100% 47/47 [01:21<00:00, 1.73s/it] Epoch 55/100 cross-entropy loss: 0.212 : 100% 47/47 [01:20<00:00, 1.76s/it] Epoch 56/100 cross-entropy loss: 0.168 : 100% 47/47 [01:21<00:00, 1.67s/it] Epoch 57/100 cross-entropy loss: 0.194 : 100% 47/47 [01:20<00:00, 1.69s/it] Epoch 58/100 cross-entropy loss: 0.152 : 100% 47/47 [01:21<00:00, 1.71s/it] Epoch 59/100 cross-entropy loss: 0.204 : 100% 47/47 [01:20<00:00, 1.73s/it] Epoch 60/100 cross-entropy loss: 0.176 : 100% 47/47 [01:21<00:00, 1.70s/it]
61 – 70	Epoch 1/40 cross-entropy loss: 0.174 : 100% 47/47 [07:15<00:00, 5.22s/it] /usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving saving_api.save_model(Epoch 2/40 cross-entropy loss: 0.162 : 100% 47/47 [12:53<00:00, 2.95s/it] Epoch 3/40 cross-entropy loss: 0.160 : 100% 47/47 [09:44<00:00, 2.51s/it] Epoch 4/40 cross-entropy loss: 0.158 : 100% 47/47 [01:35<00:00, 1.91s/it] Epoch 5/40 cross-entropy loss: 0.126 : 100% 47/47 [05:58<00:00, 1.80s/it] Epoch 6/40 cross-entropy loss: 0.140 : 100% 47/47 [01:41<00:00, 1.92s/it] Epoch 7/40 cross-entropy loss: 0.129 : 100% 47/47 [02:50<00:00, 1.75s/it] Epoch 8/40 cross-entropy loss: 0.163 : 100% 47/47 [01:21<00:00, 1.73s/it] Epoch 9/40 cross-entropy loss: 0.095 : 100% 47/47 [01:21<00:00, 1.69s/it] Epoch 10/40 cross-entropy loss: 0.174 : 100% 47/47 [01:22<00:00, 1.74s/it]
71 – 80	Epoch 11/40 cross-entropy loss: 0.110 : 100% 47/47 [01:21<00:00, 1.78s/it] Epoch 12/40 cross-entropy loss: 0.111 : 100% 47/47 [01:21<00:00, 1.69s/it] Epoch 13/40 cross-entropy loss: 0.110 : 100% 47/47 [01:21<00:00, 1.69s/it] Epoch 14/40 cross-entropy loss: 0.066 : 100% 47/47 [01:22<00:00, 1.77s/it] Epoch 15/40 cross-entropy loss: 0.077 : 100% 47/47 [01:20<00:00, 1.71s/it] Epoch 16/40 cross-entropy loss: 0.133 : 100% 47/47 [01:21<00:00, 1.68s/it] Epoch 17/40 cross-entropy loss: 0.112 : 100% 47/47 [01:20<00:00, 1.70s/it] Epoch 18/40 cross-entropy loss: 0.060 : 100% 47/47 [01:22<00:00, 1.77s/it] Epoch 19/40 cross-entropy loss: 0.087 : 100% 47/47 [01:21<00:00, 1.71s/it] Epoch 20/40 cross-entropy loss: 0.116 : 100% 47/47 [01:22<00:00, 1.72s/it]
81 – 90	Epoch 21/40 cross-entropy loss: 0.076 : 100% 47/47 [01:21<00:00, 1.72s/it] Epoch 22/40 cross-entropy loss: 0.074 : 100% 47/47 [01:22<00:00, 1.75s/it] Epoch 23/40 cross-entropy loss: 0.157 : 100% 47/47 [01:21<00:00, 1.72s/it] Epoch 24/40 cross-entropy loss: 0.123 : 100% 47/47 [01:22<00:00, 1.75s/it] Epoch 25/40 cross-entropy loss: 0.069 : 100% 47/47 [01:20<00:00, 1.73s/it] Epoch 26/40 cross-entropy loss: 0.041 : 100% 47/47 [01:21<00:00, 1.70s/it] Epoch 27/40 cross-entropy loss: 0.096 : 100% 47/47 [01:20<00:00, 1.72s/it] Epoch 28/40 cross-entropy loss: 0.075 : 100% 47/47 [01:22<00:00, 1.73s/it] Epoch 29/40 cross-entropy loss: 0.032 : 100% 47/47 [01:21<00:00, 1.76s/it] Epoch 30/40 cross-entropy loss: 0.033 : 100% 47/47 [01:21<00:00, 1.69s/it]

91 – 100	Epoch 31/40 cross-entropy loss: 0.102 : 100% 47/47 [02:43<00:00, 1.71s/it] Epoch 32/40 cross-entropy loss: 0.118 : 100% 47/47 [01:21<00:00, 1.74s/it] Epoch 33/40 cross-entropy loss: 0.054 : 100% 47/47 [01:20<00:00, 1.72s/it] Epoch 34/40 cross-entropy loss: 0.102 : 100% 47/47 [01:21<00:00, 1.68s/it] Epoch 35/40 cross-entropy loss: 0.049 : 100% 47/47 [01:20<00:00, 1.73s/it] Epoch 36/40 cross-entropy loss: 0.023 : 100% 47/47 [01:22<00:00, 1.76s/it] Epoch 37/40 cross-entropy loss: 0.071 : 100% 47/47 [01:19<00:00, 1.66s/it] Epoch 38/40 cross-entropy loss: 0.043 : 100% 47/47 [01:22<00:00, 1.71s/it] Epoch 39/40 cross-entropy loss: 0.022 : 100% 47/47 [01:21<00:00, 1.71s/it] Epoch 40/40 cross-entropy loss: 0.047 : 100% 47/47 [01:20<00:00, 1.72s/it]

(Note: My Google Colaboratory quota runs out during the middle of the training of epoch 68, since I have stored the classify-augmentation.h5 file just after finishing the training of epoch 60, so I continue the remaining 40 training epoch in a new account.)

I have trained my model for 100 epochs.

At the end of the training:

Cross-entropy loss = 0.047

[Q17]

Test the new model against the test (and un-augmented training) dataset, then report and compare the result with one before augmentation.

<u>*Accuracy & Confusion Matrix*</u>

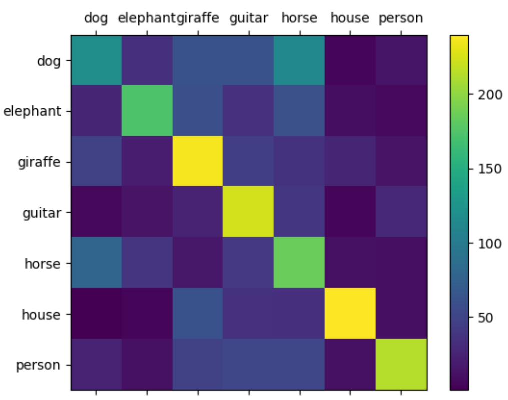
<u>For testing dataset:</u>

Accuracy = 0.5108336210250854

Confusion matrix

```
The confusion matrix is:
[[119  33  61  61 113   5  15]
 [ 26 173  59  35  60  10   7]
 [ 48  19 237  45  36  26  14]
 [  6  14  24 224  39   4  29]
 [ 79  38  16  41 185  12  11]
 [  1   5  61  35  33 240  11]
 [ 24  12  48  52  52  12 213]]
```

Visualized confusion matrix



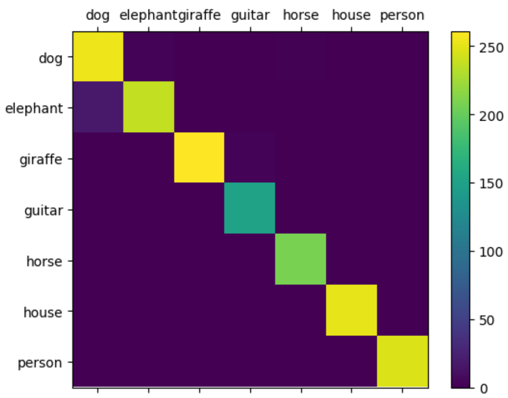
For unaugmented training dataset:

Accuracy = 0.9804996848106384

Confusion matrix

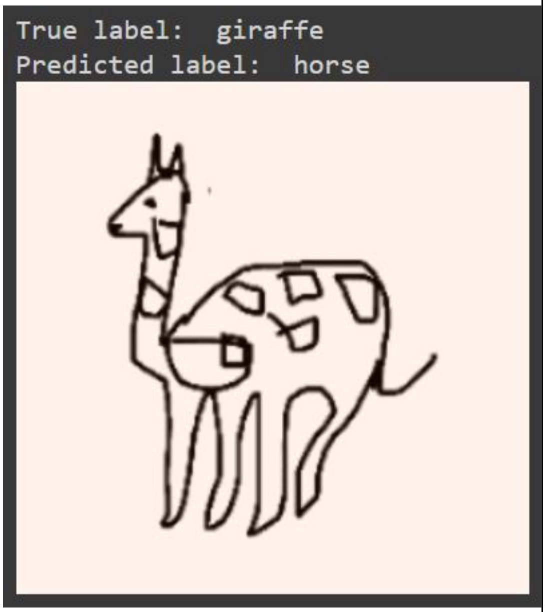
```
The confusion matrix is:
[[254   3   1   1   2   0   1]
 [ 18 237   0   1   0   0   0]
 [  0   0 261   3   1   0   0]
 [  0   0   0 150   0   0   0]
 [  1   0   0   0 209   0   0]
 [  0   0   0   0   0 251   0]
 [  0   0   0   0   0   0 247]]
```

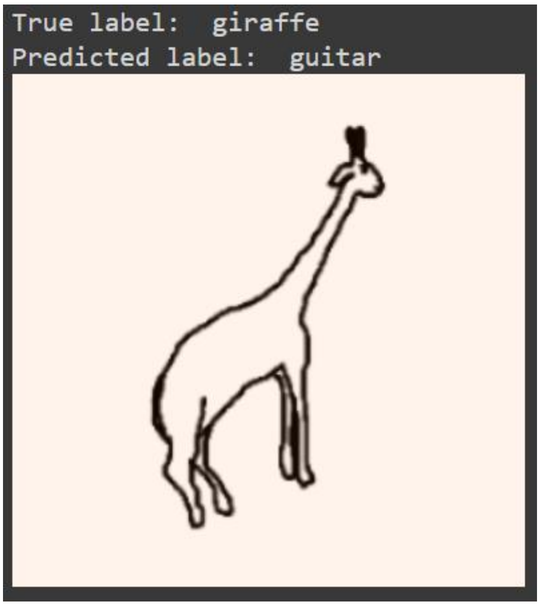
Visualized confusion matrix

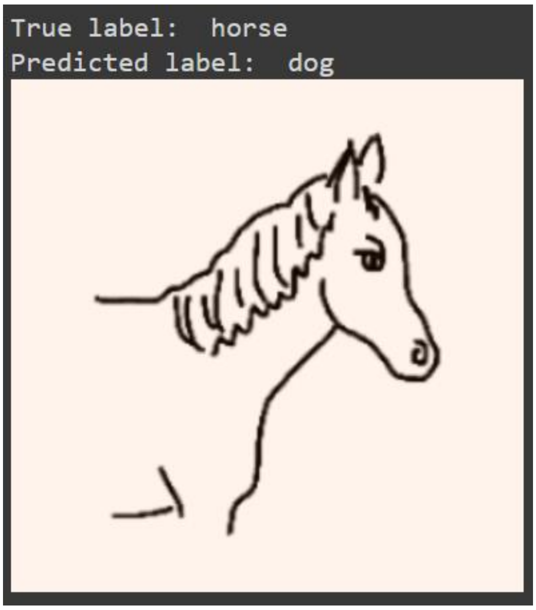


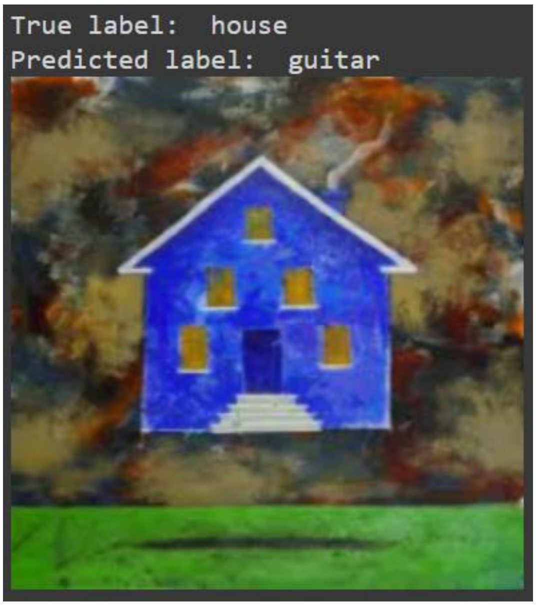
Mislabeled sample images

Some samples of the mislabeled images:






True label: horse
Predicted label: dog


True label: house
Predicted label: guitar

<u>Compare the result with one before augmentation</u>

By comparing with the result in Q13, it is obvious to see that there is a slight increase in both the accuracy of the un-augmented training dataset and the testing dataset.

For the accuracy of the un-augmented training dataset, it increases from 0.960390031337738 (in Q13) to 0.9804996848106384 (in Q17), which is increased by around 0.0201.

For the accuracy of the testing dataset, it increases from 0.47778186202049255 (in Q13) to 0.5108336210250854 (in Q17), which is increased by around 0.0331.

Also, for both the un-augmented training dataset and testing dataset, we can see that both the number of true positive cases and true negative cases in the confusion matrix increases when compared to the result in Q13. This implies both the number of false positive cases and false negatives cases have decreased.

[Q18]

Give a possible explanation on the two results (e.g., how the augmentation affected the test-time performance).

The two screenshots on the right shows the tallies of each pair of (style, label) after the dataset has the augmented images being added.

```
Tallies of each pair of (style, label):


When the style is  cartoon , the corresponding tally of each label is:
label_dog          60
label_elephant     63
label_giraffe      62
label_guitar      171
label_horse        61
label_house        62
label_person       62
dtype: int64


When the style is  art_painting , the corresponding tally of each label is:
label_dog          63
label_elephant     63
label_giraffe     281
label_guitar       60
label_horse       230
label_house        61
label_person       61
dtype: int64
```

```
When the style is  sketch , the corresponding tally of each label is:
label_dog         279
label_elephant    267
label_giraffe      60
label_guitar       59
label_horse        58
label_house        63
label_person       63
dtype: int64


When the style is  photo , the corresponding tally of each label is:
label_dog          60
label_elephant     63
label_giraffe      62
label_guitar       60
label_horse        61
label_house       265
label_person      261
dtype: int64
```

The one possible explanation of why both the accuracy for the un-augmented training dataset set and testing dataset has slightly increased a bit, as well as more true positive cases in the confusion matrix in both the un-augmented training dataset and testing dataset, can be explained by the two screenshots above.

The above two screenshots show the tally distribution of all combinations of (style, label) pairs after we have added the newly created augmented images to our dataset. In my code, I have followed the instructions from question C12 to generate 50 samples for each style-label pair, which means in total there are 4×7×50 = 1400 images generated and added to the initial training dataset. This results in total of 1641+1400 = 3041 training images in the augmented training dataset.

As we can see from the two screenshots, in each style (say S), the distribution of different labels (say L) with style S, have become more evenly distributed, as the percentage increase in the number of those least frequently occurred style-label pair (such as cartoon-dog pair, sketch-guitar pair etc.) is greater than that of those most frequently occurred style-label pair (such as photo-person pair, art_painting-giraffe pair etc.). When the model is learning the features from these images, the newly added augmented images allow the model to learn more about the features of those least frequently occurred style-label pair images. This makes the model less biased and hence less sensitive towards the features of those most frequently occurred style-label pair images. This make the sensitiveness and bias towards a few certain style-label pairs images less serious.

However, to explain just the slight increase in accuracy in testing dataset, and slight increase in true positive cases in confusion matrix of testing dataset, it is because the training dataset's skewed distribution still exist, even after we have added the augmented images to the initial training dataset. In fact, if we would like to further increase the accuracy and true positive cases of the testing dataset, we should make our training dataset distribution of each style-label pair more even, just like distribution of the testing dataset where each style-label pair is quite evenly distributed.