EEE598: Deep Learning Assignment 1

Chih-Hao Tsai

Arizona State University

September 3, 2024

1 Problem 1

In this section, I trained a classifier with the help with Teachable Machine as required. And it classifies three different species from cats, dogs, and bears.

1.1 My datasets

My datasets consist of 210 photos in total, 70 for each class, varying in size and resolution. Furthermore, my datasets include images of the same species but with different age groups or breeds, as well as images taken from different angles and motions of the three animals.

My datasets were all collected from *Pinterest*, and here's the *link* to my datasets.

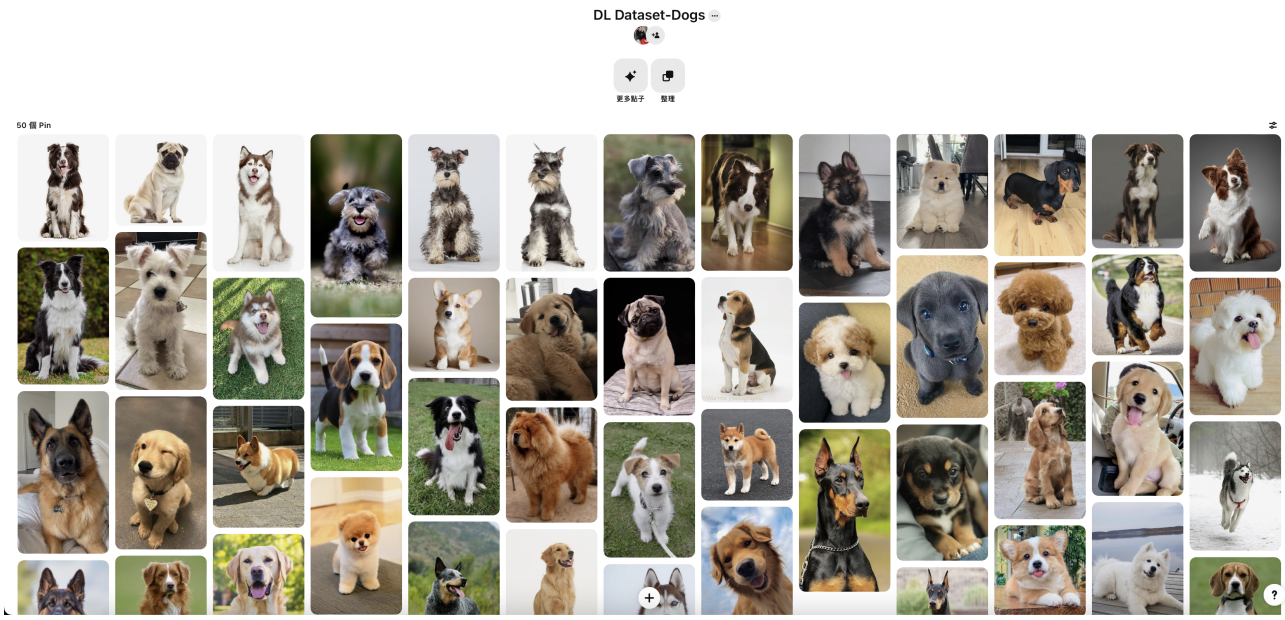

Fig. 1: This is the dataset of dogs

1.2 Split the datasets and the first try

Initially, I split my datasets into 43/7 (training/validation) for each class. With the default training settings (Epochs: 50, Batch Size: 16, Learning Rate: 0.001), most of the validations of three species got the correct answers to nearly 100%.

However, a few errors still occurred. For example, when classifying Fig.3, the accuracy drops below 85%. Furthermore, the errors occur more when classifying dogs. The accuracy declines below 90% when testing Fig.4. The accuracy even plummeted to nearly 55% when validating Fig.5.

Link to my first attempt → *First attempt*

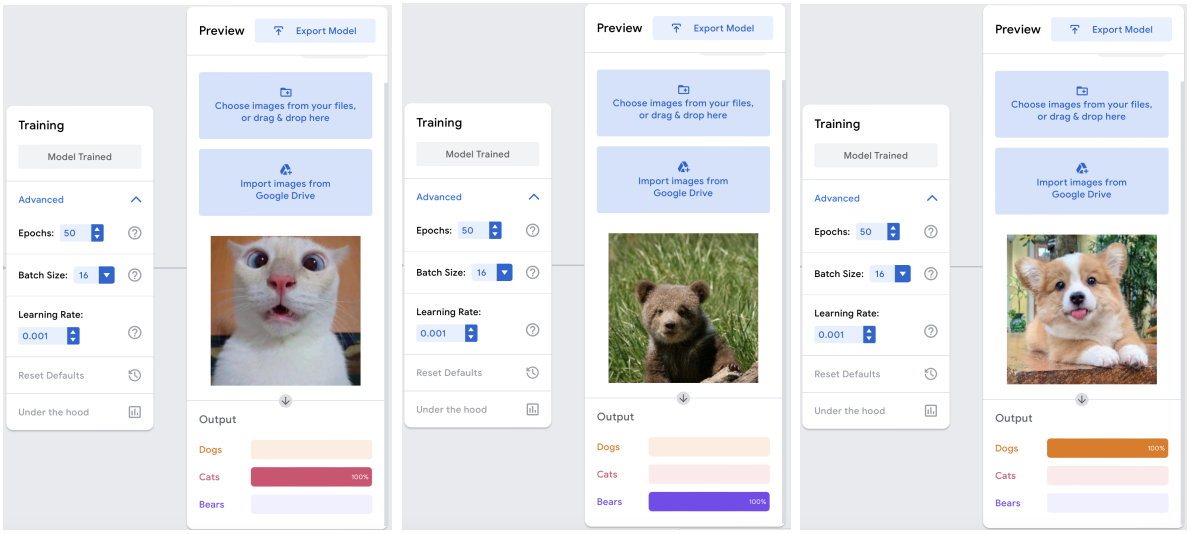

Fig. 2: Most of the validations are nearly 100%

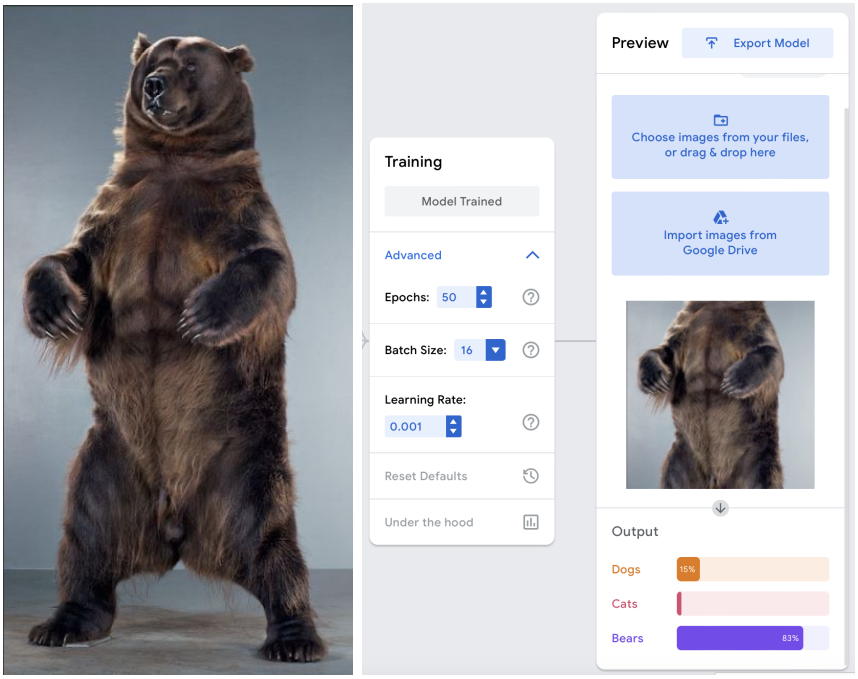

Fig. 3: The accuracy drops to 83% when classifying this picture

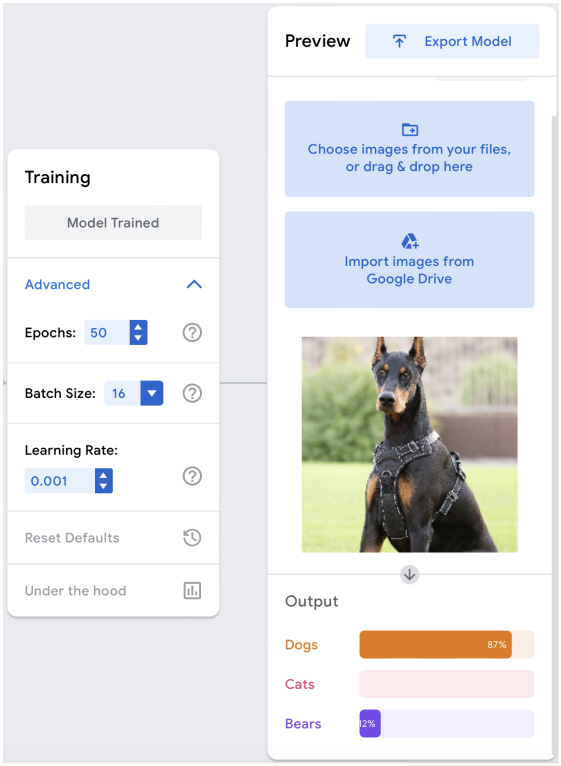

Fig. 4: The accuracy declines to 87%

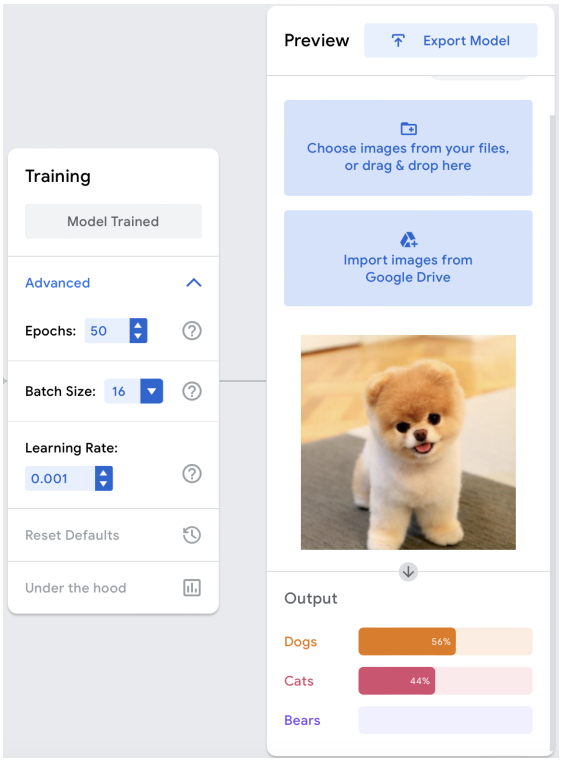

Fig. 5: The accuracy decreases to 56%

1.3 Optimize the dataset and parameters

After my first attempt, I came up with some ideas that may attribute to the inaccuracy. First, I believe the main reason is the deficiency in my datasets. Secondly, some low-resolution images may also cause the errors. Last but not least, some parameters should also be adjusted to improve the output accuracy.

I added 20 more images per class and removed a few low-resolution ones in order to optimize the output. After that, I tried a few different combinations of settings, but it didn't work properly until I set the epochs to 140, learning size to 0.00098, and the batch size remain 16. With the optimized classifier, most of the errors are corrected and the accuracy is much higher than the previous one. Surprisingly, the accuracy of classifying dogs increases the most.

Link to my optimized classifier→ *Optimized*

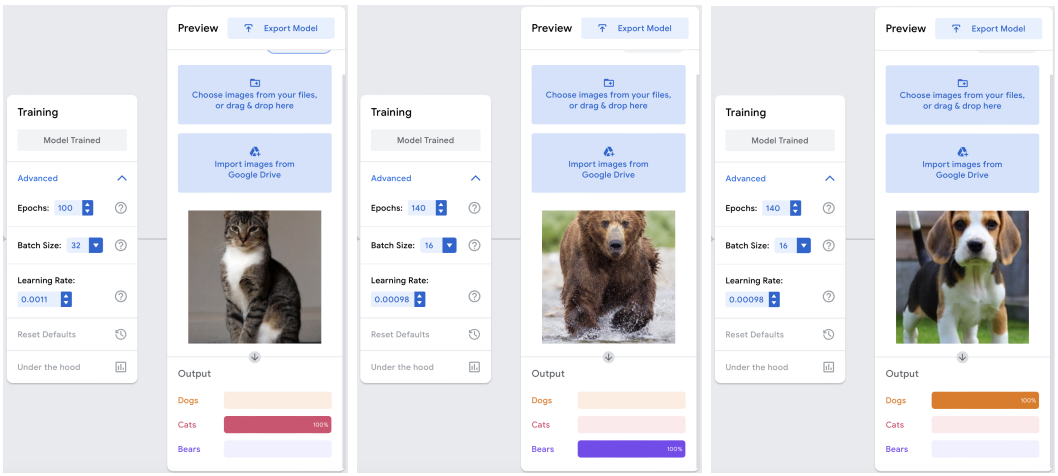

Fig. 6: The accuracy remains nearly 100%

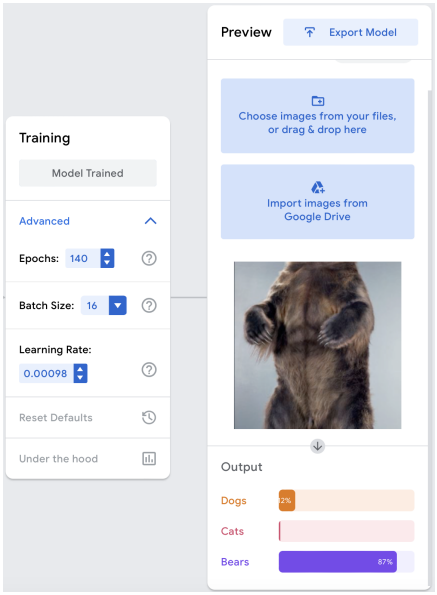

Fig. 7: The accuracy increases to 87%, which is 4% more than the previous result

The most salient feature is the accuracy of classifying dogs.

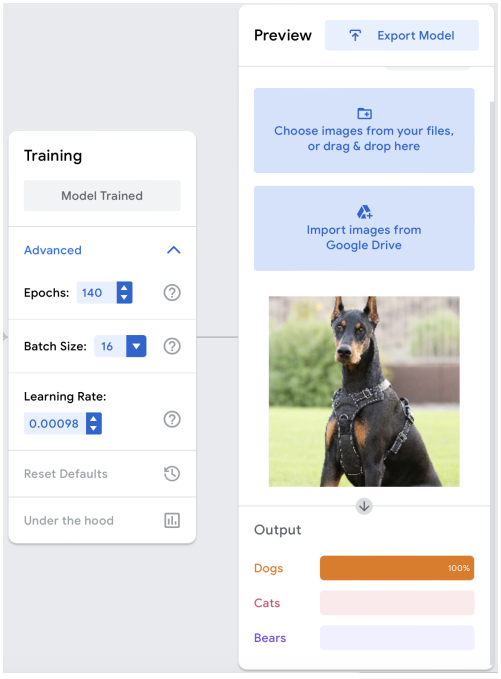

Fig. 8: The accuracy raises to 100%

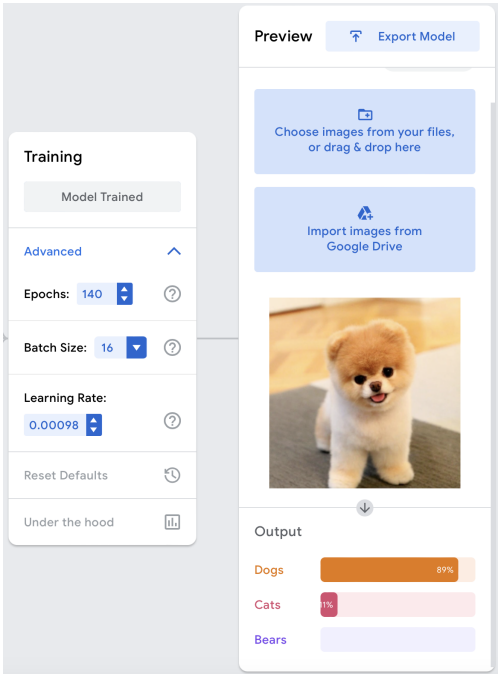

Fig. 9: The accuracy increases to 89%, 33% more than the original one

2 Problem 2

In this session, I followed the instructions from the slides provided by TA as well as seeked for assistance online.

2.1 Conda Environment Setup

I successfully created my Conda environment called "Tsai" and installed several required packages including Pytorch, tqdm. However, I was unable to install the "glob" package at first. After searching online for assistance, I figured out that the package "glob" was a standard library of Python. As the result, the "glob" module cannot show its version.

```
[7]: print('Pytorch version: ', torch.__version__)
     print('tqdm version: ', tqdm.__version__)
     print('glob version: ', glob.__version__)

Pytorch version:  2.4.1
tqdm version:  4.66.5
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[7], line 3
      1 print('Pytorch version: ', torch.__version__)
      2 print('tqdm version: ', tqdm.__version__)
----> 3 print('glob version: ', glob.__version__)

AttributeError: module 'glob' has no attribute '__version__'
```

Fig. 10: AttributeError occurs when printing the version of "glob"

2.2 GPU allocation and Jupyter Session

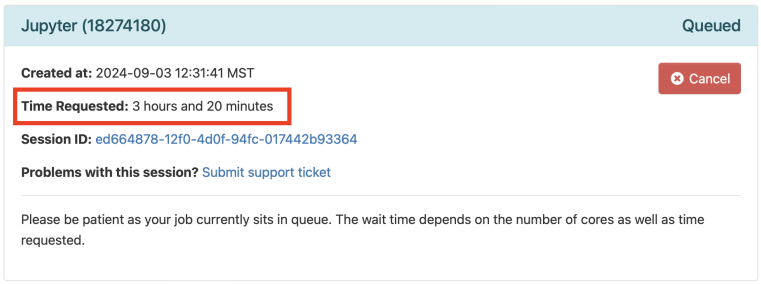

Fig. 11: Access to A100 GPU for 3 hours and 20 minutes

```
(Tsai) [ctsai67@sg040:~/EEE598]$ nvidia-smi
Tue Sep  3 14:12:12 2024
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 555.58.02              Driver Version: 555.58.02      CUDA Version: 12.5     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA A100-SXM4-80GB          On  |   00000000:41:00.0 Off |                    0 |
| N/A   28C    P0             59W /  500W |       1MiB /  81920MiB |      0%      Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
```

Fig. 12: Where the Nvdia A100 GPU is located.

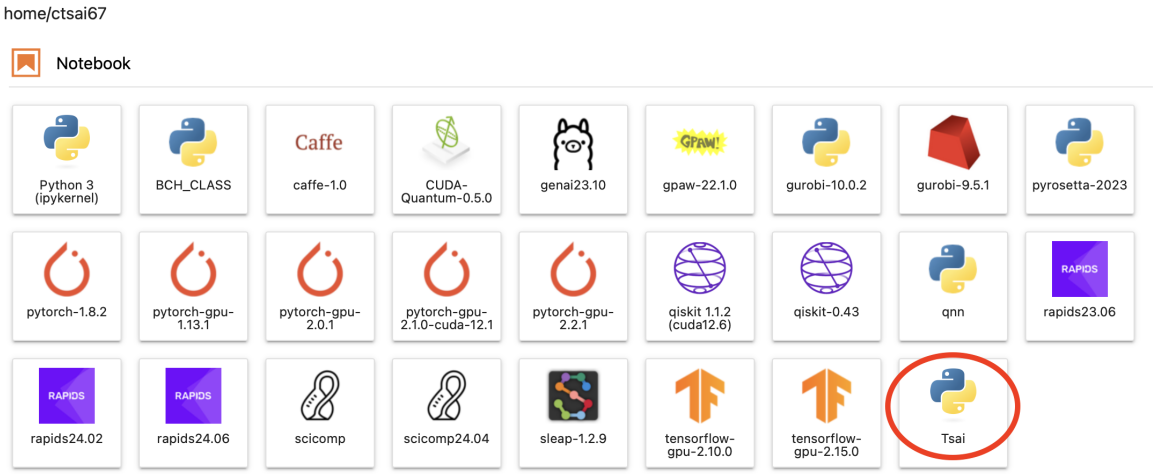

Fig. 13: My Conda envirnment is listed inside SOL

3 Problem 3

3.1 Tutorial

3.2 Tensor Operations

```python
# Create two tensors with the same shape and data type
tensor_1 = torch.rand(5, 4, dtype=torch.float32)
tensor_2 = torch.rand_like(tensor_1)

# Element-wise multiplication
tensor_3 = tensor_1 * tensor_2

# Reshape the result tensor
tensor_4 = tensor_3.reshape(10, 2)
```

Fig. 14: This is the code for creating tensor, element-wise multiplication, and reshaping.

```
The original input tensors:
tensor1:
tensor([[0.0498, 0.5944, 0.5188, 0.9836],
        [0.6118, 0.5261, 0.8757, 0.9961],
        [0.3453, 0.8656, 0.2619, 0.1507],
        [0.9784, 0.6998, 0.2307, 0.8262],
        [0.3076, 0.2869, 0.9646, 0.0600]])

tensor2:
tensor([[0.7773, 0.4111, 0.7631, 0.7681],
        [0.2427, 0.6074, 0.7404, 0.7814],
        [0.9448, 0.3802, 0.3026, 0.0189],
        [0.8006, 0.0814, 0.9669, 0.9231],
        [0.4894, 0.7302, 0.3518, 0.4726]])

The result of element-wise multiplication:
  tensor([[0.0387, 0.2443, 0.3959, 0.7555],
        [0.1485, 0.3195, 0.6483, 0.7783],
        [0.3262, 0.3291, 0.0793, 0.0029],
        [0.7834, 0.0570, 0.2231, 0.7627],
        [0.1505, 0.2095, 0.3394, 0.0283]])

The data type of the tensors:
tensor1: torch.float32,
tensor2: torch.float32,

The reshaped output tensor:
tensor([[0.0387, 0.2443],
        [0.3959, 0.7555],
        [0.1485, 0.3195],
        [0.6483, 0.7783],
        [0.3262, 0.3291],
        [0.0793, 0.0029],
        [0.7834, 0.0570],
        [0.2231, 0.7627],
        [0.1505, 0.2095],
        [0.3394, 0.0283]]),
size:torch.Size([10, 2])
```

Fig. 15: The output of Fig. 14.

3.3 Autograd Example

```python
data_1 = [[2, -1, 3], [4, 0, -2], [5, 7, -4]]
data_2 = [[-1, 2, 1], [3, -1, 2], [1, 2, -1]]

# turning array into tensor
x = torch.tensor(data_1, dtype = torch.float32, requires_grad = True)
multiplier_tensor = torch.tensor(data_2, dtype = torch.float32)
value = 3

# adding
addition = x + value
# c = torch.add(a, value)

# element-wise multiplication
multiplication = addition * multiplier_tensor

# summing elements
result = torch.sum(multiplication)

# compute gradient with backpropagation
result.backward()
```

Fig. 16: This is the code for creating tensors, adding a scale value, element-wise multiplication, summing the elements, and computing gradient.

```
The original tensor:
tensor([[ 2., -1.,  3.],
        [ 4.,  0., -2.],
        [ 5.,  7., -4.]], requires_grad=True)

The result after adding the scale value:
tensor([[ 5.,  2.,  6.],
        [ 7.,  3.,  1.],
        [ 8., 10., -1.]], grad_fn=<AddBackward0>)

The result after element-wise multiplication:
tensor([[-5.,  4.,  6.],
        [21., -3.,  2.],
        [ 8., 20.,  1.]], grad_fn=<MulBackward0>)

The scale output obtained by summing the elements: 54.0

The gradient of the scale output with respect to the original tensor:
tensor([[-1.,  2.,  1.],
        [ 3., -1.,  2.],
        [ 1.,  2., -1.]])
```

Fig. 17: The output of Fig. 16

3.4 Neural Network Parameters

```
The initail network architechture:
Net(
  (conv1): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=10, bias=True)
)

The total number of parameters: 61706

Using cuda device right now.
```

Fig. 18: This the initial network architecture, its total number of parameter, and the confirmation that it is running on GPU.

```
The initail network architechture:
Net(
  (conv1): Conv2d(1, 3, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(3, 8, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=200, out_features=40, bias=True)
  (fc2): Linear(in_features=40, out_features=23, bias=True)
  (fc3): Linear(in_features=23, out_features=14, bias=True)
)

The total number of parameters: 10005

Using cuda device right now.
```

Fig. 19: This is my modified version of network architecture with a parameter count as 10005.