西北工業大學

NORTHWESTERN POLYTECHNICAL UNIVERSITY

基于 Shufflenet 神经网络的飞机图像识别

课程名称：	Python 与机器学习
学　　院：	航空学院
专　　业：	飞行器控制与信息工程
学　　号：	2020300408
姓　　名：	齐明超
日　　期：	2022 年 11 月 27 日

目录

一、 项目开发目的

2022 年 11 月 8 日至 11 月 13 日，我国在珠海举办了中国国际航空航天博览会，简称珠海航展，在本次航展上我国展示了运 20、歼 10B、歼 20 等多款机型，其中部分机型人们虽然知道型号，但无法根据外形快速辨别。为了帮助人们快速根据飞机外形辨别出飞机型号，加深人们对于飞机的了解，我开发了基于 shufflenet 深度学习神经网络的飞机图像识别算法。

当然，本算法不仅局限于仅用于本次航展中的飞机识别，也可以用于对世界范围内常见的战斗机进行识别，可以在日常生活中帮助人们更快的辨别出飞机型号，具有一定的科普意义。更进一步，可以结合 Rcs 之类的成像技术，对空中的飞机进行探测识别。

本次项目完成的是对于七种常见战斗机歼 20、F22、F35、阵风战斗机，台风战斗机，幻影 2000，苏 57 进行的图像识别。

二、项目运行环境

Python3.10.8
Pytorch1.13.0
CUDA11.7.99

```
pygments                  2.11.2          pyhd3eb1b0_0
pyjwt                     2.4.0           py310haa95532_0
pyopenssl                 22.0.0          pyhd3eb1b0_0
pyparsing                 3.0.9           py310haa95532_0
pyqt                      5.15.7          py310hd77b12b_0
pyqt5-sip                 12.11.0         py310hd77b12b_0
pyrsistent                0.18.0          py310h2bbff1b_0
pysocks                   1.7.1           py310haa95532_0
python                    3.10.8              hbb2ffb3_0
python-dateutil           2.8.2           pyhd3eb1b0_0
python-fastjsonschema     2.16.2          py310haa95532_0
python-flatbuffers        2.0             pyhd3eb1b0_0
pytorch                   1.13.0          py3.10_cuda11.7_cudnn8_0    pytorch
pytorch-cuda              11.7                 h67b0de4_0    pytorch
pytorch-mutex             1.0                      cuda      pytorch
pytz                      2022.6                 pypi_0      pypi
pywin32                   302             py310h2bbff1b_2
pywinpty                  2.0.2           py310h5da7b33_0
pyyaml                    6.0             py310h2bbff1b_0
pyzmq                     23.2.0          py310hd77b12b_0
qt-main                   5.15.2               he8e5bd7_7
qt-webengine              5.15.9               hb9a9bb5_4
qtwebkit                  5.212                h3ad3cdb_4
requests                  2.28.1          py310haa95532_0
requests-oauthlib         1.3.0                    py_0
rsa                       4.7.2           pyhd3eb1b0_1
scikit-learn              1.1.3                  pypi_0      pypi
scipy                     1.7.3           py310h6d2d95c_2
send2trash                1.8.0           pyhd3eb1b0_1
setuptools                65.5.0          py310haa95532_0
```

三、Shufflenet 深度神经网络基本原理

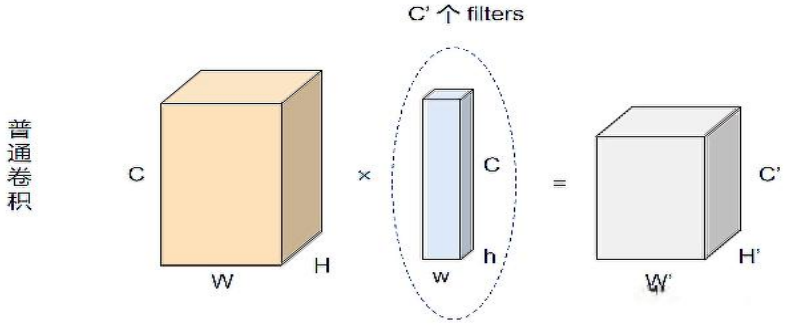
Shufflenet 是在 Resnet 神经网络上的一种改进，号称是可以在移动设备上运行的深度神经网络，主要采用 **pointwise group convolutions（逐点分组卷积）**、**channel shuffle（通道重排）**和 **depthwise separable convolution（深度可分离卷积）**，这在保持精度的同时大大降低了模型的计算量。下面分别介绍这三种操作：

3.1pointwise group convolutions（逐点分组卷积）

pointwise group convolutions 即卷积核为 1×1 的分组卷积

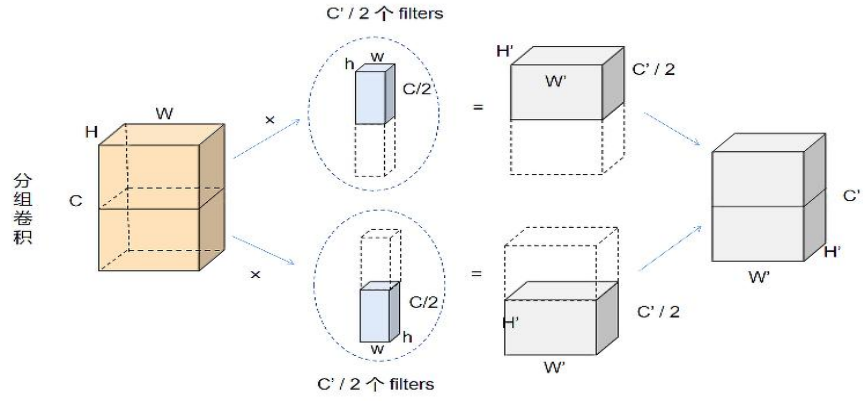
3.1.1 普通 2D 卷积和分组卷积的区别

标准的 2D 卷积步骤如下图所示，输入特征为 (H×W×C)，然后应用 C’ 个 filters[每个 filter 的大小为(h×w×c)]，输入层被转换为大小为(H’ ×W’×C’)的输出特征。



分组卷积的表示如下图(下图表示的是被拆分为 2 个 filters 组的分组卷积）：

- 首先每个 filters 组,包含 C’/2 个数量的 filter,每个 filter 的通道数为传统 2D 卷积 filter 的一半。
- 每个 filters 组作用于原来 W×H×C 对应通道的一半，也就是 W×H×C/2。
- 最终每个 filters 组对应输出 C/2 个通道的特征。
- 最后将通道堆叠得到了最终 C 个通道，实现了和上述标准 2D 卷积一样的效果。



3.1.2 分组卷积运算量的减少

假设 input.shape=[C,H,W]，output.shape=[C1,H1,W1]

输入每组 feature map 尺寸：W×H×C/g，共有 g 组；

单个卷积核每组的尺寸：k×k×C1/g；

输出 feature map 尺寸：W1×H1×g，共生成 g 个 feature map

常规卷积运算量：k×k×C×C1×W1×H1；

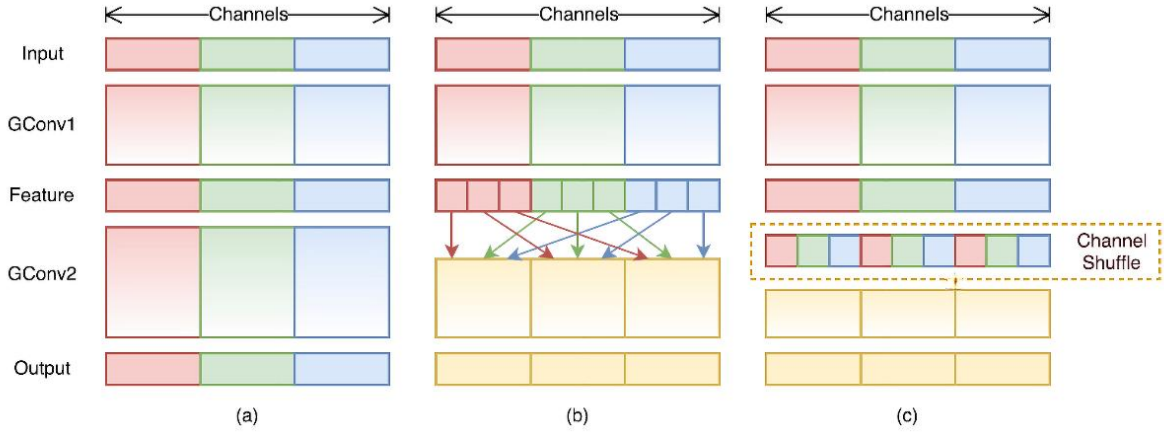
分组卷积运算量：k×k×C/g×C1×W1×H1；

结论：通过上面的计算我们可以很清楚的看到，得到相同的 output.shape，分组卷积的运算量是普通卷积运算量的 1/g

3.2.channel shuffle（通道重排）

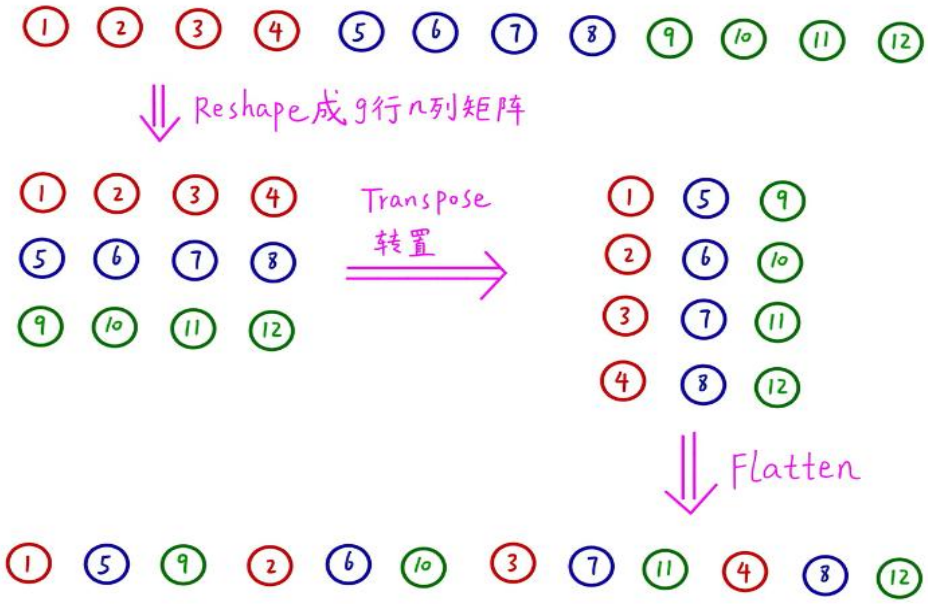
虽然分组卷积可以有效的减少运算量，但是也会带来一定的弊端：分组卷积每个 filter 不再是和输入的全部 feature map 做卷积，而是和一个 group 的 feature map 做卷积，即只会在组内进行卷积，因此组和组之间不存在信息的交互，就会产生边界效应，即某个输出 channel 仅仅来自输入 channel 的一小部分，学出来的特征会非常局限。

为了提高组与组之间的信息交流，shufflenet 深度神经网络采用 channel shufflenet（通道混洗）的方法：先看（b），在进行 GConv2 之前，对其输入 feature map 做一个分配，也就是每个 group 分成几个 subgroup，然后将不同 group 的 subgroup 作为 GConv2 的一个 group 的输入，使得 GConv2 的每一个 group 都能卷积输入的所有 group 的 feature map，（c）图和（b）图表达的是同一个意思。



通道重排具体过程：

下图展示了分 3 组进行分组卷积后的通道重排的具体过程



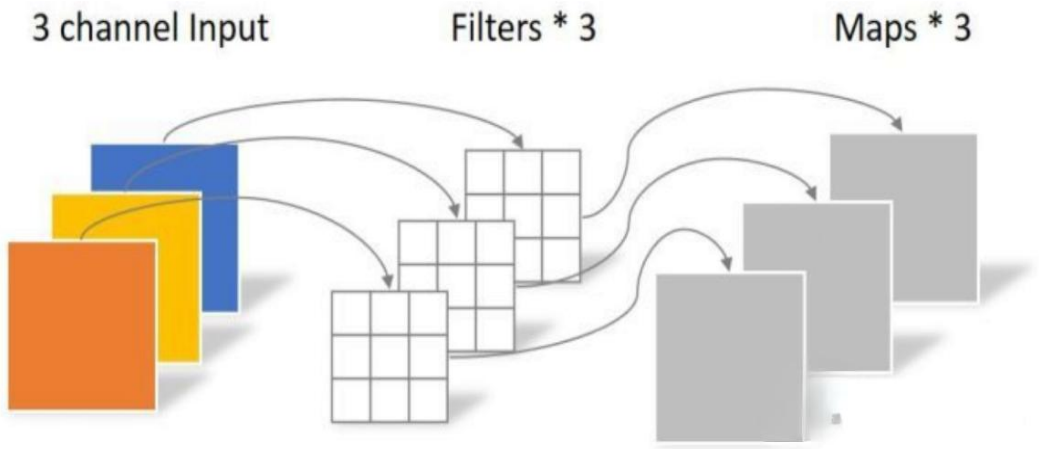
3.3 Depthwise separable convolution（深度可分离卷积）

深度可分离卷积主要分为两个过程，分别为逐通道卷积（Depthwise Convolution）和逐点卷积（Pointwise Convolution）。

- **逐通道卷积（Depthwise Convolution）**

Depthwise Convolution 的一个卷积核负责一个通道，一个通道只被一个卷积核卷积，这个过程产生的 feature map 通道数和输入的通道数完全一样。

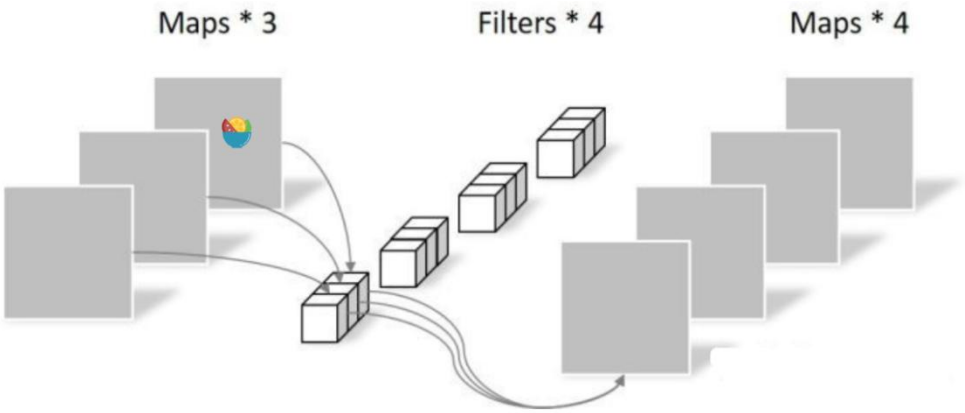
一张 5×5 像素、三通道彩色输入图片（shape 为 5×5×3），Depthwise Convolution 首先经过第一次卷积运算，DW 完全是在二维平面内进行。卷积核的数量与上一层的通道数相同（通道和卷积核一一对应）。所以一个三通道的图像经过运算后生成了 3 个 Feature map(如果有 same padding 则尺寸与输入层相同为 5×5)，如下图所示。(卷积核的 shape 即为：卷积核 W×卷积核 H ×输入通道数)。



Depthwise Convolution 完成后的 Feature map 数量与输入层的通道数相同，无法扩展 Feature map。而且这种运算对输入层的每个通道独立进行卷积运算，没有有效的利用不同通道在相同空间位置上的 feature 信息。因此需要 Pointwise Convolution 来将这些 Feature map 进行组合生成新的 Feature map。

- **逐点卷积（Pointwise Convolution）**

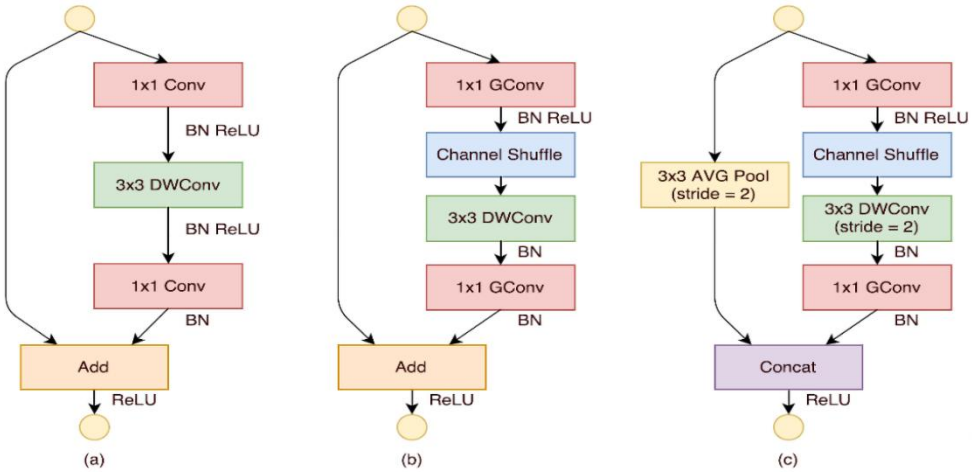
Pointwise Convolution 的运算与常规卷积运算非常相似，它的卷积核的尺寸为 1×1×M，M 为上一层的通道数。所以这里的卷积运算会将上一步的 map 在深度方向上进行加权组合，生成新的 Feature map。有几个卷积核就有几个输出 Feature map。（卷积核的 shape 即为：1 ×1 ×输入通道数×输出通道数）。



3.4 Shufflenet 深度神经网络结构

在 ResNeXt 中主要是对 3*3 的卷积做 group 操作，**但是在 ShuffleNet 中是对 1*1 的卷积做 group 的操作，因为 1*1 的卷积操作的计算量不可忽视**，原论文中论述 1x1 的 PW 卷积层占了 93.4%的计算量，因此用卷积核为 1×1 的分组卷积替代卷积核为 1×1 的普通卷积，并且在分组之后进行了 channel shuffle 操作。

下图(c)就是一个 shufflenet 块，图(a)是一个简单的残差连接块，区别在于，shufflenet 将残差连接改为了一个平均池化的操作与卷积操作之后做 **Concat，即按 channel 合并**，类似 googleNet 的 Inception 操作。



四、飞机图像分类的代码实现

4.1 数据的收集

4.1.1 图片的爬取

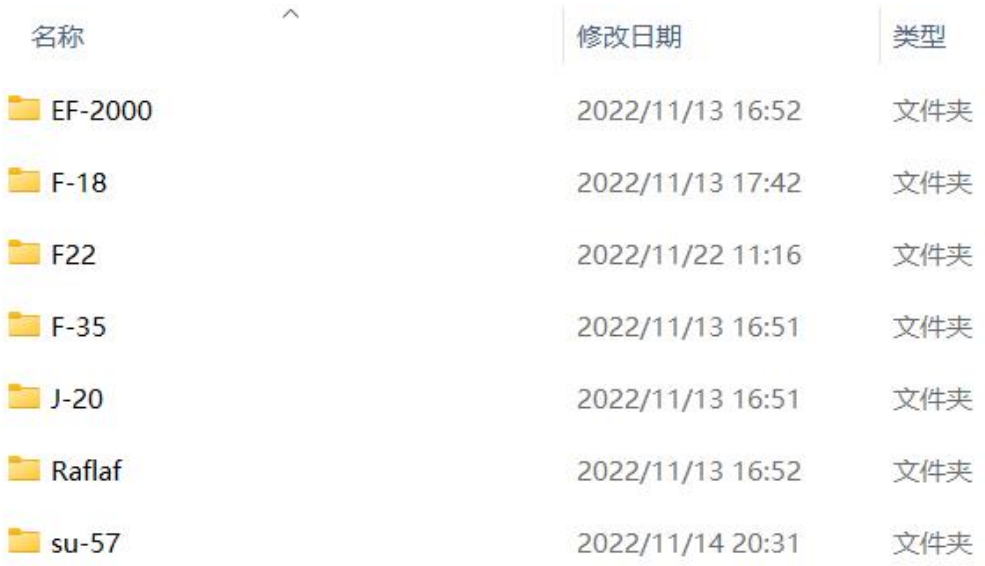
为实现七种战斗机的图像识别任务，并且为保证有较高的识别正确率，需要有较为完整的数据集，首先需要收集这七种战斗机的图片，然后人为的对这些图像进分类，即使用不同的文件名对图像进区分，最简单的办法就是将不同类别归入不同文件夹，例如建立一个“J-20”文件夹存入所有歼 20 相关图片。

由于需要收集的图片数量较大每个类别大约 600 张图片，逐个下载较为耗费时间，所以使用爬虫算法从百度图片、微软图片等资源批量爬取图片，并保存在指定的文件夹中，部分代码如图所示：

```
54  def loadpic(number, page):
55      """
56      :param number:
57      :param page:
58      :return:
59      """
60      while (True):
61          if number == 0:
62              break
63          url, params = configs(search, page, number)
64          result = requests.get(url, headers=header, params=params).json()
65          url_list = []
66          for data in result['data'][:-1]:
67              url_list.append(data['thumbURL'])
68          for i in range(len(url_list)):
69              getImg(url_list[i], 60 * page + i, path)#
70              bar.update(1)
71              number -= 1
72              if number == 0:
73                  break
74          page += 1
75      print("\nfinish!")
76
```

```
92  number = ['歼20']
93  #需要搜索的关键词
94  for number_subset in number:
95      if __name__ == '__main__':
96          search = number_subset
97          number = 800
98          #需要爬取图片的数量
99          path = 'D:/PYTHON/airplane/J-20/' + number_subset + 1
100         #图片爬取后保存的文件夹位置
101         header = {
102             'User-Agent':
103             'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome
104         }
105
106         bar = tqdm.tqdm(total=number)
107         page = 0
108         loadpic(number, page)
109
```

分别切换不同的搜索关键词和搜索路径实现不同种类飞机图片的爬取和保存，爬取后效果图如图所示：



名称	修改日期	类型
EF-2000	2022/11/13 16:52	文件夹
F-18	2022/11/13 17:42	文件夹
F22	2022/11/22 11:16	文件夹
F-35	2022/11/13 16:51	文件夹
J-20	2022/11/13 16:51	文件夹
Raflaf	2022/11/13 16:52	文件夹
su-57	2022/11/14 20:31	文件夹

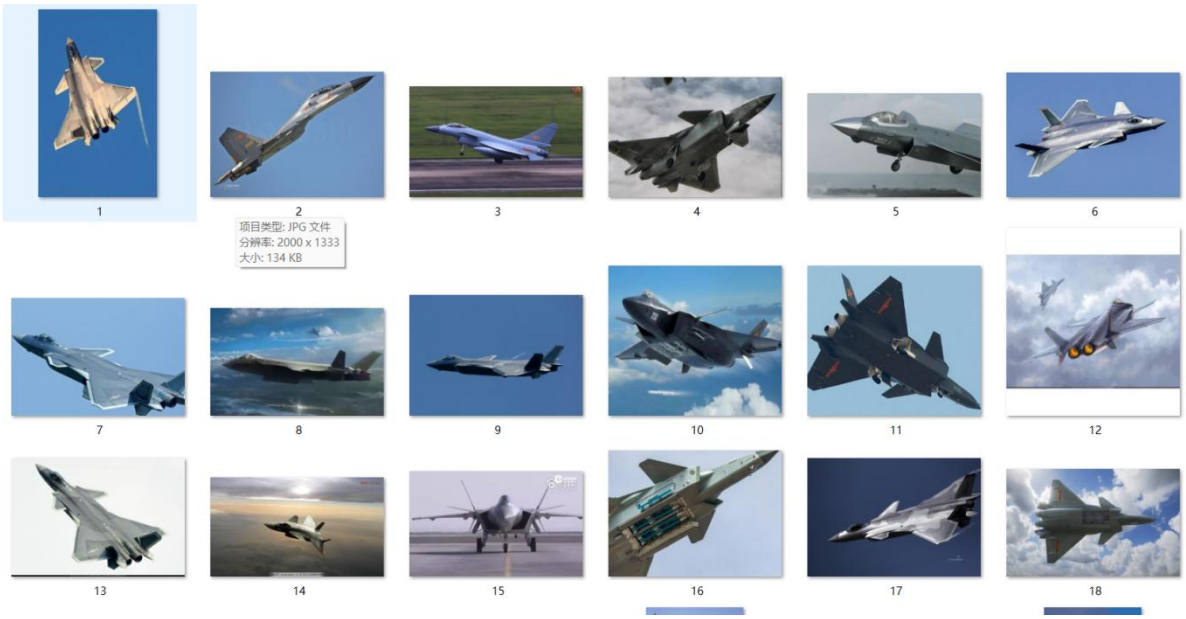
4.1.2 图片重命名

为了统一规范图片的格式，需要对已经爬取的图片进行图片格式的统一，这里我将其统一转化为 jpg 格式，并且在每个文件夹下对图片按照从 1 开始的顺序依次命名。代码如下图所示：

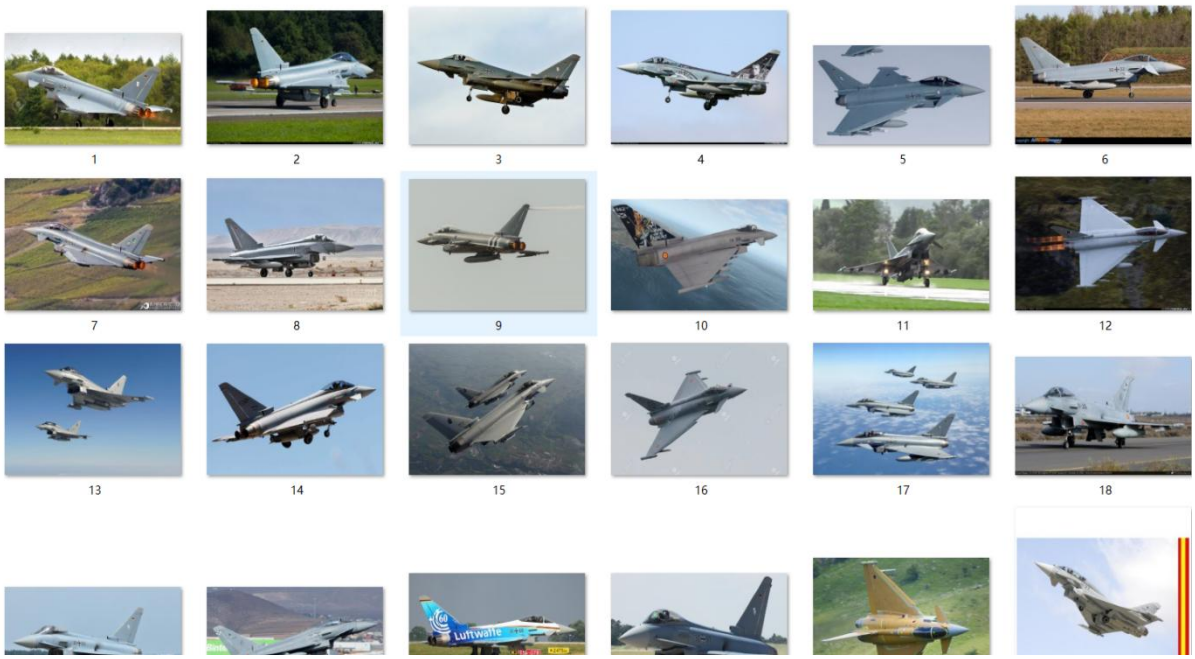
```python
#批量修改文件名，默认操作为将图片按1，2，3，，，顺序重命名
import os

path_in = 'D:/PYTHON/plane_1/su-57'  #待批量重命名的文件夹
class_name = ".jpg"  #重命名后的文件名后缀

file_in = os.listdir(path_in)  #返回文件夹包含的所有文件名
num_file_in = len(file_in)  #获取文件数目
print(file_in, num_file_in)  #输出修改前的文件名

for i in range(0, num_file_in):
    t = str(i + 1)
    new_name = os.rename(path_in + "/" + file_in[i],
                         path_in + "/" + t + class_name)
    #重命名文件名

file_out = os.listdir(path_in)
print(file_out)  #输出修改后的结果

```

文件格式统一化和重命名后效果如图所示：

歼 20：



EF-2000：



4.2 数据集的处理

4.2.1 Dataset 类的重写

```python
#重写pytorch中Dataset类，继承的抽象类Dataset
class MyDataSet(Dataset):
    def __init__(self, images_path: list, images_class: list, transform=None):
        self.images_path = images_path
        self.images_class = images_class
        self.transform = transform

    def __len__(self):
        return len(self.images_path)

    def __getitem__(self, item):
        # 根据索引item从文件中读取一个数据
        # 对数据预处理
        # 返回数据和对应标签
        img = Image.open(self.images_path[item])
        #使用爬虫从百度爬取下来的图片不一定全部为RGB图片，而我在model里定义的shufflenet网络输入为三通道RGB图片
        # 即input_channels = 3，在第一次运行程序时发生报错
        # 故在这里用一个if判断语句判断数据集中图片是否是RGB图片，如果不是，则对其进行转换
        if img.mode != 'RGB':
            img = img.convert('RGB')
        label = self.images_class[item]

        if self.transform is not None:
            img = self.transform(img)

        return img, label

```

其中有两个比较重要的函数：

- __getitem__是真正读取数据的地方，迭代器通过索引来读取数据集中的数据。实现了三个功能：

1. 根据索引 item 从文件中读取一个数据；
2. 对数据进行预处理。

在这个过程中，由于数据集是自己通过爬虫在百度图片上爬取得到，不能保证图片均为 RGB 图片，所以对其进行判断，如果不是，就将其转化为 RGB 图片。

```python
        if img.mode != 'RGB':
            img = img.convert('RGB')
```

在这里我并没有对 transform 方式做出规定，而是在 train.py 文件中在开始训练前定义了 transform 方式，并且对训练集和验证集使用了不同的 transform 方式：(各个 transform 方式具体的作用在注释中均已注明，如图)

对训练集：

```python
transforms.Compose([
    transforms.RandomResizedCrop(
        224),  #将给定图像随机裁剪为不同的大小和宽高比，然后缩放所裁剪得到的图像为制定的大小
    transforms.RandomHorizontalFlip(),  #以给定的概率随机水平旋转给定的PIL的图像，默认为0.5
    transforms.ToTensor(),#很重要的一步，将图像数据转为Tensor
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225])  # 归一化处理
]),
```

对验证集：

```
"val":
transforms.Compose([
    transforms.Resize(256),#重新设定图像大小
    transforms.CenterCrop(224),#从图像中心开始裁剪图像，224为裁剪大小
    transforms.ToTensor(),#很重要的一步，将图像数据转为Tensor
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]) # 归一化处理
])
```

3. 返回数据及对应的标签

- __len__提供这个迭代器的范围

4.2.2 训练集和验证集的划分

由于在收集数据集的时候没有在每个飞机类别对应的文件夹下再细分用于训练和用于验证的图片，所以需要对数据集进行划分，根据图片路径和索引信息进行划分，用于训练与用于验证的比例为 8：2。

首先创建空列表对图片路径和索引信息进行存储：

```
65      train_images_path = []  # 存储训练集的所有图片路径
66      train_images_label = []  # 存储训练集图片对应索引信息
67      val_images_path = []  # 存储验证集的所有图片路径
68      val_images_label = []  # 存储验证集图片对应索引信息
69      every_class_num = []  # 存储每个类别的样本总数
```

然后对文件夹中的文件路径进行遍历，获得该类别对应的索引，记录该类别的数量，最终按比例随机采样用于验证集的样本，代码中的 val_rate=0.2：

```
72      for cla in airplane_class:
73          cla_path = os.path.join(root, cla)
74          # 遍历获取supported支持的所有文件路径
75          images = [
76              os.path.join(root, cla, i) for i in os.listdir(cla_path)
77              if os.path.splitext(i)[-1] in supported
78          ]
79          # 排序，保证各平台顺序一致
80          images.sort()
81          # 获取该类别对应的索引
82          image_class = class_indices[cla]
83          # 记录该类别的样本数量
84          every_class_num.append(len(images))
85          # 按比例随机采样验证样本，这里的val_rate=0.2
86          val_path = random.sample(images, k=int(len(images) * val_rate))
87
```

由于上述是按比例随机采样验证样本，而不是将数据集前后批量按比例划分为训练集和验证集，因此为了确定将哪些数据存放入验证集，需要一个 for 循环来遍历图片，确定该图片是否属于上述按比例随机采样出用于验证集的图片，剩下的图片则用于训练集。

```
88          for img_path in images:
89              if img_path in val_path:
90              # 如果该路径在采样的验证集样本中则存入验证集
91                  val_images_path.append(img_path)
92                  val_images_label.append(image_class)
93              else:
94              # 否则存入训练集
95                  train_images_path.append(img_path)
96                  train_images_label.append(image_class)
97
```

划分结果：

```
PS D:\shufflenet - 副本> conda activate pytorch
PS D:\shufflenet - 副本> & E:/Anaconda/envs/pytorch/python.exe "d:/shufflenet - 副本/train.
Namespace(num_classes=7, epochs=10, batch_size=32, lr=0.01, lrf=0.1, data_path='D:/PYTHON/p
da:0')
数据集中共有3853张图片
其中3085张图片用于训练
768张图片用于验证
Using 8 dataloader workers every process
```

4.3 Shufflenet 深度神经网络的搭建

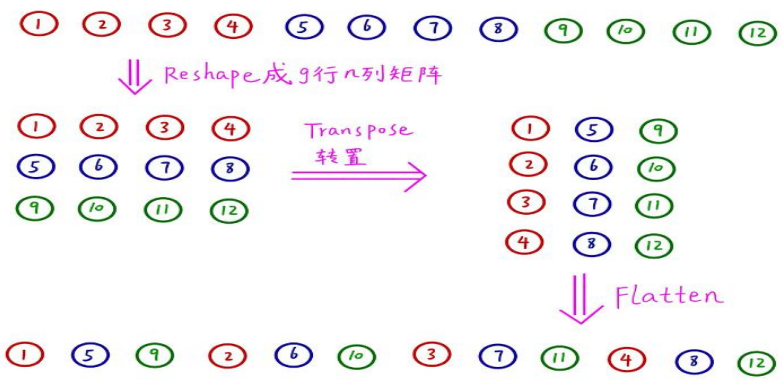
4.3.1 Channel Shuffle 的实现

Channel shuffle 操作可以看成“重塑-转置-重塑”的过程。

假定将输入层分为 g 组，总通道数为 g×n，首先将通道那个维度拆分为(g,n) 两个维度，然后将这两个维度转置变成 (n,g) ，最后重新 reshape 成一个维度 g × n。

在代码中实现的方法是：(结合手绘原理图)

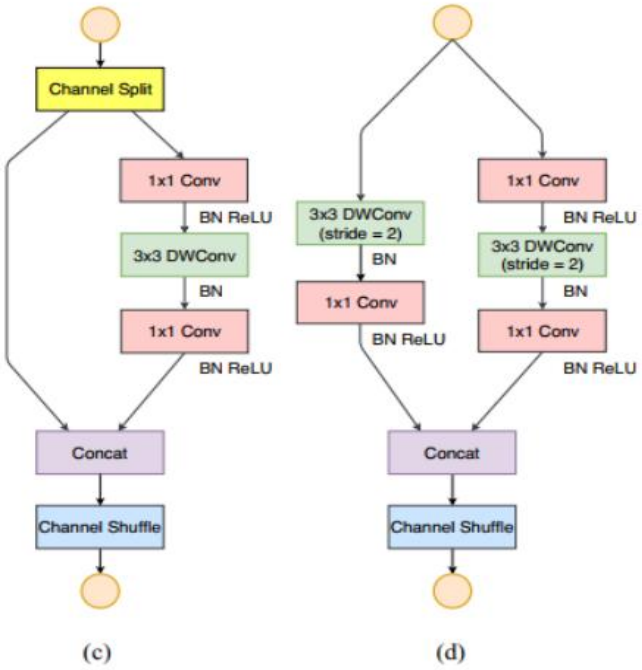
1) 通过 size()方法获取 feature map 的 size,在 pytorch 获取到的 tensor 排列顺序为[batch_size,channels,特征矩阵的高度，特征矩阵的宽度]
2) 对分组卷积结果每个组内的通道进行划分
3) 通过 veiw()方法重新定义 tensor 的维度,类似于 numy 库中的 reshape()
4) 对上述 tensor 中的维度 1 和维度 2 进行互换，类似于矩阵的转置
5) 通过 veiw()方法对经过维度互换后的 tensor 还原



```python
def channel_shuffle(x: Tensor, groups: int) -> Tensor:
#实现Shufflenet最重要的思想channel shuffle（通道重排）操作
    batch_size, num_channels, height, width = x.size()
    #获取传入进函数的feature map的size
    #在pytorch中获取得到的tensor排列顺序为
    #[batch_size, channels, 特征矩阵高度, 特征矩阵宽度]
    channels_per_group = num_channels // groups
    #对分组卷积结果每个组内的通道进行划分

    x = x.view(batch_size, groups, channels_per_group, height, width)
    #通过view()方法重新定义tensor的维度
    #[batch_size, num_channels, height, width] -> [batch_size, groups, channels_per_group, height, width]
    x = torch.transpose(x, 1, 2).contiguous()
    #通过transpose()方法对维度1和维度2进行互换，可认为是矩阵的转置

    # flatten
    x = x.view(batch_size, -1, height, width)

    return x
```

4.3.2 Block 的搭建

Shufflenet V2 网络 Block 的结构如下图所示：



(c) (d)

1) 首先判断 DWConv 的步长是否为 1 或 2，因为在上图 Shufflenet V2 网络中步长 stride 只能等于 1 或 2，图(c)是 DW 卷积步长为 1 的情况；图二是 DW 卷积步长为 2 的情况。

```python
        if stride not in [1, 2]:
            raise ValueError("illegal stride value.")
        self.stride = stride
        #判断步长是否等于1或2，因为在shufflenet网络中步长只能为1或2
```

2) 在 shufflenet V2 网络 block 中在 concat 拼接前左右分支通道数相等，因此需要判断输出特征矩阵通道数是否为 2 的整数倍，而且每个分支通道数=输出通道数/2，当 stride=1 时，input channel 需要经过 channel spilt 操作，所以此时 input channel=branch_features×2。

```
        assert output_c % 2 == 0
        #判断输出特征矩阵通道数是否为2的整数倍，因为在shufflenet V2网络block中
        #在concat拼接前左右分支通道数相等
        branch_features = output_c // 2
        #当stride为1时，input_channel应该是branch_features的两倍
        #因为当stride=1时，input_channel需要经过channel spilt操作
        assert (self.stride != 1) or (input_c == branch_features << 1)
```

3) 对右边分支(branch1)模型进行搭建。

根据原理图可以看到 stride=1 时，对右边分支没有做任何处理，所以 nn.Sequential()不添加任何模块；当 stride=2 时，按照(d)图顺序将模块依次添加到 nn.Sequential()中。（注意通道数：注释中已经注明）

```
        if self.stride == 2:  #步长为2
            self.branch1 = nn.Sequential(
                #branch1代表示意图中左边的分支
                self.depthwise_conv(
                    input_c,
                    input_c,  #因为DW卷积输出和输出特征矩阵通道数相同
                    kernel_s=3,
                    stride=self.stride,
                    padding=1),
                nn.BatchNorm2d(input_c),  #DW卷积后feature map通道数=输入通道数
                nn.Conv2d(input_c,
                          branch_features,
                          kernel_size=1,
                          stride=1,
                          padding=0,
                          bias=False),
                nn.BatchNorm2d(branch_features),
                nn.ReLU(inplace=True))
        else:
            self.branch1 = nn.Sequential()
        #stride=1时，对左边的分支没有做任何处理
```

这里面用了一个比较重要的函数 nn.Sequential()：

一个序列容器，用于搭建神经网络的模块被按照被传入构造器的顺序添加到 nn.Sequential()容器中。利用 nn.Sequential()搭建好模型架构，模型前向传播时调用 forward()方法，模型接收的输入首先被传入 Sequential()包含的第一个网络模块中。然后，第一个网络模块的输出传入第二个网络模块作为输入，按照顺序依次计算并传播，直到最后一个模块输出结果。

4) 对左边分支(branch2)模型进行搭建。

与过程三基本类似，但是(c)(d)两种情况 DW 卷积步长不同，做 DW 卷积时需要进行一下判断。

```
nn.Conv2d(
    input_c if self.stride > 1 else branch_features,
    #不论是stride=1或2，右边分支结构基本相同，不同的只是DW卷积的步长，在这里判断一下
    branch_features,
    kernel_size=1,
    stride=1,
    padding=0,
    bias=False),
```

5）前向传播

在 3）中已经介绍过用 nn.Sequential()搭建好模型架构可以接收输入后各模块依次计算并传播。

但需要注意的当 stride=1 时，需要对 input channel 进行 channel split 操作，因此调用 chunk()方法进行均分操作。

Torch.cat()实现的式 concat 拼接功能。

```
111     def forward(self, x: Tensor) -> Tensor:
112         if self.stride == 1:
113             x1, x2 = x.chunk(2, dim=1)
114             #当stride=1时，需要对input channel进行Channel split
115             out = torch.cat((x1, self.branch2(x2)), dim=1)
116             #实现concat拼接功能
117         else:
118             out = torch.cat((self.branch1(x), self.branch2(x)), dim=1)
119 
120         out = channel_shuffle(out, 2)
121 
122         return out
```

4.3.3 完整网络结构的搭建

原论文中完整的网络结构如下表所示，下面我将根据这张表对 Shufflenet V2 完整的结构进行搭建。

Layer	Output size	KSize	Stride	Repeat	Output channels			
					0.5×	1×	1.5×	2×
Image	224×224				3	3	3	3
Conv1	112×112	3×3	2	1	24	24	24	24
MaxPool	56×56	3×3	2					
Stage2	28×28		2	1	48	116	176	244
	28×28		1	3				
Stage3	14×14		2	1	96	232	352	488
	14×14		1	7				
Stage4	7×7		2	1	192	464	704	976
	7×7		1	3				
Conv5	7×7	1×1	1	1	1024	1024	1024	2048
GlobalPool	1×1	7×7						
FC					1000	1000	1000	1000
FLOPs					41M	146M	299M	591M
# of Weights					1.4M	2.3M	3.5M	7.4M

1) 根据图片定义各个需要输入的参数：

stages_repeats 代表每层结构需要的重复次数

stages_out_channels 代表每层结构的输出通道数

num_classes 代表进行分类的数目，我这里对七种飞机进行分类

inverted_residual 这里使用我上面搭建的 Block

```
126     def __init__(self,
127                  stages_repeats: List[int],#代表每层结构的重复次数
128                  stages_out_channels: List[int],#代表每层结构的输出通道数
129                  num_classes: int = 7,#代表进行分类的数目，我这里对七种飞机进行分类
130                  inverted_residual: Callable[...,
131                                              nn.Module] = InvertedResidual):
132         super(ShuffleNetV2, self).__init__()
133
```

另外，还需要对输入的参数进行检查，确定其是否符合模型需要输入的参数个数。根据上方结构图，可以看到只有在 stage2、stage3、stage4 才有卷积层的重复，因此输入到 stages_repeats 中参数长度应该为 3；同时，该网络结构输入到 stages_out_channels 参数长度应该是 5。

```
134         if len(stages_repeats) != 3:
135             raise ValueError(
136                 "expected stages_repeats as list of 3 positive ints")
137         if len(stages_out_channels) != 5:
138             raise ValueError(
139                 "expected stages_out_channels as list of 5 positive ints")
140         self._stage_out_channels = stages_out_channels
141
```

2) 第一层结构 Conv1 和 Maxpool 的搭建

首先确定输入 tensor 的通道数，因为在数据预处理的时候，我将数据集中的图片已经全部转化为 RGB 图片，故 input_channels=3；output channels 为需要人工输入的列表中的第一个变量，即 channels[0]。

```
143         # 输入的数据集图片均为RGB图片，RGB格式图片有3个channels
144         #输出的通道数为人工输入变量的第一个，即_stage_out_channels[0]
145         input_channels = 3
146         output_channels = self._stage_out_channels[0]
147
148         self.conv1 = nn.Sequential(
149             nn.Conv2d(input_channels,
150                       output_channels,
151                       kernel_size=3,
152                       stride=2,
153                       padding=1,
154                       bias=False), nn.BatchNorm2d(output_channels),
155             nn.ReLU(inplace=True))
156         input_channels = output_channels
157
158         self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
```

注：第 156 行代码表示下一层的输入通道数等于 Conv1 层的输出通道数。

3) 通过循环遍历一次性完成搭建 stage2、stage3、stage4。

观察网络结构表我们可以发现，stage2、stage3、stage4 具有很大的相似性，都是先使用一个 stride=2 的 Block，在重复使用三个 staride=1 的 Block，因此可以利用双循环来一次性搭建好这 3 个 stage。(相关细节标注在注释中)

```
        stage_names = ["stage{}".format(i) for i in [2, 3, 4]]
        for name, repeats, output_channels in zip(
                stage_names, stages_repeats, self._stage_out_channels[1:]):
        #同时遍历stage_names, stages_repeats, self._stage_out_channels
            seq = [inverted_residual(input_channels, output_channels, 2)]
        #对于每个stage层结构首先使用的都是stride=2的Block
        #每个stage进行完上面步骤后都会重复使用三个stride=1的Block，故使用循环
            for i in range(repeats - 1):
                seq.append(
                    inverted_residual(output_channels, output_channels, 1))
                #在搭建Block时已经说明当stride=1，input channels=output channels
            setattr(self, name, nn.Sequential(*seq))
            input_channels = output_channels
            #下一个stage的输入通道数等于上一个stage的输出通道数
```

4) 前向传播

剩余两层 Conv5 和 Conv1 搭建思路和实现代码基本完全一致;通过 nn.linear 方法实现全连接层 fc。全局池化直接放在前向传播里使用 mean（）方法实现。

```
        output_channels = self._stage_out_channels[-1]
        self.conv5 = nn.Sequential(
            nn.Conv2d(input_channels,
                      output_channels,
                      kernel_size=1,
                      stride=1,
                      padding=0,
                      bias=False), nn.BatchNorm2d(output_channels),
            nn.ReLU(inplace=True))

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

    def _forward_impl(self, x: Tensor) -> Tensor:
        # See note [TorchScript super()]
        x = self.conv1(x)
        x = self.maxpool(x)
        x = self.stage2(x)
        x = self.stage3(x)
        x = self.stage4(x)
        x = self.conv5(x)
        x = x.mean([2, 3])  #全局池化
        x = self.fc(x)
        return x
```

5) 传入相关参数即可完全实现神经网络。这里我是用 2×版本对应的参数。

```
def shufflenet_v2_x2_0(num_classes=7):
    model = ShuffleNetV2(stages_repeats=[4, 8, 4],
                         stages_out_channels=[24, 244, 488, 976, 2048],
                         num_classes=num_classes)

    return model
```

4.4 飞机分类模型的训练

训练模型时用于计算的设备为 GPU，可以更加快速高效的进行计算，这里我用的是 GPU0，即“cuda:0”

```
16    device = torch.device(args.device if torch.cuda.is_available() else "cpu")
17    #选定用于计算的设备，这里我用的是GPU进行计算
```

```
180   parser.add_argument('--device',
181                       default='cuda:0',
182                       help='device id (i.e. 0 or 0,1 or cpu)')
183   #规定用于计算的设备，我电脑中有两个GPU，这里我选择用英伟达的GPU进行计算，即'cuda:0'
```

在 train.py 文件中我传入参数使用的是 argparse 模块，argparse 模块是 Python 内置的一个用于命令项选项与参数解析的模块，编写用户的命令行接口。通过在程序中定义好我们需要的参数，然后 argparse 将会从 sys.argv 解析出这些参数。

在这个模块中我还调用了 Tensorboard 库对训练过程进行了数据可视化，将训练过程中训练的精度、训练的损失率、验证精度和验证损失率进行记录并绘制各自对应的图像，于此同时我还对训练过程中学习率的变化进行了记录并绘制对应图像。

1) 首先使用 SummaryWriter()方法初始化

```
19    #数据可视化，利用tensorboard
20    #初始化 SummaryWriter，无参数，默认将使用 runs/日期时间 路径来保存日志
21    #在命令行输入tensorboard --logdir runs --bind_all调用(调用的时候注意cd到runs所在上一级目录中)
22    train_writer = SummaryWriter()
```

2) 进行训练时将每次训练的训练的精度、训练的损失率、验证精度和验证损失率保存。

```
146       #利用tensorboard实现数据可视化，使用 add_scalar 方法来记录数字常量
147       #tag (string): 数据名称，不同名称的数据使用不同曲线展示
148       train_writer.add_scalar('train_acc', train_epoch_loss, epoch)
149       train_writer.add_scalar('valid_acc', train_epoch_acc, epoch)
150       train_writer.add_scalar('valid_loss', valid_epoch_loss, epoch)
151       train_writer.add_scalar('valid_acc', valid_epoch_acc, epoch)
152       train_writer.add_scalar('optimizer', optimizer.param_groups[0]["lr"],
153                               epoch)
```

4.4.1 数据训练批次划分

梯度下降的每一轮迭代下，要计算所有数据的损失函数值，然后计算每个参数的偏导数，来求出目前的梯度方向，这样求解运算量较大，可以使用 mini-batch gradient descent（小批量梯度下降），把数据分成一个一个的 batch，然后每计算一个 batch，就更新一次梯度，这样可以加快训练的速度，还有模型收敛的速度。shufflenet

实现的方法是使用 torch.utils.data.DataLoader()方法进行划分，其中：

Dataset:数据读取接口，我分别设置为 train_dataset 和 val_dataset

Batch_size:批训练数据大小，这里我通过 argparse 模块设置为 32

```
166   parser.add_argument('--batch-size', type=int, default=32)
```

Shufflenet:是否打乱数据，在训练集中设为 True，在验证集中设为 Flase

pin_memory:如果设置为 True，那么 data loader 将会在返回它们之前，将 tensors 拷贝到 CUDA 中的固定内存中。

collate_fun: 合并样本清单以形成小批量。用来处理不同情况下的输入 dataset 的封装。

完整代码如下：collate_fun

```
66    train_loader = torch.utils.data.DataLoader(
67        train_dataset,
68        batch_size=batch_size,
69        shuffle=True,
70        pin_memory=True,
71        num_workers=nw,
72        collate_fn=train_dataset.collate_fn)
73
74    val_loader = torch.utils.data.DataLoader(val_dataset,
75                                             batch_size=batch_size,
76                                             shuffle=False,
77                                             pin_memory=True,
78                                             num_workers=nw,
79                                             collate_fn=val_dataset.collate_fn)
80
```

4.4.2 训练方法与验证方法的定义

训练方法的定义：

1）定义交叉熵损失函数

```
8        criterion = torch.nn.CrossEntropyLoss()
```

2）使用 model.train()，启用 batch normalization 和 dropout ，因为 Shufflenet 网络中有 BN 层（Batch Normalization），model.train() 可以保证 BN 层能够用到每一批数据的均值和方差。

```
11       model.train()
```

3）进行前向传播和反向传播计算梯度。

```
23           # 前向传播
24           outputs = model(image)
25           # 计算损失
26           loss = criterion(outputs, labels)
27           train_running_loss += loss.item()
28           # 计算准确率
29           preds = torch.argmax(outputs.data, 1)
30           train_running_correct += (preds == labels).sum().item()
31           # 反向传播
32           loss.backward()
33           # 更新参数
34           optimizer.step()
```

4）训练过程中准确率和损失率的计算。

```
    #计算每个epoch的loss和accuracy
    #训练损失=在训练集错误的图片数/训练的图片总数
    #训练精度=(训练正确的图片数/划分出的训练集中图片总数)*100%
    epoch_loss = train_running_loss / counter
    epoch_acc = 100. * (train_running_correct / len(train_loader.dataset))
    return epoch_loss, epoch_acc
```

验证方法的定义：

验证方法与训练方法定义过程基本一致，但是有两点不同

1）使用 model.eval()，不启用 Batch Normalization 和 Dropout，model.eval() 是保证 BN 层能够用 全部训练数据 的均值和方差，即测试过程中要保证 BN 层的均值和方差不变。

```
    model.eval()
```

2）在进行验证的过程中，不需要计算梯度，也不需要进行反向传播。

```
    with torch.no_grad():  # 在计算验证集时，不需要计算梯度,也不会进行反向传播
```

4.4.3 优化器 optimizer 的使用

模型训练过程主要包括正向计算、损失计算、误差反向传播、参数更新等几个步骤。实现参数更新需要高效的优化求解器，这里我选用的 SGD 优化器，即随机梯度下降。

```
    #优化器选择SGD优化器，即随机梯度下降
    optimizer = optim.SGD(pg, lr=args.lr, momentum=0.9, weight_decay=4E-5)
```

4.4.4 学习率衰减

较大的学习率，在算法优化的前期会加速学习，使得模型更容易接近局部或全局最优解。但是在后期会有较大波动，甚至出现损失函数的值围绕最小值徘徊，波动很大，始终难以达到最优。

所以需要使用学习率衰减，就是在模型训练初期，会使用较大的学习率进行模型优化，加快模型学习速度，随着迭代次数增加，学习率会逐渐进行减小，保证模型在训练后期不会有太大的波动，从而更加接近最优解。

这里我选用余弦衰减，表达式如下所示：

$$\alpha_t = 0.5\alpha_0\left[1+\cos\left(\frac{\pi t}{T}\right)\right]$$

其中，α_0 代表初始学习率，t 是指当前是第几个 epoch，T 是指多少个 epoch 之后学习率衰减为 0

```
lf = lambda x: ((1 + math.cos(x * math.pi / args.epochs)) / 2) * (
    1 - args.lrf) + args.lrf
```

4.4.5 预训练权重加载

在搭建完 Shufflenet 神经网络后，训练七种常见战斗机模型过程中，由于准备的数据集只有几千张图片，数据量比较小，从 0 开始训练，权值比较随机，特征提取效果不明显，网络在前几个 epoch 的 Loss 可能非常大，并且多次训练得到的训练结果相差很大。

为了提高模型训练的准确度，需要使用预训练权重，即是在大型基准数据集上训练的模型权重，用于解决相似的问题。

```
86      if args.weights != "":
87          if os.path.exists(args.weights):
88              weights_dict = torch.load(args.weights, map_location=device)
89              load_weights_dict = {
90                  k: v
91                  for k, v in weights_dict.items()
92                  if model.state_dict()[k].numel() == v.numel()
93              }
94              print(model.load_state_dict(load_weights_dict, strict=False))
95          else:
96              raise FileNotFoundError("not found weights file: {}".format(
97                  args.weights))
```

导入预训练权重路径：

```
179     parser.add_argument('--weights',
180                         type=str,
181                         default='./shufflenetv2_x2_0.pth',
182                         help='initial weights path')
```

4.5 应用模型进行飞机图像识别

模型训练完毕后需要应用模型对输入的图片中的飞机进行识别。

1) 首先需要加载指定路径的图片，并对其进行裁剪大小、转换为 tensor、进行归一化等。

Torch.unsqueeze()是进行拓展维度

```
    data_transform = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])

    # load image
    img_path = "D:/PYTHON/plane_1/F-35/19.jpg"
    assert os.path.exists(img_path), "file: '{}' dose not exist.".format(
        img_path)
    img = Image.open(img_path)
    plt.imshow(img)
    # [N, C, H, W]
    img = data_transform(img)
    # expand batch dimension
    img = torch.unsqueeze(img, dim=0)
```

2) 读取模型训练过程中生成的飞机分类类别文件

```
    json_path = './class_indices.json'
    assert os.path.exists(json_path), "file: '{}' dose not exist.".format(
        json_path)

    with open(json_path, "r") as f:
        class_indict = json.load(f)
```

3) 加载训练好的模型，利用 pytorch 中的 model.load_state_dict()方法读取模型字典。

```
    model_weight_path = "./weights/model-29.pth"
    model.load_state_dict(torch.load(model_weight_path, map_location=device))
    model.eval()
```

4) 对图片中飞机的类型进行预测并给出置信度，在这过程中不需要计算梯度

```
    with torch.no_grad():
        #在预测是不需要计算梯度
        output = torch.squeeze(model(img.to(device))).cpu()
        predict = torch.softmax(output, dim=0)
        predict_cla = torch.argmax(predict).numpy()

    print_res = "class: {}      proability: {:.3}".format(
        class_indict[str(predict_cla)], predict[predict_cla].numpy())
    plt.title(print_res)
    for i in range(len(predict)):
        print("class: {:10}     pr (variable) predict: Tensor
            class_indict[str(i)], predict[i].numpy()))
    plt.show()
```

五、结果展示与模型评价

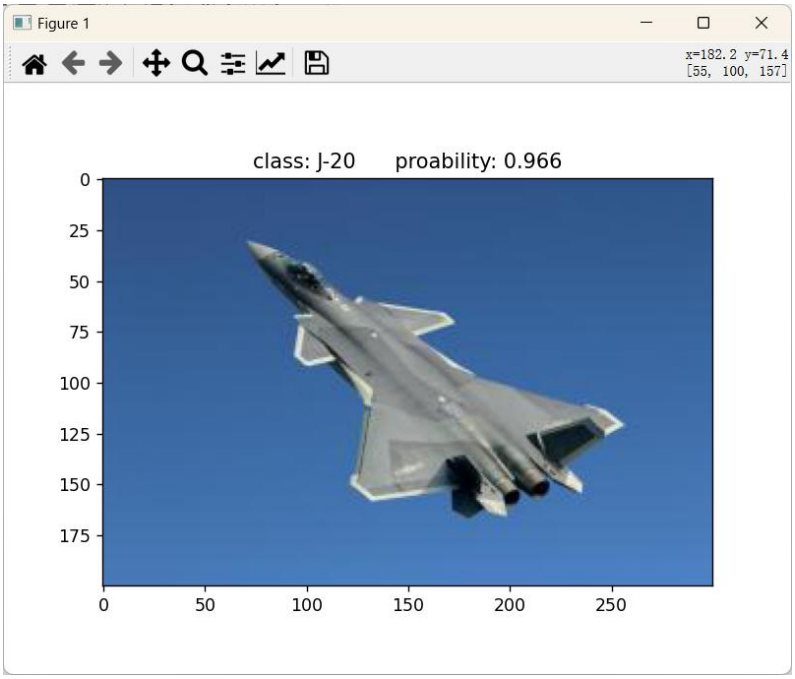
在这一部分首先展示利用训练好的模型对几个不同类别的战斗机分别进行识别效果，并给出相应的置信度，其次会应用 tensorboard 数据可视化、Grad CAM 技术绘制的神经网络特征提取的热力图等对搭建的神经网络学习效果进行评价。

5.1 模型预测结果展示

训练模型时生成的记录七种飞机类别的文件：

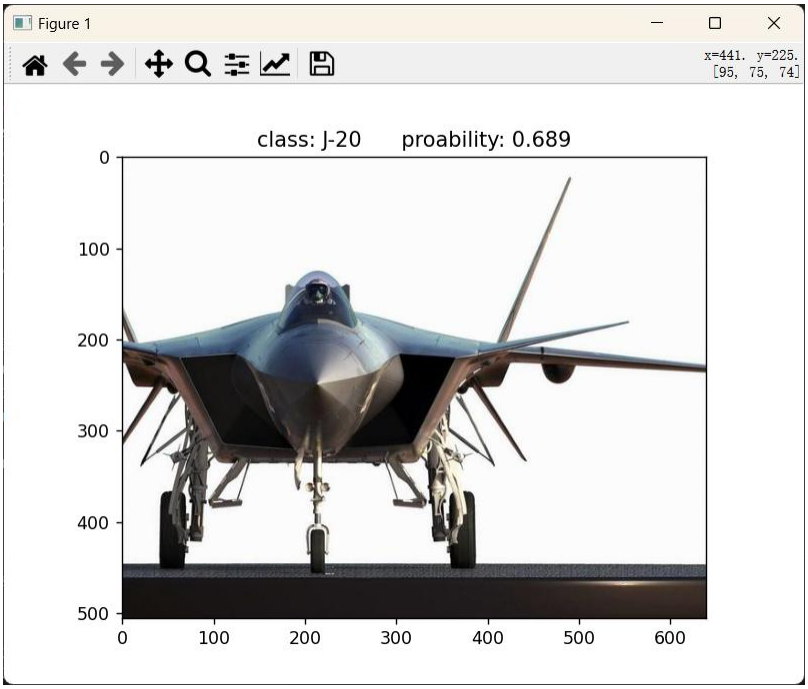
```
{} class_indices.json > ...
{
    "0": "EF-2000",
    "1": "F-18",
    "2": "F-35",
    "3": "F22",
    "4": "J-20",
    "5": "Raflaf",
    "6": "su-57"
}
```

对歼 20 的识别结果(1)：（全身）



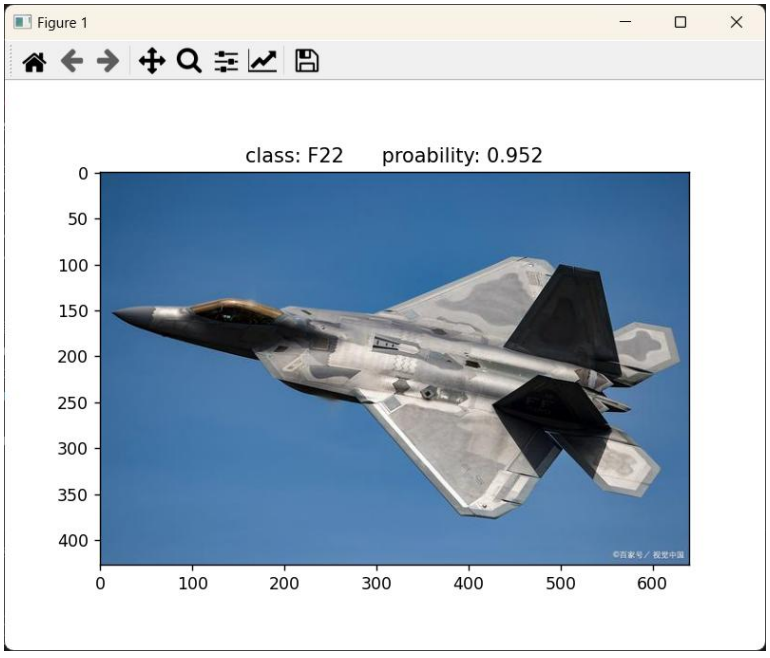

```
问题  输出  调试控制台  终端  JUPYTER

PS D:\shufflenet_2> conda activate pytorch
PS D:\shufflenet_2> & E:/Anaconda/envs/pytorch/python.exe d:/shufflenet_2/predict.py
class: EF-2000       probability: 0.000535
class: F-18          probability: 0.000835
class: F-35          probability: 0.00201
class: F22           probability: 0.0152
class: J-20          probability: 0.966
class: Raflaf        probability: 0.00122
class: su-57         probability: 0.0144
PS D:\shufflenet_2>
```

对歼 20 的识别结果(2)：（头部）



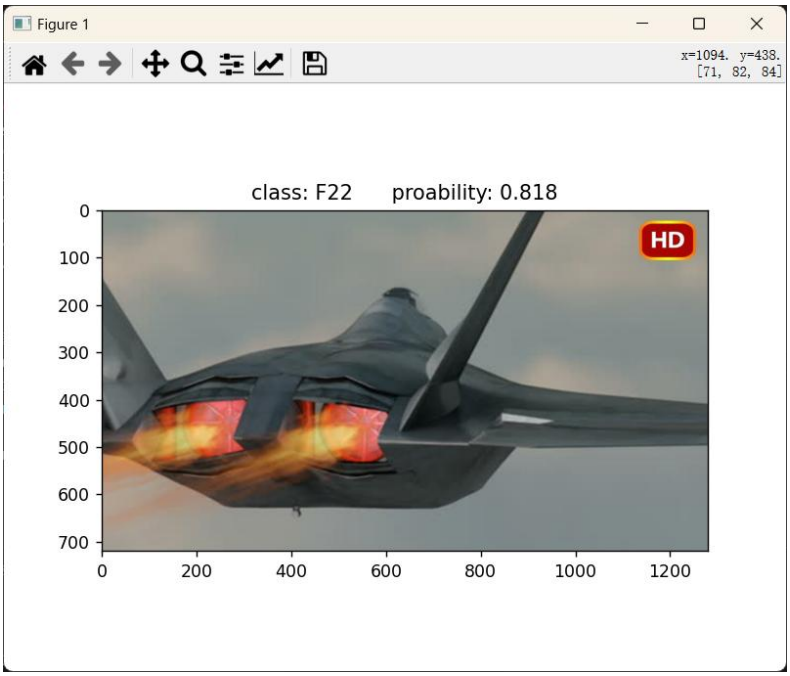

```
问题  输出  调试控制台  终端  JUPYTER

PS D:\shufflenet_2> conda activate pytorch
PS D:\shufflenet_2> & E:/Anaconda/envs/pytorch/python.exe d:/shufflenet_2/predict.py
class: EF-2000        probability: 0.0261
class: F-18           probability: 0.0278
class: F-35           probability: 0.0738
class: F22            probability: 0.166
class: J-20           probability: 0.689
class: Raflaf         probability: 0.0104
class: su-57          probability: 0.00708
PS D:\shufflenet_2> & E:/Anaconda/envs/pytorch/python.exe d:/shufflenet_2/predict.py
```

对 F22 的识别结果(1)：（全身）



```
问题  输出  调试控制台  终端  JUPYTER

PS D:\shufflenet_2> conda activate pytorch
PS D:\shufflenet_2> & E:/Anaconda/envs/pytorch/python.exe d:/shufflenet_2/predict.py
class: EF-2000        probability: 0.000169
class: F-18           probability: 0.00148
class: F-35           probability: 0.0325
class: F22            probability: 0.952
class: J-20           probability: 0.00416
class: Raflaf         probability: 0.000134
class: su-57          probability: 0.0098
PS D:\shufflenet_2>
```

对 F22 的识别结果(1)：（尾部）




```
问题  输出  调试控制台  终端  JUPYTER

PS D:\shufflenet_2> conda activate pytorch
PS D:\shufflenet_2> & E:/Anaconda/envs/pytorch/python.exe d:/shufflenet_2/predict.py
class: EF-2000        probability: 0.00412
class: F-18           probability: 0.0181
class: F-35           probability: 0.136
class: F22            probability: 0.818
class: J-20           probability: 0.00948
class: Raflaf         probability: 0.00238
class: su-57          probability: 0.0122
```

上面四幅图是我选取了选取了两张歼 20 和两张 F22 的图片进行预测的结果，歼 20 两张图片分别是全身和正面头部照片，F22 两张图片分别是全身和尾部的照片，可以看到预测结果全部正确，并且给出了较高的置信度。

从上述预测结果可以看到通过搭建的 Shufflenet V2 深度神经网络学习的模型对七种常见战斗机的特征进行了比较全面和准确的提取，能偶较好的完成对这七种飞机的识别分类任务。

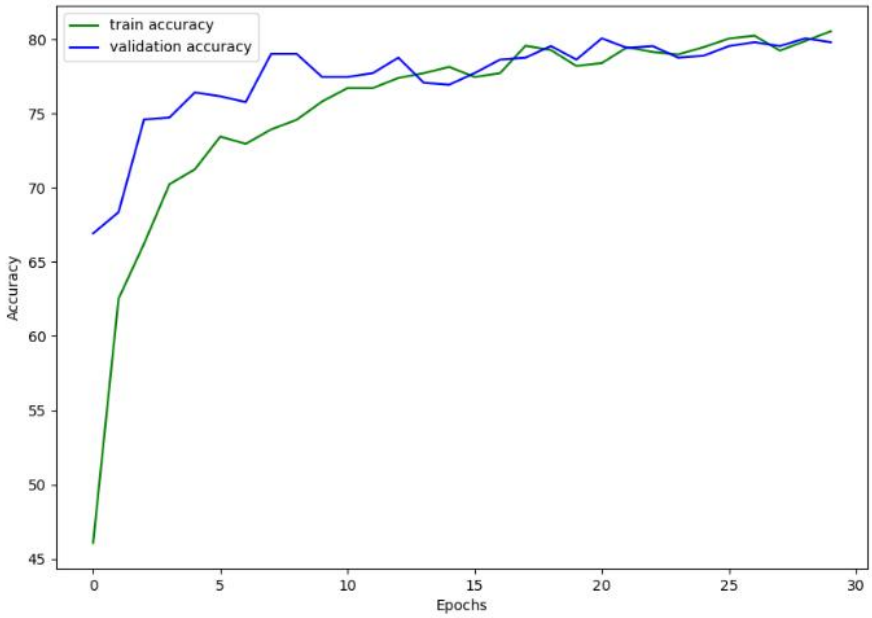
5.2 模型评价

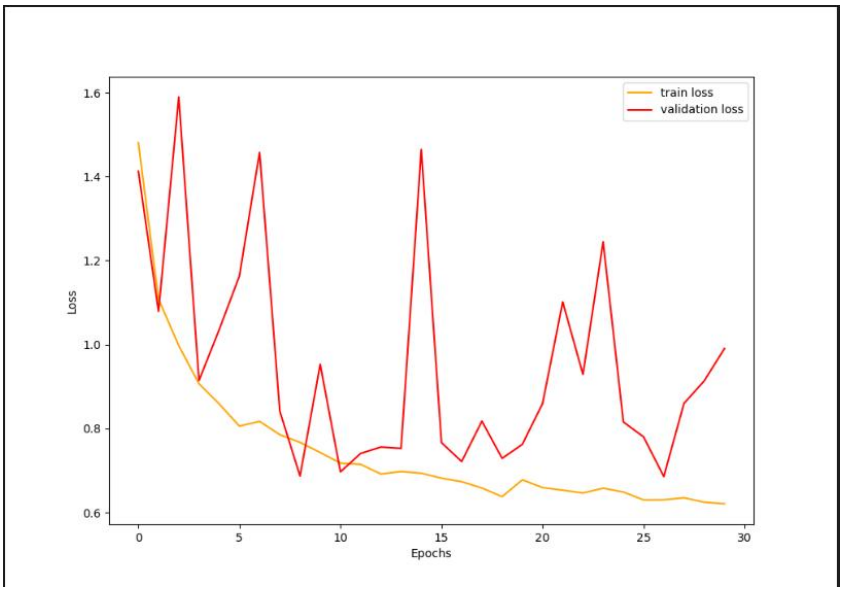
在 train.py 函数中，我将训练过程中每个 epoch 的训练精度，训练损失、验证精度、验证损失都存入指定列表中，并在 utils.py 文件中利用 matplotlib 绘制成对应的训练精度和训练损失图，保存至 outputs 文件夹中。

```python
        train_epoch_loss, train_epoch_acc = train_one_epoch(
            model=model,
            optimizer=optimizer,
            train_loader=train_loader,
            device=device)

        train_loss.append(train_epoch_loss)
        train_acc.append(train_epoch_acc)
        #将每一次的训练损失和训练精度存放到列表中
        print(
            f"Training loss: {train_epoch_loss:.3f}, training acc(%): {train_epoch_acc:.3f}"
        )
        #显示本次训练的训练损失和训练精度
        scheduler.step()

        #开始验证，这里直接调用在utils.py文件已经定义好的函数'validate'
        valid_epoch_loss, valid_epoch_acc = validate(model=model,
                                                     valid_loader=val_loader,
                                                     device=device)
        valid_loss.append(valid_epoch_loss)
        valid_acc.append(valid_epoch_acc)
```

```python
# 定义一个函数，用来保存和生成损失图和精度图
def save_plots(train_acc, valid_acc, train_loss, valid_loss):
    """
    将损失和准确度图保存
    """
    #生成精度曲线
    plt.figure(figsize=(10, 7))
    #规定生成图像的大小
    plt.plot(train_acc, color='green', linestyle='-', label='train accuracy')
    #规定训练精度曲线颜色为绿色，曲线样式为'-'，曲线标签为'train accuracy'
    plt.plot(valid_acc,
             color='blue',
             linestyle='-',
             label='validation accuracy')
    #规定训练精度曲线颜色为蓝色，曲线样式为'-'，曲线标签为'validation accuracy'
    plt.xlabel('Epochs')
    #规定横坐标为'Epochs'
    plt.ylabel('Accuracy')
    #规定纵坐标为'Accuracy'
    plt.legend()
    plt.savefig('outputs/accuracy.png')
```

运行结果为：






```
第29次训练
Training
100%|                                                                                | 97/97
Training loss: 0.628, training acc(%): 80.324
Validation
100%|                                                                                | 24/24
Validation loss: 1.559, validation acc(%): 80.208
第30次训练
Training
100%|                                                                                | 97/97
Training loss: 0.619, training acc(%): 80.583
Validation
100%|                                                                                | 24/24
Validation loss: 1.442, validation acc(%): 79.818
PS D:\shufflenet_2>
```

上方图片显示的是训练过程中训练精度和验证精度，下方图片显示的是训练过程中训练损失率和验证损失率，可以看到训练 30 次过后精度大约在 80 左右，损失率在 1.6 以下。

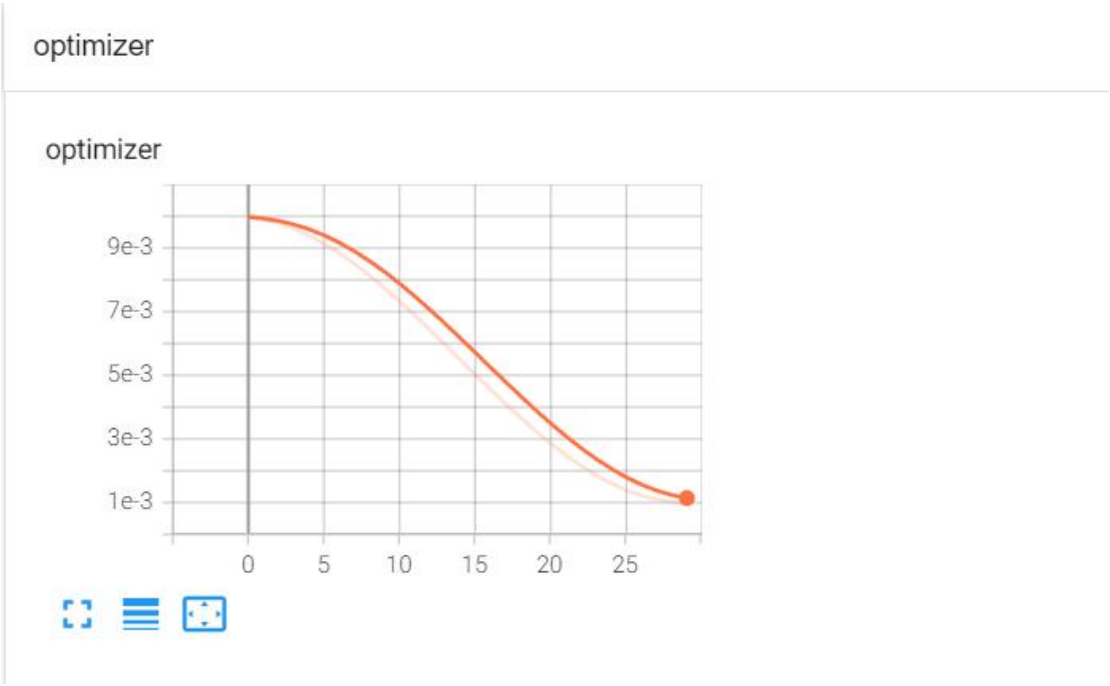
利用 tensorboard 实现可视化：

在 train.py 文件中我利用 tensorbaoad 实现数据可视化，分别记录存储了学习率变化曲线、训练精度曲线、训练损失曲线等。

```
149         #利用tensorboard实现数据可视化，使用 add_scalar 方法来记录数字常量
150         #tag (string): 数据名称，不同名称的数据使用不同曲线展示
151         train_writer.add_scalar('train_acc', train_epoch_loss, epoch)
152         train_writer.add_scalar('valid_acc', train_epoch_acc, epoch)
153         train_writer.add_scalar('valid_loss', valid_epoch_loss, epoch)
154         train_writer.add_scalar('valid_acc', valid_epoch_acc, epoch)
155         train_writer.add_scalar('optimizer', optimizer.param_groups[0]["lr"],
156                                 epoch)
```

```
(pytorch) D:\shufflenet_2>tensorboard --logdir runs --bind_all
I1130 00:31:36.179365 317848 plugin.py:429] Monitor runs begin
TensorBoard 2.9.0 at http://LAPTOP-U3FPJQ8Q:6006/ (Press CTRL+C to quit)
```

学习率变化曲线如图：



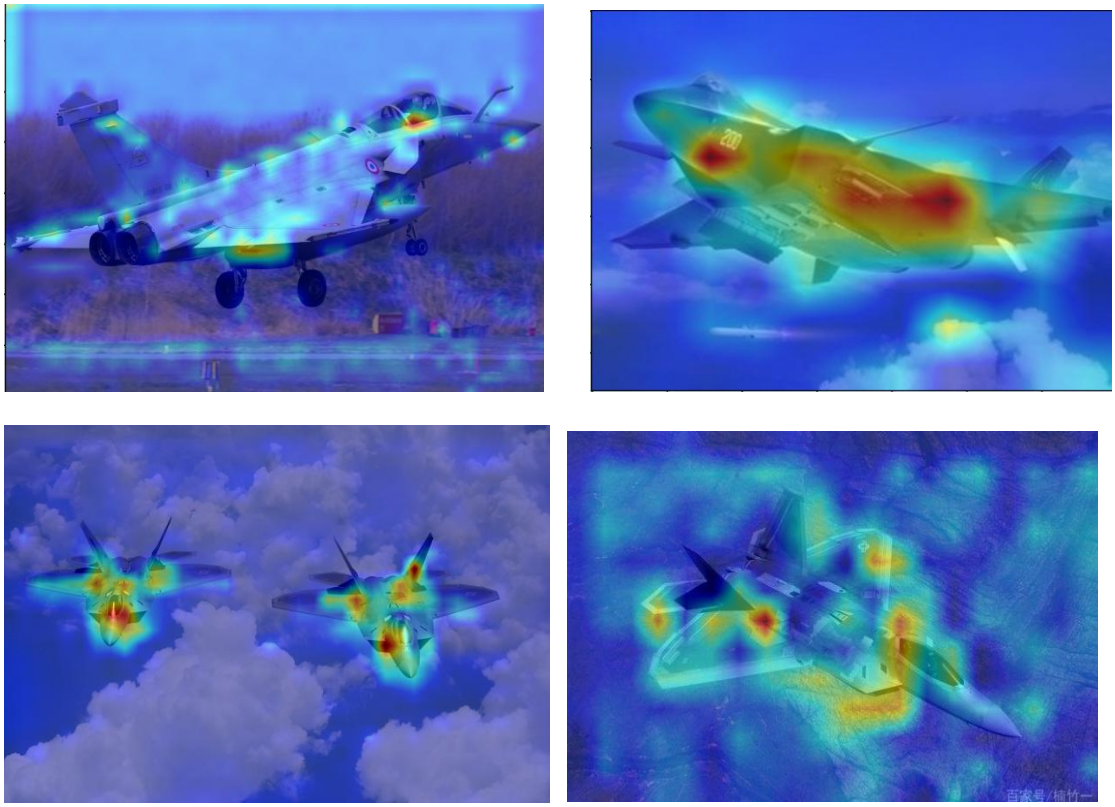
可以看到在使用学习率衰减后，在模型训练初期，使用了较大的学习率进行模型优化，加快模型学习速度，随着迭代次数增加，学习率逐渐进行减小，保证模型在训练后期不会有太大的波动，从而更加接近最优解。

使用 Grad CAM 技术对模型提取的特征进行分析：

根据搭建的 Shufflenet 深度神经网络完成对七种常见战斗机的分类模型训练后，想要了解在训练过程中神经网络究竟在关注数据集图片的哪一部分特征，神经网络是否提取到了有效的特征，因此我使用了 Grad CAM 技术对学习过程中 Shufflenet 的 stage4 层的特征提取进行了可视化。

```python
def main():
    model = shufflenet_v2_x2_0(num_classes=7)
    target_layers = [model.stage4[-1]]

    data_transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])
    # load image
    img_path = "D:/PYTHON/plane_1/F-35/23.jpg"
    assert os.path.exists(img_path), "file: '{}' dose not exist.".format(
        img_path)
    img = Image.open(img_path).convert('RGB')
    img = np.array(img, dtype=np.uint8)
    # img = center_crop_img(img, 224)

    # [C, H, W]
    img_tensor = data_transform(img)
    # expand batch dimension
    # [C, H, W] -> [N, C, H, W]
    input_tensor = torch.unsqueeze(img_tensor, dim=0)

    cam = GradCAM(model=model, target_layers=target_layers, use_cuda=False)

    target_category = 3

    grayscale_cam = cam(input_tensor=input_tensor,
                        target_category=target_category)

    grayscale_cam = grayscale_cam[0, :]
    visualization = show_cam_on_image(img.astype(dtype=np.float32) / 255.,
                                      grayscale_cam,
                                      use_rgb=True)
    plt.imshow(visualization)
    plt.show()
```

结果如图所示：



可以看到在歼 20、F22、F35 和阵风战斗机中随机抽取一张图片绘制特征抽取热力图，神经网络抽取的特征基本都位于飞机上，说明搭建的 Shufflenet 神经网络有着不错的特征提取能力。

六、总结

本次项目对我来说是一次很大提升，是我第一次真正完成了了从现实中的问题出发，到模型抽象再到代码实现整个项目过程，也让我对于 Python 语法有了更加深入的了解和更加熟练的应用。

对于机器学习，我一边跟随老师的课堂，一边跟随 B 站李宏毅老师进行学习，对机器学习的一些基本原理，例如梯度下降、回归、反向传播、卷积神经网络、异常检测、对抗网络等等知识有了更深入的理解。与此同时，我也学会了用 Pytorch 库实现机器学习代码，了解了其中部分重要的函数，掌握了其应用，为以后项目的实现奠定了基础。

这门课程为我以后的科研奠定了基础，在学院的顶峰计划中我参与的项目也是与机器学习有关的，前期的机器学习基础为我节省了不少力气，我也会在方面继续深入学习。