

# Computer Vision Assignment 01: Introduction to PyTorch Report

---

## Computer Vision Assignment 01: Introduction to PyTorch Report

### 2 Simple 2D Classifier

#### 2.3 Training loop

#### 2.4 Multi-layer perceptron

#### 2.5 Feature transform

### 3 Digit classifier

#### 3.3 Multi-layer perceptron

#### 3.4 Convolutional network

#### 3.5 Comparison of number of parameters

#### 3.6 Confusion matrix

## 2 Simple 2D Classifier

---

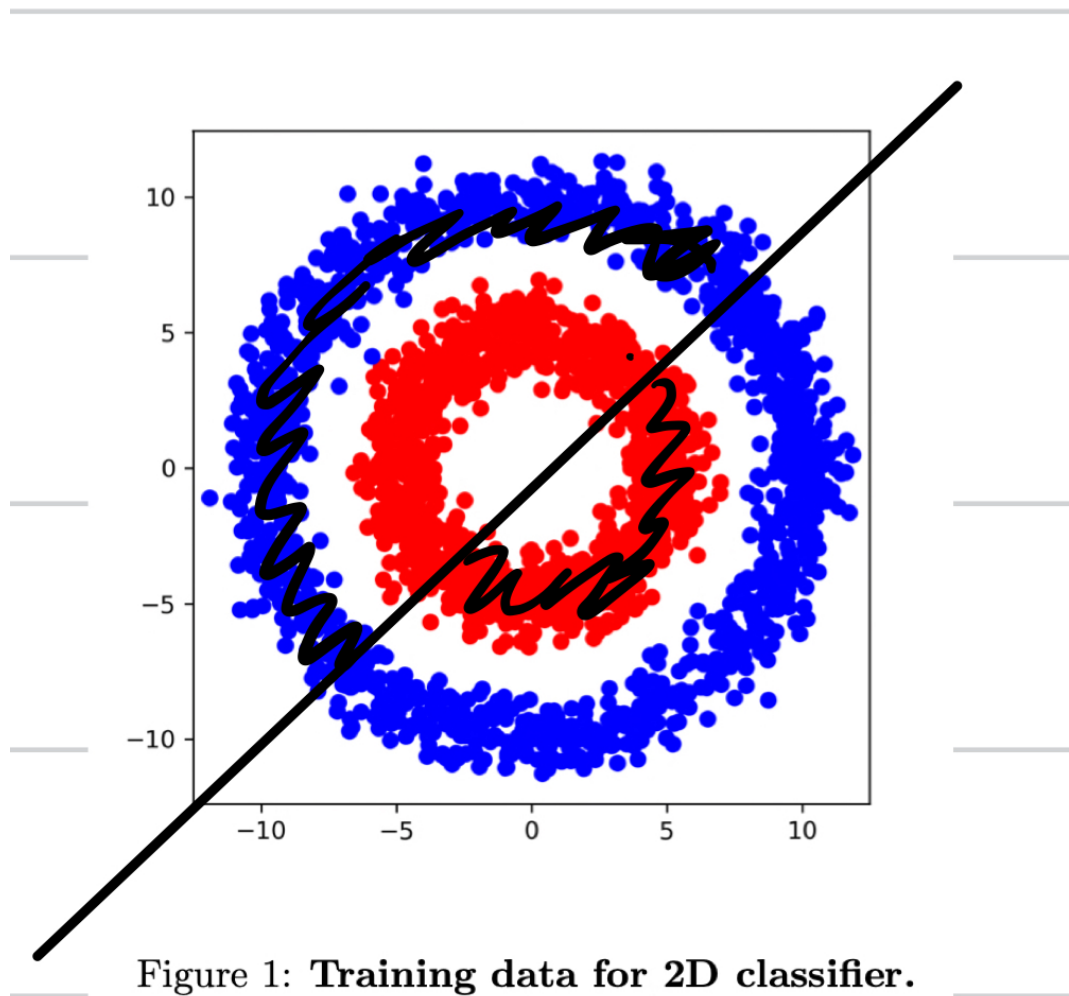
### 2.3 Training loop

**Question:** Once this is done, you should be able to run the `train.py` script. What accuracy do you achieve with the linear classifier? Is this an expected result? Justify your answer.

```
1 >>> python train.py
2 [Epoch 01] Acc.: 43.0556%
3 [Epoch 02] Loss: 0.7339
4 [Epoch 02] Acc.: 40.6746%
5 [Epoch 03] Loss: 0.7000
6 [Epoch 03] Acc.: 49.8016%
7 [Epoch 04] Loss: 0.6960
8 [Epoch 04] Acc.: 32.9365%
9 [Epoch 05] Loss: 0.6942
10 [Epoch 05] Acc.: 40.6746%
11 [Epoch 06] Loss: 0.6934
12 [Epoch 06] Acc.: 41.8651%
13 [Epoch 07] Loss: 0.6933
14 [Epoch 07] Acc.: 44.6429%
15 [Epoch 08] Loss: 0.6933
16 [Epoch 08] Acc.: 47.0238%
17 [Epoch 09] Loss: 0.6932
18 [Epoch 09] Acc.: 46.6270%
19 [Epoch 10] Loss: 0.6933
20 [Epoch 10] Acc.: 48.6111%
```

- The accuracy is `48.6111%` which is lower than expected (`50%` for random guess). However, the data shown below is not linearly separable.

- If we analyze the distribution of the data, we can find that there are three cases (we assume the sizes of two classes are the same):
  - the decision boundary go through the center of circle => accuracy equals to 50%



- the decision boundary have intersection with circle, but does not go through the center => accuracy larger than 50%

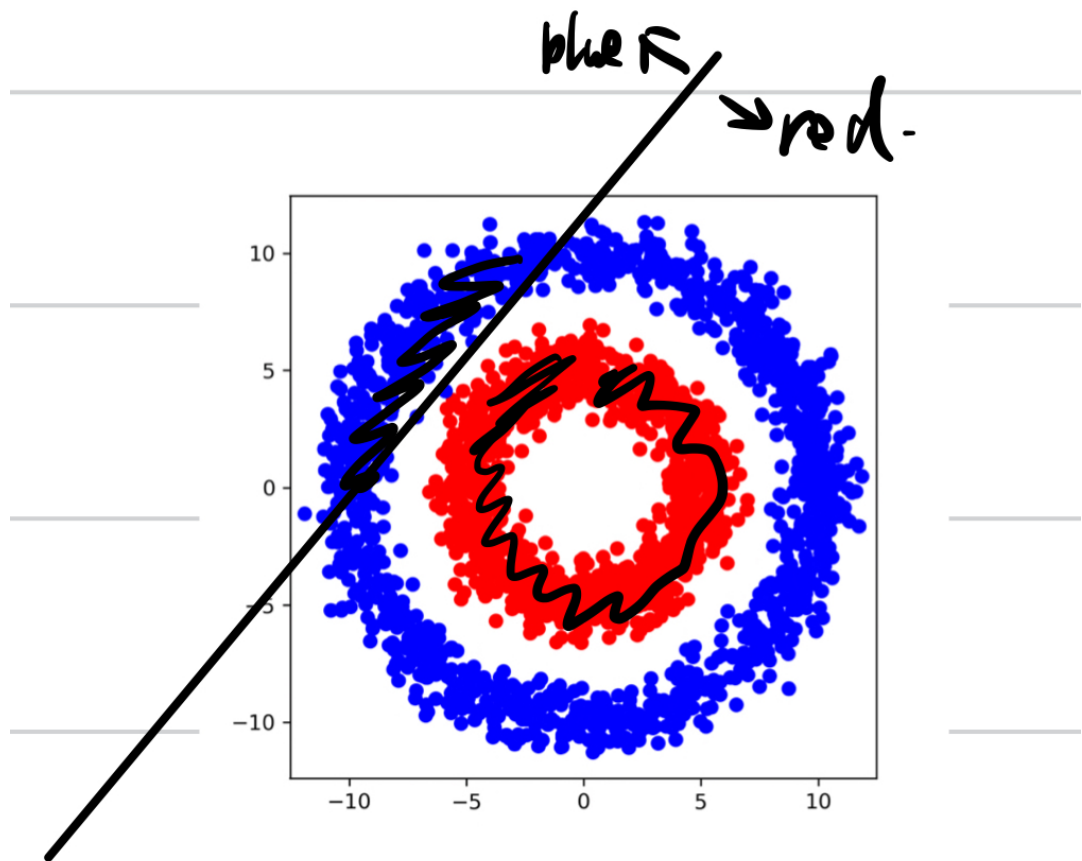


Figure 1: **Training data for 2D classifier.**

- the decision boundary does not have intersection with circle => accuracy equals to 50%

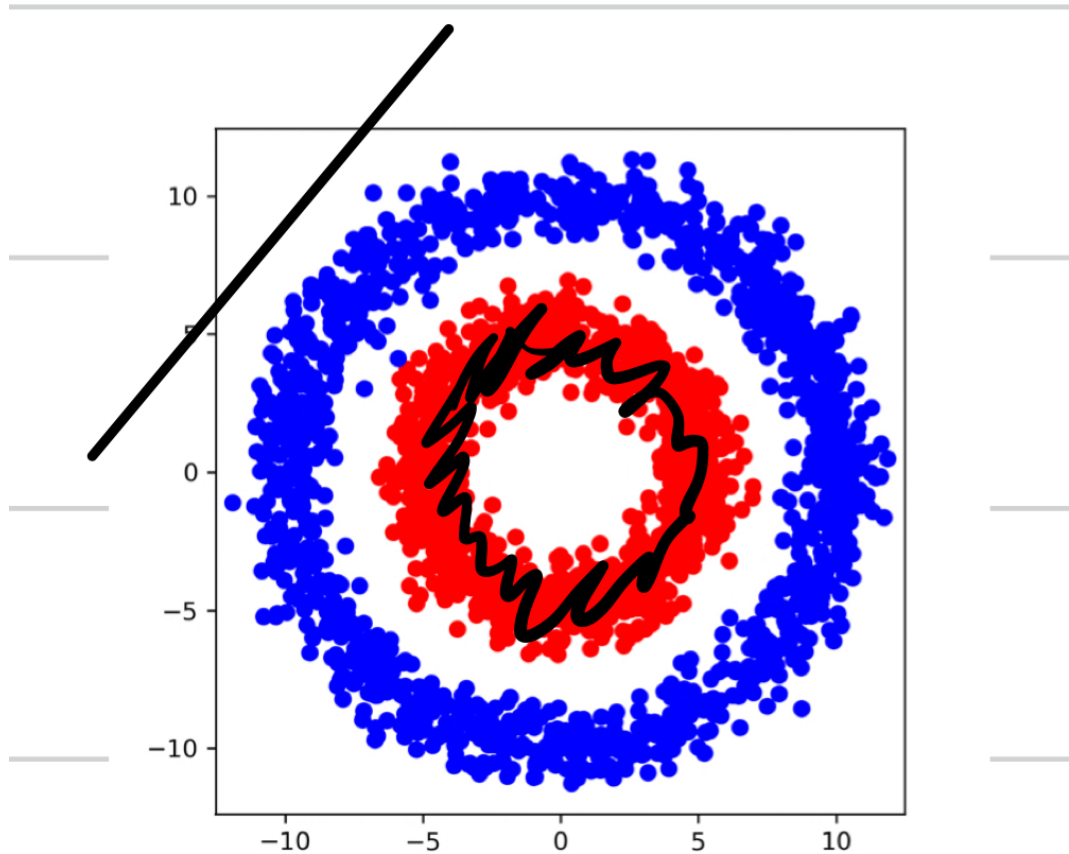


Figure 1: Training data for 2D classifier.

- Therefore, the accuracy should be larger than 50%, but the result accuracy is less than 50%, which is **not expected**. This implies that the linear model is not well converged (may be land in some local minimum) when the dataset is not linearly separable.

## 2.4 Multi-layer perceptron

**Question:** Switch to the new network by uncommenting `L83` in `train.py`. What accuracy does this network obtain? Why are the results better compared to the previous classifier?

```
1 >>> python train.py
2 [Epoch 01] Loss: 0.6409
3 [Epoch 01] Acc.: 51.3889%
4 [Epoch 02] Loss: 0.5039
5 [Epoch 02] Acc.: 95.2381%
6 [Epoch 03] Loss: 0.2507
7 [Epoch 03] Acc.: 99.4048%
8 [Epoch 04] Loss: 0.0961
9 [Epoch 04] Acc.: 99.6032%
10 [Epoch 05] Loss: 0.0484
11 [Epoch 05] Acc.: 99.6032%
```

```

12 [Epoch 06] Loss: 0.0299
13 [Epoch 06] Acc.: 99.6032%
14 [Epoch 07] Loss: 0.0210
15 [Epoch 07] Acc.: 99.6032%
16 [Epoch 08] Loss: 0.0160
17 [Epoch 08] Acc.: 99.6032%
18 [Epoch 09] Loss: 0.0128
19 [Epoch 09] Acc.: 99.8016%
20 [Epoch 10] Loss: 0.0111
21 [Epoch 10] Acc.: 99.8016%

```

- The final accuracy is 99.8016%, which is better than the linear classifier.
- The hidden layers and activation functions introduce non-linearity into the model. According to the Universal Approximation Theorem, any continuous function  $C(I_n)$  on the n-dimensional unit cube  $I_n$  can be approximated by a MLP with a hidden layer (with enough number of neurons). That means the front layers of MLP will automatically transform the coordinate to make the data linearly separable for the last layer. Therefore, this structure improves the performance.

## 2.5 Feature transform

**Question:** Think of a coordinate system that renders the two classes linearly separable and justify your choice.

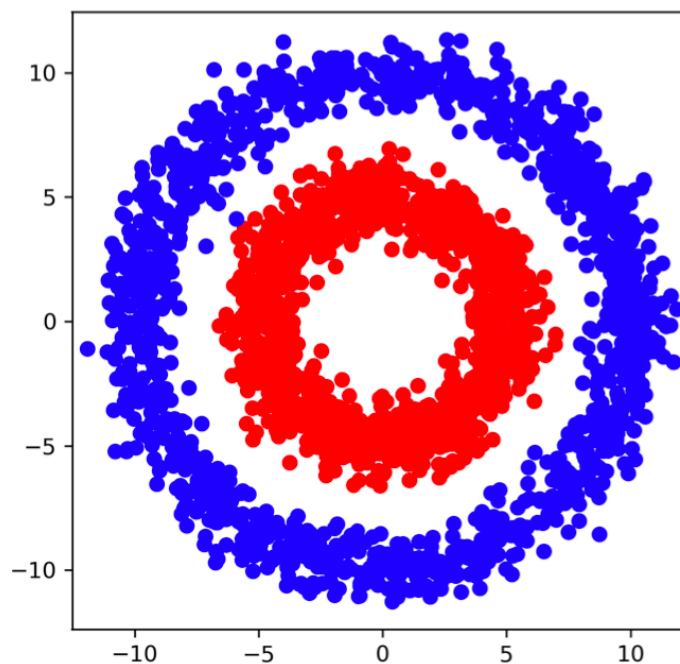


Figure 1: Training data for 2D classifier.

- By observation, the data is not linearly separable since it is a circle. However, by changing the coordinate system to polar system or simply use  $(x, y) \mapsto (x^2, y^2)$ . Then the data will distribute on two lines:

$$\begin{aligned}x^2 + y^2 &= r_1 \\x^2 + y^2 &= r_2\end{aligned}\tag{1}$$

Therefore, the classes become linearly separable.

**Question:** Verify the hypothesis by training a linear classifier on the new representation.

```
1 >>> python train.py
2 [Epoch 01] Loss: 0.7104
3 [Epoch 01] Acc.: 50.0000%
4 [Epoch 02] Loss: 0.6177
5 [Epoch 02] Acc.: 50.0000%
6 [Epoch 03] Loss: 0.5581
7 [Epoch 03] Acc.: 50.1984%
8 [Epoch 04] Loss: 0.5060
9 [Epoch 04] Acc.: 53.3730%
10 [Epoch 05] Loss: 0.4578
11 [Epoch 05] Acc.: 71.4286%
12 [Epoch 06] Loss: 0.4140
13 [Epoch 06] Acc.: 76.9841%
14 [Epoch 07] Loss: 0.3766
15 [Epoch 07] Acc.: 91.0714%
16 [Epoch 08] Loss: 0.3412
17 [Epoch 08] Acc.: 93.6508%
18 [Epoch 09] Loss: 0.3106
19 [Epoch 09] Acc.: 97.8175%
20 [Epoch 10] Loss: 0.2824
21 [Epoch 10] Acc.: 99.6032%
```

- Although it does not converge as fast as the MLP model, the accuracy is significantly improved.

## 3 Digit classifier

### 3.3 Multi-layer perceptron

**Question:** Implement a linear classifier by filling in the `MLPClassifier` class from `lib/networks.py`. What performance do you obtain? `91.2600%`

```
1 self.layers = nn.Sequential(
2     nn.Linear(784, 10)
3 )
```

```
1 >>> python train.py
2 100%|██████████| 3750/3750 [00:30<00:00, 122.82it/s]
3 [Epoch 01] Loss: 0.4088
4 [Epoch 01] Acc.: 90.8600%
5 100%|██████████| 3750/3750 [00:30<00:00, 121.56it/s]
6 [Epoch 02] Loss: 0.3350
```

```

7 [Epoch 02] Acc.: 89.1800%
8 100%|████████████████████████████████████████| 3750/3750 [00:30<00:00, 121.43it/s]
9 [Epoch 03] Loss: 0.3208
10 [Epoch 03] Acc.: 90.8300%
11 100%|████████████████████████████████████████| 3750/3750 [00:33<00:00, 112.44it/s]
12 [Epoch 04] Loss: 0.3146
13 [Epoch 04] Acc.: 91.8800%
14 100%|████████████████████████████████████████| 3750/3750 [00:32<00:00, 113.81it/s]
15 [Epoch 05] Loss: 0.3092
16 [Epoch 05] Acc.: 91.2600%

```

**Question:** Next, use an MLP with one hidden layer of dimension 32 followed by ReLU and then the final linear prediction layer. What is the new testing accuracy? **93.6400%**

```

1 self.layers = nn.Sequential(
2     nn.Linear(784, 32),
3     nn.ReLU(),
4     nn.Linear(32, 10)
5 )

```

```

1 >>> python train.py
2 100%|████████████████████████████████████████| 3750/3750 [00:32<00:00, 114.94it/s]
3 [Epoch 01] Loss: 0.4166
4 [Epoch 01] Acc.: 91.4300%
5 100%|████████████████████████████████████████| 3750/3750 [00:32<00:00, 117.09it/s]
6 [Epoch 02] Loss: 0.2859
7 [Epoch 02] Acc.: 92.6200%
8 100%|████████████████████████████████████████| 3750/3750 [00:33<00:00, 113.34it/s]
9 [Epoch 03] Loss: 0.2429
10 [Epoch 03] Acc.: 93.5000%
11 100%|████████████████████████████████████████| 3750/3750 [00:31<00:00, 117.46it/s]
12 [Epoch 04] Loss: 0.2165
13 [Epoch 04] Acc.: 94.3700%
14 100%|████████████████████████████████████████| 3750/3750 [00:32<00:00, 115.52it/s]
15 [Epoch 05] Loss: 0.1986
16 [Epoch 05] Acc.: 93.6400%

```

## 3.4 Convolutional network

**Question:** What testing accuracy do you obtain with this architecture? **98.4500%**

```

1 100%|████████████████████████████████████████| 3750/3750 [00:42<00:00, 88.20it/s]
2 [Epoch 01] Loss: 0.2677
3 [Epoch 01] Acc.: 97.3400%
4 100%|████████████████████████████████████████| 3750/3750 [00:41<00:00, 89.81it/s]
5 [Epoch 02] Loss: 0.0810
6 [Epoch 02] Acc.: 97.7400%

```

```
7 | 100%|██████████| 3750/3750 [00:49<00:00, 76.11it/s]
8 | [Epoch 03] Loss: 0.0614
9 | [Epoch 03] Acc.: 98.4200%
10| 100%|██████████| 3750/3750 [00:41<00:00, 89.31it/s]
11| [Epoch 04] Loss: 0.0495
12| [Epoch 04] Acc.: 98.3700%
13| 100%|██████████| 3750/3750 [00:41<00:00, 89.70it/s]
14| [Epoch 05] Loss: 0.0440
15| [Epoch 05] Acc.: 98.4500%
```

### 3.5 Comparison of number of parameters

**Question:** Compute the number of parameters of the MLP with one hidden layer. Compute the number of parameters of the convolutional network. **You should count both the weights and biases.** Provide detailed explanations and computations.

- MLP: For each layer, number of weight is the number of input dimension times the number of output dimension, number of bias is the output dimension, therefore:

$$Parameter\_Num = \sum_i^{layers} (in\_channel_i + 1)(out\_channel_i) \quad (2)$$

- Convolution: For each layer, number of weight equals to the number of kernel times the number of weight in each kernel. Number of bias is the output channel number:

$$Parameter\_Num = \sum_i^{layers} (in\_channel_i \times width_i \times height_i + 1)(out\_channel_i) \quad (3)$$

- The result is shown as below:

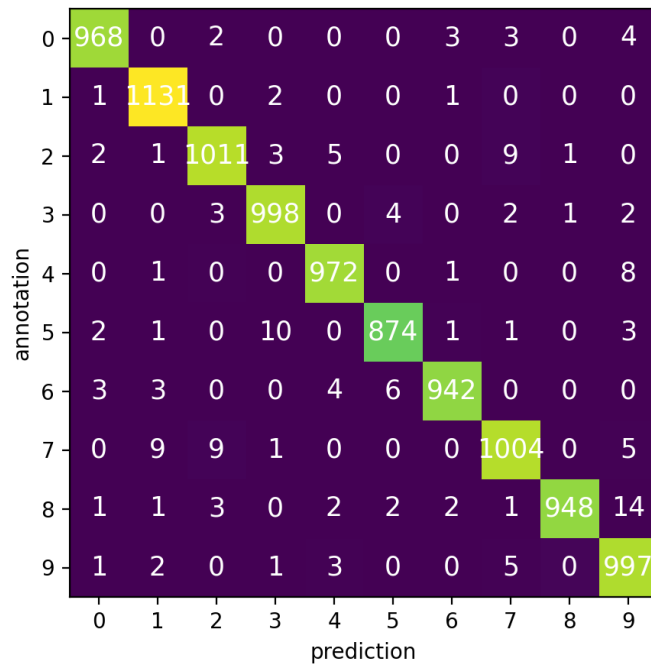
$$\begin{aligned} Parameter\_Num_{MLP} &= (784 + 1) \times 32 + (32 + 1) \times 10 = 25450 \\ Parameter\_Num_{convolution} &= 8 \times (1 + 3 \times 3 \times 1) + 16 \times (1 + 3 \times 3 \times 8) \\ &\quad + 32 \times (1 + 3 \times 3 \times 16) + (32 + 1) \times 10 = 6218 \end{aligned} \quad (4)$$

### 3.6 Confusion matrix

**Question:** The confusion matrix  $M_{i,j}$  is a useful tool for understanding the performance of a classifier. It is defined as follows:  $M_{i,j}$  is the number of test samples for which the network predicted  $i$ , but the ground-truth label was  $j$ . Ideally, in the case of 100% accuracy, this will be a diagonal matrix. Based on the run validation epoch function from `train.py`, update `plot_confusion_matrix.py` to compute the confusion matrix. Provide the plot in the report and comment the results.

- **Confusion matrix for CNN (left):** The summation of the matrix equals to the size of the dataset. The performance of the model is good because the matrix is very closed to a diagonal matrix.





- Confusion matrix for MLP (right):** The summation of the matrix equals to the size of the dataset. But the values are more dense than the CNN result. Therefore the performance is not as good as CNN model.

