利用 MLP 及 CNN 实现图像分类

程远 2234412848

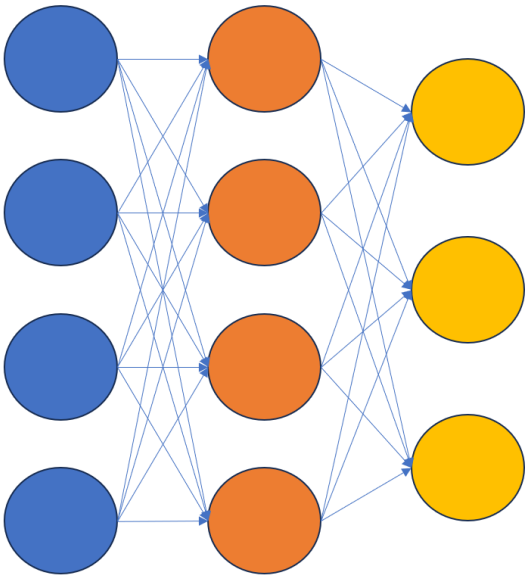
目录

1 引入

计算机视觉（Computer Vesion，下面简称为 CV）是一个重要的计算机科学研究领域，也是我十分感兴趣的领域。CV 在如今的自动驾驶，流水线分类等方面发挥着关键性作用。CV 最基本的任务便是图像分类，该任务要求运用合适的算法，对输入图像进行分类。例如输入手写数字图像，将其分类为 0-9 的阿拉伯数字；又或者输入不同的动物图片，通过计算机来分辨动物种类。最初的 CV 往往采用逻辑学的符号主义手段，即通过特定的识别算法，依靠各个像素点之间的差异来识别不同的对象。然而世间万物的特征无穷无尽，识别算法却十分有限，因此 CV 研究陷入了停滞。然而，随着芯片算力大幅提高，神经科学中的连接主义方法的可行性逐渐提高。程序员只需设定简单的学习模式，即可让机器从数据中自主学习出人工神经网络，从而完成图像处理任务。不难看出，神经网络在 CV 中起到了革命性的影响，因此自己尝试利用神经网络完成简单的 CV 任务能够帮助我们对 CV 研究与应用的过程产生更深刻的了解，并为之后的科研工作打下基础。

2 MLP 简介

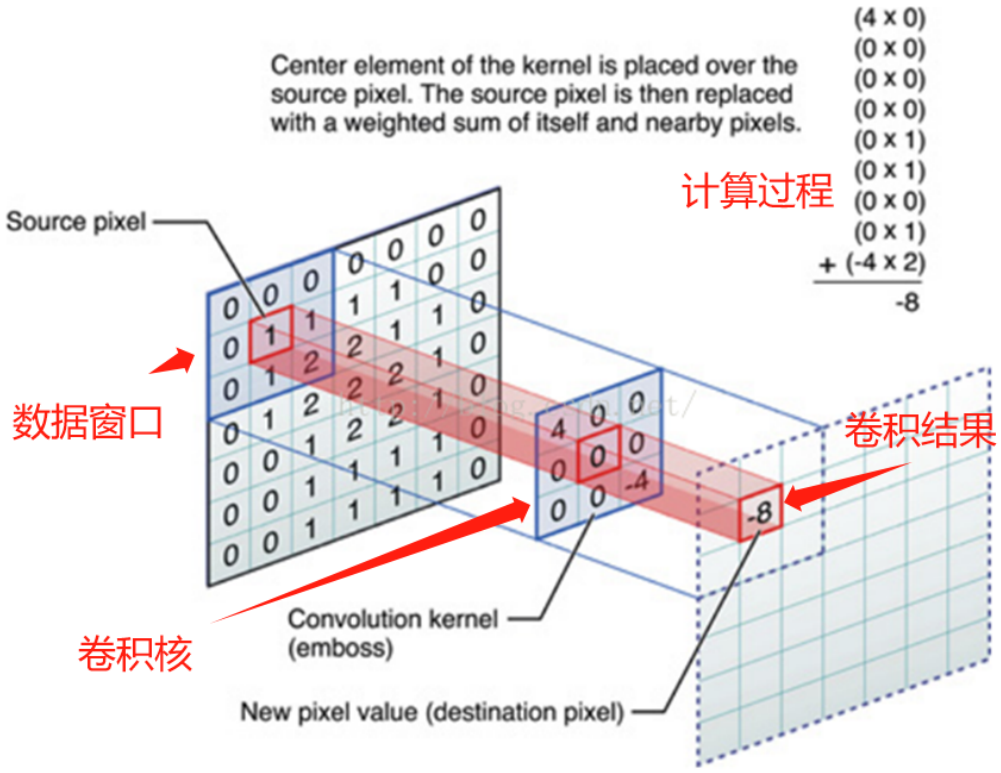
MLP（Multilayer Perceptron，中文名多层感知机）是一种神经网络模型，本质上可以理解为一个从输入层映射到输出层的函数。具体来说，它由一个输入层、一个或多个隐藏层和一个输出层组成。每一层中的每个神经元与下一层中的每个神经元完全连接，这种连接被称为全连接层。除了层，激活函数也是 MLP 的一个重要组件，用于在模型中引入非线性。由于层与层之间都是通过线性的矩阵运算连接，因此如果不人为破坏线性，最终的输出层一定是输入层的线性组合，这对拟合五花八门的特征显然是毁灭性的打击。所以我们们通过激活函数破坏线性，常见的激活函数有 Sigmoid 函数，ReLU 函数等，此处不展开。



MLP 结构图

3 CNN 简介

CNN（Convolutional Neural Network，中文名 CNN）也是一种神经网络模型，与 MLP 不同的是，CNN 特别擅长处理具有网格状拓扑结构的数据，如图像和视频。CNN 通过卷积层、池化层和全连接层来提取和处理数据中的特征。它在 CV 任务（如图像分类、目标检测和图像分割）中表现出色。CNN 核心组件是卷积核，卷积核是一个矩阵，每一层上滑动，对应位置的数字相乘得到输出。每个卷积核产生一个特征图，多个卷积核可以提取不同的特征。为了减少计算量和参数数量，可以通过池化层来粗略提取信息。在神经网络的最后几层，通常会添加若干层全连接层，即把 CNN 与 MLP 结合起来以提高训练效果。



CNN 计算过程演示

4 MLP 与 CNN 模型搭建

4.1 环境简介

Windows 11 WSL2 Ubuntu 20.04 下的 Pytorch，锐龙 R7-5800H，RTX3050laptop。如果老师也想尝试训练模型，可能需要先行配置好 pytorch 环境，详见 https://blog.csdn.net/iwanvan/article/details/122119595

4.2 主函数文件编写

主函数文件为 main.py，该文件下包含了训练器 Trainer 类的详细定义及实现。Trainer 类中依次包含如下功能：构造函数、模型选择、设置检查点、在验证集上验证模型性能、在测试集上测试模型

性能、训练函数、日志记录、绘制训练效果图。main.py 的核心函数是 Trainer 类中的 train 方法，下面对其进行详细解释。首先进入外层循环，即训练 epoch 轮。在每一个 epoch 中，通过 torch 自带的 dataloader 方法加载训练集数据并遍历数据。每一遍遍历中，先将数据加载到计算设备上（我选择的是 CPU），通过模型计算出输出，再计算损失函数，并通过反向转播更新模型参数。同时每 200 个 step 打印一次训练信息。最后将损失转换为标量保存，并进行最终的验证与测试，写入训练日志。

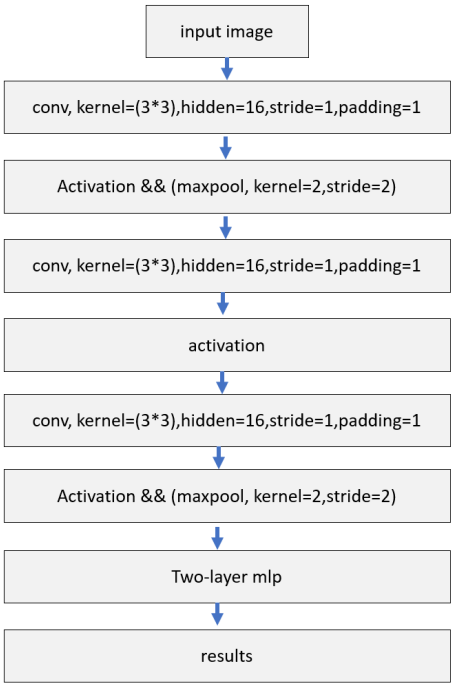
4.3 模型文件

MLP 与 CNN 模型位于 model.py 中，下面依次介绍 model.py 中的三个模型。

第一个模型为 MLP，它是一个简单的感知机，具有两个全连接层，fc1 与 fc2。再初始化连接层后，需要对连接层参数进行初始化。我选择用均值为 0 的高斯分布初始化参数。初始化完成后，需要定义前向传播函数。MLP 的前向传播十分简单，依次进行以下操作即可：将 x 展平为一维张量，通过第一个全连接层的映射，应用 relu 激活函数，通过第二个全连接层，返回 x。

第二个模型是 MLP_D，它与 MLP 最大的不同是添加了 dropout。dropout 是一种正则化技术，可以减少过拟合。dropout 会在训练过程中随机丢弃神经网络中的一部分神经元，包括其输入和输出连接。因此在每次训练迭代中，每个神经元都有一定的概率被临时排除在外，但是在下一次迭代中，这些神经元又有可能被包括进来。这样一来，神经网络被迫在不同的神经元子集上学习，从而有效地防止神经网络在训练集上过拟合，并且通常可以显著提高在测试集上的表现。得益于 pytorch 的完整框架支持，只需要额外添加一行语句 “x = self.dropout(x)” 就可以实现 dropout 的应用。

第三个模型是 CNN，它是一个卷积神经网络，由若干层组成。CNN 的初始化和前面的模型类似，不多赘述。下面用图展示 CNN 的各层神经元。除了 main.py 和 model.py，还有两个必要 python 文件：util.py 与 dataset.py，分别用于设置随机数与数据处理，较为简单，不展开。



CNN 各层结构

5 MLP 与 CNN 模型测试

5.1 MLP 模型测试

在 Ubuntu 中输入命令 python main.py，再输入 MLP 即可运行。前两轮运行结果如下：

```
Epoch [0/50], Step [0/625], Train Loss: 2.302915096282959
Epoch [0/50], Step [200/625], Train Loss: 1.9990676641464233
Epoch [0/50], Step [400/625], Train Loss: 2.1364216804504395
Epoch [0/50], Step [600/625], Train Loss: 1.819313406944275
Validation Loss: 1.8432, Accuracy: 0.3490, F1 Score: 0.3362
Validatoin Confusion Matrix:
[[348 157  64  12   7  23  20  32 304  47]
 [ 26 623  15  18   8  34  38  23 147  82]
 [ 81 137 277  17  48 128 115  44  86  19]
 [ 23 236  82 124  26 279  94  36  88  28]
 [ 57 143 175  26 150 126 159  80  61  20]
 [ 16 179 120  92  18 413  76  49  49  13]
 [ 10 206  95  48  44 160 360  15  25  17]
 [ 33 151  93  48  66  85  61 283  70  87]
 [ 72 200   8  14   2  37   3   8 611  48]
 [ 39 383  11  19   3  28  25  24 189 301]]

Epoch [1/50], Step [0/625], Train Loss: 1.9400862455368042
Epoch [1/50], Step [200/625], Train Loss: 1.6846601963043213
Epoch [1/50], Step [400/625], Train Loss: 1.7122489213943481
Epoch [1/50], Step [600/625], Train Loss: 1.7564774751663208
Validation Loss: 1.7371, Accuracy: 0.3872, F1 Score: 0.3805
Validatoin Confusion Matrix:
[[521  70  73  15  17  24  13  39 174  68]
 [ 78 490  29  23  20  39  32  26 103 174]
 [110  50 398  30  45 113  78  58  44  26]
 [ 69  76 131 192  36 279  71  54  50  58]
 [103  44 308  32 188  88  88  86  18  42]
 [ 54  63 179 129  25 399  60  66  28  22]
 [ 26  47 193 105  52 142 314  37  13  51]
 [ 68  61 121  43  82  96  39 364  24  79]
 [168 104  18  13   6  44   4  10 538  98]
 [106 183  26  27   3  34  24  28 123 468]]
```

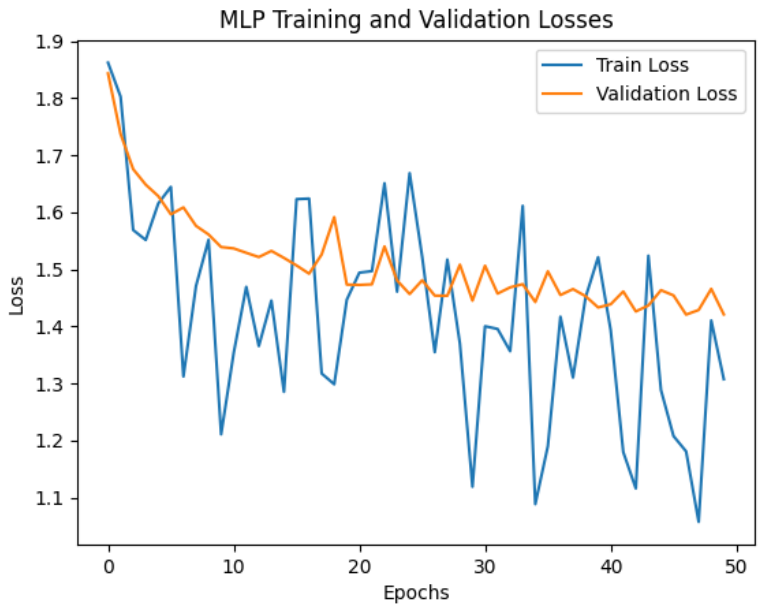
MLP 前两轮运行结果

训练结束后，查看日志文件，得到 MLP 模型最终性能如下：

```
Test Loss: 1.4058, Accuracy: 0.5074, F1 Score: 0.5028
Test Confusion Matrix:
[[571  21  51  31  26  13  32  38 128  89]
 [ 55 507  14  22  10  14  23  27  69 259]
 [ 83  16 313 102 145  66 124  95  18  38]
 [ 29  13  55 369  53 154 152  80  25  70]
 [ 50  11 107  65 405  45 155 115  24  23]
 [ 24   6  71 219  63 364  84 102  28  39]
 [  8  12  45 101  91  32 634  32  16  29]
 [ 29  14  31  77  78  67  30 605  12  57]
 [120  51  14  29  16  19   6  22 632  91]
 [ 42 107  11  32  11  18  19  40  46 674]]
```

MLP 训练结果

画出损失函数图像如下：



MLP 损失函数图像

5.2 MLP_D 模型测试

在 Ubuntu 中输入命令 python main.py，再输入 MLP_D 即可运行。前两轮运行结果如下：

```
Epoch [0/50], Step [0/625], Train Loss: 2.339503049850464
Epoch [0/50], Step [200/625], Train Loss: 2.1220743656158447
Epoch [0/50], Step [400/625], Train Loss: 2.1482632160186768
Epoch [0/50], Step [600/625], Train Loss: 1.9407024383544922
Validation Loss: 1.8859, Accuracy: 0.3274, F1 Score: 0.3008
Validatoin Confusion Matrix:
[[483  81  15   2  35  39  28  40 139 152]
 [ 61 368  14   4  27  56  61  14  92 317]
 [162  76  64   1 223 153 122  62  36  53]
 [ 76 109  35  25  80 347 127  60  39 118]
 [113  59  46   4 312 144 173  52  19  75]
 [ 74  82  27  16  96 460 106  77  24  63]
 [ 26  75  21   7 133 178 386  46  15  93]
 [ 83  75  33   7 161 129  84 227  20 158]
 [212 116   3   4   4  64   5   6 368 221]
 [ 73 160   3   2   9  37  40  25  92 581]]

Epoch [1/50], Step [0/625], Train Loss: 1.9209638833999634
Epoch [1/50], Step [200/625], Train Loss: 1.7790684700012207
Epoch [1/50], Step [400/625], Train Loss: 1.9119181632995605
Epoch [1/50], Step [600/625], Train Loss: 1.938104510307312
Validation Loss: 1.8133, Accuracy: 0.3637, F1 Score: 0.3506
Validatoin Confusion Matrix:
[[438 103  44   7  40  32  26  50 170 104]
 [ 51 480  18  15  40  42  50  20  92 206]
 [121  66 151  16 212 131 135  57  32  31]
 [ 49 103  55  95  80 328 139  49  52  66]
 [ 86  59  84  13 350 122 171  56  18  38]
 [ 33  72  71  63  88 442 128  69  31  28]
 [ 17  67  53  54 126 135 439  33  15  41]
 [ 49  75  49  30 181 117  84 282  19  91]
 [174 132   7   5   9  50   5  12 468 141]
 [ 58 213  12  17   8  35  43  34 110 492]]
```

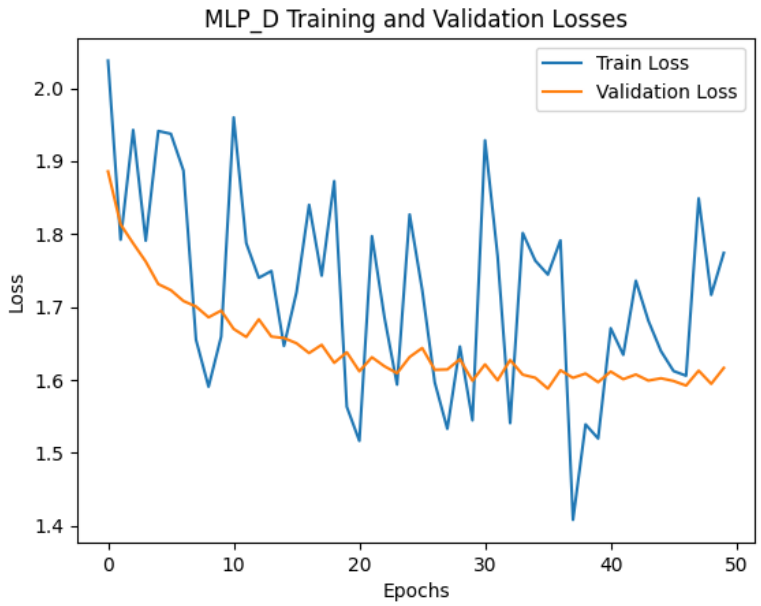
MLP_D 前两轮运行结果

训练结束后，查看日志文件，得到 MLP_D 模型最终性能如下：

```
Test Loss: 1.5955, Accuracy: 0.4382, F1 Score: 0.4273
Test Confusion Matrix:
[[423  41  40  42  11   9  30  16 329  59]
 [ 37 559  16  30  10  15  21  12 128 172]
 [112  33 200 125  89  52 271  29  64  25]
 [ 28  31  60 359  26 144 203  22  55  72]
 [ 52  12  89  79 258  34 351  39  52  34]
 [ 20  19  91 246  24 309 155  33  67  36]
 [  9  27  39 123  37  35 672   7  22  29]
 [ 53  24  64 101  85  77  84 348  55 109]
 [ 62  52   4  33  12  20   9   3 726  79]
 [ 33 170   7  43   7  14  31  26 141 528]]
```

MLP_D 训练结果

画出损失函数图像如下：



MLP_D 损失函数图像

5.3 CNN 模型测试

在 Ubuntu 中输入命令 python main.py，再输入 CNN 即可运行。前两轮运行结果如下：

```
Epoch [0/50], Step [0/625], Train Loss: 2.298011064529419
Epoch [0/50], Step [200/625], Train Loss: 2.097132444381714
Epoch [0/50], Step [400/625], Train Loss: 1.9563944339752197
Epoch [0/50], Step [600/625], Train Loss: 1.6506541967391968
Validation Loss: 1.8629, Accuracy: 0.3153, F1 Score: 0.2962
Validatoin Confusion Matrix:
[[329  80  71  18   5  36  27  26 256 166]
 [ 24 350  38  37   6  69  66  39 168 217]
 [ 60  47 162  97  17 164 253  44  50  58]
 [ 29  40  90 182  16 272 206  48  51  82]
 [ 32  25  71 114  34 121 420  57  65  58]
 [ 17  50  95 181  18 339 173  44  42  66]
 [  3  22  79 127  14 107 538  45   8  37]
 [  9  87  82 112  16 107 142 192  33 197]
 [130  81  33  12   1  48  11  17 499 171]
 [ 17  94  24  52   8  38  46  54 161 528]]

Epoch [1/50], Step [0/625], Train Loss: 1.8591079711914062
Epoch [1/50], Step [200/625], Train Loss: 1.7484936714172363
Epoch [1/50], Step [400/625], Train Loss: 1.8170567750930786
Epoch [1/50], Step [600/625], Train Loss: 1.5760688781738281
Validation Loss: 1.6804, Accuracy: 0.4084, F1 Score: 0.3999
Validatoin Confusion Matrix:
[[441  59  73  21  46  25  34  89 155  71]
 [ 51 452  25  23  32  30  51  58  96 196]
 [ 54  58 167  60 193 114 170  74  35  27]
 [ 14  35  65 150 125 273 209  61  37  47]
 [ 39  13  84  31 388  86 199 104  33  20]
 [ 13  28  83  89  91 406 170  87  24  34]
 [  3  19  53  37 135  86 577  39   8  23]
 [ 24  46  51  38 151  89  74 437  15  52]
 [101  78  18  23  34  32  18  26 567 106]
 [ 53 136  25  26  14  26  65  82  96 499]]
```

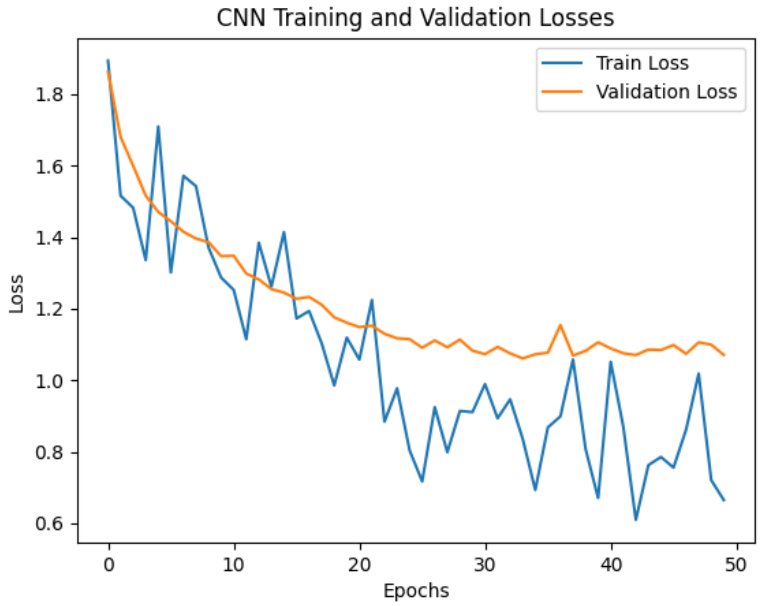
CNN 前两轮运行结果

训练结束后，查看日志文件，得到 CNN 模型最终性能如下：

```
Test Loss: 1.0747, Accuracy: 0.6401, F1 Score: 0.6379
Test Confusion Matrix:
[[769  21  34  17  13  11  17  15  69  34]
 [ 49 718  12   9   3   6  16   8  59 120]
 [ 96   8 498  80 113  58  72  40  22  13]
 [ 43  17  79 478  83 117  90  45  22  26]
 [ 38   5 111  59 576  25  79  78  20   9]
 [ 17   6  83 208  65 475  47  69  17  13]
 [  9   7  51  70  60   9 758  10  13  13]
 [ 36   5  39  49  78  63   8 695   3  24]
 [116  46  20  20  10   5  12   5 740  26]
 [ 65 123   8  18   6  13  15  23  35 694]]
```

CNN 训练结果

画出损失函数图像如下：



CNN 损失函数图像

5.4 结果分析

MLP 是三个模型中最简单的模型，准确度最终收敛至 0.5 左右。然而观察其损失函数图像，发现该模型存在一定的过拟合现象。从 MLP 的损失函数图像可以看出，测试集与验证集的损失数值抖动剧烈，这可能是因为模型已经毕竟拟合极限，在最优拟合左右反复导致的。再添加 dropout 后，出现了两个变化。第一是测试集损失显著提高，这个容易理解。因为 dropout 忽略了部分神经元（我设置了忽略一半神经元），所以模型性能有所下降，最终收敛的准确率较低。第二是验证集抖动情况得到改善，这说明模型的过拟合现象得到抑制。这两个变化基本符合理论上的 dropout 作用。而 CNN 模型显然显示出了更强大的拟合能力，这一方面归功于卷积核能够更好地提取图像中临近像素的特征关系，另一方面也因为我的 CNN 模型比 MLP 复杂不少，甚至 CNN 模型最后嵌入了两个全连接层，相当于加入了一个 MLP。除了损失更低，准确率更高，CNN 的过拟合现象也更为不明显，无论是测试集损失还是验证机损失，相较 MLP 与 MLP_D 都显得平滑不少。

6 总结

通过完成这次大作业，我将我对 CV 的兴趣转化为了现实的模型，我更好地理解了 MLP 和 CNN 的基本原理、模型结构及其在图像分类任务中的表现。我在构建和测试 MLP 和 CNN 模型的过程中，学会了如何配置训练环境、编写训练和测试代码、以及如何分析模型的性能。尤其是，通过对比 MLP、MLP_D（包含 Dropout 的 MLP）和 CNN 模型的测试结果，我发现了不同模型在处理图像分类任务时的优缺点，以及如何通过正则化技术（如 Dropout）来减轻过拟合现象。

除了实现基本的 MLP 与 CNN 框架，我还希望未来能在更复杂的计算机视觉任务中进一步应用和优化这些模型，例如图像分割和目标检测。此外，我还计划了解并实践其他先进的神经网络结构和技术，如生成对抗网络和注意力机制。通过不断的学习和实践，我期待能为计算机视觉领域的发展贡献自己的力量。

7 附录：程序使用说明

想要运行程序，主要的难点在于安装 pytorch，详见 https://blog.csdn.net/iwanvan/article/details/122119595。pytorch 安装完毕后，直接将我提供的压缩包解压，运行 main.py 即可。运行 main.py 后，程序会要求用户输入模型名称，用户需要在 MLP、MLP_D、CNN 中三选一，输入后程序自动开始训练模型，训练完毕后自动测试模型并生成日志与损失函数图。

需要注意的是，用户需要基于自己的操作系统修改文件路径格式，默认为 Linux 路径格式。此外，用户需要在 main.py 的第 20、21、182、184 行输入想要的文件名称以及图像标题，以区分不同模型的结果。