前馈神经网络实验报告

- 姓名：陈睿颖
- 学号：2013544
- 专业：计算机科学与技术

1. 实验要求

- 掌握前馈神经网络（FFN）的基本原理
- 学会使用PyTorch搭建简单的FFN实现MNIST数据集分类
- 掌握如何改进网络结构、调试参数以提升网络识别性能

2. 报告内容

- 运行原始版本MLP，查看网络结构、损失和准确度曲线
- 尝试调节MLP的全连接层参数（深度、宽度等）、优化器参数等，以提高准确度
- 分析与总结格式不限
- 挑选MLP-Mixer，ResMLP，Vision Permutator中的一种进行实现（加分项）

3. 实验内容

3.1 原始MLP

网络结构

通过运行原始版本的MLP，可以看到网络结构如下：

```
Net(
  (fc1): Linear(in_features=784, out_features=100, bias=True)
  (fc1_drop): Dropout(p=0.2, inplace=False)
  (fc2): Linear(in_features=100, out_features=80, bias=True)
  (fc2_drop): Dropout(p=0.2, inplace=False)
  (fc3): Linear(in_features=80, out_features=10, bias=True)
)
```

输入层：

- 输入特征的维度为784，这表明模型接受的输入是一个大小为28x28的图像（通常是手写数字图像），将其展平为784维的向量。

隐藏层：

- 第一个隐藏层（fc1）是一个线性层（Linear），它有784个输入特征和100个输出特征。这意味着它将输入特征的维度从784降低到100。
- 在第一个隐藏层之后，应用了一个丢弃层（dropout层），其丢弃概率为0.2。丢弃层的目的是在训练过程中随机丢弃一部分神经元，以减少过拟合的风险。

第二隐藏层：

- 第二隐藏层（fc2）是一个线性层，有100个输入特征和80个输出特征。这进一步减少了特征的维度。

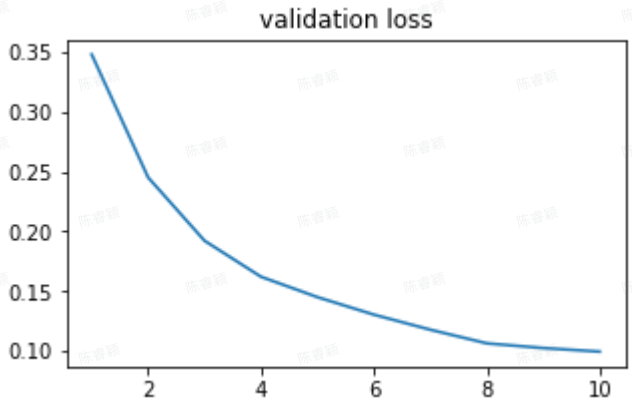
在第二隐藏层之后，再次应用了一个丢弃层（dropout层），丢弃概率为0.2。

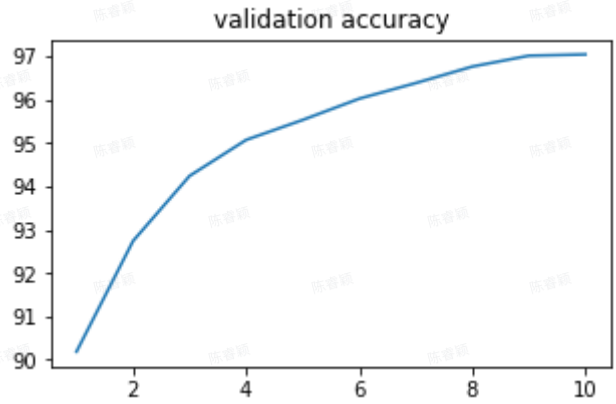
输出层：

- 输出层（fc3）是一个线性层，有80个输入特征和10个输出特征。这表明模型输出一个包含10个元素的向量，对应于10个可能的类别（0到9之间的数字）。

该模型有两个隐藏层，分别具有100个和80个神经元。它还包括丢弃层以减少过拟合的风险。输出层具有10个神经元，用于表示10个可能的类别。该模型在手写数字分类任务中使用，输入图像的大小应为28x28像素。

训练结果





可以看到，在经历十轮迭代训练后，在验证集上的准确率在 97% 左右.

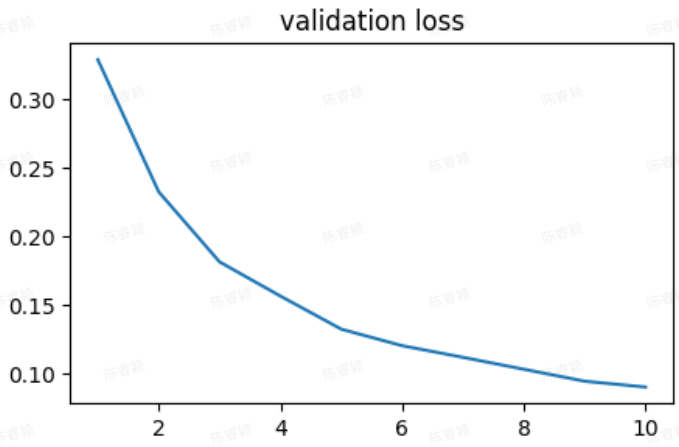
3.2 改进模型

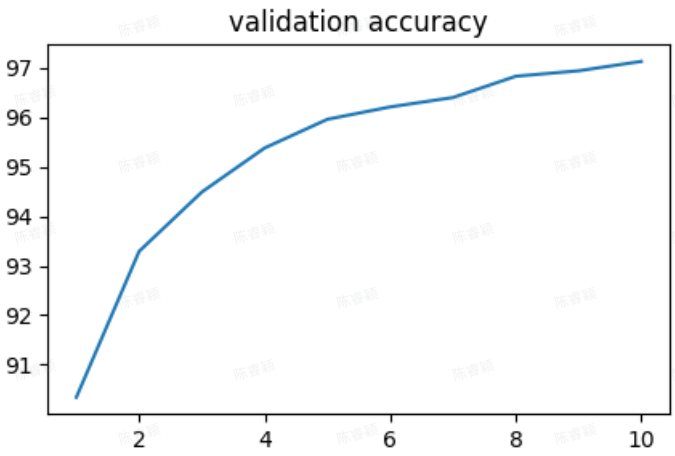
增加隐藏层的宽度

在模型定义的部分，增加了fc1的宽度，更改如下：

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(28*28, 200)  # weight: [28*28, 50]   bias: [50, ]
        self.fc1_drop = nn.Dropout(0.2)
        self.fc2 = nn.Linear(200, 80)
        self.fc2_drop = nn.Dropout(0.2)
        self.fc3 = nn.Linear(80, 10)

#         self.relu1 = nn.ReLU()

    def forward(self, x):
        x = x.view(-1, 28*28)    # [32, 28*28]
        x = F.relu(self.fc1(x))
        x = self.fc1_drop(x)
        x = F.relu(self.fc2(x))
        x = self.fc2_drop(x)    # [32, 10]
        return F.log_softmax(self.fc3(x), dim=1)

model = Net().to(device)
optimizer = torch.optim.SGD(model.parameters(), lr=0.01, momentum=0.5)
criterion = nn.CrossEntropyLoss()

print(model)
```

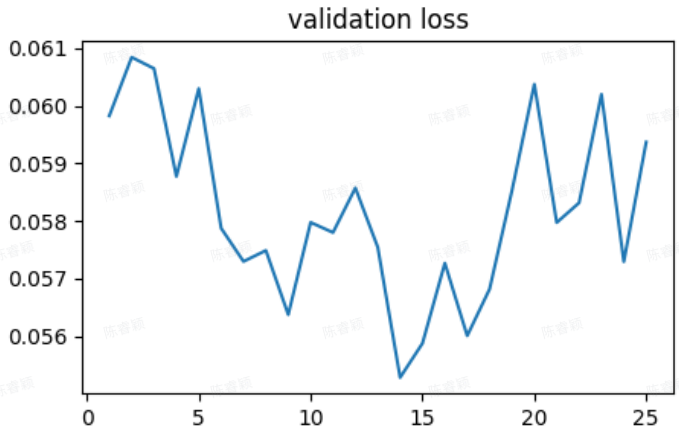
Epoch=10的运行结果如下：

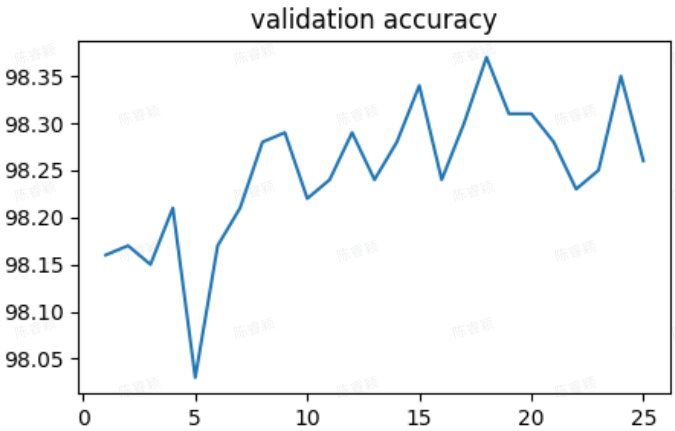




最后的准确率可达到98%

Epoch=25的运行结果如下：



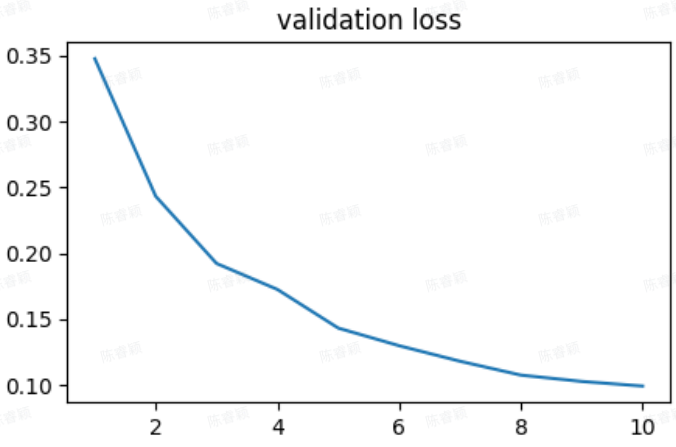


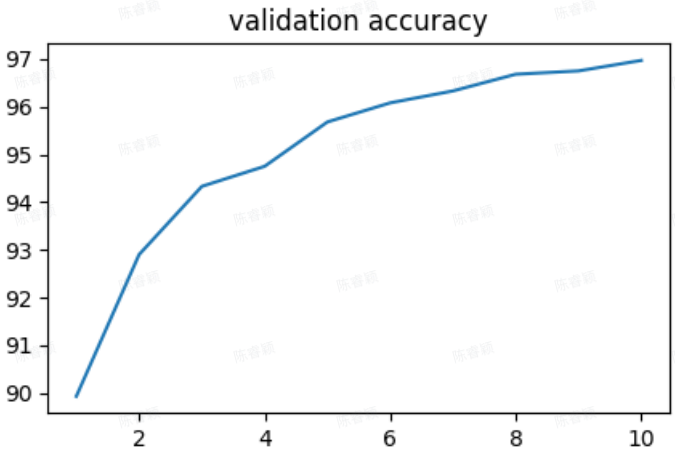
上述可能出现了过拟合现象，所以训练轮次为10轮左右较合适。

增加隐藏层的深度

```
        self.fc1 = nn.Linear(28*28, 100)
        self.fc1_drop = nn.Dropout(0.2)
        self.fc2 = nn.Linear(100, 90)  # Add an additional hidden layer with wid
        self.fc2_drop = nn.Dropout(0.2)
        self.fc3 = nn.Linear(90, 80)
        self.fc3_drop = nn.Dropout(0.2)
        self.fc4 = nn.Linear(80, 10)
```

运行结果如下





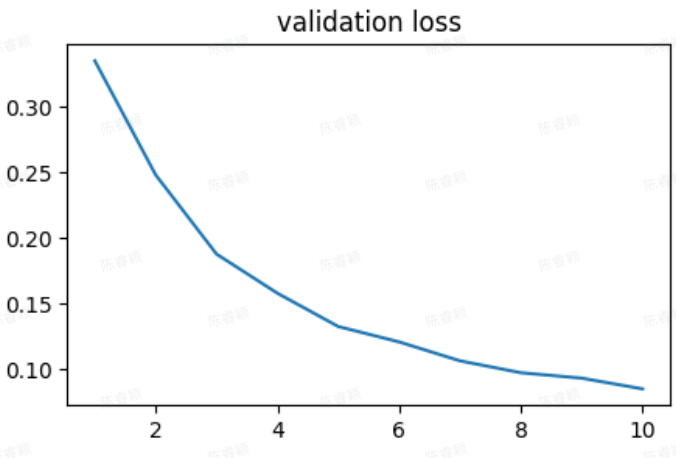
可以看到最后的正确率在97%左右。优化效果并不如增加隐藏层的宽度。

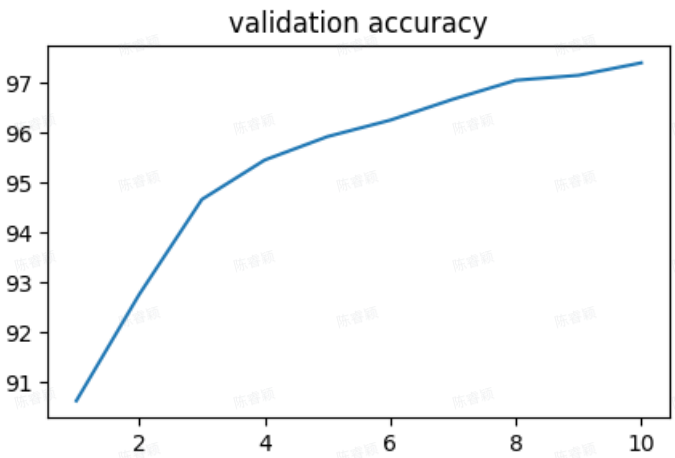
原因可能是MNIST数据集的相对简单性。MNIST数据集包含手写数字图像，每个图像都比较简单，且类别之间的区分度较高。因此，增加隐藏层的深度可能会导致模型过度拟合训练数据，而在验证集或测试集上的性能并不会有显著的提升。

增加隐藏层的深度会增加模型的复杂性，使其具备更多的参数和非线性变换能力。然而，在相对简单的任务上，增加深度可能会引入过多的模型复杂性，导致模型难以泛化到未见过的数据。这可能会导致过拟合问题，使得在验证集上的性能无法达到预期。

相比之下，增加隐藏层的宽度可以提供更多的神经元用于学习特征，使得模型能够更好地捕捉数据的变化和模式。在MNIST数据集这样相对简单的任务上，增加隐藏层的宽度可以为模型提供更多的表示能力，进而提高模型的准确度。

下面同时进行深度和宽度的增加：





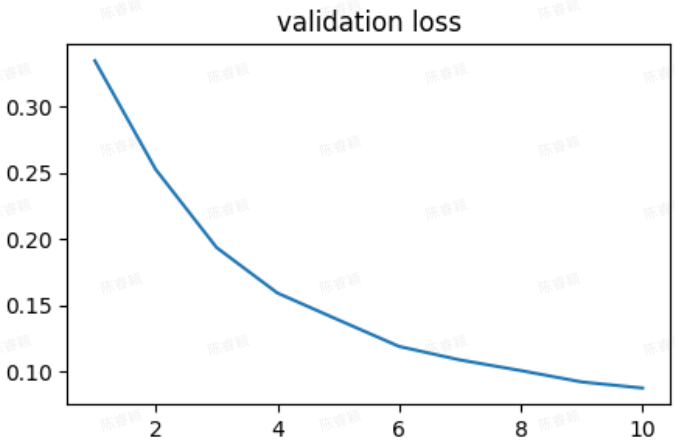
正确率在97%左右。

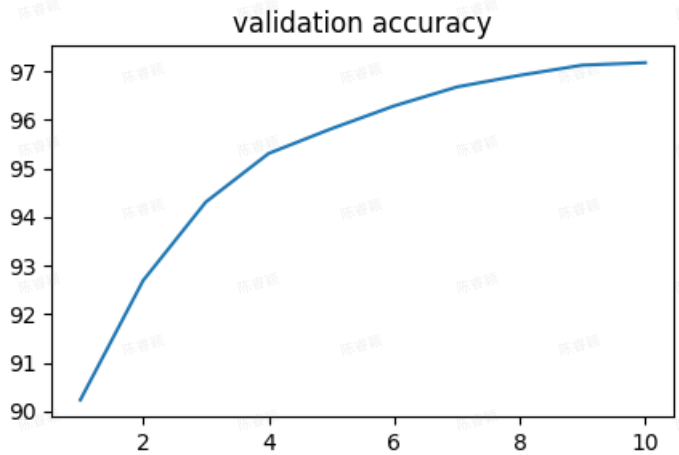
更改dropout值

Dropout是一种正则化技术，用于减少模型的过拟合。它在训练过程中随机地将一部分神经元的输出置为0，以防止它们过度依赖于其他特定神经元，从而增加了模型的鲁棒性。

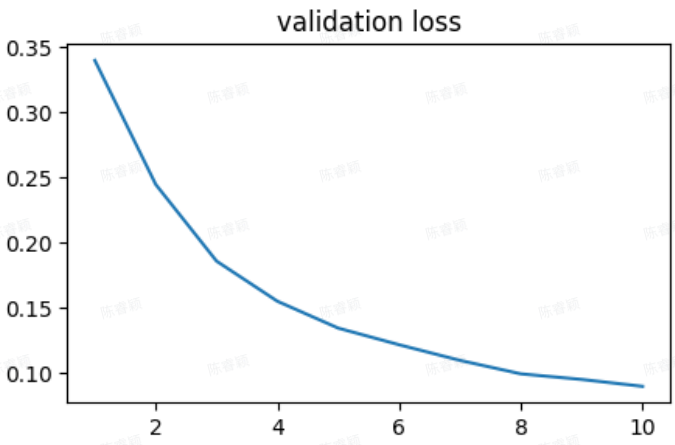
设置dropout的丢弃概率可以控制在训练过程中丢弃神经元的比例。较高的丢弃概率会增加模型的鲁棒性，但也可能导致信息损失。较低的丢弃概率可能会减少模型的鲁棒性，但也有可能提供更多的信息来进行训练。

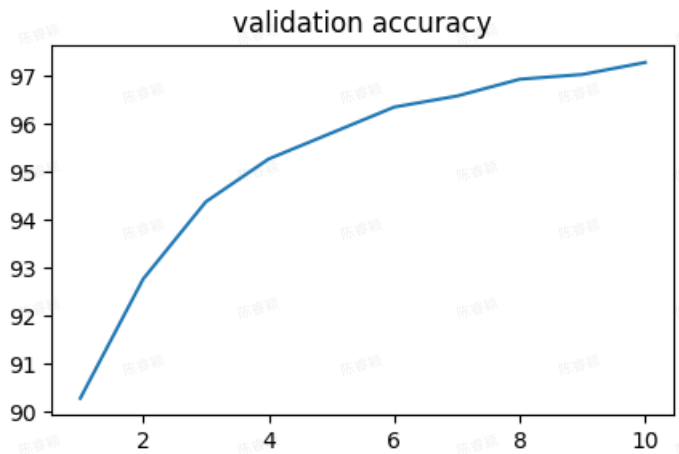
设置dropout=0.1：





设置dropout=0.3：





正确率均在97%左右，优化效果并不是很明显。

3.3 实现ResMLP

模型结构

ResMLP（Residual Multi-Layer Perceptron）是一种基于多层感知机（Multi-Layer Perceptron，MLP）的残差连接架构，用于图像分类和计算机视觉任务。它的设计灵感来自于残差网络（Residual Network）中的残差连接思想。

传统的 MLP 通常由多个全连接层组成，每个层之间使用非线性激活函数进行转换。但在深层的 MLP 中，可能会面临梯度消失和过拟合等问题。为了解决这些问题，ResMLP 引入了残差连接机制。

ResMLP 的核心思想是将残差连接应用于 MLP 的每一层。具体而言，每个 MLP Block（由多个全连接层组成）的输出与其输入进行元素级的相加操作，形成残差连接。这样可以使信息在网络中更好地传递，有效地减轻梯度消失问题，提高模型的训练效果和表达能力。

与传统的 MLP 不同，ResMLP 并不使用卷积层，而是仅由全连接层组成。这使得 ResMLP 在处理图像数据时具有更高的灵活性和可扩展性。

ResMLP 的整体架构通常包括多个 ResMLP Block，每个 Block 内部包含多个全连接层和残差连接。在训练过程中，通过反向传播算法更新网络的参数，以最小化损失函数，从而实现模型的训练和优化。

ResMLP 结合了 MLP 和残差连接的优势，在图像分类和计算机视觉任务中取得了较好的性能表现。它是一种有前景的深度学习模型架构，为处理图像数据提供了一种新的选择。

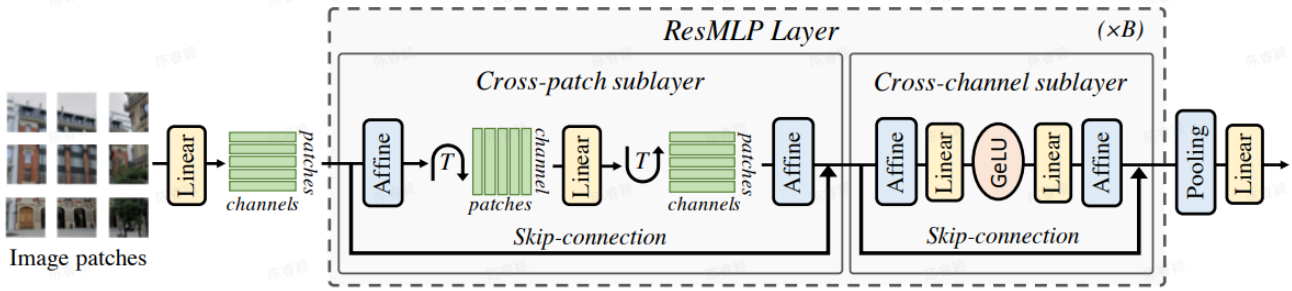


Figure 1: **The ResMLP architecture.** After linearly projecting the image patches into high dimensional embeddings, ResMLP sequentially processes them with (1) a cross-patch linear sublayer; (2) a cross-channel two-layer MLP. The MLP is the same as the FCN sublayer of a Transformer. Each sublayer has a residual connection and two Affine element-wise transformations.

ResMLP结构[1]

代码设计

ResMLP 模型的代码设计思路如下：

1. 数据预处理：模型首先通过卷积层（`to_patch_embedding`）将输入图像转换为一系列的图像 patch。这是为了将图像数据转换为序列数据，以便后续的处理。
2. MLP 模块设计：模型使用了多个 MLP 模块（`MLPblock`）组成的堆叠。每个 MLP 模块由三个关键组件组成：
   - `pre_affine`：一个仿射变换操作，对输入进行缩放和偏移，以引入一定的灵活性。
   - `token_mix`：通过线性层对输入的 patch 序列进行混合，以促进信息交换和整合。

- `ff`：一个前馈神经网络，用于对混合后的序列进行进一步的特征提取和非线性变换。
- `post_affine`：类似于 `pre_affine`，对输出进行缩放和偏移。

3. 特征整合和分类：在多个 MLP 模块的堆叠之后，通过仿射变换 (`affine`) 对最后一个 MLP 模块的输出进行缩放和偏移操作，以进一步调整特征表示。最后，通过线性层 (`mlp_head`) 对调整后的特征进行分类，输出最终的预测结果。

```
from einops.layers.torch import Rearrange

class Aff(nn.Module):
    def __init__(self, dim):
        super().__init__()
        self.alpha = nn.Parameter(torch.ones([1, 1, dim]))
        self.beta = nn.Parameter(torch.zeros([1, 1, dim]))

    def forward(self, x):
        x = x * self.alpha + self.beta
        return x

class FeedForward(nn.Module):
    def __init__(self, dim, hidden_dim, dropout = 0.):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(dim, hidden_dim),
            nn.GELU(),
            nn.Dropout(dropout),
            nn.Linear(hidden_dim, dim),
            nn.Dropout(dropout)
        )
    def forward(self, x):
        return self.net(x)

class MLPblock(nn.Module):

    def __init__(self, dim, num_patch, mlp_dim, dropout = 0., init_values=1e-4):
        super().__init__()
        self.pre_affine = Aff(dim)
        self.token_mix = nn.Sequential(
            Rearrange('b n d -> b d n'),
            nn.Linear(num_patch, num_patch),
            Rearrange('b d n -> b n d'),
        )
        self.ff = nn.Sequential(
            FeedForward(dim, mlp_dim, dropout),
        )
```

```
        self.post_affine = Aff(dim)
        self.gamma_1 = nn.Parameter(init_values * torch.ones((dim)), requires_grad=True)
        self.gamma_2 = nn.Parameter(init_values * torch.ones((dim)), requires_grad=True)

    def forward(self, x):
        x = self.pre_affine(x)
        x = x + self.gamma_1 * self.token_mix(x)
        x = self.post_affine(x)
        x = x + self.gamma_2 * self.ff(x)
        return x

class ResMLP(nn.Module):

    def __init__(self, in_channels, dim, num_classes, patch_size, image_size, depth, mlp_dim):
        super().__init__()
        assert image_size % patch_size == 0, 'Image dimensions must be divisible by the patch size.'
        self.num_patch =  (image_size// patch_size) ** 2
        self.to_patch_embedding = nn.Sequential(
            nn.Conv2d(in_channels, dim, patch_size, patch_size),
            Rearrange('b c h w -> b (h w) c'),
        )
        self.mlp_blocks = nn.ModuleList([])
        for _ in range(depth):
            self.mlp_blocks.append(MLPblock(dim, self.num_patch, mlp_dim))
        self.affine = Aff(dim)
        self.mlp_head = nn.Sequential(
            nn.Linear(dim, num_classes)
        )

    def forward(self, x):
        x = self.to_patch_embedding(x)
        for mlp_block in self.mlp_blocks:
            x = mlp_block(x)
        x = self.affine(x)
        x = x.mean(dim=1)
        return self.mlp_head(x)
model = ResMLP(in_channels=1, image_size=28, patch_size=7, num_classes=10, dim=384, depth=2, mlp_dim=384*4).to(device)
```

通过将图像数据转换为序列数据，并利用多个 MLP 模块对序列进行处理，实现图像分类任务。MLP 模块中的仿射变换、线性层和非线性激活函数等操作能够引入非线性和灵活性，从而提取和整合图像的

特征。通过堆叠多个 MLP 模块，可以逐步深化特征表示。最后，通过线性层进行分类，输出预测结果。

模型结构输出如下：

```
ResMLP(
  (to_patch_embedding): Sequential(
    (0): Conv2d(1, 384, kernel_size=(7, 7), stride=(7, 7))
    (1): Rearrange('b c h w -> b (h w) c')
  )
  (mlp_blocks): ModuleList(
    (0-1): 2 x MLPblock(
      (pre_affine): Aff()
      (token_mix): Sequential(
        (0): Rearrange('b n d -> b d n')
        (1): Linear(in_features=16, out_features=16, bias=True)
        (2): Rearrange('b d n -> b n d')
      )
      (ff): Sequential(
        (0): FeedForward(
          (net): Sequential(
            (0): Linear(in_features=384, out_features=1536, bias=True)
            (1): GELU(approximate='none')
            (2): Dropout(p=0.0, inplace=False)
            (3): Linear(in_features=1536, out_features=384, bias=True)
            (4): Dropout(p=0.0, inplace=False)
          )
        )
      )
      (post_affine): Aff()
    )
  )
  (affine): Aff()
  (mlp_head): Sequential(
    (0): Linear(in_features=384, out_features=10, bias=True)
  )
)
```

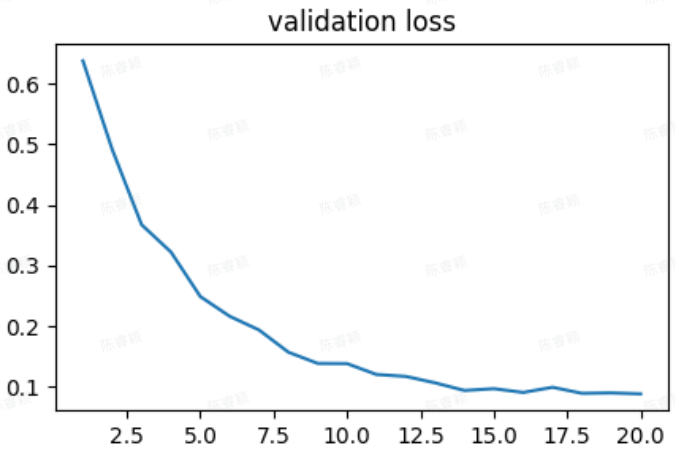
其中：

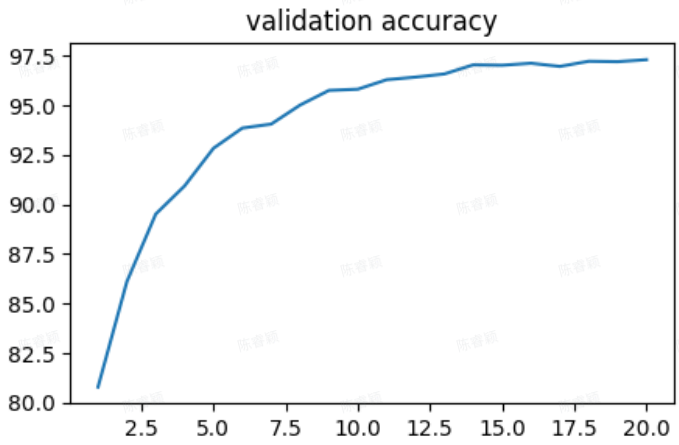
- `to_patch_embedding` 是一个序列化操作，它将输入图像的通道数从 1 扩展为 384，并通过 7x7 的卷积核对图像进行卷积，步长为 7，以生成图像 patch 的序列。然后，`Rearrange` 操作将生成的序列进行重排，使其符合 MLP 模块的输入要求。

- `mlp_blocks` 是一个 `ModuleList`，其中包含两个 `MLPblock` 模块的实例。每个 `MLPblock` 模块包括三个主要组件：`pre_affine`、`token_mix` 和 `ff`。这些组件通过一定的变换和操作对输入进行处理，以提取和整合特征。
- `affine` 是一个仿射变换操作，对最后一个 MLP 模块的输出进行缩放和偏移。
- `mlp_head` 是一个线性层，将 MLP 模块的输出特征映射到类别标签的预测结果。

运行结果

下面为Epoch=20，learning rate=0.03的训练结果





最后准确率收敛在97.5%左右，可以看出 ResMLP 模型对于 MNIST 数据集的特征提取和分类具有较好的能力。

4. 参考文献

[1] Touvron, H., Bojanowski, P., Caron, M., Cord, M., El-Nouby, A., Grave, E., Izacard, G., Joulin, A., Synnaeve, G., Verbeek, J., & Jégou, H. (2021). ResMLP: Feedforward networks for image classification with data-efficient training. Retrieved from arXiv:2105.03404. (Primary Class: cs.CV)