

# **COMP 4211**

## **Programming Assignment 2 Report**

**Name: LOKESWARA, Lovera**



**Student ID: 20798275**

**Lecture Section: L2**

## 5. Datasets and Data Loaders

### [Q1] \_\_len\_\_ Implementation (ImageDataset)

- COCO dataset: 3557
- WikiArt dataset: 7492
- Documentation:

 <code>obj = ImageDataset(coco_path)</code> <code>obj.__len__()</code> 3557	 <code>obj2 = ImageDataset(wikiart_path)</code> <code>obj2.__len__()</code> 7492
--	---

### [Q2] \_\_len\_\_ Implementation (ClassificationDataset)

- PACS training dataset: 1641
- PACS test dataset: 2723
- Documentation:

```
print(f"Length of PACS Training Dataset: {train_ds.__len__()}")  
print(f"Length of PACS Test Dataset: {test_ds.__len__()}")
```

```
Length of PACS Training Dataset: 1641  
Length of PACS Test Dataset: 2723
```

## 6. Style Transfer

### [Q3] What is the usage of upsampling layers?

Upsampling layers is used to increase the spatial dimensions (width and height) of a feature map while keeping the 2D representation of an image. This is the opposite of downsampling (such as max-pooling), where spatial dimensions are reduced. In this context, encoders usually reduce the spatial dimensions of the input image through convolutions to encode higher-level features. The upsampling layers are used to reconstruct the original image dimensions that were reduced by the previous steps.

Upsampling layers is also useful to refine the resolution of feature maps or images to reconstruct the details of the original input. This is essential in style transfer, because we want to generate an image that combines the content of one image with the style of another image. We need to be able to reconstruct the original image features even after we have put a new style into the image.

Also, because we are using convolutional neural network model, it captures more complex features, reducing depth (this is achieved by using convolutional layers with fewer filters than the preceding layer) while increasing spatial dimensions (which is exactly what upsampling does), helps in maintaining a balance between the representational capacity and spatial extent of feature maps. This is very important for reconstructing images. Upsampling can gradually transform the abstract, complex features captured by the encoder back into a detailed spatial representation necessary for reconstructing an image.

In the decoder architecture for style transfer, upsampling layers are specifically used to progressively increase the size of the feature maps back to the original dimensions of the input image, allowing the network to reconstruct the image with the transferred style. This is done by gradually increasing the resolution of the feature maps and decreasing their depth, eventually leading to an output with the desired spatial dimensions (usually the same as the original input image) and a depth of 3, corresponding to the RGB channels of the output image.

### [Q4] Report the number of trainable parameters in the decoder.

Number of trainable parameters: 3505219

```
# Build model
vgg19 = build_vgg19_enc(INPUT_SHAPE, vgg_path) # encoder
decoder = build_decoder(vgg19.output.shape[1:]) # input shape == encoder output shape
print(f"Decoder Trainable Parameter: {decoder.count_params()}")
model = build_model(vgg19, decoder, INPUT_SHAPE)

# Get loss
vgg19_relus = build_vgg19_relus(vgg19)
loss = get_loss(vgg19, vgg19_relus, epsilon=EPSILON, style_weight=STYLE_WEIGHT)

return model, loss
```

```
[42] content_ds, style_ds = part1_prepare_datasets()
```

```
model, loss = part1_setup()
```

Decoder Trainable Parameter: 3505219

**[Q5] Compare the architectures of the encoder and decoder. Discuss the usage of the decoder architecture.**

The architecture of the encoder is used to extract the content and style features of an image. This encoder is a convolutional neural network. The encoder consists of several convolutional layers followed by max-pooling layers. Each convolutional layer increases the depth (indicated by the number of filters), capturing more complex and abstract features. While max-pooling layers reduce the spatial dimensions (width and height) to increase the feature abstraction level. The spatial dimensions of the feature maps are progressively reduced through the max-pooling layers. It focuses on the high-level content and style information and ignoring the less relevant spatial details. The encoder that we use has several convolutional layers that started with 64 filters and gradually increases the number of filters used. It also has 3 max-pooling layers that reduces the spatial dimensions.

While the decoder architecture is used to reconstruct an image from the encoded features. This decoder applies the style of one image to another image while maintaining the content of the image. The decoder progressively upscales the feature maps and decrease the depth to reconstruct the important spatial details of the image while applying a style (of another image). This encoder consists of several convolutional layers with a decreasing number of filters and upsampling layers to increase the spatial dimensions. This decoder aims to restore the image size from the encoded features. The decoder only has 3 convolutional layers, each corresponding to RGB color channels. The last layer of the decoder usually does not use an activation function to output the raw pixel values of the reconstructed image. The decoder started with 256 filters and gradually decreases the number of filters used. It also has 3 upsampling layers that increases the size of the feature maps back to the original dimensions of the input image

The decoder is used to reverse the encoder's process. It reconstructs an image by gradually increases the spatial dimensions and decreases the feature complexity, to rebuild the image's original spatial structure while applying the style captured from another image. The last step of this decoder is a layer that maps the features back to the 3 color channels of an RGB image, producing the final styled output. So, the decoder takes the features created by the encoder and reconstructs the image while applying the desire style.

**[Q6] In the training of a style transfer model, why is it necessary to use a combination of style loss and content loss?**

The combination of both style and content loss is essential to transfer one image style to another image because content loss ensures that the generated image still maintains the same content as the original content image and the style loss ensure that the style of the generated image still has the resemblance of the reference style image.

The content loss measure how much the content of the generated image deviates from the content of the original image. It is usually calculated by comparing the feature representations of the generated image and the content image at one or more layers within a pre-trained convolutional

neural network. The assumption is that higher-level layers of the network capture the high-level content (such as objects and overall structure) but are invariant to the exact pixel values or textures.

Style loss, on the other hand, measures the style such as textures, colors, brush strokes, and other visual patterns characteristic of the style image. This is usually calculated by measuring the difference between the style representations of the generated image and the referenced style image.

Style transfer model that we are training is trying to find the right balance between both losses. If we have too much content loss, then we may end up generating an image that does not have the resemblance of the object of the original image. Conversely, if we have too much style loss, then we can end up generating an image that has the desired style but loses the content structure and details. During training, the model iteratively adjusts the generated image to minimize both losses, so that it can generate an image that still has the content of the original image while having the desired style of another 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.**

Loss obtained at the end of training: (Total Loss: 12674.256, Content Loss: 9113.542, Style Loss: 3560.720)

Epoch 1/15 total loss: 48976.746 content loss: 25764.078 style loss: 23212.641 : 100%	<div></div>	223/223 [14:29<00:00, 2.66s/it]
Epoch 2/15 total loss: 26209.648 content loss: 17849.488 style loss: 8360.160 : 100%	<div></div>	223/223 [1:03:06<00:00, 2.68s/it]
Epoch 3/15 total loss: 21182.285 content loss: 14797.994 style loss: 6384.292 : 100%	<div></div>	223/223 [52:58<00:00, 2.81s/it]
Epoch 4/15 total loss: 19625.424 content loss: 13724.279 style loss: 5901.140 : 100%	<div></div>	223/223 [42:53<00:00, 2.76s/it]
Epoch 5/15 total loss: 18065.221 content loss: 12699.059 style loss: 5366.173 : 100%	<div></div>	223/223 [32:49<00:00, 2.74s/it]
Epoch 6/15 total loss: 16272.464 content loss: 11552.032 style loss: 4720.429 : 100%	<div></div>	223/223 [22:42<00:00, 2.70s/it]
Epoch 7/15 total loss: 15744.856 content loss: 11165.249 style loss: 4579.604 : 100%	<div></div>	223/223 [12:38<00:00, 2.74s/it]
Epoch 8/15 total loss: 15264.040 content loss: 10866.781 style loss: 4397.255 : 100%	<div></div>	223/223 [1:01:52<00:00, 2.63s/it]
Epoch 9/15 total loss: 14672.465 content loss: 10418.860 style loss: 4253.597 : 100%	<div></div>	223/223 [51:44<00:00, 2.86s/it]
Epoch 10/15 total loss: 14408.471 content loss: 10216.730 style loss: 4191.749 : 100%	<div></div>	223/223 [41:37<00:00, 2.84s/it]
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file v saving_api.save_model()		
Epoch 11/15 total loss: 13672.497 content loss: 9787.842 style loss: 3884.652 : 100%	<div></div>	223/223 [31:30<00:00, 2.67s/it]
Epoch 12/15 total loss: 13564.698 content loss: 9635.744 style loss: 3928.961 : 100%	<div></div>	223/223 [21:24<00:00, 2.79s/it]
Epoch 13/15 total loss: 13372.546 content loss: 9520.195 style loss: 3852.343 : 100%	<div></div>	223/223 [11:18<00:00, 2.65s/it]
Epoch 14/15 total loss: 12835.462 content loss: 9190.096 style loss: 3645.370 : 100%	<div></div>	223/223 [10:09<00:00, 2.70s/it]
Epoch 15/15 total loss: 12674.256 content loss: 9113.542 style loss: 3560.720 : 100%	<div></div>	223/223 [10:08<00:00, 2.64s/it]

**[Q8] Demonstrate the model's ability to combine the content and style of images effectively, showcasing several examples in your report.**



## 7. Classification Style

**[Q9] Compare the distribution of image styles and labels between the training and test datasets. What is the difference between the two datasets?**

Training dataset				Test dataset			
	style	label	count		style	label	count
0	art_painting	dog	13	0	art_painting	dog	119
1	art_painting	elephant	13	1	art_painting	elephant	89
2	art_painting	giraffe	231	2	art_painting	giraffe	110
3	art_painting	guitar	10	3	art_painting	guitar	82
4	art_painting	horse	180	4	art_painting	horse	90
5	art_painting	house	11	5	art_painting	house	110
6	art_painting	person	11	6	art_painting	person	96
7	cartoon	dog	10	7	cartoon	dog	95
8	cartoon	elephant	13	8	cartoon	elephant	83
9	cartoon	giraffe	12	9	cartoon	giraffe	109
10	cartoon	guitar	121	10	cartoon	guitar	82
11	cartoon	horse	11	11	cartoon	horse	81
12	cartoon	house	12	12	cartoon	house	101
13	cartoon	person	12	13	cartoon	person	95
14	photo	dog	10	14	photo	dog	81
15	photo	elephant	13	15	photo	elephant	83
16	photo	giraffe	12	16	photo	giraffe	102
17	photo	guitar	10	17	photo	guitar	81
18	photo	horse	11	18	photo	horse	103
19	photo	house	215	19	photo	house	95
20	photo	person	211	20	photo	person	110
21	sketch	dog	229	21	sketch	dog	112
22	sketch	elephant	217	22	sketch	elephant	115
23	sketch	giraffe	10	23	sketch	giraffe	104
24	sketch	guitar	9	24	sketch	guitar	95
25	sketch	horse	8	25	sketch	horse	108
26	sketch	house	13	26	sketch	house	80
27	sketch	person	13	27	sketch	person	112

Both dataset has the same pairs of (style, label). However, both have different counts of these pairs. We can see that the test dataset has significantly more counts for each label than the training dataset. Also, we can see that some (style, label) pairs have significantly more data than some other pairs. For instance, (sketch, dog) and (photo, house) have lots of data, but (sketch, guitar) and (sketch, horse) has very less data (in the training dataset).

**[Q10] Write down your prediction on how the above difference would affect test-time performance.**

The above difference would affect test-time performance in several ways. The first one is this can cause overfitting. Because our model only has small training dataset, it cannot generalize well with unseen data (such as the noise or outliers in the test data). This is because our training data does not represent the full complexity and variability of the data. Thus, our model might fit our training

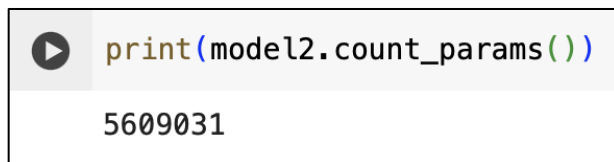
data too well and cannot perform comparably well on our test data. Aside from that, this might also cause imbalance class, where some classes are underrepresented because they do not have enough training samples. This can further decrease the performance of our model on the test set.

Aside from that, our model might also have higher variance in its output because it does not have strong predictive ability due to the lack of training that it gets. Hence, the model's performance metrics (such as accuracy in this case) might be significantly lower on the test set compared to the training set. Lastly, this can also be a problem if we are using deep neural network model with many layers. This means our model can easily overfit the limited training data that it has.

Thus, I predict that the test-time performance of our model might be significantly lower than its performance on the training dataset. Our model might have a very high training accuracy but much lower test accuracy. It might also perform very badly on imbalanced classes (those classes that are underrepresented in the training dataset). Thus, doing data augmentation (like what we did later in the project) might help increase the variability of the training data and increase our model performance.

**[Q11] Report the number of trainable parameters in the model.**

Number of trainable parameters: 5609031



```
print(model2.count_params())
```
















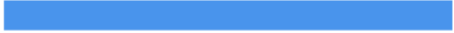




5609031




















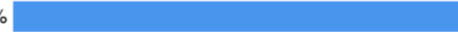


**[Q12] Train the model with the given training dataset for at least 100 epochs. Record the loss obtained at the end of training (or better, throughout training).**

The loss at the start of training is: **1.709**

The loss at the end of training is: **0.101**

Epoch 1/100 cross-entropy loss: 1.709 : 100%		26/26 [01:19<00:00, 1.54s/it]
Epoch 2/100 cross-entropy loss: 1.482 : 100%		26/26 [00:39<00:00, 1.56s/it]
Epoch 3/100 cross-entropy loss: 1.382 : 100%		26/26 [00:40<00:00, 1.57s/it]
Epoch 4/100 cross-entropy loss: 1.302 : 100%		26/26 [13:34<00:00, 1.55s/it]
Epoch 5/100 cross-entropy loss: 1.182 : 100%		26/26 [12:54<00:00, 1.54s/it]
Epoch 6/100 cross-entropy loss: 1.120 : 100%		26/26 [12:14<00:00, 1.55s/it]
Epoch 7/100 cross-entropy loss: 1.104 : 100%		26/26 [00:41<00:00, 1.55s/it]
Epoch 8/100 cross-entropy loss: 1.078 : 100%		26/26 [10:53<00:00, 1.55s/it]
Epoch 9/100 cross-entropy loss: 1.059 : 100%		26/26 [00:40<00:00, 1.55s/it]
Epoch 10/100 cross-entropy loss: 1.043 : 100%		26/26 [09:32<00:00, 1.57s/it]
Epoch 91/100 cross-entropy loss: 0.204 : 100%		26/26 [00:41<00:00, 1.61s/it]
Epoch 92/100 cross-entropy loss: 0.204 : 100%		26/26 [02:43<00:00, 1.58s/it]
Epoch 93/100 cross-entropy loss: 0.201 : 100%		26/26 [00:41<00:00, 1.57s/it]
Epoch 94/100 cross-entropy loss: 0.192 : 100%		26/26 [01:21<00:00, 1.55s/it]
Epoch 95/100 cross-entropy loss: 0.171 : 100%		26/26 [00:40<00:00, 1.56s/it]
Epoch 96/100 cross-entropy loss: 0.215 : 100%		26/26 [00:40<00:00, 1.56s/it]
Epoch 97/100 cross-entropy loss: 0.224 : 100%		26/26 [00:40<00:00, 1.56s/it]
Epoch 98/100 cross-entropy loss: 0.235 : 100%		26/26 [05:46<00:00, 1.57s/it]
Epoch 99/100 cross-entropy loss: 0.226 : 100%		26/26 [00:41<00:00, 1.57s/it]
Epoch 100/100 cross-entropy loss: 0.217 : 100%		26/26 [04:25<00:00, 1.58s/it]

Epoch 1/20 cross-entropy loss: 0.271 : 100%		26/26 [00:46<00:00, 1.63s/it]
Epoch 2/20 cross-entropy loss: 0.210 : 100%		26/26 [07:09<00:00, 1.65s/it]
Epoch 3/20 cross-entropy loss: 0.198 : 100%		26/26 [00:43<00:00, 1.62s/it]
Epoch 4/20 cross-entropy loss: 0.193 : 100%		26/26 [05:44<00:00, 1.63s/it]
Epoch 5/20 cross-entropy loss: 0.150 : 100%		26/26 [00:43<00:00, 1.65s/it]
Epoch 6/20 cross-entropy loss: 0.133 : 100%		26/26 [04:18<00:00, 1.65s/it]
Epoch 7/20 cross-entropy loss: 0.135 : 100%		26/26 [00:43<00:00, 1.68s/it]
Epoch 8/20 cross-entropy loss: 0.157 : 100%		26/26 [02:52<00:00, 1.67s/it]
Epoch 9/20 cross-entropy loss: 0.197 : 100%		26/26 [00:42<00:00, 1.67s/it]
Epoch 10/20 cross-entropy loss: 0.142 : 100%		26/26 [01:26<00:00, 1.64s/it]
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are s saving_api.save_model( Epoch 11/20 cross-entropy loss: 0.110 : 100%		
Epoch 11/20 cross-entropy loss: 0.110 : 100%		26/26 [00:43<00:00, 1.63s/it]
Epoch 12/20 cross-entropy loss: 0.097 : 100%		26/26 [00:42<00:00, 1.62s/it]
Epoch 13/20 cross-entropy loss: 0.113 : 100%		26/26 [00:43<00:00, 1.63s/it]
Epoch 14/20 cross-entropy loss: 0.089 : 100%		26/26 [00:42<00:00, 1.63s/it]
Epoch 15/20 cross-entropy loss: 0.108 : 100%		26/26 [00:43<00:00, 1.64s/it]
Epoch 16/20 cross-entropy loss: 0.112 : 100%		26/26 [00:42<00:00, 1.64s/it]
Epoch 17/20 cross-entropy loss: 0.099 : 100%		26/26 [00:45<00:00, 1.66s/it]
Epoch 18/20 cross-entropy loss: 0.102 : 100%		26/26 [00:45<00:00, 1.71s/it]
Epoch 19/20 cross-entropy loss: 0.116 : 100%		26/26 [00:43<00:00, 1.65s/it]
Epoch 20/20 cross-entropy loss: 0.101 : 100%		26/26 [00:43<00:00, 1.66s/it]

Note: The last picture is an additional 20 epochs for training the model. So, I trained the model for a total of 120 epochs.

**[Q13] Report the accuracy and confusion matrix of the model trained in the previous section, when tested against the training and test datasets respectively.**

Training dataset:

```
For Training Dataset
Accuracies: 0.91375
Confusion Matrix:
[[196  47   3   1   7   1   1]
 [  9 237   0   0   1   1   0]
 [  6   0 248   2   5   0   0]
 [  2   0   8 128   8   0   0]
 [  2   2   3   0 199   0   0]
 [  2   1   7   0   5 226   0]
 [  8   0   2   0   3   1 228]]
```

Test dataset:

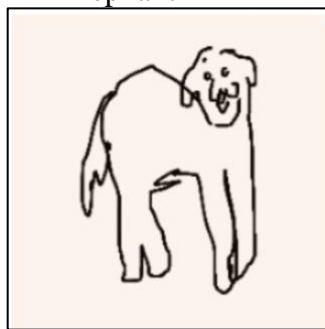
```
For Test Dataset
Accuracies: 0.45498511904761907
Confusion Matrix:
[[ 92  59  90  18 121   9  12]
 [ 31 175  66   9  61   9  15]
 [ 63  19 243  33  28  24   9]
 [ 23  12  42 168  73   4  14]
 [ 90  32  43  14 173  12  14]
 [  5  10 108  12  40 204   4]
 [ 50  20  79   9  58  21 168]]
```

**[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.**

Mislabeled sample images in Training Dataset:



1. The actual label is: Class 0 = Dog
2. The predicted label is: Class 1 = Elephant



1. The actual label is: Class 0 = Dog
2. The predicted label is: Class 1 = Elephant

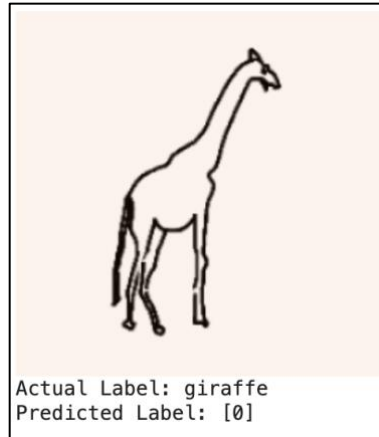
Mislabeled sample images in Test Dataset:



1. The actual label is: Class 0 = Dog
2. The predicted label is: Class 4 = Horse



1. The actual label is: Class 3 = Guitar
2. The predicted label is: Class 4 = Horse









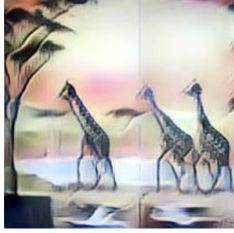


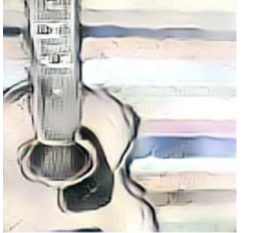

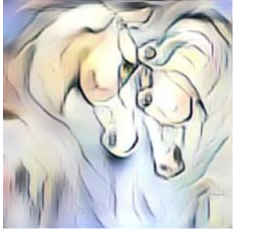
1. The actual label is: Class 2 = Giraffe
2. The predicted label is: Class 0 = Dog

```
Label Names: ['label_dog', 'label_elephant', 'label_giraffe',  
'label_guitar', 'label_horse', 'label_house', 'label_person']
```









We can see that the accuracy for the training dataset is close to 100% but the accuracy for the test dataset is significantly lower than that (around 45.5%). This is possible because the model that we have is trained based on the training dataset. So, it is expected that our model will perform very well when we pass the same training dataset into it. However, this is not the case for the test dataset. When we pass the test dataset to our model, we are passing a data that have not been seen before. Thus, it is very likely that the model will give us the wrong output every once a while. So, this explains why the accuracy score for the unseen data (test dataset) can be lower than the accuracy score for the training dataset and our model has more incorrect predictions in the test dataset towards the training dataset.

These failure cases might happen because of several reasons that confuses our model and causes it to give the wrong label. For the first training dataset failure, I think this might happen because the image of the dog has a comparably long nose, which causes the model to label it as an elephant. For the second training dataset failure, I think this might shows some features that are similar to those of elephant. For the first test dataset failure, I think the model might confuse the dog as a horse because most dog pictures in the training dataset are sketches and this particular picture has a more realistic feature. I think this might cause the model to think of the dog as another picture, which is horse. The second test dataset failure might be caused by the available training dataset for guitar are mostly cartoons. So, something realistic like the given picture is unfamiliar for the model. Thus, the model labeled it incorrectly, not as a guitar, but as a horse. For the last failure example, similar to previous reasons. This failure might happen because most sketches that are given in the training dataset are dog pictures. Thus, when given this picture, our model thought it is a dog picture and labeled it as a dog, when it is actually a giraffe.

**[Q15] Showcase some samples from the generated dataset, with at least one per object label, and one per style.**

Object Label	Art Painting	Cartoon	Photo	Sketch
Dog	 dog_art_painting_19.jpg	 dog_cartoon_12.jpg	 dog_photo_1.jpg	 dog_sketch_6.jpg
Elephant	 elephant_art_painting_10.jpg	 elephant_cartoon_19.jpg	 elephant_photo_0.jpg	 elephant_sketch_1.jpg
Giraffe	 giraffe_art_painting_0.jpg	 giraffe_cartoon_0.jpg	 giraffe_photo_0.jpg	 giraffe_sketch_13.jpg
Guitar	 guitar_art_painting_10.jpg	 guitar_cartoon_10.jpg	 guitar_photo_1.jpg	 guitar_sketch_15.jpg
Horse	 horse_art_painting_10.jpg	 horse_cartoon_1.jpg	 horse_photo_13.jpg	 horse_sketch_14.jpg




















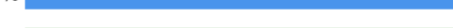



House	 <p>house_art_painting_0.jpg</p>	 <p>house_cartoon_1.jpg</p>	 <p>house_photo_10.jpg</p>	 <p>house_sketch_1.jpg</p>
Person	 <p>person_art_painting_0.jpg</p>	 <p>person_cartoon_11.jpg</p>	 <p>person_photo_16.jpg</p>	 <p>person_sketch_19.jpg</p>

**[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.**

The loss at the start of the training: **0.608**

The loss at the end of the training: **0.002**

Epoch 1/100 cross-entropy loss: 0.608 : 100%		35/35 [01:23<00:00, 1.73s/it]
Epoch 2/100 cross-entropy loss: 0.376 : 100%		35/35 [03:54<00:00, 1.66s/it]
Epoch 3/100 cross-entropy loss: 0.349 : 100%		35/35 [00:58<00:00, 1.68s/it]
Epoch 4/100 cross-entropy loss: 0.343 : 100%		35/35 [01:56<00:00, 1.66s/it]
Epoch 5/100 cross-entropy loss: 0.273 : 100%		35/35 [00:58<00:00, 1.66s/it]
Epoch 6/100 cross-entropy loss: 0.262 : 100%		35/35 [00:58<00:00, 1.66s/it]
Epoch 7/100 cross-entropy loss: 0.279 : 100%		35/35 [00:59<00:00, 1.64s/it]
Epoch 8/100 cross-entropy loss: 0.270 : 100%		35/35 [21:14<00:00, 1.65s/it]
Epoch 9/100 cross-entropy loss: 0.253 : 100%		35/35 [00:57<00:00, 1.63s/it]
Epoch 10/100 cross-entropy loss: 0.223 : 100%		35/35 [19:19<00:00, 1.63s/it]
Epoch 90/100 cross-entropy loss: 0.001 : 100%		35/35 [01:00<00:00, 1.91s/it]
Epoch 91/100 cross-entropy loss: 0.003 : 100%		35/35 [00:57<00:00, 1.65s/it]
Epoch 92/100 cross-entropy loss: 0.000 : 100%		35/35 [01:01<00:00, 1.65s/it]
Epoch 93/100 cross-entropy loss: 0.001 : 100%		35/35 [00:58<00:00, 1.75s/it]
Epoch 94/100 cross-entropy loss: 0.002 : 100%		35/35 [00:58<00:00, 1.67s/it]
Epoch 95/100 cross-entropy loss: 0.003 : 100%		35/35 [00:59<00:00, 1.65s/it]
Epoch 96/100 cross-entropy loss: 0.002 : 100%		35/35 [00:59<00:00, 1.73s/it]
Epoch 97/100 cross-entropy loss: 0.000 : 100%		35/35 [00:58<00:00, 1.66s/it]
Epoch 98/100 cross-entropy loss: 0.000 : 100%		35/35 [00:57<00:00, 1.64s/it]
Epoch 99/100 cross-entropy loss: 0.001 : 100%		35/35 [00:57<00:00, 1.65s/it]
Epoch 100/100 cross-entropy loss: 0.002 : 100%		35/35 [01:00<00:00, 1.80s/it]



**[Q17] Test the new model against the test (and un-augmented training) dataset, then report and compare the result with one before augmentation.**

For the augmented training dataset:

For the test dataset:

<b>For Training Dataset</b> <b>Accuracies: 0.9967830882352942</b> <b>Confusion Matrix:</b> [[356 2 0 0 0 0 5] [ 0 322 0 0 0 0 0] [ 0 0 317 0 0 0 0] [ 0 0 0 230 0 0 0] [ 0 0 0 0 283 0 0] [ 0 0 0 0 0 349 0] [ 0 0 0 0 0 0 312]]	<b>For Test Dataset</b> <b>Accuracies: 0.4973958333333333</b> <b>Confusion Matrix:</b> [[125 29 60 42 111 15 22] [ 39 149 65 15 60 12 23] [ 57 15 235 35 33 35 8] [ 21 6 28 211 41 9 21] [ 90 26 35 28 164 23 11] [ 9 9 74 28 19 228 14] [ 19 25 60 22 47 10 225]]
---	---

For the unaugmented training dataset:

<b>For Training Dataset</b> <b>Accuracies: 0.9975</b> <b>Confusion Matrix:</b> [[247 1 0 0 0 0 3] [ 0 251 0 0 0 0 0] [ 0 0 259 0 0 0 0] [ 0 0 0 148 0 0 0] [ 0 0 0 0 209 0 0] [ 0 0 0 0 0 243 0] [ 0 0 0 0 0 0 239]]
---

From these results, we can see that the new model's accuracy is higher than previous model (the model trained with un-augmented training dataset). This is similar to what I have suggested before: the model performance might get better if we do some augmentation for the training dataset to increase the variability of the training dataset. Thus, this leads to the new model performing better than previous model.

For the confusion matrix, the diagonal values represent the number of correct predictions for each class. From here, we can see that the higher the accuracies, the higher the values in the diagonal are. For the off-diagonal values, it represents the number of incorrect predictions (misclassifications). Lower values are more desired for the off-diagonal and higher values are more desired for the diagonal values. In both training datasets, we can see that the off-diagonal values are mostly zeros, which means there are not a lot of misclassifications. Aside from the diagonals, we can also analyze the rows of the confusion matrix. If the off-diagonal values of the row are high, it means that class is frequently misclassified. Whereas for the column analysis, columns with a high off-diagonal values means that other class's objects are frequently misclassified as that class. From these results, we can see that our new model generally has higher diagonal values than previous model and generally lower off-diagonal values (in most cases). This means that our new model has better predictive abilities than the previous model.

**[Q18] Give a possible explanation on the two results (e.g., how the augmentation affected the test-time performance)**

Augmentation affected our model's test-time performance in a good way. It increases the diversity of our training dataset, allowing our model to be trained using more variety of training data. By increasing the number of training data, our model is prevented from memorizing the exact details of the training set and leads to better generalization and reduced overfitting. Also, our model is going to be more robust to variations and can learn better due to an increase in the training data variations. Augmentation also addresses the class imbalance problem by generating more examples of underrepresented classes. Thus, increasing our model's predictive ability on these classes. Thus, by using augmentation, our new model is able to increase its predictive ability.