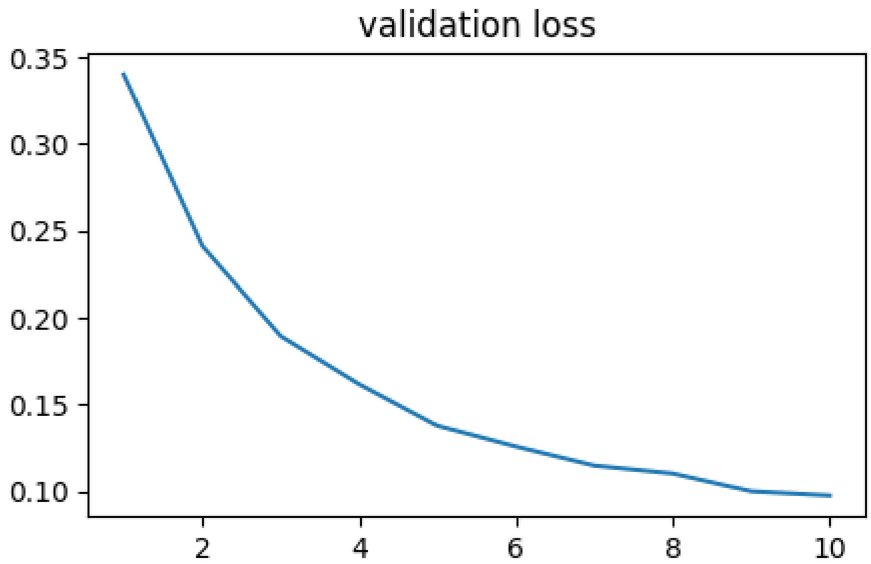
深度学习报告w2：前馈神经网络

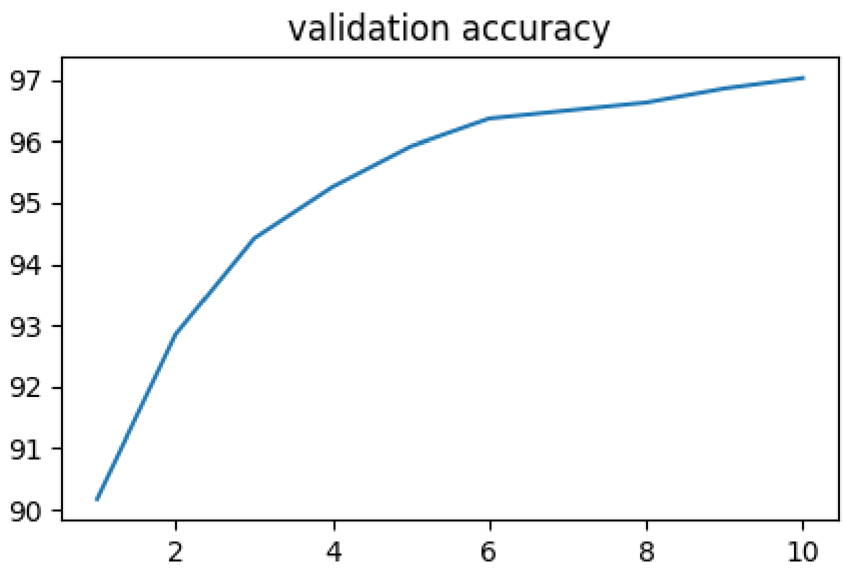
1 原模型复现

原模型为一个3层全连接层的MLP，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)
)
```

复现结果如下：





2 模型参数修改

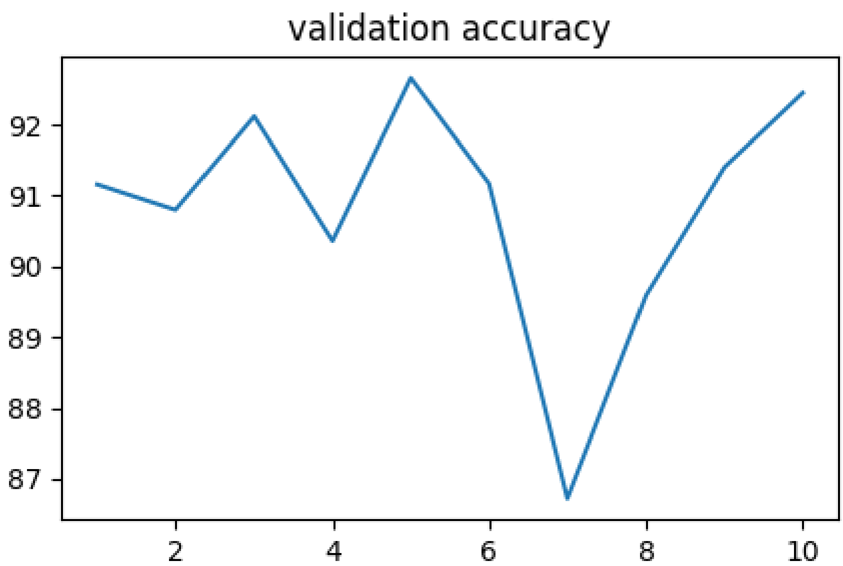
2.1 换用更好的优化器（Adam优化器）优化梯度下降

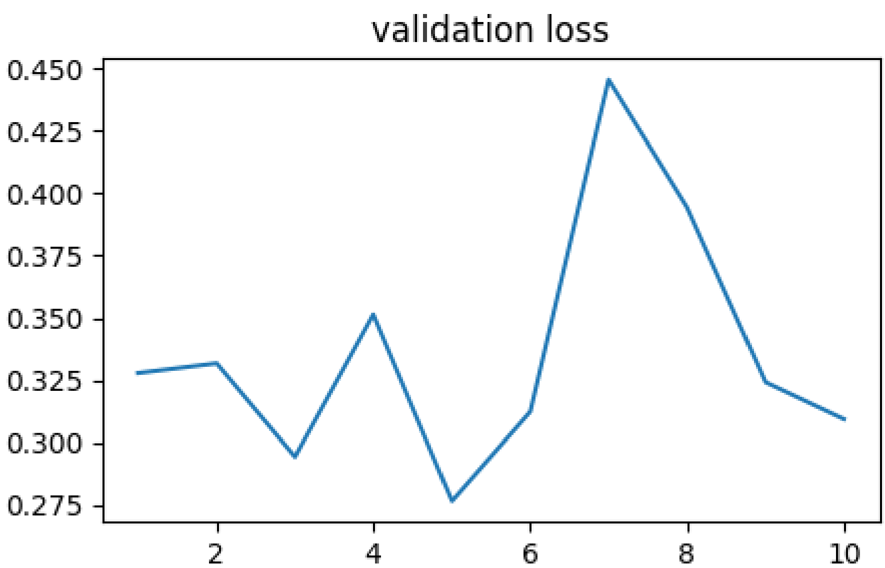
发现模型验证曲线极差（如下），阅读部分文献之后得出如下的结论：

Adam优化器在训练中能够快速下降，但有时并非找到的是局部最优解，而是一个poor solution，我的分析是在minst这种小数据集中过快的收敛很可能会导致欠拟合。

https://ar5iv.labs.arxiv.org/html/1711.05101

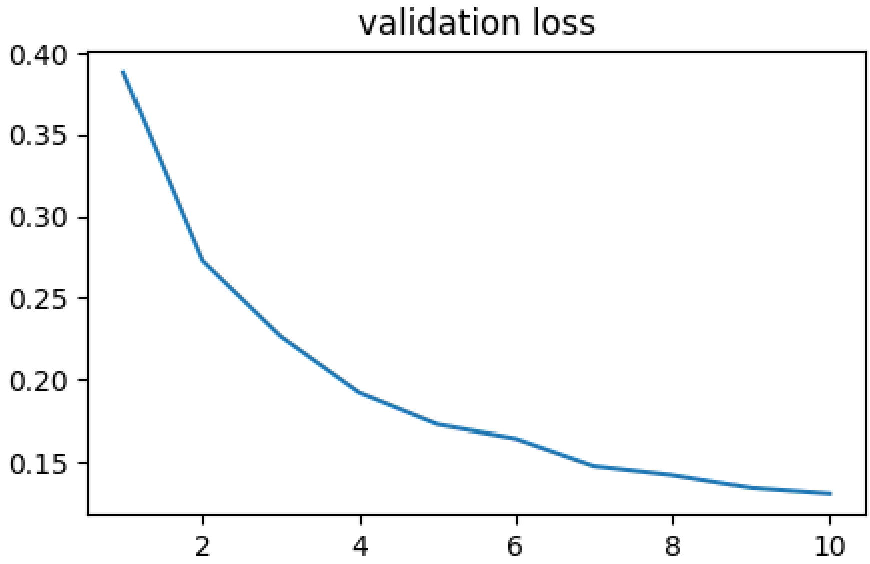
https://arxiv.org/abs/1705.08292

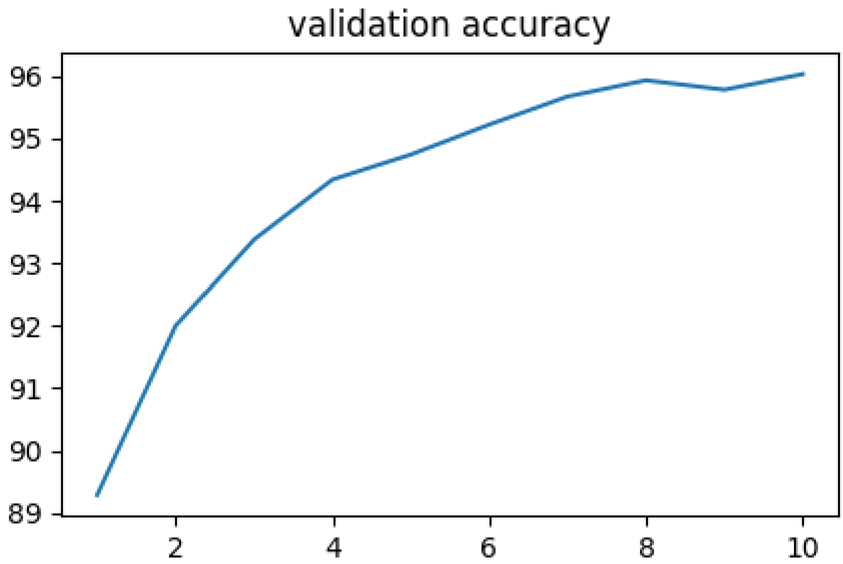




2.2 正则化项的验证

修改dropout，设为0.5进行观察：





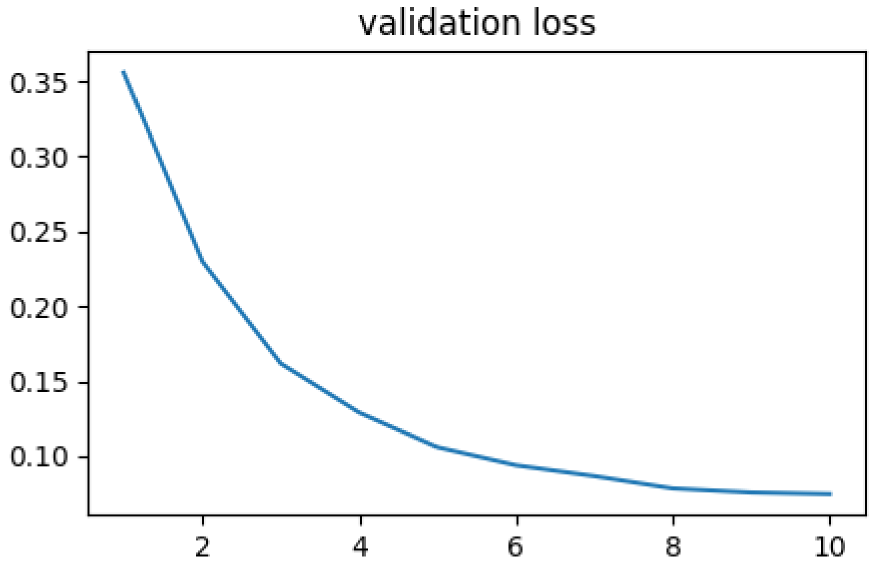
基本没有区别

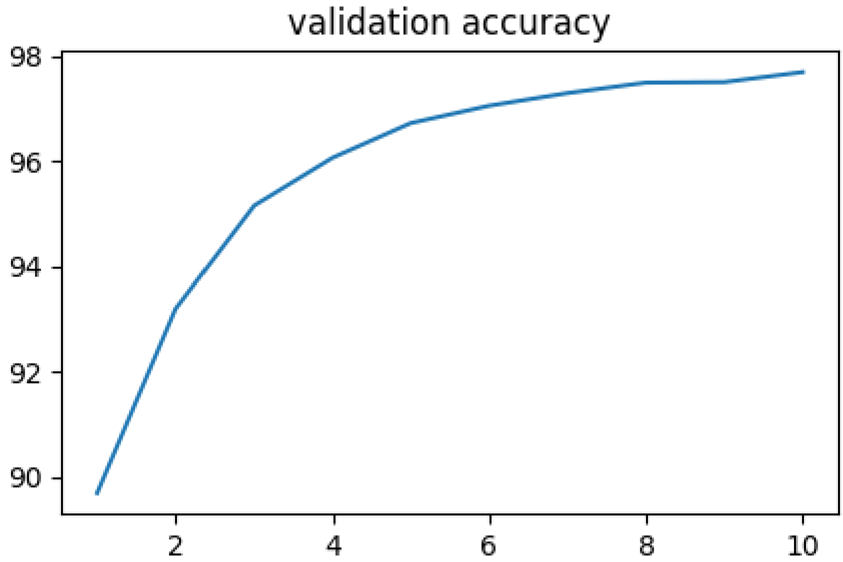
2.3 模型层数修改

修改模型为4层，其他参数保持不变：

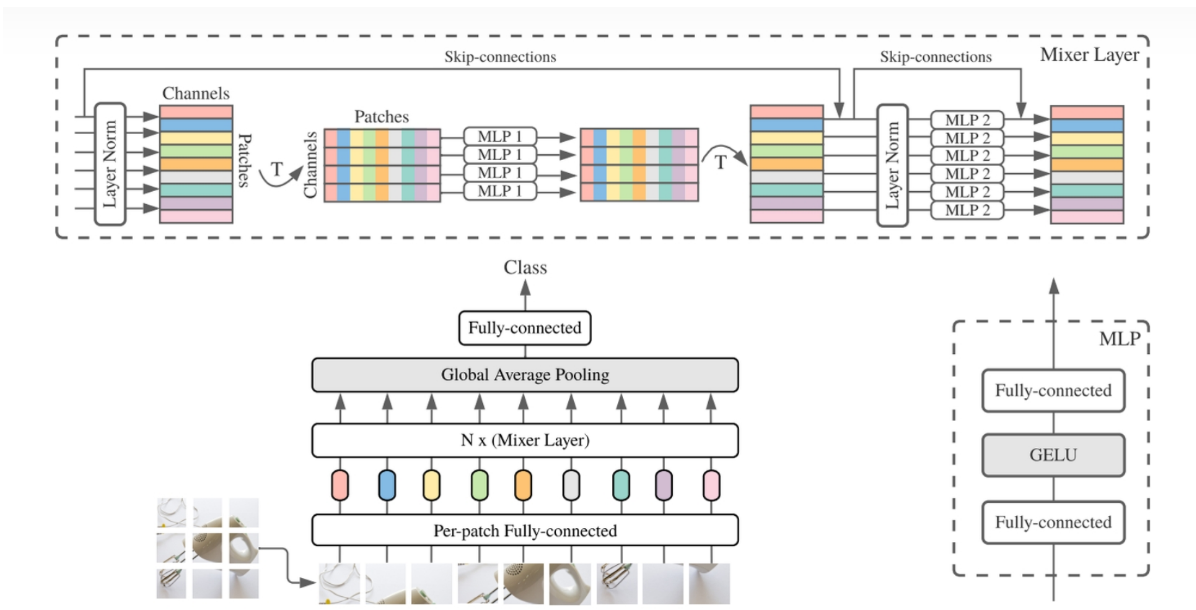
```
Net(
  (fc1): Linear(in_features=784, out_features=500, bias=True)
  (fc1_drop): Dropout(p=0.2, inplace=False)
  (fc2): Linear(in_features=500, out_features=200, bias=True)
  (fc2_drop): Dropout(p=0.2, inplace=False)
  (fc3): Linear(in_features=200, out_features=80, bias=True)
  (fc3_drop): Dropout(p=0.2, inplace=False)
  (fc4): Linear(in_features=80, out_features=10, bias=True)
)
```

训练结果如下，相对于原来3层线性层，确实有一点提升，这是模型参数量带来的





3 MLPMixer



上图是MLPMixer的结构图，有类似于ViT的设计。首先先将图片分解为patch，然后送入全连接层做一个类似patch-embedding的操作，然后将patch拼起来送入Mixer MLP层，最后通过全局平均池化合全连接层输出类别，我的实现如下：

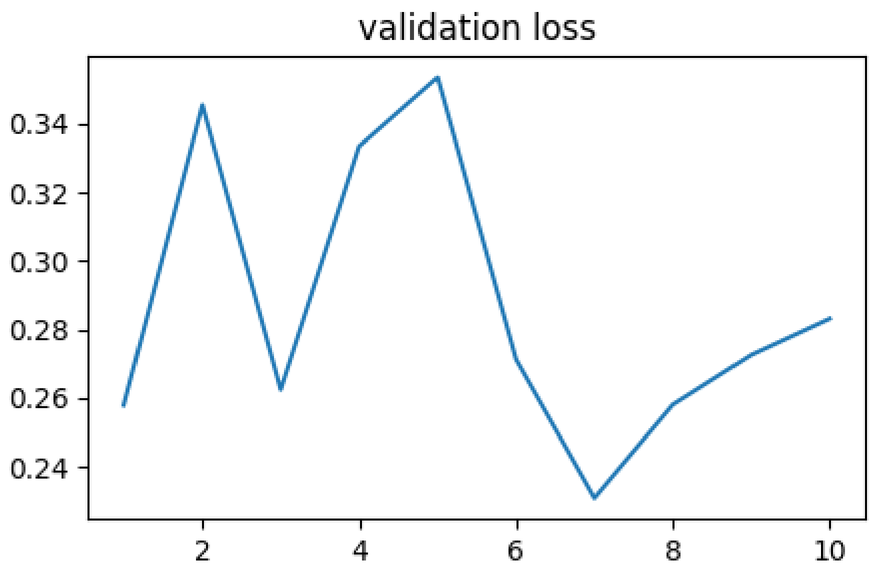
```python
# 定义 MlpMixer 模型
class MlpBlock(nn.Module):
    def __init__(self, dim, mlp_dim, dropout=0.):
        super(MlpBlock, self).__init__()
        self.fc1 = nn.Linear(dim, mlp_dim)
        self.fc2 = nn.Linear(mlp_dim, dim)
        self.dropout = nn.Dropout(dropout)
        self.act = nn.GELU()

    def forward(self, x):
        x = self.fc1(x)
        x = self.act(x)
        x = self.dropout(x)
        x = self.fc2(x)
        x = self.act(x)
        x = self.dropout(x)
        return x

class MixerLayer(nn.Module):
    def __init__(self, num_patches, hidden_dim, token_mlp_dim, channel_mlp_dim,
dropout=0.):
        super(MixerLayer, self).__init__()
        self.token_mixing = nn.Sequential(
            nn.LayerNorm(num_patches),
            nn.Linear(num_patches, token_mlp_dim),
            nn.GELU(),
            nn.Dropout(dropout),
            nn.Linear(token_mlp_dim, num_patches),
            nn.Dropout(dropout)
        )
        self.channel_mixing = nn.Sequential(
            nn.LayerNorm(hidden_dim),
            nn.Linear(hidden_dim, channel_mlp_dim),
            nn.GELU(),
            nn.Dropout(dropout),
            nn.Linear(channel_mlp_dim, hidden_dim),
            nn.Dropout(dropout)
        )

    def forward(self, x):
        # Token mixing
        y = x.permute(0, 2, 1)
        y = self.token_mixing(y)
        y = y.permute(0, 2, 1)
        x = x + y
        # Channel mixing
        y = self.channel_mixing(x)
        x = x + y
        return x

class MlpMixer(nn.Module):
    def __init__(self, image_size, patch_size, hidden_dim, num_layers,
num_classes, token_mlp_dim, channel_mlp_dim, dropout=0.):
        super(MlpMixer, self).__init__()
```

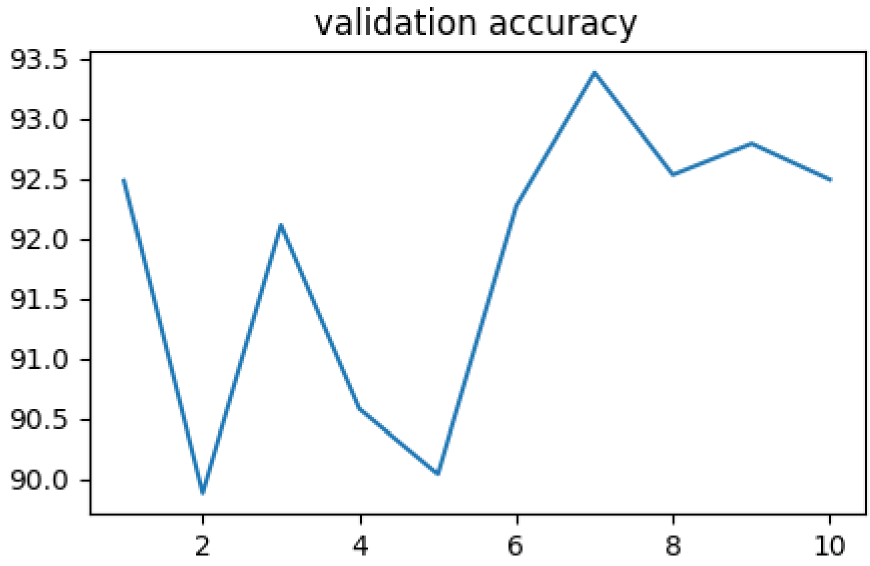
```python
        assert image_size % patch_size == 0, 'Image dimensions must be divisible by the patch size.'
        num_patches = (image_size // patch_size) ** 2

        self.patch_embedding = nn.Conv2d(1, hidden_dim, kernel_size=patch_size, stride=patch_size)
        self.mixer_layers = nn.ModuleList([
            MixerLayer(num_patches, hidden_dim, token_mlp_dim, channel_mlp_dim, dropout) for _ in range(num_layers)
        ])
        self.layer_norm = nn.LayerNorm(hidden_dim)
        self.fc = nn.Linear(hidden_dim, num_classes)

    def forward(self, x):
        x = self.patch_embedding(x).flatten(2).permute(0, 2, 1)  # [batch_size, num_patches, hidden_dim]
        for layer in self.mixer_layers:
            x = layer(x)
        x = self.layer_norm(x)  # Apply LayerNorm on the last dimension
        x = x.mean(dim=1)  # Global average pooling
        return self.fc(x)
```

2层的Mlp-Mixer实验结果（patch=7）：

```
Train Epoch: 1 [0/60000 (0%)]	Loss: 0.602157
Train Epoch: 1 [6400/60000 (11%)]	Loss: 0.583576
Train Epoch: 1 [12800/60000 (21%)]	Loss: 0.415777
Train Epoch: 1 [19200/60000 (32%)]	Loss: 0.303204
Train Epoch: 1 [25600/60000 (43%)]	Loss: 0.261712
```

训练第一轮loss就这么低，感觉有过拟合的嫌疑





最后也取得了不如MLP差不多的性能（汗），推测是参数量太大欠拟合了