实验三： 图像分类及经典CNN实现

10215501433 仲韦萱

实验目的和任务

了解、熟悉并使用CNN不同卷积神经网络模型实现MNIST手写数字识别任务使用多种CNN模型（至少四种）：必选的三种架构：LeNet[1], AlexNet[2], ResNet[3]其它：自己挑选感兴趣的一个或多个架构实现。

实验过程：

（详细代码见py文件，篇幅原因以用torchinfo.summary 可视化网络结构代替代码解释）

本次实验MNIST数据集一共70000张图像，其中训练集包含60000张，测试集10000张图像。其中图像为单通道的黑白图像，尺寸28*28。需要通过搭建不同的CNN卷积网络模型对数据集进行训练，然后对于测试集的数据进行预测，和给定的标签进行对比，给出预测的准确率。

数据预处理：

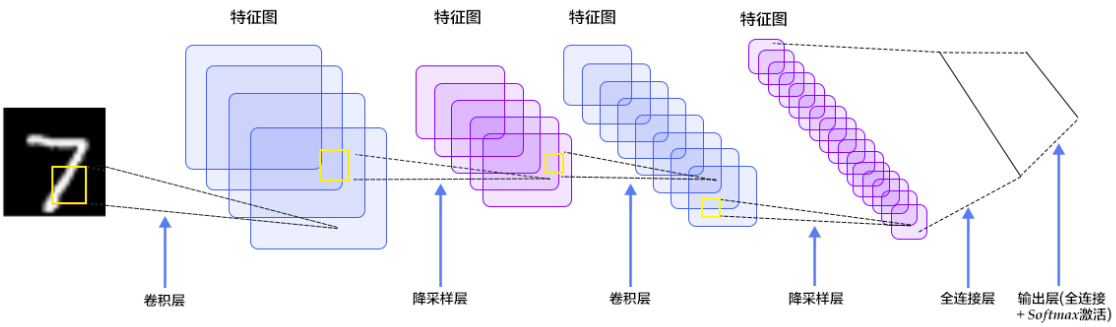
Pytorch 里面的 torchvision 已经实现了 MNIST 方法用于加载 MNIST 数据集，并对其进行张量化和标准化。同时可以使用 torch.utils.data.random_split 将数据集8：2划分为训练集与验证集。（也可以手动加载处理数据集，由于篇幅原因代码详见mnist.py）

搭建网络

本次选取LeNet、AlexNet、ResNet、VGGNet、DenseNet、MobileNet模型

LeNet

LeNet-5的基本结构包括7层网络结构（不含输入层），其中包括2个卷积层、2个池化层、2个全连接层和输出层。



1、卷积+池化层（前五层）

- 输入层接收大小为 32×32 的手写数字图像，其中包括灰度值（0-255）
- 卷积层C1：C1包括6个卷积核，每个卷积核的大小为 5×5 ，步长为1，填充为0。每个卷积核会产生一个大小为 28×28 的特征图。
- 池化层S2：采用最大池化操作，每个窗口的大小为 2×2 ，步长为2。因此，每个池化操作会从4个相邻的特征图中选择最大值，产生一个大小为 14×14 的特征图（输出通道数为6）。这样可以减少特征图的大小，提高计算效率，并且对于轻微的位置变化可以保持一定的不变性。

- 卷积层C3：卷积层C3包括16个卷积核，每个卷积核的大小为 5×5 ，步长为1，填充为0。因此，每个卷积核会产生一个大小为 10×10 的特征图（输出通道数为16）。
- 池化层S4：采样层S4采用最大池化操作，每个窗口的大小为 2×2 ，步长为2，每个池化操作会从4个相邻的特征图中选择最大值，产生一个大小为 5×5 的特征图（输出通道数为16）。

2、全连接层（后三层）

- 全连接层C5：C5将每个大小为 5×5 的特征图拉成一个长度为400的向量，并通过一个带有120个神经元的全连接层进行连接。120是由LeNet-5的设计者根据实验得到的最佳值。
- 全连接层F6：全连接层F6将120个神经元连接到84个神经元。
- 输出层：10个神经元组成，每个神经元对应0-9中数字，输出最终的分类结果。训练过程中，使用交叉熵损失函数计算输出层的误差，通过反向传播算法更新卷积核和全连接层的权重参数。

可视化本次实现的网络：

```
==========================================================================
LeNet                                    --                       --
├─Sequential: 1-1                        [64, 16, 5, 5]           --
│    └─Conv2d: 2-1                       [64, 6, 28, 28]          156
│    └─BatchNorm2d: 2-2                  [64, 6, 28, 28]          12
│    └─Sigmoid: 2-3                      [64, 6, 28, 28]          --
│    └─MaxPool2d: 2-4                    [64, 6, 14, 14]          --
│    └─Conv2d: 2-5                       [64, 16, 10, 10]         2,416
│    └─Sigmoid: 2-6                      [64, 16, 10, 10]         --
│    └─BatchNorm2d: 2-7                  [64, 16, 10, 10]         32
│    └─MaxPool2d: 2-8                    [64, 16, 5, 5]           --
├─Sequential: 1-2                        [64, 10]                 --
│    └─Linear: 2-9                       [64, 120]                48,120
│    └─ReLU: 2-10                        [64, 120]                --
│    └─Linear: 2-11                      [64, 84]                 10,164
│    └─ReLU: 2-12                        [64, 84]                 --
│    └─Linear: 2-13                      [64, 10]                 850
│    └─Dropout: 2-14                     [64, 10]                 --
==========================================================================
Total params: 61,750
Trainable params: 61,750
Non-trainable params: 0
Total mult-adds (M): 27.08
==========================================================================
Input size (MB): 0.20
Forward/backward pass size (MB): 6.56
Params size (MB): 0.25
Estimated Total Size (MB): 7.01
```

AlexNet：

1、卷积+池化层（前五层）

- 卷积层C1：使用96个核—224 × 224 × 3的输入图像，卷积核大小为11 × 11 × 3，步长为4。将一对55×55×48的特征图分别放入ReLU激活函数，生成激活图。激活后的图像进行最大池化，size为3×3，stride为2，池化后的特征图size为27×27×48（一对）。池化后进行LRN处理。
- 卷积层C2：使用卷积层C1的输出（响应归一化和池化）作为输入，并使用256个卷积核，核大小为 5 × 5 × 48。
- 卷积层C3：有384个核，核大小为3 × 3 × 256，与卷积层C2的输出（归一化，池化）相连。
- 卷积层C4：有384个核，核大小为3 × 3 × 192。
- 卷积层C5：有256个核，核大小为3 × 3 × 192。卷积层C5与C3、C4层相比多了个池化，池化核size同样为3×3，stride为2。
  其中，卷积层C3、C4、C5互相连接，中间没有接入池化层或归一化层。

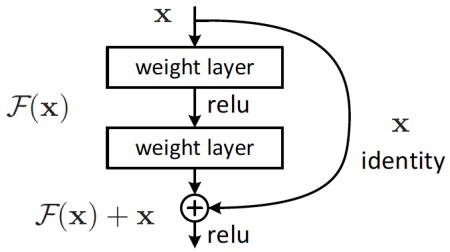
2、全连接层（后三层）

- 全连接层F6：因为是全连接层，卷积核size为6×6×256，4096个卷积核生成4096个特征图，尺寸为1×1。然后放入ReLU函数、Dropout处理。值得注意的是AlexNet使用了Dropout层，以减少过拟合现象的发生。
- 全连接层F7：同F6层。
- 全连接层F8：最后一层全连接层的输出。

可视化本次实现的网络：

```
Layer (type:depth-idx)                                  Output Shape              Param #
==========================================================================================
AlexNet                                                 --                        --
├─Sequential: 1-1                                       [64, 256, 2, 2]           --
│    └─Conv1: 2-1                                       [64, 96, 30, 30]          --
│    │    └─Sequential: 3-1                             [64, 96, 30, 30]          11,712
│    └─Pool1: 2-2                                       [64, 96, 14, 14]          --
          --
│    │    └─Sequential: 3-8                             [64, 256, 2, 2]           --
├─Sequential: 1-2                                       [64, 10]                  --
│    └─FC6: 2-9                                         [64, 4096]                --
│    │    └─Sequential: 3-9                             [64, 4096]                4,198,400
│    └─FC7: 2-10                                        [64, 4096]                --
│    │    └─Sequential: 3-10                            [64, 4096]                16,781,312
│    └─FC8: 2-11                                        [64, 10]                  --
│    │    └─Sequential: 3-11                            [64, 10]                  40,970
==========================================================================================
Total params: 24,744,650
Trainable params: 24,744,650
Non-trainable params: 0
Total mult-adds (G): 16.87
==========================================================================================
Input size (MB): 0.20
Forward/backward pass size (MB): 93.00
Params size (MB): 98.98
Estimated Total Size (MB): 192.18
```

ResNet：



ResNet是一种深度卷积神经网络模型。ResNet于2015年由微软亚洲研究院的Kaiming He等人提出，以解决深度卷积神经网络中梯度消失和梯度爆炸的问题。ResNet通过引入残差块的方式构建网络模型，使得网络在学习过程中可以直接传递某些层的输入信息到后续层的输出中，从而避免了信息丢失和梯度消失的问题。残差块的设计使得网络不仅能够学习到浅层特征，还能够学习到更深层次的特征

1、卷积层+池化层

- 卷积层Conv1: 输入通道数为1，输出通道数为64，卷积核大小为7×7，步长为2，填充为3。
- 批归一化层BN: 用于对卷积层输出进行批量归一化。
- 最大池化层MaxPool: 使用3×3的池化核，步长为2，填充为1。

2、残差块层（Layer1, Layer2, Layer3, Layer4）

- 残差块层Layer1: 包含两个ResNetBlock，每个块内部包含两个3×3的卷积层，输出通道数为64，不进行下采样。
- 残差块层Layer2: 包含一个进行下采样的ResNetBlock和一个不进行下采样的ResNetBlock，卷积层输出通道数分别为128和64。
- 残差块层Layer3: 类似Layer2，包含一个进行下采样的ResNetBlock和一个不进行下采样的ResNetBlock，卷积层输出通道数分别为256和128。
- 残差块层Layer4: 类似Layer2和Layer3，包含一个进行下采样的ResNetBlock和一个不进行下采样的ResNetBlock，卷积层输出通道数分别为512和256。

3、全局平均池化层+全连接层

- 全局平均池化层AvgPool: 对特征图进行自适应平均池化，输出尺寸为1×1。
- 全连接层FC: 输入节点数为512，输出节点数为10，用于分类任务。

可视化本次实现的网络：

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
ResNet                                   --                        --
├─Conv2d: 1-1                            [64, 64, 14, 14]          3,200
├─BatchNorm2d: 1-2                       [64, 64, 14, 14]          128
├─MaxPool2d: 1-3                         [64, 64, 7, 7]            --
├─Sequential: 1-4                        [64, 64, 7, 7]            --
│    └─ResNetBlock: 2-1                  [64, 64, 7, 7]            --
│    │    └─Sequential: 3-1              [64, 64, 7, 7]            37,056
│    │    └─Sequential: 3-2              [64, 64, 7, 7]            37,056
│    └─ResNetBlock: 2-2                  [64, 64, 7, 7]            --
│    │    └─Sequential: 3-3              [64, 64, 7, 7]            37,056
│    │    └─Sequential: 3-4              [64, 64, 7, 7]            37,056
├─Sequential: 1-5                        [64, 128, 4, 4]           --
│    └─ResNetBlock: 2-3                  [64, 128, 4, 4]           --
│    │    └─Sequential: 3-5              [64, 128, 4, 4]           74,112
│    │    └─Sequential: 3-6              [64, 128, 4, 4]           147,840
│    │    └─Sequential: 3-7              [64, 128, 4, 4]           8,576
│    └─ResNetBlock: 2-4                  [64, 128, 4, 4]           --
│    │    └─Sequential: 3-8              [64, 128, 4, 4]           147,840
│    │    └─Sequential: 3-9              [64, 128, 4, 4]           147,840
├─Sequential: 1-6                        [64, 256, 2, 2]           --
│    └─ResNetBlock: 2-5                  [64, 256, 2, 2]           --
│    │    └─Sequential: 3-10             [64, 256, 2, 2]           295,680
│    │    └─Sequential: 3-11             [64, 256, 2, 2]           590,592
│    │    └─Sequential: 3-12             [64, 256, 2, 2]           33,536
│    └─ResNetBlock: 2-6                  [64, 256, 2, 2]           --
│    │    └─Sequential: 3-13             [64, 256, 2, 2]           590,592
│    │    └─Sequential: 3-14             [64, 256, 2, 2]           590,592
├─Sequential: 1-7                        [64, 512, 1, 1]           --
│    └─ResNetBlock: 2-7                  [64, 512, 1, 1]           --
│    │    └─Sequential: 3-15             [64, 512, 1, 1]           1,181,184
│    │    └─Sequential: 3-16             [64, 512, 1, 1]           2,360,832
│    │    └─Sequential: 3-17             [64, 512, 1, 1]           132,608
│    └─ResNetBlock: 2-8                  [64, 512, 1, 1]           --
│    │    └─Sequential: 3-18             [64, 512, 1, 1]           2,360,832
│    │    └─Sequential: 3-19             [64, 512, 1, 1]           2,360,832
├─AdaptiveAvgPool2d: 1-8                 [64, 512, 1, 1]           --
├─Linear: 1-9                            [64, 10]                  5,130
├─Dropout: 1-10                          [64, 10]                  --
==========================================================================================
```

VGGNet

VGGNet是由牛津大学计算机视觉组的研究团队开发和提出。主要特点是采用了非常小的卷积核（3×3）和更深的网络结构。它的基本思想是通过多个卷积层和池化层的堆叠来构建更深的网络，以实现更好的特征提取能力。VGGNet通常包含16或19个卷积层，其中包括5个卷积块以及3个全连接层。在每个卷积块中，VGGNet使用相同数量的卷积层和池化层。这种设计使得网络结构非常规整和可扩展，方便进行改进和调整。同时，VGGNet的卷积层都使用了较小的卷积核，这样可以减少参数数量，增加网络的深度，并且保持对感受野的高度敏感性。

1、卷积层+ReLU激活函数+最大池化层

- 卷积层Conv1: 输入通道数为1，输出通道数为64，卷积核大小为3×3，填充为1。
- ReLU激活函数ReLU: 对卷积层输出进行非线性激活。
- 卷积层Conv2: 输入通道数为64，输出通道数为64，卷积核大小为3×3，填充为1。
- ReLU激活函数ReLU: 对卷积层输出进行非线性激活。
- 最大池化层MaxPool: 使用2×2的池化核，步长为2。

2、卷积层+ReLU激活函数+最大池化层

- 卷积层Conv3: 输入通道数为64，输出通道数为128，卷积核大小为3×3，填充为1。
- ReLU激活函数ReLU: 对卷积层输出进行非线性激活。
- 卷积层Conv4: 输入通道数为128，输出通道数为128，卷积核大小为3×3，填充为1。
- ReLU激活函数ReLU: 对卷积层输出进行非线性激活。
- 最大池化层MaxPool: 使用2×2的池化核，步长为2。

3、全连接层+ReLU激活函数

- Dropout层: 对输入进行随机失活，防止过拟合。
- 全连接层FC1: 输入节点数为128×7×7，输出节点数为512。
- ReLU激活函数ReLU: 对全连接层输出进行非线性激活。
- Dropout层: 对输入进行随机失活，防止过拟合。
- 全连接层FC2: 输入节点数为512，输出节点数为512。
- ReLU激活函数ReLU: 对全连接层输出进行非线性激活。
- 全连接层FC3: 输入节点数为512，输出节点数为10，用于分类任务。

可视化实现的网络：

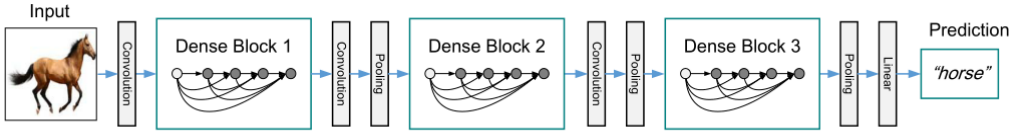
```
VGGNet                                   --                        --
├─Sequential: 1-1                       [64, 128, 7, 7]           --
│    └─Conv2d: 2-1                      [64, 64, 28, 28]          640
│    └─ReLU: 2-2                        [64, 64, 28, 28]          --
│    └─Conv2d: 2-3                      [64, 64, 28, 28]          36,928
│    └─ReLU: 2-4                        [64, 64, 28, 28]          --
│    └─MaxPool2d: 2-5                   [64, 64, 14, 14]          --
│    └─Conv2d: 2-6                      [64, 128, 14, 14]         73,856
│    └─ReLU: 2-7                        [64, 128, 14, 14]         --
│    └─Conv2d: 2-8                      [64, 128, 14, 14]         147,584
│    └─ReLU: 2-9                        [64, 128, 14, 14]         --
│    └─MaxPool2d: 2-10                  [64, 128, 7, 7]           --
├─Sequential: 1-2                       [64, 10]                  --
│    └─Dropout: 2-11                    [64, 6272]                --
│    └─Linear: 2-12                     [64, 512]                 3,211,776
│    └─ReLU: 2-13                       [64, 512]                 --
│    └─Dropout: 2-14                    [64, 512]                 --
│    └─Linear: 2-15                     [64, 512]                 262,656
│    └─ReLU: 2-16                       [64, 512]                 --
│    └─Linear: 2-17                     [64, 10]                  5,130
==========================================================================
Total params: 3,738,570
Trainable params: 3,738,570
Non-trainable params: 0
Total mult-adds (G): 4.89
==========================================================================
Input size (MB): 0.20
Forward/backward pass size (MB): 77.60
Params size (MB): 14.95
Estimated Total Size (MB): 92.75
```

DenseNet：

主要特点是在网络中连接了密集块，这些块中的每个层都与前面所有层直接相连，也就是说，每个层的输入不仅包括前一个层的输出，还包括前面所有层的输出。

DenseNet相比于其他的卷积神经网络结构具有以下优点：

1. 参数少：通过密集连接的方式，DenseNet在训练过程中能够更充分地利用之前层的特征，减少了参数的数量，使得模型更加轻量级。
2. 特征重用：在DenseNet中，每个层都可以访问所有之前层的特征图，这样可以更好地利用之前层的信息，提高了特征的重用性能力。
3. 梯度传播更加稳定：由于每个层都可以访问之前层的特征图，在反向传播时，梯度会在所有层之间进行流动，从而使得梯度传播更加稳定。



1. **ConvLayer**: 这是DenseNet的第一层，它包含一个卷积层(nn.Conv2d)和一个批归一化层(nn.BatchNorm2d)。卷积层使用7x7的卷积核，步长为2，并填充为3。它将输入的单通道图像转换为64个特征图。然后，通过ReLU激活函数进行非线性变换，并使用最大池化层(nn.MaxPool2d)缩小特征图的大小。
2. **DenseLayer**: 这是DenseNet中的一个密集层，用于构建密集块。它包含了两个卷积层和两个批归一化层。首先，通过批归一化和ReLU激活函数对输入的通道进行归一化和非线性变换。然后，使

用1x1的卷积核将输入通道数减少为瓶颈尺寸(bottle_neck_size)。接下来，再次进行批归一化和ReLU激活函数操作，并使用3x3的卷积核进行特征提取。这样，每个密集层都会生成一个增长率(growth_rate)大小的特征图。

3. **DenseBlock**: 这是由多个密集层组成的密集块。在初始化时，根据`layer_counts`参数的值，重复添加密集层。每个密集层的输入通道数是前面所有密集层输出的特征图的通道数之和。
4. **TransitionLayer**: 这是DenseNet中的一个过渡层，用于减小特征图的大小。它包含一个批归一化层、ReLU激活函数、1x1卷积层和2x2的平均池化层。通过批归一化和ReLU激活函数对输入通道进行归一化和非线性变换。然后，使用1x1的卷积核将特征图的通道数减少为指定的输出通道数。最后，通过平均池化层缩小特征图的大小。
5. **DenseNet**: 这是整个DenseNet模型的主要部分。它包含了多个ConvLayer、DenseBlock和TransitionLayer，用于构建特征提取器(feature_extractor)。特征提取器通过堆叠多个密集块和过渡层来逐步提取图像特征。最后，通过全局自适应平均池化(nn.AdaptiveAvgPool2d)将特征图压缩为1x1的大小。然后，将压缩后的特征图通过展平操作(view)转换为向量，并通过全连接层进行分类。最后，添加了一个dropout层以防止过拟合。

可视化实现的网络：

```
DenseNet                                 --                        --
│    └─TransitionLayer: 2-7              [64, 512, 1, 1]           --
│    │    └─Sequential: 3-4              [64, 512, 1, 1]           526,336
│    └─DenseBlock: 2-8                   [64, 1024, 1, 1]          --
├─AdaptiveAvgPool2d: 1-2                 [64, 1024, 1, 1]          --
├─Sequential: 1-3                        [64, 10]                  --
│    └─Linear: 2-9                       [64, 10]                  10,250
│    └─Dropout: 2-10                     [64, 10]                  --
==========================================================================================
Total params: 705,290
Trainable params: 705,290
Non-trainable params: 0
Total mult-adds (M): 455.98
==========================================================================================
Input size (MB): 0.20
Forward/backward pass size (MB): 38.80
Params size (MB): 2.82
Estimated Total Size (MB): 41.82
```

MobileNet

MobileNet是一种轻量级的卷积神经网络结构，由Google于2017年提出，旨在解决在移动设备和嵌入式设备上进行实时图像识别和分析的需求。MobileNet的设计目标是在保持较高的准确性的同时大幅减少模型的参数量和计算量，以便在资源受限的环境下实现高效的推理。

MobileNet的主要特点包括：

1. 深度可分离卷积（Depthwise Separable Convolution）：MobileNet使用深度可分离卷积来替代标准卷积，这种卷积操作将标准卷积分解为深度卷积和逐点卷积两步，从而降低了参数数量和计算复杂度。深度可分离卷积分别对输入的每个通道进行卷积，然后再将结果进行逐点卷积，以此来实现卷积操作。
2. 轻量级设计：MobileNet采用了一系列设计来减少模型的参数数量和计算量，包括窄的卷积核，少量的通道数和全局平均池化等。
3. 灵活的结构：MobileNet可以根据需求进行扩展或者缩减，通过调整超参数来得到不同大小和性能的模型，以满足不同场景下的资源限制和准确性要求。

本次实现的网络：

- 上采样层：比例因子：8，双线性插值上采样，输出尺寸：1792 x 1792 x 1
- 卷积层C1：输入通道数：1，输出通道数：32，卷积核大小：3 x 3，步长：2，填充：1，BatchNormalization层，ReLU6激活函数，输出尺寸：896 x 896 x 32

MobileBlock块1：输入通道数：32，输出通道数：64，步长：1

- 第一个卷积层：输入通道数：32， 输出通道数：32，卷积核大小：3 x 3，步长：1，填充：1， 分组卷积(每个通道有自己的卷积核)，BatchNormalization层，ReLU6激活函数
- 第二个卷积层：输入通道数：32，输出通道数：64，卷积核大小：1 x 1，步长：1，BatchNormalization层，ReLU6激活函数，输出尺寸：896 x 896 x 64

（……此处省略中间5个MobileBlock块分析）

MobileBlock块7：输入通道数：512，输出通道数：512，步长：1

- 第一个卷积层： 输入通道数：512，输出通道数：512，卷积核大小：3 x 3，步长：1，填充：1，分组卷积，BatchNormalization层，ReLU6激活函数
- 第二个卷积层：，输入通道数：512，输出通道数：512，卷积核大小：1 x 1，步长：1，BatchNormalization层，ReLU6激活函数，输出尺寸：112 x 112 x 512
  平均池化层：池化窗口大小：7 x 7，平均池化，输出尺寸：1 x 1 x 512
  Flatten层：将输入展平成一维向量，输出尺寸：512
  全连接层FC1：输入特征数：512，输出特征数：10，Dropout处理，输出尺寸：10

可视化实现的网络：

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
MobileNet                                --                        --
├─Sequential: 1-1                        [64, 10]                  --
│    └─Upsample: 2-1                     [64, 1, 224, 224]         --
│    └─Conv2d: 2-2                       [64, 32, 112, 112]        288
│    └─BatchNorm2d: 2-3                  [64, 32, 112, 112]        64
│    └─ReLU6: 2-4                        [64, 32, 112, 112]        --
│    └─MobileBlock: 2-5                  [64, 64, 112, 112]        --
│    │    └─Sequential: 3-1              [64, 64, 112, 112]        2,528
│    └─MobileBlock: 2-6                  [64, 128, 56, 56]         --
│    │    └─Sequential: 3-2              [64, 128, 56, 56]         9,152
│    └─MobileBlock: 2-7                  [64, 128, 56, 56]         --
│    │    └─Sequential: 3-3              [64, 128, 56, 56]         18,048
│    └─MobileBlock: 2-8                  [64, 256, 28, 28]         --
│    │    └─Sequential: 3-4              [64, 256, 28, 28]         34,688
│    └─MobileBlock: 2-9                  [64, 256, 28, 28]         --
│    │    └─Sequential: 3-5              [64, 256, 28, 28]         68,864
│    └─MobileBlock: 2-10                 [64, 512, 14, 14]         --
│    │    └─Sequential: 3-6              [64, 512, 14, 14]         134,912
│    └─MobileBlock: 2-11                 [64, 512, 14, 14]         --
│    │    └─Sequential: 3-7              [64, 512, 14, 14]         268,800
│    └─AvgPool2d: 2-12                   [64, 512, 2, 2]           --
│    └─Flatten: 2-13                     [64, 2048]                --
│    └─Linear: 2-14                      [64, 10]                  20,490
│    └─Dropout: 2-15                     [64, 10]                  --
==========================================================================================
Total params: 557,834
Trainable params: 557,834
Non-trainable params: 0
Total mult-adds (G): 17.51
==========================================================================================
Input size (MB): 0.20
Forward/backward pass size (MB): 4161.80
Params size (MB): 2.23
Estimated Total Size (MB): 4164.24
==========================================================================================
```

实验结果：

优化

通过调整超参数：

以LeNet为例：（其他模型调参过程一样，但因cpu运行时间较长，只在LeNet模型上尝试了10+epoch）

尝试一组lr和dropout参数— 修改lr得到更好的准确率—尝试增加训练轮次

```
PS E:\大三上\当代人工智能\code\3 (12.8) \new>  & "c:/Program Files/PsychoPy/python.exe" e:/
大三上/当代人工智能/code/3 (12.8) /new/main.py --model LeNet --lr 0.004 --batch_size 128 --
epoch 10 -dropout 0.1
Validation - Epoch: 1, Accuracy: 0.974417
Validation - Epoch: 2, Accuracy: 0.976083
Validation - Epoch: 3, Accuracy: 0.978667
Validation - Epoch: 4, Accuracy: 0.982750
Validation - Epoch: 5, Accuracy: 0.985250
Validation - Epoch: 6, Accuracy: 0.984417
Validation - Epoch: 7, Accuracy: 0.984250
Validation - Epoch: 8, Accuracy: 0.986750
Validation - Epoch: 9, Accuracy: 0.970667
Validation - Epoch: 10, Accuracy: 0.979917
Test - Accuracy: 0.981200
```

```
PS E:\大三上\当代人工智能\code\3 (12.8) \new>  & "c:/Program Files/PsychoPy/python.exe" e:/大三上/当代人工智能/code/3 (12.8) /new/main_.py --model LeNet --lr 0.0001 --batch_size 128 --epoch 10 -dropout 0.1
Validation - Epoch: 1, Accuracy: 0.923667
Validation - Epoch: 2, Accuracy: 0.951917
Validation - Epoch: 3, Accuracy: 0.961083
Validation - Epoch: 4, Accuracy: 0.968167
Validation - Epoch: 5, Accuracy: 0.971667
Validation - Epoch: 6, Accuracy: 0.974000
Validation - Epoch: 7, Accuracy: 0.977167
Validation - Epoch: 8, Accuracy: 0.977083
Validation - Epoch: 9, Accuracy: 0.977583
Validation - Epoch: 10, Accuracy: 0.981250
Test - Accuracy: 0.983800
```

```
PS E:\大三上\当代人工智能\code\3 (12.8) \new>  & "c:/Program Files/PsychoPy/python.exe" e:/大三上/当代人工智能/code/3 (12.8) /new/main_.py --model LeNet --lr 0.0001 --batch_size 128 --epoch 20 -dropout 0.1
Validation - Epoch: 1, Accuracy: 0.903500
Validation - Epoch: 2, Accuracy: 0.935167
Validation - Epoch: 3, Accuracy: 0.953083
Validation - Epoch: 4, Accuracy: 0.961167
Validation - Epoch: 5, Accuracy: 0.966333
Validation - Epoch: 6, Accuracy: 0.970833
Validation - Epoch: 7, Accuracy: 0.973083
Validation - Epoch: 8, Accuracy: 0.975083
Validation - Epoch: 9, Accuracy: 0.976250
Validation - Epoch: 10, Accuracy: 0.979667
Validation - Epoch: 11, Accuracy: 0.978833
Validation - Epoch: 12, Accuracy: 0.979417
Validation - Epoch: 13, Accuracy: 0.981500
Validation - Epoch: 14, Accuracy: 0.982250
Validation - Epoch: 15, Accuracy: 0.981500
Validation - Epoch: 16, Accuracy: 0.982583
Validation - Epoch: 17, Accuracy: 0.984000
Validation - Epoch: 18, Accuracy: 0.983083
Validation - Epoch: 19, Accuracy: 0.984167
Validation - Epoch: 20, Accuracy: 0.985000
Test - Accuracy: 0.986200
```

结果

以下顺序为AlexNet、ResNet、VGGNet、MobileNet、DenseNet

```
PS E:\大三上\当代人工智能\code\3 (12.8) \new>  & "c:/Program Files/PsychoPy/python.exe" e:/大三上/当代人工智能/code/3 (12.8) /new/main.py --model AlexNet --lr 0.004 --batch_size 128 --epoch 10 -dropout 0.2
Validation - Epoch: 1, Accuracy: 0.921000
Validation - Epoch: 2, Accuracy: 0.959417
Validation - Epoch: 3, Accuracy: 0.967417
Validation - Epoch: 4, Accuracy: 0.961917
Validation - Epoch: 5, Accuracy: 0.972917
Validation - Epoch: 6, Accuracy: 0.973167
Validation - Epoch: 7, Accuracy: 0.972167
Validation - Epoch: 8, Accuracy: 0.972000
Validation - Epoch: 9, Accuracy: 0.975250
Validation - Epoch: 10, Accuracy: 0.976833
Test - Accuracy: 0.981300
================================================================================
PS E:\大三上\当代人工智能\code\3 (12.8) \new>  & "c:/Program Files/PsychoPy/python.exe" e:/大三上/当代人工智能/code/3 (12.8) /new/main.py --model ResNet --lr 0.004 --batch_size 128 --epoch 10 -dropout 0.1
Validation - Epoch: 1, Accuracy: 0.946500
Validation - Epoch: 2, Accuracy: 0.978417
Validation - Epoch: 3, Accuracy: 0.977250
Validation - Epoch: 4, Accuracy: 0.980083
Validation - Epoch: 5, Accuracy: 0.985250
Validation - Epoch: 6, Accuracy: 0.954250
Validation - Epoch: 7, Accuracy: 0.987833
Validation - Epoch: 8, Accuracy: 0.987917
Validation - Epoch: 9, Accuracy: 0.988417
Validation - Epoch: 10, Accuracy: 0.986750
Test - Accuracy: 0.987500
```

```
PS E:\大三上\当代人工智能\code\3 (12.8) \new>  & "c:/Program Files/PsychoPy/python.exe" e:/大三上/当代人工智能/code/3 (12.8) /new/main.py --model VGGNet --lr 0.004 --batch_size 128 --epoch 10 -dropout 0.4
Validation - Epoch: 1, Accuracy: 0.941000
Validation - Epoch: 2, Accuracy: 0.964833
Validation - Epoch: 3, Accuracy: 0.972083
Validation - Epoch: 4, Accuracy: 0.976667
Validation - Epoch: 5, Accuracy: 0.980833
Validation - Epoch: 6, Accuracy: 0.981583
Validation - Epoch: 7, Accuracy: 0.981250
Validation - Epoch: 8, Accuracy: 0.984917
Validation - Epoch: 9, Accuracy: 0.983333
Validation - Epoch: 10, Accuracy: 0.984500
Test - Accuracy: 0.983800
```

```
PS E:\大三上\当代人工智能\code\3 (12.8) \new>  & "c:/Program Files/PsychoPy/python.exe" e:/大三上/当代人工智能/code/3 (12.8) /new/main.py --model MobileNet --lr 0.004 --batch_size 128 --epoch 10 -dropout 0.4
C:\Users\47517\AppData\Roaming\Python\Python36\site-packages\torch\nn\functional.py:3635: UserWarning: Default upsampling behavior when mode=bilinear is changed to align_corners=False since 0.4.0. Please specify align_corners=True if the old behavior is desired. See the documentation of nn.Upsample for details.
  "See the documentation of nn.Upsample for details.".format(mode)
Validation - Epoch: 1, Accuracy: 0.976000
Validation - Epoch: 2, Accuracy: 0.985000
Validation - Epoch: 3, Accuracy: 0.983250
Validation - Epoch: 4, Accuracy: 0.989667
Validation - Epoch: 5, Accuracy: 0.990417
Validation - Epoch: 6, Accuracy: 0.989083
Validation - Epoch: 7, Accuracy: 0.989750
Validation - Epoch: 8, Accuracy: 0.989417
Validation - Epoch: 9, Accuracy: 0.989083
Validation - Epoch: 10, Accuracy: 0.989583
Test - Accuracy: 0.991800
PS E:\大三上\当代人工智能\code\3 (12.8) \new>  & "c:/Program Files/PsychoPy/python.exe" e:/大三上/当代人工智能/code/3 (12.8) /new/main.py --model DenseNet --lr 0.004 --batch_size 128 --epoch 10 -dropout 0.2
Validation - Epoch: 1, Accuracy: 0.976833
Validation - Epoch: 2, Accuracy: 0.982583
Validation - Epoch: 3, Accuracy: 0.982583
Validation - Epoch: 4, Accuracy: 0.983667
Validation - Epoch: 5, Accuracy: 0.984667
Validation - Epoch: 6, Accuracy: 0.986583
Validation - Epoch: 7, Accuracy: 0.984750
Validation - Epoch: 8, Accuracy: 0.987500
Validation - Epoch: 9, Accuracy: 0.986333
Validation - Epoch: 10, Accuracy: 0.987167
Test - Accuracy: 0.988300
```

LeNet	AlexNet	ResNet	MobileNet	DenseNet	VGGNet
0.987	0.981	0.988	0.992	0.989	0.984

其中

- 几种模型准确率相差不大，都在98%以上，说明拟合的很不错。
- LeNet是因为增加了训练轮数至20，所以小范围提高了一点准确率，本次实验MobileNet的结果最好，其次是DenseNet，第三是ResNet。同时AlexNet和VGGNet相对表现不佳.
  猜测实验结果的原因（除了偶然性和训练次数以及更多种参数组合）
- MobileNet表现优异，主要得益于其采用的深度可分离卷积结构，这种结构能够显著减少参数数量和计算量，使得在资源受限的移动设备和嵌入式设备上能够高效地进行图像识别和分析。因此，MobileNet在计算资源有限（本次cpu设备）的情况下能取得较好的性能。DenseNet在利用密集连接的设计思想，使得特征复用更加充分，有利于信息的传递和梯度的流动，提高了网络的训练效率和特征表达能力，因此取得较好的性能。ResNet引入了残差连接，通过跨层的直接连接，解决了深层网络训练中的梯度消失和梯度爆炸问题，有利于训练非常深的网络，因此通常能够在大规模数据集上能够取得较好的性能。
- 相比之下，AlexNet和VGGNet是早期的深度学习模型，在参数量和计算量方面相对较大，且没有采用后续提出的一些轻量级设计思想，如深度可分离卷积、密集连接等。因此，在资源受限的环境下，它们可能无法充分发挥作用，表现相对较差。

根据以上我认为：可能影响模型性能的原因有：

- 网络深度和宽度：模型的深度和宽度对性能有显著影响。较深的网络可以提取更高级别的抽象特征，但可能会导致梯度消失或梯度爆炸等问题。较宽的网络可以提供更多的特征通道，增加模型的表达能力。
- 卷积操作的设计：不同的模型在卷积操作的设计上有所差异，如卷积核尺寸、步幅、填充等。
- 激活函数的选择：激活函数在神经网络中起着非常重要的作用，能够引入非线性，增强网络的表达能力。不同的激活函数会对网络的性能产生影响，如ReLU、Sigmoid、Tanh等。
- 正则化方法：正则化方法可以帮助防止过拟合现象，提高模型的泛化能力。常见的正则化方法包括L1、L2正则化、Dropout、批标准化等。

问题&解决

1. 本次实验多个模型运行时间较长，因此增大batch_size参数（最小取到128），减小训练轮数，希望后续能够使用gpu进行参数优化
2. 每个模型输入尺寸不同，需要检查好及时调整输入图片大小（使用x = F.interpolate(x, size=32, mode='bilinear', align_corners=False)）