In [1]:
```python
from utils import *
```

In [2]:
```python
import torch
import torch.nn as nn
import torchvision
import torch.nn.functional as F
import torchvision.datasets as datasets
import torchvision.transforms as transforms
```

In [41]:
```python
transform_train = transforms.Compose([
    transforms.RandomResizedCrop(32),  # 随机裁剪，将图像裁剪为 32x32 大小
    transforms.RandomHorizontalFlip(),
    transforms.RandomAffine(degrees = 30, translate = (0.1, 0.1), scale = (0.8, 1.2), shear = 10),    # 随机水平翻转
    transforms.ToTensor(),             # 将 PIL 图像转换为 Tensor
])

transform_val = transforms.Compose([
    transforms.Resize(32),             # 缩放图像到 32x32 大小
    transforms.ToTensor(),
])
```

In [42]:
```python
batch_size = 128 # 设定batch_size
train_set = datasets.CIFAR100(root = "../dataset", train = True, download = True, transform = transform_train) # 加载训练集
val_set = datasets.CIFAR100(root = "../dataset", train = False, download = True, transform = transform_val) # 加载测试集
train_loader = torch.utils.data.DataLoader(train_set, batch_size=batch_size, shuffle=True, num_workers=10) # 生成训练集loader
val_loader = torch.utils.data.DataLoader(val_set, batch_size=batch_size, shuffle=False, num_workers=10) # 生成测试集loader
```

```
Files already downloaded and verified
Files already downloaded and verified
```

In [27]:
```python
# VGG19
class VGG19(nn.Module):
    def __init__(self):
        super().__init__() # 继承nn.Module的性质
        self.cnn = nn.Sequential(
            # conv3-64 + conv3-64 + maxpool
            nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1), nn.BatchNorm2d(64), nn.ReLU(inplace=True),
            nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1), nn.BatchNorm2d(64), nn.ReLU(inplace=True),
            nn.MaxPool2d(2, 2),

            # conv3-128 + conv3-128 + maxpool
            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1), nn.BatchNorm2d(128), nn.ReLU(inplace=True),
            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1), nn.BatchNorm2d(128), nn.ReLU(inplace=True),
            nn.MaxPool2d(2, 2),

            # conv3-256 + conv3-256 + conv3-256 + conv3-256 + maxpool
            nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1), nn.BatchNorm2d(256), nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1), nn.BatchNorm2d(256), nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1), nn.BatchNorm2d(256), nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1), nn.BatchNorm2d(256), nn.ReLU(inplace=True),
            nn.MaxPool2d(2, 2),

            # conv3-512 + conv3-512 + conv3-512 + conv3-512 + maxpool
            nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1), nn.BatchNorm2d(512), nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1), nn.BatchNorm2d(512), nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1), nn.BatchNorm2d(512), nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1), nn.BatchNorm2d(512), nn.ReLU(inplace=True),
            nn.MaxPool2d(2, 2),

            # conv3-512 + conv3-512 + conv3-512 + conv3-512 + maxpool
            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1), nn.BatchNorm2d(512), nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1), nn.BatchNorm2d(512), nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1), nn.BatchNorm2d(512), nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1), nn.BatchNorm2d(512), nn.ReLU(inplace=True),
            nn.MaxPool2d(2, 2),
        )

        # 全连接层 进行分类
        self.fc = nn.Linear(512, 100)

    def forward(self, x):
        out = self.cnn(x) # 经过vgg卷积层
        out = out.view(out.size(0), -1) # 将张量拉直
        out = self.fc(out) # 最终全连接
        return out
```

In [28]:
```python
from torchsummary import summary

IMSIZE = 32
vgg19_model = VGG19().cuda()
summary(vgg19_model, (3, IMSIZE, IMSIZE)) # 模型参数总结
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 32, 32]           1,792
       BatchNorm2d-2           [-1, 64, 32, 32]             128
```

```
            ReLU-3          [-1, 64, 32, 32]               0
          Conv2d-4          [-1, 64, 32, 32]          36,928
     BatchNorm2d-5          [-1, 64, 32, 32]             128
            ReLU-6          [-1, 64, 32, 32]               0
       MaxPool2d-7          [-1, 64, 16, 16]               0
          Conv2d-8         [-1, 128, 16, 16]          73,856
     BatchNorm2d-9         [-1, 128, 16, 16]             256
           ReLU-10         [-1, 128, 16, 16]               0
         Conv2d-11         [-1, 128, 16, 16]         147,584
    BatchNorm2d-12         [-1, 128, 16, 16]             256
           ReLU-13         [-1, 128, 16, 16]               0
      MaxPool2d-14           [-1, 128, 8, 8]               0
         Conv2d-15           [-1, 256, 8, 8]         295,168
    BatchNorm2d-16           [-1, 256, 8, 8]             512
           ReLU-17           [-1, 256, 8, 8]               0
         Conv2d-18           [-1, 256, 8, 8]         590,080
    BatchNorm2d-19           [-1, 256, 8, 8]             512
           ReLU-20           [-1, 256, 8, 8]               0
         Conv2d-21           [-1, 256, 8, 8]         590,080
    BatchNorm2d-22           [-1, 256, 8, 8]             512
           ReLU-23           [-1, 256, 8, 8]               0
         Conv2d-24           [-1, 256, 8, 8]         590,080
    BatchNorm2d-25           [-1, 256, 8, 8]             512
           ReLU-26           [-1, 256, 8, 8]               0
      MaxPool2d-27           [-1, 256, 4, 4]               0
         Conv2d-28           [-1, 512, 4, 4]       1,180,160
    BatchNorm2d-29           [-1, 512, 4, 4]           1,024
           ReLU-30           [-1, 512, 4, 4]               0
         Conv2d-31           [-1, 512, 4, 4]       2,359,808
    BatchNorm2d-32           [-1, 512, 4, 4]           1,024
           ReLU-33           [-1, 512, 4, 4]               0
         Conv2d-34           [-1, 512, 4, 4]       2,359,808
    BatchNorm2d-35           [-1, 512, 4, 4]           1,024
           ReLU-36           [-1, 512, 4, 4]               0
         Conv2d-37           [-1, 512, 4, 4]       2,359,808
    BatchNorm2d-38           [-1, 512, 4, 4]           1,024
           ReLU-39           [-1, 512, 4, 4]               0
      MaxPool2d-40           [-1, 512, 2, 2]               0
         Conv2d-41           [-1, 512, 2, 2]       2,359,808
    BatchNorm2d-42           [-1, 512, 2, 2]           1,024
           ReLU-43           [-1, 512, 2, 2]               0
         Conv2d-44           [-1, 512, 2, 2]       2,359,808
    BatchNorm2d-45           [-1, 512, 2, 2]           1,024
           ReLU-46           [-1, 512, 2, 2]               0
         Conv2d-47           [-1, 512, 2, 2]       2,359,808
    BatchNorm2d-48           [-1, 512, 2, 2]           1,024
           ReLU-49           [-1, 512, 2, 2]               0
         Conv2d-50           [-1, 512, 2, 2]       2,359,808
    BatchNorm2d-51           [-1, 512, 2, 2]           1,024
           ReLU-52           [-1, 512, 2, 2]               0
      MaxPool2d-53           [-1, 512, 1, 1]               0
         Linear-54                 [-1, 100]          51,300
================================================================
Total params: 20,086,692
Trainable params: 20,086,692
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 7.18
Params size (MB): 76.62
Estimated Total Size (MB): 83.81
----------------------------------------------------------------
```

卷积核参数量 = （kernel_size *kernel_size* in_channels + 1） * out_channels

batch_norm参数量 = 2 * channels

linear参数量 = （512+1） * 100

In [35]:

```python
# 训练VGG19模型
lr1 = 3e-3 # 学习率
optimizer1 = torch.optim.Adam(vgg19_model.parameters(), lr=lr1)# 优化器
epochs = 60
train_result1 = train(vgg19_model, optimizer1, train_loader, val_loader, epochs=epochs) # 训练结果
```

```
Epoch [1/60], time: 17.60s, loss: 3.5789, acc: 0.1511, val_loss: 2.7906, val_acc: 0.2746
Epoch [2/60], time: 17.91s, loss: 3.1608, acc: 0.2199, val_loss: 2.4581, val_acc: 0.3562
Epoch [3/60], time: 17.84s, loss: 2.9378, acc: 0.2654, val_loss: 2.4104, val_acc: 0.3739
Epoch [4/60], time: 17.98s, loss: 2.8362, acc: 0.2915, val_loss: 2.3133, val_acc: 0.3945
Epoch [5/60], time: 17.94s, loss: 2.7770, acc: 0.3054, val_loss: 2.3216, val_acc: 0.3948
Epoch [6/60], time: 18.05s, loss: 2.7149, acc: 0.3190, val_loss: 2.2247, val_acc: 0.4226
Epoch [7/60], time: 17.96s, loss: 2.6644, acc: 0.3279, val_loss: 2.1801, val_acc: 0.4334
Epoch [8/60], time: 17.90s, loss: 2.6162, acc: 0.3376, val_loss: 2.1839, val_acc: 0.4334
Epoch [9/60], time: 18.02s, loss: 2.6039, acc: 0.3417, val_loss: 2.2153, val_acc: 0.4245
Epoch [10/60], time: 17.88s, loss: 2.5651, acc: 0.3532, val_loss: 2.1613, val_acc: 0.4417
Epoch [11/60], time: 17.70s, loss: 2.5327, acc: 0.3614, val_loss: 2.1090, val_acc: 0.4468
Epoch [12/60], time: 17.96s, loss: 2.5135, acc: 0.3641, val_loss: 2.0934, val_acc: 0.4546
Epoch [13/60], time: 18.00s, loss: 2.4823, acc: 0.3710, val_loss: 2.2026, val_acc: 0.4420
Epoch [14/60], time: 18.06s, loss: 2.4518, acc: 0.3770, val_loss: 2.0695, val_acc: 0.4632
Epoch [15/60], time: 18.02s, loss: 2.4269, acc: 0.3838, val_loss: 2.1145, val_acc: 0.4616
Epoch [16/60], time: 17.73s, loss: 2.4129, acc: 0.3896, val_loss: 2.0119, val_acc: 0.4804
Epoch [17/60], time: 18.07s, loss: 2.3907, acc: 0.3964, val_loss: 2.1261, val_acc: 0.4670
Epoch [18/60], time: 17.98s, loss: 2.3708, acc: 0.3965, val_loss: 1.9590, val_acc: 0.4856
Epoch [19/60], time: 18.00s, loss: 2.3510, acc: 0.4006, val_loss: 1.9990, val_acc: 0.4798
Epoch [20/60], time: 17.86s, loss: 2.3339, acc: 0.4053, val_loss: 1.9641, val_acc: 0.4836
Epoch [21/60], time: 17.80s, loss: 2.3089, acc: 0.4101, val_loss: 1.9809, val_acc: 0.4890
Epoch [22/60], time: 17.92s, loss: 2.2950, acc: 0.4156, val_loss: 1.9114, val_acc: 0.4999
Epoch [23/60], time: 17.97s, loss: 2.2747, acc: 0.4181, val_loss: 1.9367, val_acc: 0.4970
Epoch [24/60], time: 17.97s, loss: 2.2624, acc: 0.4196, val_loss: 1.9261, val_acc: 0.5011
Epoch [25/60], time: 17.91s, loss: 2.2511, acc: 0.4241, val_loss: 1.9512, val_acc: 0.5010
```

```
Epoch [26/60], time: 17.87s, loss: 2.2298, acc: 0.4294, val_loss: 1.9397, val_acc: 0.4995
Epoch [27/60], time: 17.93s, loss: 2.2225, acc: 0.4314, val_loss: 1.8768, val_acc: 0.5166
Epoch [28/60], time: 18.01s, loss: 2.2013, acc: 0.4347, val_loss: 1.9332, val_acc: 0.5053
Epoch [29/60], time: 17.96s, loss: 2.1895, acc: 0.4399, val_loss: 1.9486, val_acc: 0.5019
Epoch [30/60], time: 18.05s, loss: 2.1871, acc: 0.4391, val_loss: 1.8845, val_acc: 0.5160
Epoch [31/60], time: 17.91s, loss: 2.1605, acc: 0.4470, val_loss: 1.8704, val_acc: 0.5163
Epoch [32/60], time: 17.88s, loss: 2.1607, acc: 0.4455, val_loss: 1.9210, val_acc: 0.5152
Epoch [33/60], time: 17.96s, loss: 2.1402, acc: 0.4493, val_loss: 1.8968, val_acc: 0.5145
Epoch [34/60], time: 17.87s, loss: 2.1314, acc: 0.4559, val_loss: 1.9356, val_acc: 0.5100
Epoch [35/60], time: 17.96s, loss: 2.1286, acc: 0.4532, val_loss: 1.8698, val_acc: 0.5249
Epoch [36/60], time: 17.96s, loss: 2.0966, acc: 0.4613, val_loss: 1.8657, val_acc: 0.5223
Epoch [37/60], time: 17.78s, loss: 2.0889, acc: 0.4641, val_loss: 1.8834, val_acc: 0.5184
Epoch [38/60], time: 17.90s, loss: 2.0779, acc: 0.4647, val_loss: 1.8236, val_acc: 0.5359
Epoch [39/60], time: 17.79s, loss: 2.0762, acc: 0.4689, val_loss: 1.8757, val_acc: 0.5239
Epoch [40/60], time: 17.96s, loss: 2.0591, acc: 0.4689, val_loss: 1.9094, val_acc: 0.5210
Epoch [41/60], time: 17.95s, loss: 2.0492, acc: 0.4728, val_loss: 1.8240, val_acc: 0.5315
Epoch [42/60], time: 17.71s, loss: 2.0357, acc: 0.4781, val_loss: 1.8782, val_acc: 0.5233
Epoch [43/60], time: 17.86s, loss: 2.0142, acc: 0.4824, val_loss: 1.8612, val_acc: 0.5277
Epoch [44/60], time: 17.73s, loss: 2.0182, acc: 0.4812, val_loss: 1.8132, val_acc: 0.5392
Epoch [45/60], time: 17.85s, loss: 2.0117, acc: 0.4820, val_loss: 1.8141, val_acc: 0.5392
Epoch [46/60], time: 17.84s, loss: 2.0026, acc: 0.4856, val_loss: 1.8116, val_acc: 0.5464
Epoch [47/60], time: 17.86s, loss: 1.9870, acc: 0.4888, val_loss: 1.8648, val_acc: 0.5295
Epoch [48/60], time: 18.07s, loss: 1.9705, acc: 0.4926, val_loss: 1.7838, val_acc: 0.5482
Epoch [49/60], time: 17.96s, loss: 1.9680, acc: 0.4918, val_loss: 1.7580, val_acc: 0.5482
Epoch [50/60], time: 18.03s, loss: 1.9568, acc: 0.4974, val_loss: 1.7907, val_acc: 0.5501
Epoch [51/60], time: 18.01s, loss: 1.9508, acc: 0.4974, val_loss: 1.9167, val_acc: 0.5301
Epoch [52/60], time: 17.90s, loss: 1.9380, acc: 0.5000, val_loss: 1.7492, val_acc: 0.5477
Epoch [53/60], time: 17.93s, loss: 1.9347, acc: 0.5019, val_loss: 1.7747, val_acc: 0.5559
Epoch [54/60], time: 18.00s, loss: 1.9133, acc: 0.5039, val_loss: 1.7831, val_acc: 0.5505
Epoch [55/60], time: 17.91s, loss: 1.9070, acc: 0.5070, val_loss: 1.7596, val_acc: 0.5567
Epoch [56/60], time: 17.76s, loss: 1.9003, acc: 0.5079, val_loss: 1.8217, val_acc: 0.5481
Epoch [57/60], time: 17.94s, loss: 1.8899, acc: 0.5133, val_loss: 1.7886, val_acc: 0.5507
Epoch [58/60], time: 17.83s, loss: 1.8804, acc: 0.5151, val_loss: 1.7513, val_acc: 0.5557
Epoch [59/60], time: 17.89s, loss: 1.8800, acc: 0.5151, val_loss: 1.7580, val_acc: 0.5594
Epoch [60/60], time: 17.91s, loss: 1.8719, acc: 0.5159, val_loss: 1.7597, val_acc: 0.5601
```

In [36]:

```python
# ResNet18

class BasicBlock(nn.Module): # resnet的一个残差学习模块
    def __init__(self, in_channel, out_channel, stride):
        super(BasicBlock, self).__init__()
        # 每个模块内有两层卷积+两层批量归一化
        self.conv1 = nn.Conv2d(in_channel, out_channel, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(out_channel)
        self.conv2 = nn.Conv2d(out_channel, out_channel, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channel)
        # 如果stride为1，则不需要对张量进行调整
        self.shortcut = nn.Sequential()
        if stride != 1:
            self.shortcut = nn.Sequential(
                # 通过尺寸为1的卷积核，将输入尺寸匹配，得以相加
                nn.Conv2d(in_channel, out_channel, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(out_channel)
            )

    def forward(self, x):
        out = self.conv1(x) # 经过第一层卷积
        out = self.bn1(out) # 归一化
        out = F.relu(out) # relu
        out = self.conv2(out) # 第二层卷积
        out = self.bn2(out) # 归一化
        out += self.shortcut(x) # F(x) + x
        out = F.relu(out) # relu
        return out
```

In [37]:

```python
class ResNet18(nn.Module):
    def __init__(self, block, num_blocks, num_classes=100):
        super(ResNet18, self).__init__()
        # 设定初始的in_channel，之后会更新
        self.in_channel = 64
        # 3-64层的same卷积，批量归一化，ReLU激活
        self.pre_conv = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU()
        )
        # 第一个block，64-64，尺寸不变
        self.layer1 = self._make_layer(block, 64, num_blocks[0], stride=1)
        # 第二个block，64-128，尺寸减半
        self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)
        # 第三个block，128-256，尺寸减半
        self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)
        # 第四个block，256-512，尺寸减半
        self.layer4 = self._make_layer(block, 512, num_blocks[3], stride=2)

        self.fc = nn.Linear(512, num_classes)

    def _make_layer(self, block, channels, num_blocks, stride): # 用于生成神经网络层
        strides = [stride] + [1] * (num_blocks - 1) # 除了第一组卷积步长可以为2，后续全为1
        layers = [] # 用于存储层的信息
        for i in strides:
            # layer里添加一个block，输入层数为in_channel，输出层数为channels，步长为i给定
            layers.append(block(self.in_channel, channels, i))
            # 将输入层数改为上一block的输出层数，当作下一block的输入
            self.in_channel = channels
```

```
            # 用*提取layer的全部信息，用nn.Sequential包装起来
            return nn.Sequential(*layers)

    def forward(self, x):
        out = self.pre_conv(x) # 预卷积，变成64层
        out = self.layer1(out) # 从上至下一致经过四个block
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = F.avg_pool2d(out, 4) # 用一个和输出尺寸大小一样的average pooling，将输出变成1x1大小
        out = out.view(out.size(0), -1) # 拉直
        out = self.fc(out) # 全连接，分类
        return out
```

In [38]:

```
resnet18_model = ResNet18(BasicBlock, [2, 2, 2, 2]).cuda() # block由BasicBlock组成，每个block含有两个BasicBlock
summary(resnet18_model, (3, IMSIZE, IMSIZE))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 32, 32]           1,728
       BatchNorm2d-2           [-1, 64, 32, 32]             128
              ReLU-3           [-1, 64, 32, 32]               0
            Conv2d-4           [-1, 64, 32, 32]          36,864
       BatchNorm2d-5           [-1, 64, 32, 32]             128
            Conv2d-6           [-1, 64, 32, 32]          36,864
       BatchNorm2d-7           [-1, 64, 32, 32]             128
        BasicBlock-8           [-1, 64, 32, 32]               0
            Conv2d-9          [-1, 128, 16, 16]          73,728
      BatchNorm2d-10          [-1, 128, 16, 16]             256
           Conv2d-11          [-1, 128, 16, 16]         147,456
      BatchNorm2d-12          [-1, 128, 16, 16]             256
           Conv2d-13          [-1, 128, 16, 16]           8,192
      BatchNorm2d-14          [-1, 128, 16, 16]             256
       BasicBlock-15          [-1, 128, 16, 16]               0
           Conv2d-16            [-1, 256, 8, 8]         294,912
      BatchNorm2d-17            [-1, 256, 8, 8]             512
           Conv2d-18            [-1, 256, 8, 8]         589,824
      BatchNorm2d-19            [-1, 256, 8, 8]             512
           Conv2d-20            [-1, 256, 8, 8]          32,768
      BatchNorm2d-21            [-1, 256, 8, 8]             512
       BasicBlock-22            [-1, 256, 8, 8]               0
           Conv2d-23            [-1, 512, 4, 4]       1,179,648
      BatchNorm2d-24            [-1, 512, 4, 4]           1,024
           Conv2d-25            [-1, 512, 4, 4]       2,359,296
      BatchNorm2d-26            [-1, 512, 4, 4]           1,024
           Conv2d-27            [-1, 512, 4, 4]         131,072
      BatchNorm2d-28            [-1, 512, 4, 4]           1,024
       BasicBlock-29            [-1, 512, 4, 4]               0
           Linear-30                  [-1, 100]          51,300
================================================================
Total params: 4,949,412
Trainable params: 4,949,412
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 7.06
Params size (MB): 18.88
Estimated Total Size (MB): 25.96
----------------------------------------------------------------
```

In [57]:

```
# 对resnet18模型进行训练
lr2 = 1e-4
optimizer2 = torch.optim.Adam(resnet18_model.parameters(), lr=lr2)
epochs = 20
train_result2 = train(resnet18_model, optimizer2, train_loader, val_loader, epochs=epochs)
```

```
Epoch [1/20], time: 19.25s, loss: 1.9965, acc: 0.5047, val_loss: 1.7664, val_acc: 0.5370
Epoch [2/20], time: 19.13s, loss: 1.4601, acc: 0.6136, val_loss: 1.8332, val_acc: 0.5662
Epoch [3/20], time: 19.31s, loss: 1.2589, acc: 0.6650, val_loss: 1.9422, val_acc: 0.5617
Epoch [4/20], time: 19.33s, loss: 1.1971, acc: 0.6805, val_loss: 1.9654, val_acc: 0.5606
Epoch [5/20], time: 19.38s, loss: 1.1489, acc: 0.6906, val_loss: 2.1156, val_acc: 0.5600
Epoch [6/20], time: 19.36s, loss: 1.1204, acc: 0.6975, val_loss: 2.1329, val_acc: 0.5622
Epoch [7/20], time: 19.36s, loss: 1.0983, acc: 0.7046, val_loss: 2.0989, val_acc: 0.5657
Epoch [8/20], time: 19.47s, loss: 1.0742, acc: 0.7069, val_loss: 2.2327, val_acc: 0.5575
Epoch [9/20], time: 19.32s, loss: 1.0763, acc: 0.7081, val_loss: 2.2316, val_acc: 0.5603
Epoch [10/20], time: 19.29s, loss: 1.0599, acc: 0.7134, val_loss: 2.2421, val_acc: 0.5613
Epoch [11/20], time: 19.23s, loss: 1.0427, acc: 0.7193, val_loss: 2.2357, val_acc: 0.5615
Epoch [12/20], time: 19.31s, loss: 1.0334, acc: 0.7206, val_loss: 2.2595, val_acc: 0.5660
Epoch [13/20], time: 19.28s, loss: 1.0212, acc: 0.7253, val_loss: 2.3225, val_acc: 0.5629
Epoch [14/20], time: 19.38s, loss: 1.0128, acc: 0.7242, val_loss: 2.3289, val_acc: 0.5580
Epoch [15/20], time: 19.30s, loss: 1.0097, acc: 0.7264, val_loss: 2.3598, val_acc: 0.5610
Epoch [16/20], time: 19.32s, loss: 1.0075, acc: 0.7250, val_loss: 2.3300, val_acc: 0.5603
Epoch [17/20], time: 19.13s, loss: 0.9914, acc: 0.7317, val_loss: 2.3587, val_acc: 0.5668
Epoch [18/20], time: 19.23s, loss: 1.0014, acc: 0.7285, val_loss: 2.3620, val_acc: 0.5676
Epoch [19/20], time: 19.25s, loss: 0.9884, acc: 0.7325, val_loss: 2.3354, val_acc: 0.5663
Epoch [20/20], time: 19.29s, loss: 0.9688, acc: 0.7390, val_loss: 2.4522, val_acc: 0.5669
```

最高精度56.69%

In [49]:

```
from torchvision.models import resnet18
transferred_resnet = torchvision.models.resnet18(pretrained=True) # 导入预训练好的resnet18模型
transferred_resnet.fc = nn.Linear(transferred_resnet.fc.in_features, 100) # 将全连接层与100分类连接
```

```
Downloading: "https://download.pytorch.org/models/resnet18-5c106cde.pth" to /root/.cache/torch/hub/checkpoints/resnet18-5c106cde.pth
```

In [50]:
```
transferred_resnet = transferred_resnet.cuda()
summary(transferred_resnet, (3, IMSIZE, IMSIZE))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 16, 16]           9,408
       BatchNorm2d-2           [-1, 64, 16, 16]             128
              ReLU-3           [-1, 64, 16, 16]               0
         MaxPool2d-4             [-1, 64, 8, 8]               0
            Conv2d-5             [-1, 64, 8, 8]          36,864
       BatchNorm2d-6             [-1, 64, 8, 8]             128
              ReLU-7             [-1, 64, 8, 8]               0
            Conv2d-8             [-1, 64, 8, 8]          36,864
       BatchNorm2d-9             [-1, 64, 8, 8]             128
             ReLU-10             [-1, 64, 8, 8]               0
       BasicBlock-11             [-1, 64, 8, 8]               0
           Conv2d-12             [-1, 64, 8, 8]          36,864
      BatchNorm2d-13             [-1, 64, 8, 8]             128
             ReLU-14             [-1, 64, 8, 8]               0
           Conv2d-15             [-1, 64, 8, 8]          36,864
      BatchNorm2d-16             [-1, 64, 8, 8]             128
             ReLU-17             [-1, 64, 8, 8]               0
       BasicBlock-18             [-1, 64, 8, 8]               0
           Conv2d-19            [-1, 128, 4, 4]          73,728
      BatchNorm2d-20            [-1, 128, 4, 4]             256
             ReLU-21            [-1, 128, 4, 4]               0
           Conv2d-22            [-1, 128, 4, 4]         147,456
      BatchNorm2d-23            [-1, 128, 4, 4]             256
           Conv2d-24            [-1, 128, 4, 4]           8,192
      BatchNorm2d-25            [-1, 128, 4, 4]             256
             ReLU-26            [-1, 128, 4, 4]               0
       BasicBlock-27            [-1, 128, 4, 4]               0
           Conv2d-28            [-1, 128, 4, 4]         147,456
      BatchNorm2d-29            [-1, 128, 4, 4]             256
             ReLU-30            [-1, 128, 4, 4]               0
           Conv2d-31            [-1, 128, 4, 4]         147,456
      BatchNorm2d-32            [-1, 128, 4, 4]             256
             ReLU-33            [-1, 128, 4, 4]               0
       BasicBlock-34            [-1, 128, 4, 4]               0
           Conv2d-35            [-1, 256, 2, 2]         294,912
      BatchNorm2d-36            [-1, 256, 2, 2]             512
             ReLU-37            [-1, 256, 2, 2]               0
           Conv2d-38            [-1, 256, 2, 2]         589,824
      BatchNorm2d-39            [-1, 256, 2, 2]             512
           Conv2d-40            [-1, 256, 2, 2]          32,768
      BatchNorm2d-41            [-1, 256, 2, 2]             512
             ReLU-42            [-1, 256, 2, 2]               0
       BasicBlock-43            [-1, 256, 2, 2]               0
           Conv2d-44            [-1, 256, 2, 2]         589,824
      BatchNorm2d-45            [-1, 256, 2, 2]             512
             ReLU-46            [-1, 256, 2, 2]               0
           Conv2d-47            [-1, 256, 2, 2]         589,824
      BatchNorm2d-48            [-1, 256, 2, 2]             512
             ReLU-49            [-1, 256, 2, 2]               0
       BasicBlock-50            [-1, 256, 2, 2]               0
           Conv2d-51            [-1, 512, 1, 1]       1,179,648
      BatchNorm2d-52            [-1, 512, 1, 1]           1,024
             ReLU-53            [-1, 512, 1, 1]               0
           Conv2d-54            [-1, 512, 1, 1]       2,359,296
      BatchNorm2d-55            [-1, 512, 1, 1]           1,024
           Conv2d-56            [-1, 512, 1, 1]         131,072
      BatchNorm2d-57            [-1, 512, 1, 1]           1,024
             ReLU-58            [-1, 512, 1, 1]               0
       BasicBlock-59            [-1, 512, 1, 1]               0
           Conv2d-60            [-1, 512, 1, 1]       2,359,296
      BatchNorm2d-61            [-1, 512, 1, 1]           1,024
             ReLU-62            [-1, 512, 1, 1]               0
           Conv2d-63            [-1, 512, 1, 1]       2,359,296
      BatchNorm2d-64            [-1, 512, 1, 1]           1,024
             ReLU-65            [-1, 512, 1, 1]               0
       BasicBlock-66            [-1, 512, 1, 1]               0
AdaptiveAvgPool2d-67            [-1, 512, 1, 1]               0
           Linear-68                  [-1, 100]          51,300
================================================================
Total params: 11,227,812
Trainable params: 11,227,812
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 1.29
Params size (MB): 42.83
Estimated Total Size (MB): 44.13
----------------------------------------------------------------
```

In [60]:
```
# 训练迁移的resnet18模型
lr3 = 1e-4
optimizer3 = torch.optim.Adam(transferred_resnet.parameters(), lr=lr3)
epochs = 50
train_result3 = train(transferred_resnet, optimizer3, train_loader, val_loader, epochs=epochs)
```

```
Epoch [1/50], time: 11.91s, loss: 2.3321, acc: 0.4007, val_loss: 1.9221, val_acc: 0.4929
Epoch [2/50], time: 11.95s, loss: 2.2968, acc: 0.4054, val_loss: 1.9869, val_acc: 0.4823
Epoch [3/50], time: 11.89s, loss: 2.2145, acc: 0.4197, val_loss: 2.0636, val_acc: 0.4746
Epoch [4/50], time: 11.83s, loss: 2.1650, acc: 0.4340, val_loss: 2.0003, val_acc: 0.4850
Epoch [5/50], time: 11.87s, loss: 2.1499, acc: 0.4388, val_loss: 1.9658, val_acc: 0.4916
Epoch [6/50], time: 11.84s, loss: 2.1221, acc: 0.4423, val_loss: 1.9210, val_acc: 0.4990
Epoch [7/50], time: 11.84s, loss: 2.1136, acc: 0.4463, val_loss: 2.0402, val_acc: 0.4809
Epoch [8/50], time: 11.89s, loss: 2.0904, acc: 0.4510, val_loss: 1.9503, val_acc: 0.5008
```

```
Epoch [9/50], time: 11.85s, loss: 2.0557, acc: 0.4602, val_loss: 1.9933, val_acc: 0.4942
Epoch [10/50], time: 11.90s, loss: 2.0438, acc: 0.4621, val_loss: 1.9729, val_acc: 0.4990
Epoch [11/50], time: 11.92s, loss: 2.0311, acc: 0.4624, val_loss: 2.0021, val_acc: 0.4955
Epoch [12/50], time: 11.83s, loss: 2.0146, acc: 0.4672, val_loss: 1.9706, val_acc: 0.5023
Epoch [13/50], time: 11.92s, loss: 1.9972, acc: 0.4701, val_loss: 1.9629, val_acc: 0.4994
Epoch [14/50], time: 11.78s, loss: 1.9917, acc: 0.4746, val_loss: 2.0209, val_acc: 0.4986
Epoch [15/50], time: 11.88s, loss: 1.9657, acc: 0.4798, val_loss: 1.9943, val_acc: 0.4979
Epoch [16/50], time: 12.01s, loss: 1.9617, acc: 0.4810, val_loss: 1.9556, val_acc: 0.5019
Epoch [17/50], time: 11.65s, loss: 1.9451, acc: 0.4815, val_loss: 2.0382, val_acc: 0.4892
Epoch [18/50], time: 12.13s, loss: 1.9188, acc: 0.4896, val_loss: 2.0189, val_acc: 0.5003
Epoch [19/50], time: 11.92s, loss: 1.9156, acc: 0.4937, val_loss: 2.0213, val_acc: 0.5007
Epoch [20/50], time: 12.03s, loss: 1.8980, acc: 0.4951, val_loss: 2.0446, val_acc: 0.4963
Epoch [21/50], time: 11.89s, loss: 1.8821, acc: 0.5008, val_loss: 2.0642, val_acc: 0.5012
Epoch [22/50], time: 11.93s, loss: 1.8677, acc: 0.5021, val_loss: 2.0016, val_acc: 0.5027
Epoch [23/50], time: 11.82s, loss: 1.8589, acc: 0.5046, val_loss: 2.0131, val_acc: 0.5080
Epoch [24/50], time: 11.83s, loss: 1.8487, acc: 0.5064, val_loss: 2.0942, val_acc: 0.4963
Epoch [25/50], time: 11.93s, loss: 1.8435, acc: 0.5112, val_loss: 2.0172, val_acc: 0.5071
Epoch [26/50], time: 12.03s, loss: 1.8208, acc: 0.5134, val_loss: 2.0165, val_acc: 0.5081
Epoch [27/50], time: 11.82s, loss: 1.8254, acc: 0.5140, val_loss: 2.0550, val_acc: 0.4966
Epoch [28/50], time: 11.72s, loss: 1.8044, acc: 0.5174, val_loss: 2.0797, val_acc: 0.5031
Epoch [29/50], time: 11.84s, loss: 1.7906, acc: 0.5207, val_loss: 2.0768, val_acc: 0.4997
Epoch [30/50], time: 11.76s, loss: 1.7771, acc: 0.5250, val_loss: 2.0481, val_acc: 0.5082
Epoch [31/50], time: 11.79s, loss: 1.7697, acc: 0.5256, val_loss: 2.0775, val_acc: 0.5050
Epoch [32/50], time: 11.97s, loss: 1.7582, acc: 0.5285, val_loss: 2.0966, val_acc: 0.5088
Epoch [33/50], time: 11.83s, loss: 1.7525, acc: 0.5319, val_loss: 2.1001, val_acc: 0.5005
Epoch [34/50], time: 11.74s, loss: 1.7408, acc: 0.5329, val_loss: 2.0950, val_acc: 0.5095
Epoch [35/50], time: 12.02s, loss: 1.7047, acc: 0.5418, val_loss: 2.0919, val_acc: 0.5082
Epoch [36/50], time: 11.90s, loss: 1.7026, acc: 0.5444, val_loss: 2.0785, val_acc: 0.5149
Epoch [37/50], time: 11.92s, loss: 1.7081, acc: 0.5416, val_loss: 2.1019, val_acc: 0.5078
Epoch [38/50], time: 11.90s, loss: 1.6940, acc: 0.5463, val_loss: 2.1577, val_acc: 0.5103
Epoch [39/50], time: 11.97s, loss: 1.6836, acc: 0.5482, val_loss: 2.1493, val_acc: 0.5113
Epoch [40/50], time: 11.90s, loss: 1.6754, acc: 0.5487, val_loss: 2.1555, val_acc: 0.5114
Epoch [41/50], time: 11.84s, loss: 1.6835, acc: 0.5482, val_loss: 2.1741, val_acc: 0.5073
Epoch [42/50], time: 11.83s, loss: 1.6733, acc: 0.5502, val_loss: 2.1823, val_acc: 0.5080
Epoch [43/50], time: 11.75s, loss: 1.6596, acc: 0.5557, val_loss: 2.1601, val_acc: 0.5116
Epoch [44/50], time: 12.02s, loss: 1.6517, acc: 0.5574, val_loss: 2.2525, val_acc: 0.5039
Epoch [45/50], time: 11.97s, loss: 1.6422, acc: 0.5591, val_loss: 2.1960, val_acc: 0.5021
Epoch [46/50], time: 11.94s, loss: 1.6229, acc: 0.5637, val_loss: 2.1760, val_acc: 0.5075
Epoch [47/50], time: 11.86s, loss: 1.6173, acc: 0.5651, val_loss: 2.2224, val_acc: 0.5095
Epoch [48/50], time: 11.78s, loss: 1.6090, acc: 0.5643, val_loss: 2.3064, val_acc: 0.4957
Epoch [49/50], time: 11.73s, loss: 1.6016, acc: 0.5699, val_loss: 2.2338, val_acc: 0.5058
Epoch [50/50], time: 12.03s, loss: 1.5903, acc: 0.5720, val_loss: 2.2627, val_acc: 0.5111
```

In [61]:

```python
# 绘制子图
from matplotlib import pyplot as plt
fig,ax = plt.subplots(1,2)     # 一行两列
fig.set_figwidth(15)           # 宽度为 15
fig.set_figheight(6)           # 高度为 6

ax[0].grid(linestyle = 'dashed')                                # 添加网格线
ax[0].plot(torch.arange(1, len(train_result1[0])+1), train_result1[0])    # 绘制vgg的实验结果
ax[0].plot(torch.arange(1, len(train_result2[0])+1), train_result2[0])    # 绘制resnet18的实验结果
ax[0].plot(torch.arange(1, len(train_result3[0])+1), train_result3[0])    # 绘制迁移的resnet18的实验结果
ax[0].set_xlabel("Epochs", fontsize = 15)                       # 添加 x-label文字
ax[0].set_ylabel("Loss", fontsize = 15)                         # 添加 y-label文字
ax[0].set_title("Train", fontsize = 15)                         # 添加标题
ax[0].legend(labels = ['VGG19', 'ResNet18', 'Transferred ResNet18'])            # 添加图例

ax[1].grid(linestyle = 'dashed')
ax[1].plot(torch.arange(1, len(train_result1[3])+1), train_result1[3])
ax[1].plot(torch.arange(1, len(train_result2[3])+1), train_result2[3])
ax[1].plot(torch.arange(1, len(train_result3[3])+1), train_result3[3])
ax[1].set_xlabel("Epochs", fontsize = 15)
ax[1].set_ylabel("Accuracy", fontsize = 15)
ax[1].set_title("Validation", fontsize = 15)
ax[1].legend(labels = ['VGG19', 'ResNet18', 'Transferred ResNet18'])
```

Out[61]: <matplotlib.legend.Legend at 0x7f3b8cf096d0>