**表情识别**

**摘要**

目录

[第一章 引言 3](#_Toc134552720)

[1.1 背景 3](#_Toc134552721)

[第二章 数据处理 3](#_Toc134552722)

[2.1 Fer2013数据集 3](#_Toc134552723)

[2.2 数据集基本情况 4](#_Toc134552724)

[2.3 数据增强 5](#_Toc134552725)

[第三章 Gabor滤波器与AdaBoost分类器 5](#_Toc134552726)

[3.1 Gabor滤波器 5](#_Toc134552727)

[3.2 PCA 降维 6](#_Toc134552728)

[3.3 AdaBoost 分类器 7](#_Toc134552729)

[3.4 实验结果 8](#_Toc134552730)

[第四章 AlexNet 9](#_Toc134552731)

[4.1 AlexNet 9](#_Toc134552732)

[4.2 网络实现 10](#_Toc134552733)

[第五章 VGG 11](#_Toc134552734)

[5.1 VGG 11](#_Toc134552735)

[5.2 实验 12](#_Toc134552736)

[5.2.1 vgg11 12](#_Toc134552737)

[5.2.2 vgg16 12](#_Toc134552738)

[5.2.3 vgg19 13](#_Toc134552739)

[第六章 GoogLeNet 14](#_Toc134552740)

[6.1 GoogLeNet 14](#_Toc134552741)

[6.2 实验结果 15](#_Toc134552742)

[第七章 ResNet 15](#_Toc134552743)

[7.1 ResNet 15](#_Toc134552744)

[7.2 网络实现 16](#_Toc134552745)

[7.3 实验 18](#_Toc134552746)

[7.3.1 数据预处理 18](#_Toc134552747)

[7.3.2 训练 19](#_Toc134552748)

[7.4 实验结果 20](#_Toc134552749)

[第六章 结束语 22](#_Toc134552750)

[附录 22](#_Toc134552751)

[Resnet18 结构 22](#_Toc134552752)

# 第一章 引言

## 1.1 背景

情绪通过始终如一的面部表情表现出来。这意味着，无论语言和文化障碍如何，总会有一套基本的面部表情可供人们评估和交流。经过广泛的研究，现在人们普遍认为，人类共有七种反映基本情绪体验的面部表情。这些基本情绪是愤怒、厌恶、恐惧、快乐、悲伤、惊讶和中立。表情是人类传达情感状态和意图的最有力、最自然、最普遍的信号之一。计算机视觉领域已探索了各种面部表情识别（FER）系统编码来自面部表示的表情信息。

FER系统根据特征表示可被分为静态图像FER和动态序列FER，基于静态的方法用当前图像的空间信息进行编码，基于动态的方法考虑了表情序列之间的时间关系，现在也存在通过增加音频等信息构建多模态系统等方法。

本文主要讨论静态图像FER，静态图像FER的大多数传统方法使用手工特征或浅层学习方法，随着人工智能领域的飞速发展和情感识别比赛的不断举办，面部表情训练数据不断丰富，各个领域逐渐向深度学习方法转变，效果大大超过了传统方法。

静态图像FER当前存在的问题：深度神经网络需要大量的训练数据，以避免过拟合；受试者的个人属性差异较大；此外，还存在姿势、光照、遮挡等因素的影响；

# 第二章 数据处理

## 2.1 Fer2013数据集

Fer2013人脸表情数据集是非常常用的表情检测数据集，由 35886 张人脸表情图片组成。其中训练图（Training）28708张，公共验证图（PublicTest）和私有验证图（PrivateTest）各 3589 张，每张图片是由大小固定为 48×48 的灰度图像组成。数据集共有 7 种表情类别，具体表情对应的标签和中英文如下表。

|  |  |  |
| --- | --- | --- |
| 类别标号 | 英文含义 | 中文含义 |
| 0 | anger | 生气 |
| 1 | disgust | 厌恶 |
| 2 | fear | 恐惧 |
| 3 | happy | 开心 |
| 4 | sad | 伤心 |
| 5 | surprised | 惊讶 |
| 6 | normal | 中性 |

数据集中部分样本如下图所示：



Figure 1数据集图片示例

关于数据集的详细情况与下载地址可见<https://www.kaggle.com/datasets/msambare/fer2013>。

## 2.2 数据集基本情况

通过对数据集进行简单的统计性分析，注意到，该数据集的7个类别分布并不均匀，在测试图（Training）、验证图（Test）（由公共验证图（PublicTest）和私有验证图（PrivateTest）合并组成，共 7178 张图片）上各表情类别的分布情况大致如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 表情标号 | 表情含义 | 训练图（Training）数量 | 验证图Test）数量 |
| 0 | 生气 | 3995 | 958 |
| 1 | 厌恶 | 436 | 111 |
| 2 | 恐惧 | 4097 | 1025 |
| 3 | 开心 | 7215 | 1774 |
| 4 | 伤心 | 4830 | 1247 |
| 5 | 惊讶 | 3171 | 831 |
| 6 | 中性 | 4965 | 1233 |

通过画出各表情类别在训练图与验证图上的柱状图，可以发现在七个类别中，开心类别的图片数量明显高于其他类别，而关于厌恶类型的图片数量大概只有其他类别数量的十分之一，这对于后期整个表情识别系统性能影响是巨大的。

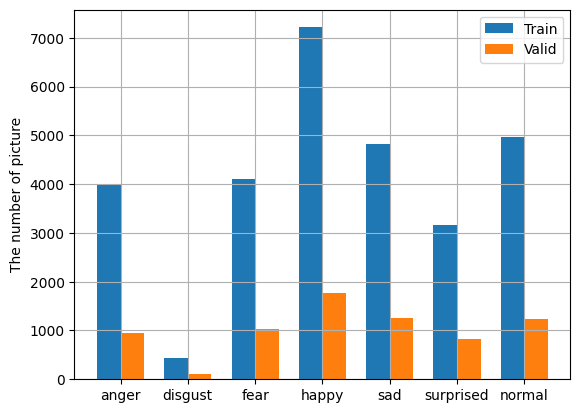


Figure 2 数据集各类别数据分布情况

## 2.3 数据增强

数据增强是深度学习中的一种技术，它通过从现有数据生成新的训练数据来扩展原数据集。数据增强工具通过操作现有数据的参数，将数据转换为新的、独特的样本，从而有效解决训练深度学习模型数据不足的问题，并赋予模型更多的多样性和灵活性，提高模型的泛化能力与鲁棒性。

对于训练图，我们主要进行了如下几种数据增强：

* 将图片尺寸拉伸到（Resize）
* 对图片进行随机仿射变换（RandomAffine）
* 将图片随机水平翻转（RandomHorizontalFlip）

鉴于后续使用的深度模型都是以 的图片作为输入，我们对于测试图也进行了Resize操作。

# 第三章 Gabor滤波器与AdaBoost分类器

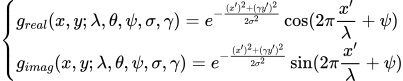
## 3.1 Gabor滤波器

Gabor是 D.Gabor 于 1946 年提出的一个用于边缘提取的线性滤波器，其频率和方向表达与人类视觉系统类似，能够提供良好的方向选择和尺度选择特性，而且对于光照变化不敏感。

在空间域，一个二维的Gabor滤波器是一个正弦平面波和高斯核函数的乘积，其公式如下：

descript

进一步的上式可以分为实部与虚部的形式



其中

descript

二维Gabor滤波器中各参数的含义可见下表：

|  |  |  |
| --- | --- | --- |
| 参数 | 物理意义 | 描述 |
| descript | 波长 | 直接影响滤波器的滤波尺度，通常大于等于2 |
| descript | 方向 | 滤波器的方向 |
| descript | 相位偏移 | 调谐函数的相位偏移，取值-180到180 |
| descript | 空间纵横比 | 决定滤波器的形状，取1时为圆形，通常取0.5 |
| descript | 带宽 | Gauss滤波器的方差，通常取2descript |

## 3.2 PCA 降维

PCA（Principal Component Analysis） 是一种常见的数据分析方式，常用于高维数据的降维，可用于提取数据的主要特征分量。

在PCA中，数据从原来的坐标系转换到了新的坐标系。新坐标系的选择是由数据本身决定的。第一个新坐标轴选择的是原始数据中方差最大的方向，第二个新坐标轴的选择和第一个坐标轴正交且具有最大方差的方向。该过程一直重复，重复次数为原始数据中特征的数目。会发现，大部分方差都包含在最前面的几个新坐标轴中。因此我们可以只选择前面几个坐标轴，即对数据进行了降维处理。

通过降维操作，我们可以识别出最重要的多个特征，使得数据集更容易使用，同时降低很多算法的计算开销，提升分类算法的性能。但同时降维有可能会损失掉有用信息。

PCA算法的主要流程如下：

1. 对输入数据去除平均值
2. 计算去除平均值后的数据协方差矩阵的特征值和特征向量
3. 按照大到小的顺序排列，取特征值的前n个对应的特征向量
4. 最后按照逆过程重建数据

## 3.3 AdaBoost 分类器

AdaBoost 是 Adaptive Boosting 的简称，属于集成算法(Ensemble Method)中Boosting 类别中的一种，其通过结合多个弱学习器获得比任意单个学习器都要好的性能。其基本思想为：

* 提高上一轮被错误分类的样本的权值，降低被正确分类的样本的权值；
* 线性加权求和。误差率小的基学习器拥有较大的权值，误差率大的基学习器拥有较小的权值。

AdaBoost 算法的基本流程为：

1. 确定样本集
2. 初始化样本数据的权重，如每个样本的权重为1/n(假设样本个数为n)
3. 进行1，2，...，T轮迭代

a. 归一化样本权重

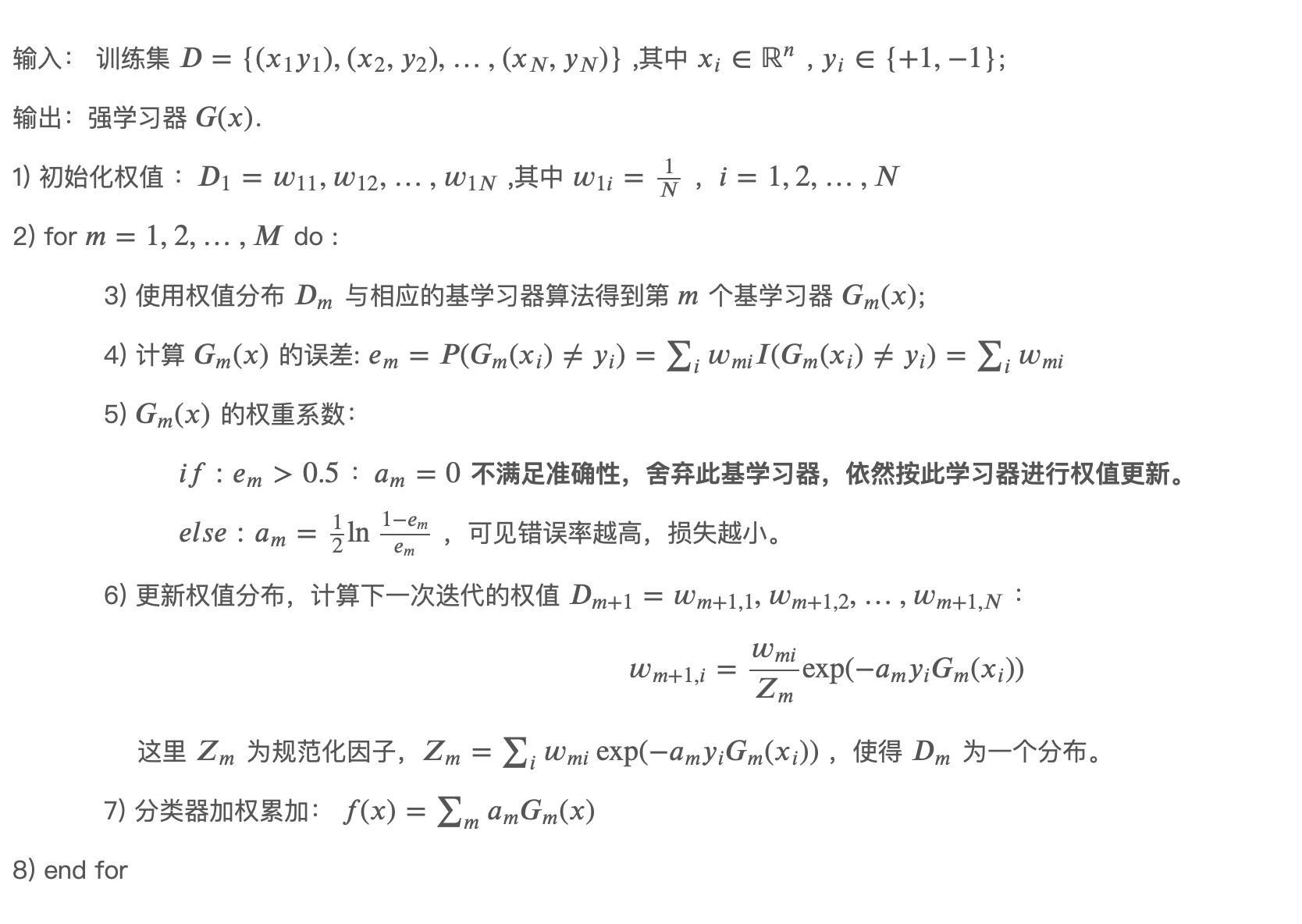
b. 对样本集进行训练，并计算训练误差

c. 选择误差最小的分类器作为本轮的分类器

d. 根据预测结果更新样本数据的权重：预测错误样本增加权重，预测正确样本降低权重

e. 计算分类器的权重

4. 根据分类器的预测结果及其权重加权表决最终结果



该算法的主要优点为：

* 不容易发生过拟合。
* Adaboost是一种有很高精度的分类器。
* 当使用简单分类器时，计算出的结果是可理解的。
* 可以使用各种方法构建子分类器，Adaboost算法提供的是框架。

相应地，其主要缺点为：

* 训练时间过长。
* 执行效果依赖于弱分类器的选择。
* 对样本敏感，异常样本在迭代中可能会获得较高的权重，影响最终的强学习器的预测准确性。

## 3.4 实验结果

在实验中，我们选取方向为 0，30，60，90，120，150，频率为 0.5 的一组 Gabor 滤波器进行特征提取，将滤波获得的特征展平，并进行归一化和特征拼接，从而将每张图转化为864维的特征。对于高维特征，我们再采用PCA方法进行降维，将维度压缩至100维。最后我们采用AdaBoost算法，在训练集上进行训练。最终在测试集上获得的分类结果如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 表情类别（标签） | 精度（Precision） | Recall（召回率） | F1分数（F1 Score） |
| 生气（0） | 0.17 | 0.09 | 0.12 |
| 厌恶（1） | 0.08 | 0.05 | 0.06 |
| 恐惧（2） | 0.19 | 0.08 | 0.12 |
| 开心（3） | 0.33 | 0.56 | 0.42 |
| 伤心（4） | 0.27 | 0.24 | 0.25 |
| 惊讶（5） | 0.18 | 0.18 | 0.18 |
| 中性（6） | 0.28 | 0.28 | 0.28 |
| 宏观指标 | | | |
| 准确率 | 0.27 | | |
| 宏平均 | 0.21 | 0.21 | 0.20 |
| 权重平均 | 0.25 | 0.27 | 0.25 |

通过实验数据可以发现，传统的机器学习方法在该数据集上获得的效果并不理想，对于开心类别的分类准确率相对于其他表情而言略高一些。

# 第四章 AlexNet

## 4.1 AlexNet

AlexNet由G.E.Hinton的学生A.Krizhevsky于2012年提出，并以显著的优势获得了当年ILSVRC比赛的冠军(Top-5错误率16.4%)，从而一举奠定了卷积神经网络在计算机视觉领域的统治地位。

AlexNet 的网络结构可以看作是早期的 LeNet 网络的更深更宽的版本，其采用了 8 层的卷积神经网络结构，拥有 5 个卷积层，其中 3 个卷积层后面接入了 MaxPooling 层，然后接入 3 层全连接层，最后一层是代表 n 类输出的 Softmax 层用作分类层。

AlexNet 网络结构包含 65 万个神经元、 6000万个网络参数和 6.3 亿个网络连接，其首次采用了 GPU 运算加速实现了如此大规模的训练运算。 AlexNet 的网络结构图如图 4.1 所示。

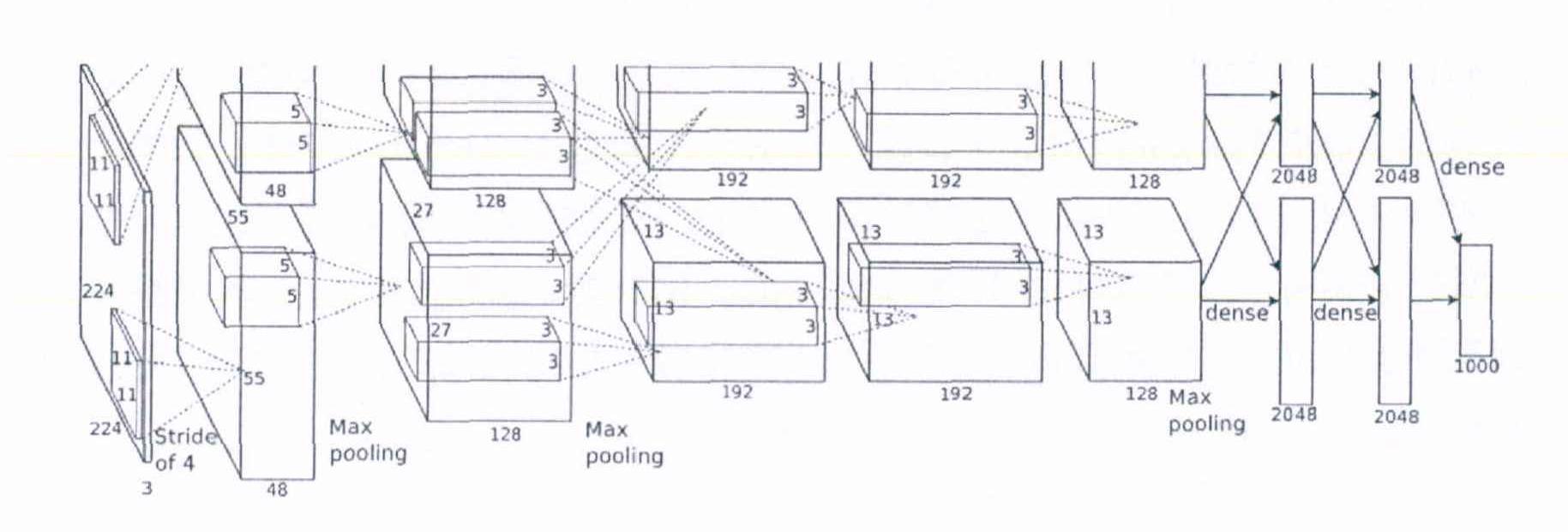


Figure 3 AlexNet的网络结构

相比与早期的 LeNet 网络，AlexNet 采用了一系列新技术点如下：

(1) 采用了 ReLU 激活函数替代早期的 Sigmoid 激活函数，成功解决了早期神经网络在网络层数较深时遇到的梯度弥散问题。

(2) 提出了 Dropout 方法在训练时随机忽略掉部分神经元以避免模型过拟合 AlexNet。首次将 Dropout 算法实用化并证明其有效性，其网络结构中最后几个全连接层都使用了 Dropout 层。

(3) 采用 Max-Pooling 替代以往普遍使用的 Mean-Pooling，避免了后者带来的模糊化效果，并且提出了使步长比 pooling 核尺寸小的思路，使得池化层输出之间存在重叠和覆盖，从而提升了特征的丰富性。

(4) 提出了 LRN 层(Local Response Normalization Layer)，对局部神经元的活动创建竞争机制，放大响应较大的神经元并抑制反馈小的神经元活动，从而增强网络的泛化性能。其通过前后几层对应位置的神经元对中间层做平滑约束，其计算方法为:

公式4.1

其中……其中a表示在(x)这个位置上，在第i层正向传播经过激励后的输出，N为该层的feature map 总数取该 feature map 为中心前后各n/2个feature map 求均值，最后得出下一层的输入b。AlexNet 网络中 LRN 被接入第一个及第二个卷积层之后

(5) 首次使用 CUDA 加速深度卷积神经网络的训练，利用了 GPU 的并行计算能力处理神经网络训练时的大量矩阵计算。由于当时 GPU 运算能力限制，原始的 AlexNet 训练时采用了两块 GPU 并行训练的方式，导致早期的 AlexNet 网络结构不少组件都被拆分为两个部分；目前单个 GPU的显存积可以容纳AlexNet 的全部网络模型参数，但多个 GPU 并行运算仍然是大规模网络训练提速的重要手段。

## 4.2 网络实现

在实验中，我们采用SGD优化器，学习率取0.001，动量取0.9，采用交叉熵作为分类损失函数。在批量大小取为128的情况下，训练120轮后，模型基本趋于收敛。训练中误差曲线，训练集准确率曲线和测试集准确率曲线如下图：

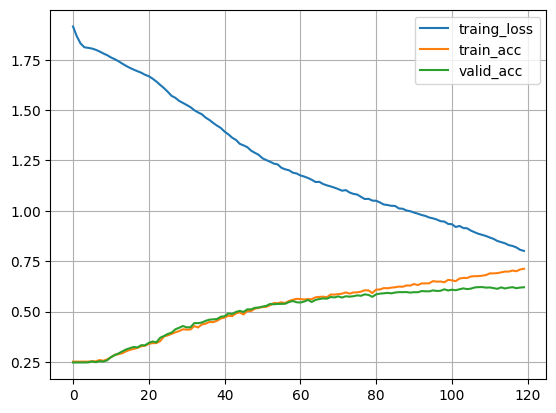


Figure 4 AlexNet 训练曲线

AlexNet在训练集上达到了71.3%的准确率，在测试集上达到了62.2%的准确率。对于各个类别而言，在测试集上的表现可见下表：

|  |  |
| --- | --- |
| 表情类别（标签） | 准确率 |
| 生气（0） | 68% |
| 厌恶（1） | 50% |
| 恐惧（2） | 21% |
| 开心（3） | 92% |
| 伤心（4） | 30% |
| 惊讶（5） | 60% |
| 中性（6） | 72% |

值得注意的是，分类器对于开心的检测能力比较出众，达到了92%，对于小样本的厌恶检测能力也还不错，但对于恐惧和伤心的检测能力略显欠缺。总体而言，相对于传统方法，AlexNet在性能上取得了很大的突破。

# 第五章 VGG

## 5.1 VGG

VGG于2014年由英国牛津大学 Visual Geometry Group 组提出，主要工作是证明了增加网络的深度能够在一定程度上影响网络最终的性能。VGG有两种结构，分别是VGG16和VGG19，两者除了网络深度不一样，其本质并没有什么区别。相对于2012年AlexNet，VGG的一个改进是采用连续的3x3小卷积核来代替AlexNet中较大的卷积核（AlexNet 采用了11x11，7x7与5x5大小的卷积核）。两个3x3步长为1的卷积核的叠加，其感受野相当与一个5x5的卷积核。但是采用堆积的小卷积核是由于大卷积核的，因为层数的增加，增加了网络的非线性，从而能让网络来学习更复杂的模型，并且小卷积核的参数更少。

VGG的输入是224x224大小的RGB图像，在输入网络之前，作者先在训练集图像上对所有的图像计算RGB三通道的均值，并且减去对应均值。下图为VGG的网络结构，其中VGG16包含了13个卷积层和3个全连层，如图中的网络D，VGG19包含了16个卷积层和3个全连层，如图中的网络E。

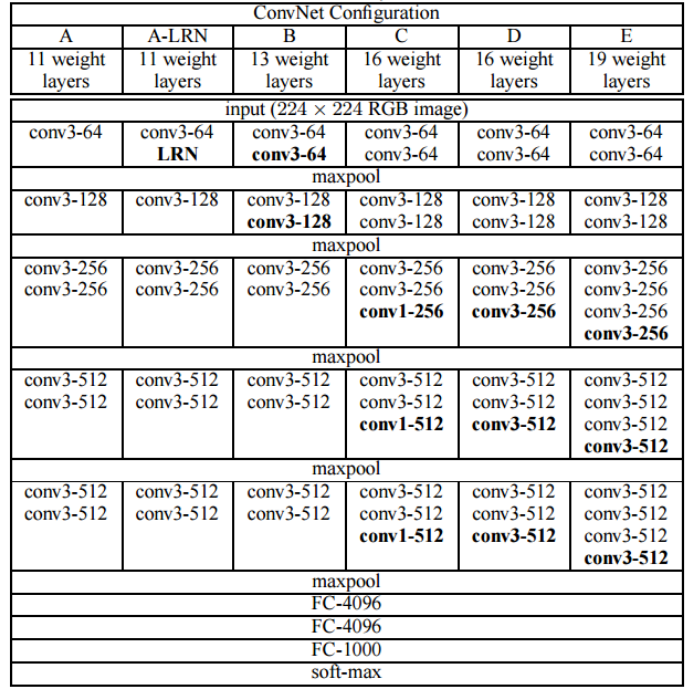


Figure 5 VGG网络结构图

## 5.2 实验

在实验中，我们采用SGD优化器，学习率取0.001，动量取0.9，采用交叉熵作为分类损失函数。在批量大小取为128的情况下，使用预训练参数初始化模型，依次对vgg11、vgg16和vgg19进行训练。

### 5.2.1 vgg11

训练30轮后，模型基本趋于收敛。vgg11在数据集FER2013上测试集准确率达到89.19%，验证集准确率达到65.71%，出现了严重的过拟合现象。训练中误差曲线，训练集准确率曲线和测试集准确率曲线如下图：

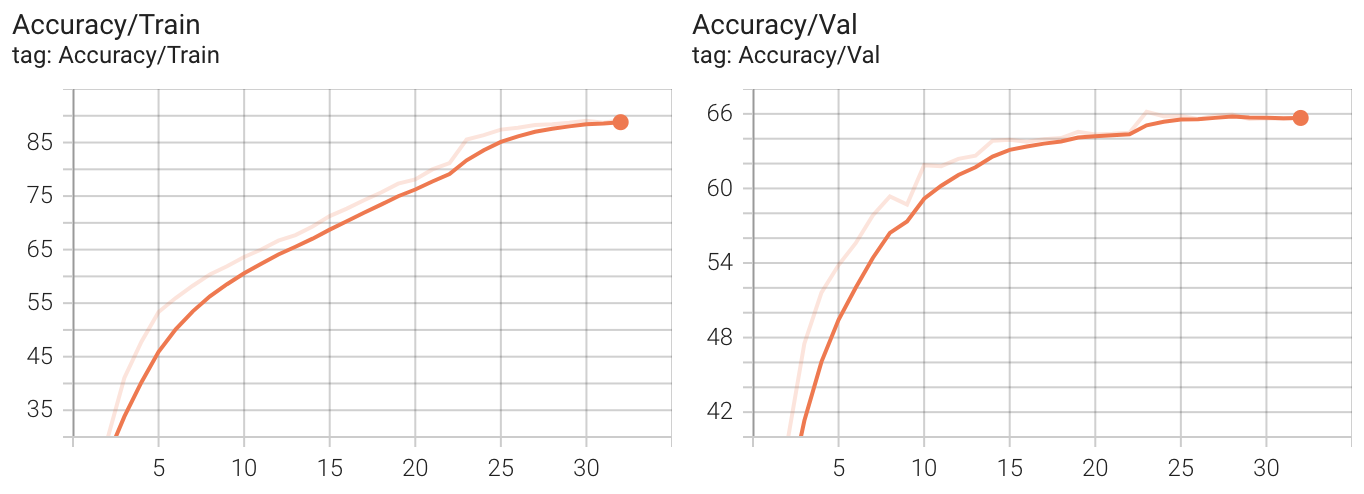


Figure 6 vgg11在FER2013上准确率图



Figure 7 vgg11在FER2013上损失图

### 5.2.2 vgg16

训练30轮后，模型基本趋于收敛。vgg16在数据集FER2013上测试集准确率达到77.38%，验证集准确率达到65.72%，出现了严重的过拟合现象，测试集性能与vgg11相当且训练集性能弱于vgg11，说明更复杂的网络没有带来更好的性能，需要使用数据增强、正则化等方法限制模型等复杂度、减弱过拟合现象、增强模型等泛化性。训练中误差曲线，训练集准确率曲线和测试集准确率曲线如下图：

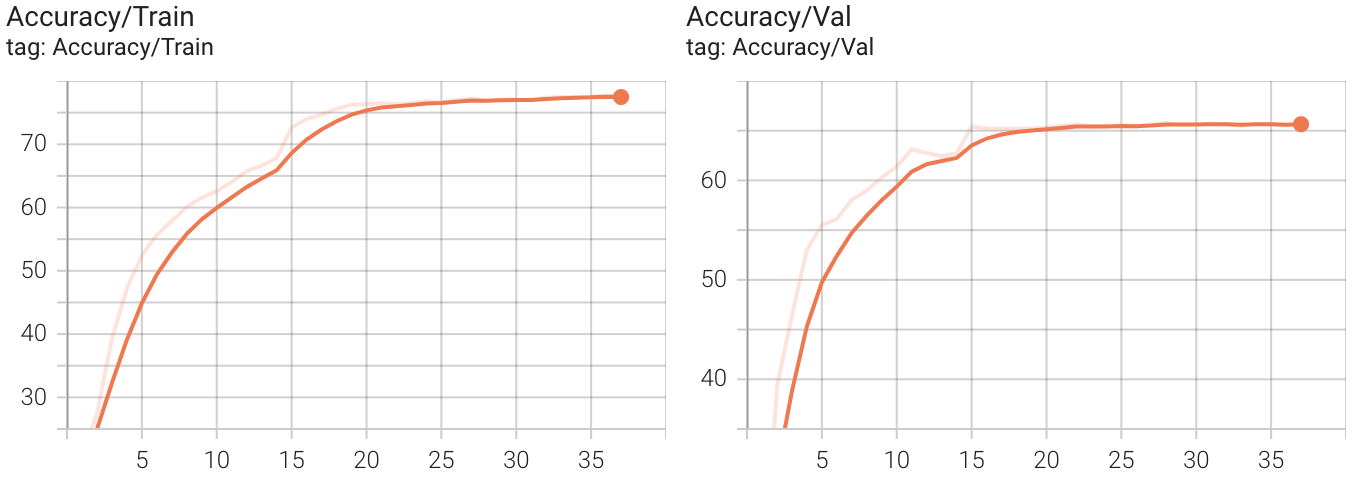


Figure 8 vgg16在FER2013上准确率图

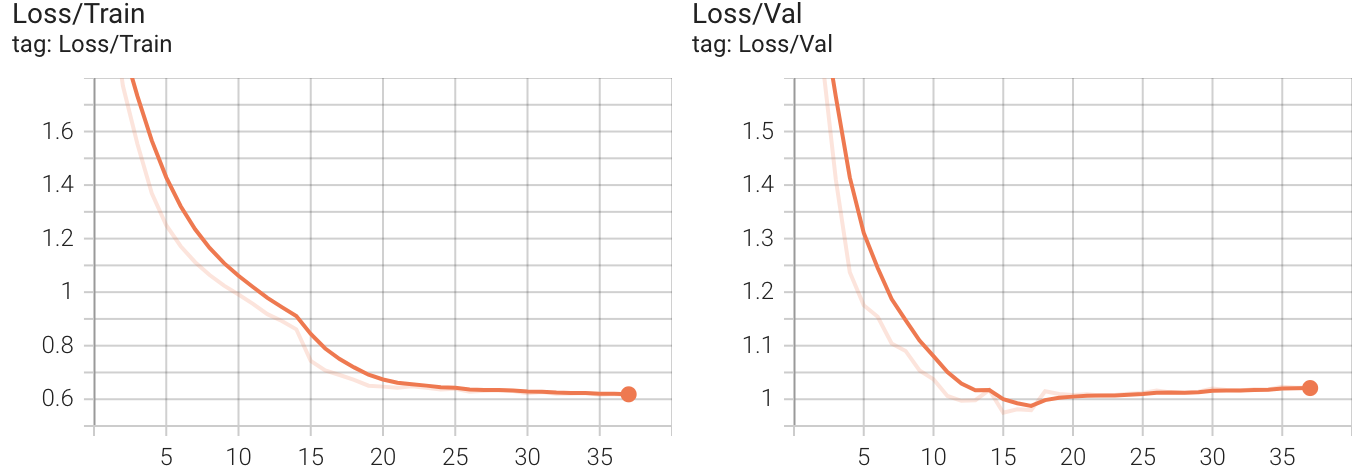


Figure 9 vgg16在FER2013上损失图

### 5.2.3 vgg19

训练30轮后，模型基本趋于收敛。vgg19在数据集FER2013上测试集准确率达到93.45%，验证集准确率达到69.75%，出现了严重的过拟合现象。训练中误差曲线，训练集准确率曲线和测试集准确率曲线如下图：

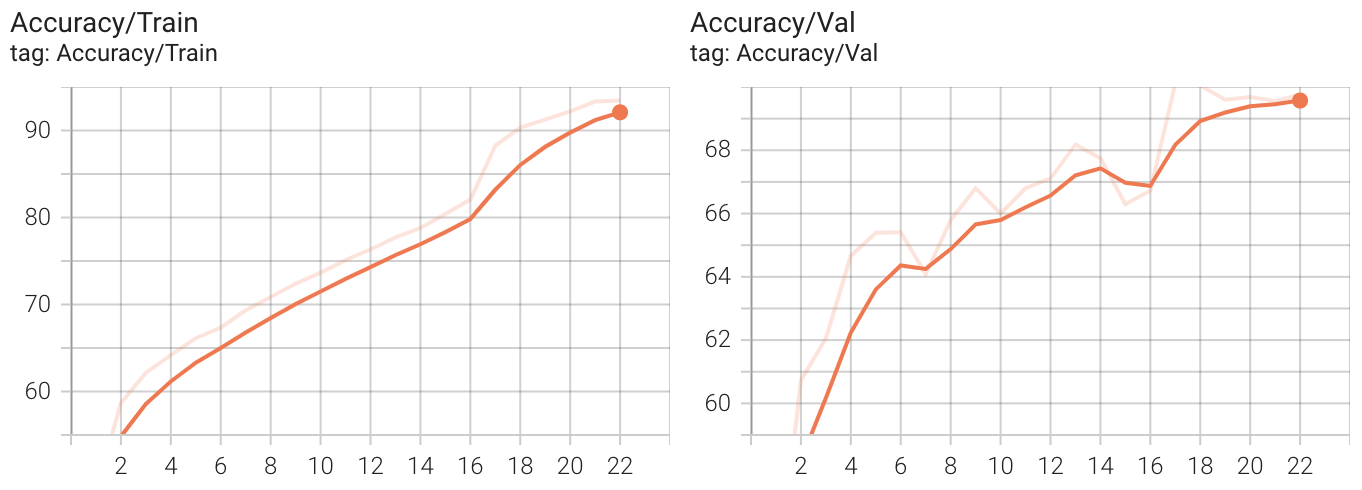


Figure 10 vgg19在FER2013上损失图

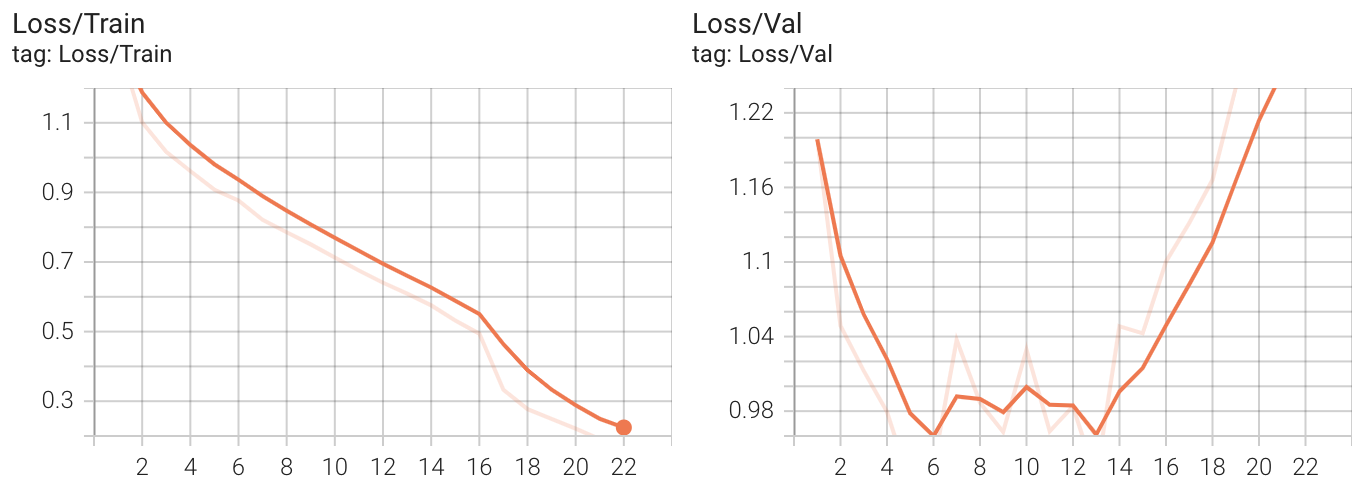


Figure 11 vgg19在FER2013上损失图

# 第六章 GoogLeNet

## 6.1 GoogLeNet

GoogLeNet是2014年Christian Szegedy等人在2014年大规模视觉挑战赛(ILSVRC-2014)上使用的一种全新卷积神经网络结构，并以6.65%的错误率力压VGGNet等模型取得了ILSVRC-2014在分类任务上的冠军，于2015年在CVPR发表了论文《Going Deeper with Convolutions》。在这之前的AlexNet、VGG等结构都是通过增大网络的深度（层数）来获得更好的训练效果，但层数的增加会带来很多负作用，比如overfitting、梯度消失、梯度爆炸等，GoogLeNet则做了更加大胆的网络结构尝试，Inception的提出则从另一种角度来提升训练结果：能更高效的利用计算资源，在相同的计算量下能提取到更多的特征，从而提升训练结果，采用了Inception结构的GoogLeNet深度只有22层，其参数约为AlexNet的1/12，是同时期VGGNet的1/3。

下面给出了GoogLeNet架构的缩略图，相比于以前的卷积神经网络结构，除了在深度上进行了延伸，还对网络的宽度进行了扩展，整个网络由许多块状子网络的堆叠而成，这个子网络即Inception模块，采用了模块化的设计，方便层的添加与修改。

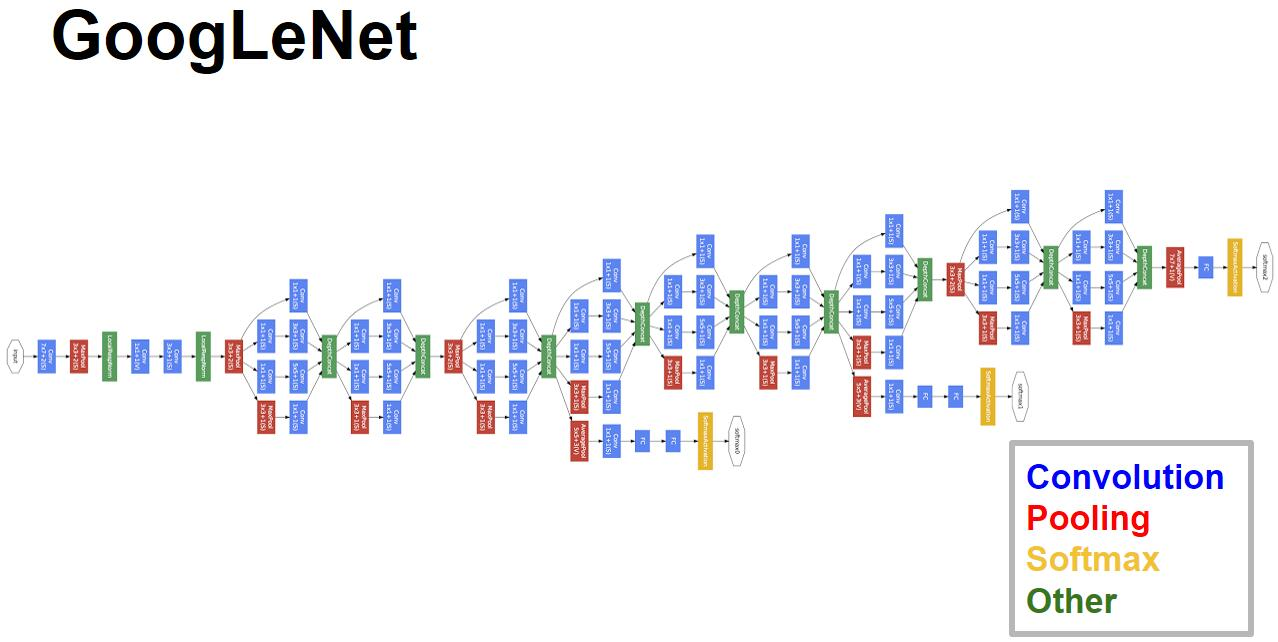


Figure 12 GoogLeNet结构图

## 6.2 实验结果

在实验中，我们采用SGD优化器，学习率取0.001，动量取0.9，采用交叉熵作为分类损失函数。在批量大小取为128的情况下，使用预训练参数初始化模型，训练120轮后，模型基本趋于收敛。GoogleNet在数据集FER2013上测试集准确率达到91.71%，验证集准确率达到69.77%，出现了严重的过拟合现象。训练中误差曲线，训练集准确率曲线和测试集准确率曲线如下图：

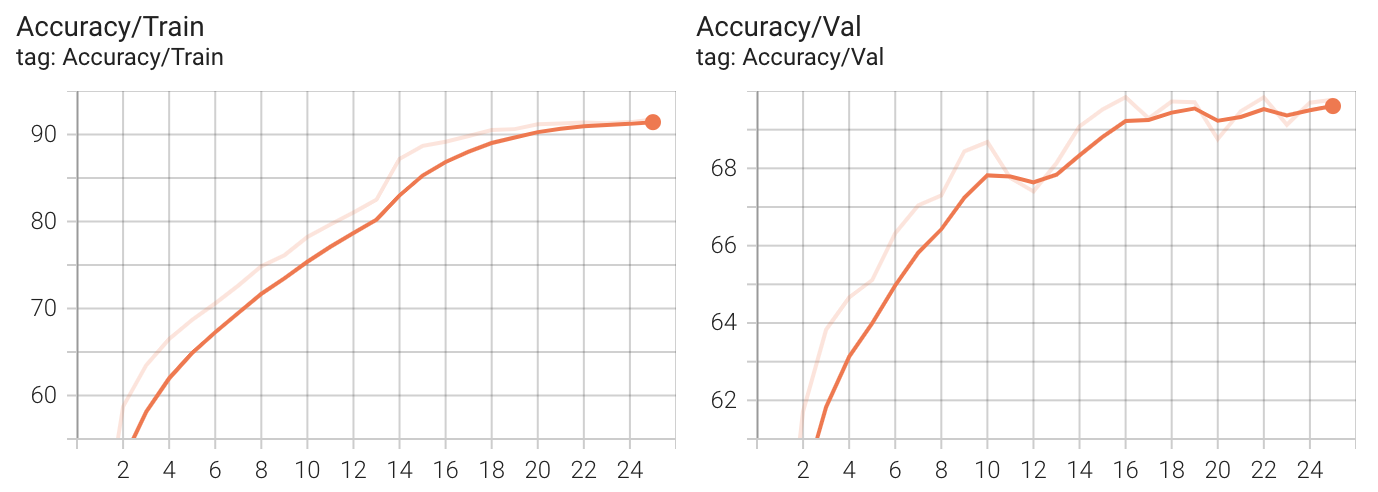


Figure 13 GoogleNet在FER2013上准确率图

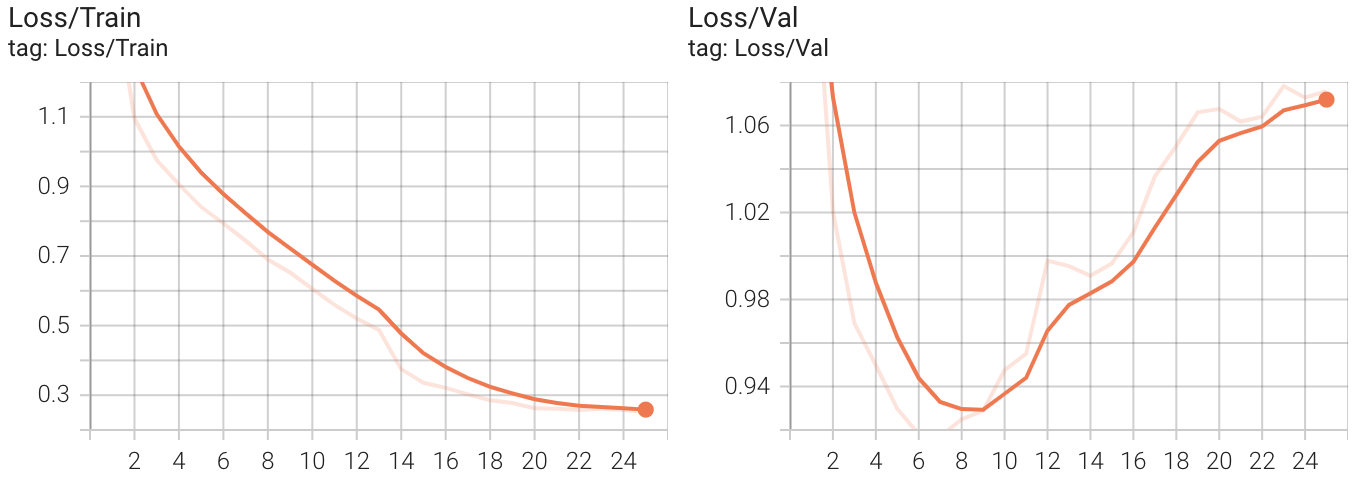


Figure 14 GoogleNet在FER2013上损失图

# 第七章 ResNet

## 7.1 ResNet

ResNet(Residual Neural Network)通过采用残差单元(Residual Unit)训练超深层次的神经网络，取得了 2015 年 ILSVRC 大赛的分类准确率第一名，其最深的网络结构可达 152 层。

ResNet 的核心思想是残差单元的引入。假定某段神经网络的输入是 x，期望输出是 H(x)，如果把 x 直接传到输出作为初始结果，此时需要学习的目标就转化为 F(x) = H(x) - x。ResNet 即是利用残差单元将学习目标从完整的输出 H(x) 转化为残差 H(x) - x。下图展示了一个基本的残差学习模块:

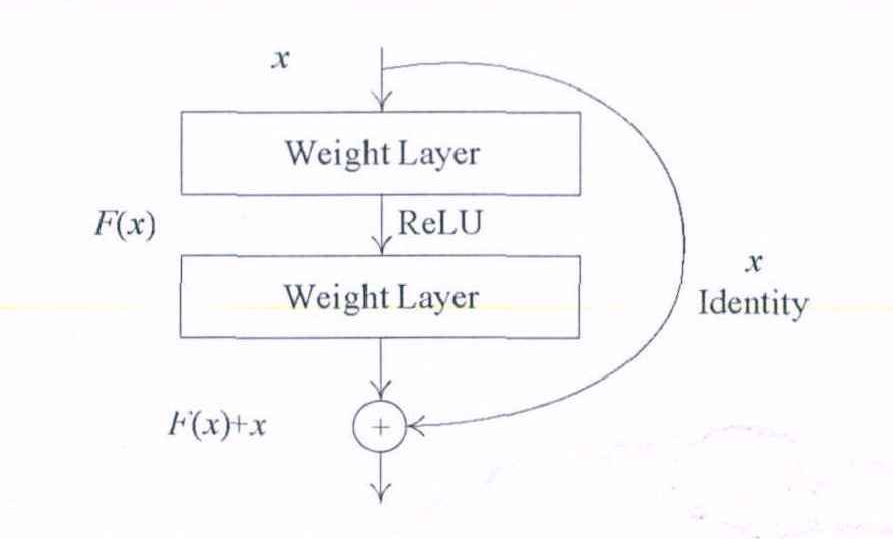


Figure 15 残差学习模块示意图

与直连卷积神经网络不同，ResNet通过多个旁路的支线将输入连接到后面的层，从而引入残差学习，与传统的卷积层信息传递相比保存了信息的完整性，简化了学习目标和难度。下图展示了不同层数的ResNet的网络结构，都是通过基本的两层和三层残差模块连接而成：

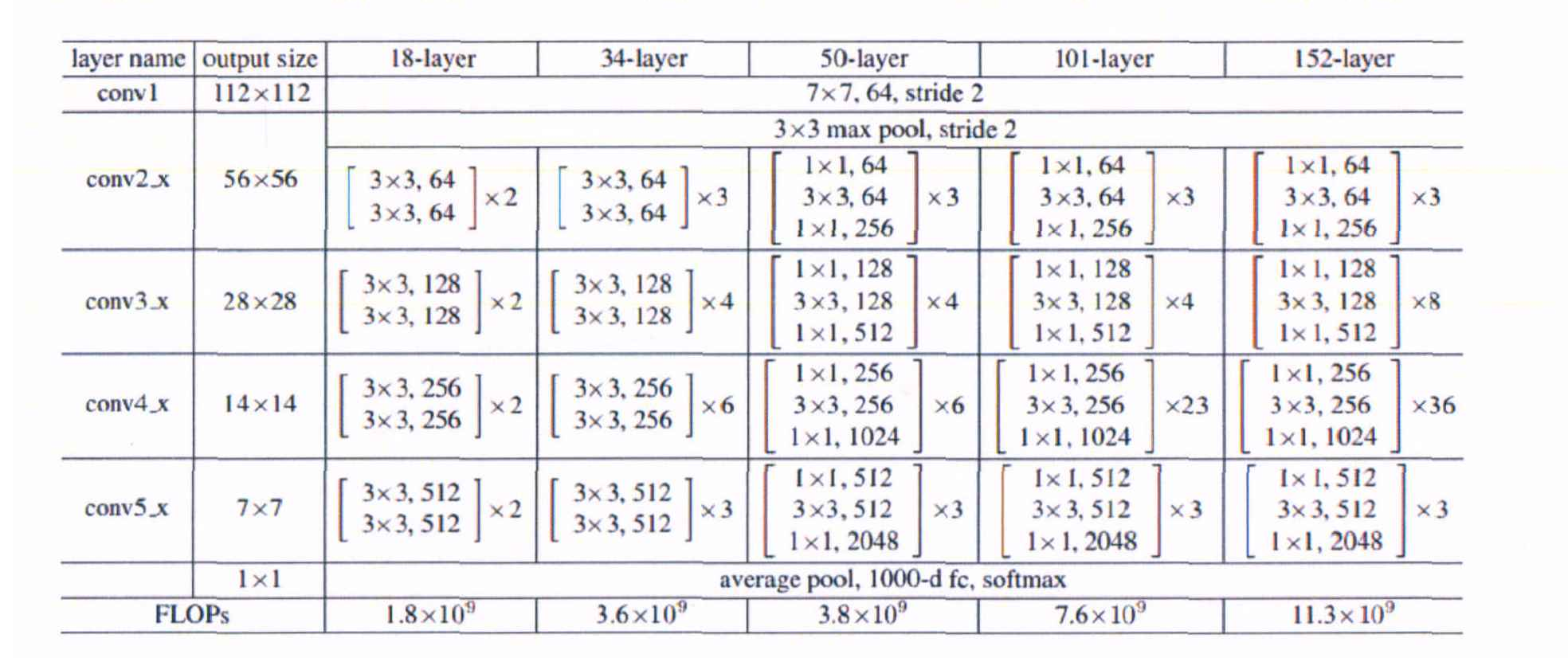


Figure 16 ResNet网络结构配置

## 7.2 网络实现

ResNet包含多种不同的网络结构，其差异主要在于网络层数不同，经过多轮实验对比我们发现ResNet18的效果较好，所以选择了ResNet18进行训练。ResNet18的结构如下：



Figure 17 ResNet18 网络结构图

与之前的网络相比，这次我们的CNN模型的一个关键变化是增加了resudial块，它将原始输入添加回通过一个或多个卷积层传递输入获得的输出特征图。这个看似很小的变化使模型的性能有了很大的提高。此外，在每个卷积层之后，我们将添加一个批量归一化层，该层对前一层的输出进行归一化。

残差连接：添加残差块将原始输入添加回通过一个或多个卷积层传递输入获得的输出特征图。

批量归一化：在每个卷积层之后，我们添加一个批量归一化层，该层对前一层的输出进行归一化。这有点类似于数据规范化，只是它应用于层的输出，并且平均值和标准差是学习的参数。

## 7.3 实验

### 7.3.1 数据预处理

对于ResNet的实验，我们对训练集进行了如下数据处理：

1. 将图像转化为灰度图。

2. 将图像于0.8倍到1.2倍之间进行随机裁剪并将分辨率重置为[40,40]。

3. 每次处理有50%的概率将图像的亮度、对比度和饱和度分别修改为0.5、0.5、0.5。

4. 每次处理有50%的概率对图像进行水平和垂直偏移量为 0.2 的仿射变换。

5.每次处理有50%的概率对图像进行水平翻转。

6. 每次处理有50%的概率对图像进行角度为-10度到10度之间的旋转。

7. 对图片进行上下左右以及中心裁剪，获得5张图片。

8. 转化为Tensor并进行归一化，接着将数据按通道进行标准化。我们可以通过减去平均值并除以每个通道中像素的标准偏差来归一化图像张量。对数据进行归一化可防止来自任何一个通道的像素值不成比例地影响损失和梯度。

9. 使用Random Erasing方法，在图片中随机选择一个矩形框，在随机的位置上使用随机的值来擦除图片原来的像素。通过该方法能够给图片加入不同程度的遮挡，通过这样的训练数据，可以减少模型过拟合的风险同时对遮挡具有一定的鲁棒性。

对验证集和测试集进行如下操作：

1. 将图像转化为灰度图。

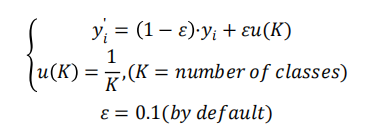
2． 上下左右中心裁剪后翻转：在图片的四个角和中心各截取一幅大小为 size 的图片，并进行水平或竖直翻转。

3. 转化为Tensor并进行归一化，接着将数据按通道进行标准化。

### 7.3.2 训练

混合精度：在训练时通过在模型中同时使用16 位和32 位浮点类型，加快运行速度，以减少内存对使用。 通过让模型的某些部分保持使用32 位类型以保持数值稳定性，可以缩短模型的单步用时，而在评估指标（如准确率）方面仍可以获得同等的训练效果。混合精度训练能够有效降低对显存的消耗，可只占用之前一半左右的显存，但是训练速度会比之前稍慢一些。

标签平滑：采用标签平滑方法进行正则化以防止模型在训练时过于自信地预测标签，改善泛化能力差的问题。公式如下图所示，y’\_i表示在原标签基础上修改后的标签值，y\_i为原标签值。以二分类为例，原标签值为[0,1]，经过计算之后，标签值就变为了[0.05,0.95]。相当于重置了label，标签平滑一定程度上缩小 label 中 min 和 max 的差距，鼓励模型接近对应的label，越接近损失越小，可减少过拟合，让两端的极值向中间靠拢，可有效增强泛化能力。



标签平滑公式

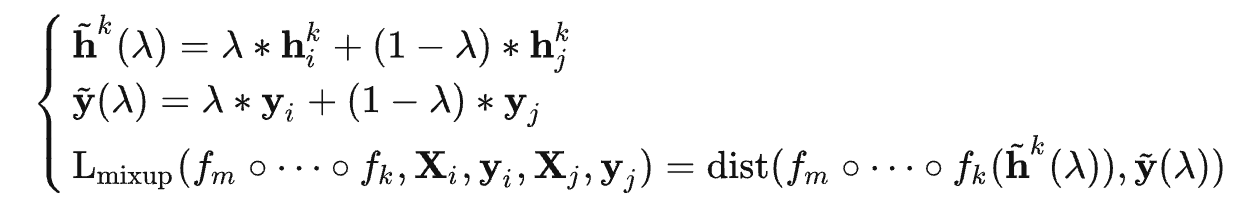
优化器：采用SDG作为优化器，学习率为0.1，动量为0.9，使用L2正则化方法通过向损失函数添加附加项来防止权重变得太大以防止过拟合，权衰量为0.001，并使用nesterov动量，在凸批量梯度的情况下可有效改进收敛率。

学习率调度：使用学习率调度器，而不是使用固定的学习率，它会在每批训练后改变学习率。应用ReduceLROnPlateau策略时，当特定的度量指标，如训练损失、验证损失或准确率不再变化时，学习率就会改变。采取max模式，factor为0.75，patience为5，意味着当指标5次没有改进时，学习率下降为原来的75%。

梯度裁剪：我们还添加了梯度裁剪，这有助于将梯度值限制在较小的范围内，以防止在训练期间由于较大的梯度值而导致模型参数的意外更改。

数据预处理时为进行数据增强进行了随机裁剪，在训练时需要对数据维数和标签进行对应调整。

使用Mixup对输入数据进行简单线性变化以防止过拟合，可极大提升模型的泛化能力，其公式如下所示。Mixup与label smoothing结合能产生非常好的效果。



## 7.4 实验结果

经过优化的ResNet18在FER2013数据集上测试集准确率达到了74.28%，相较于优化前的66.34%提升显著。

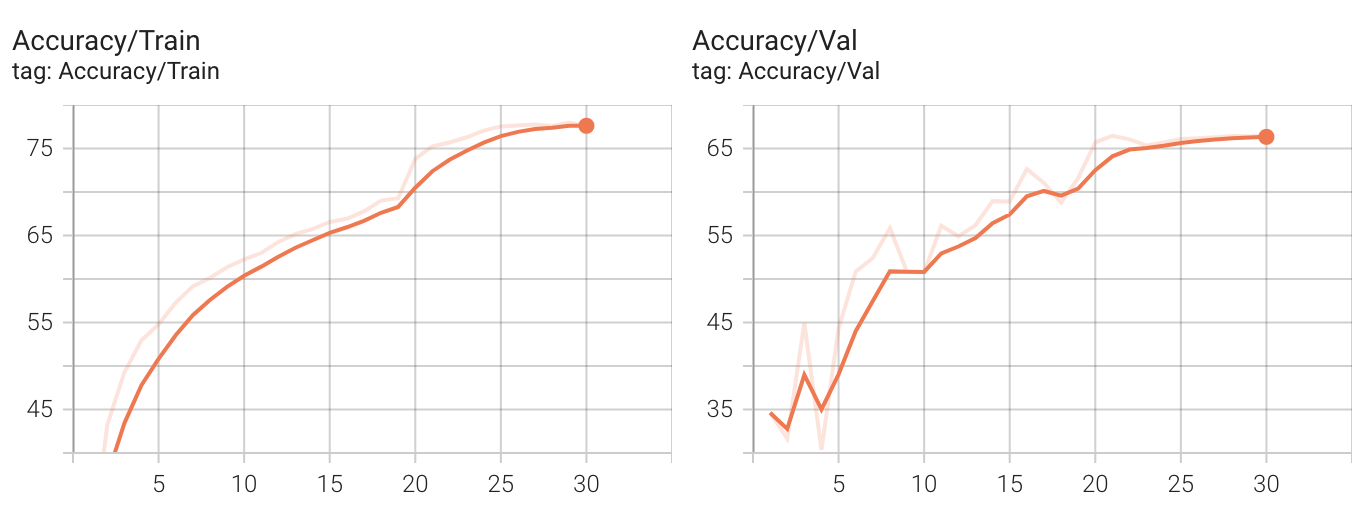


Figure 18 优化前ResNet18在FER2013上准确率图

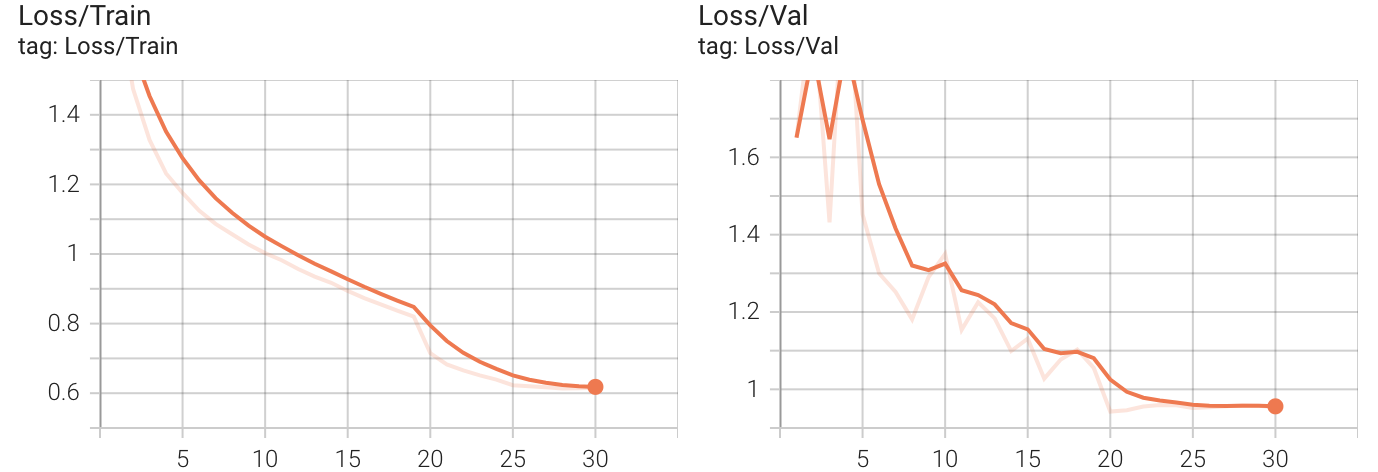


Figure 19 优化前ResNet18在FER2013上损失图

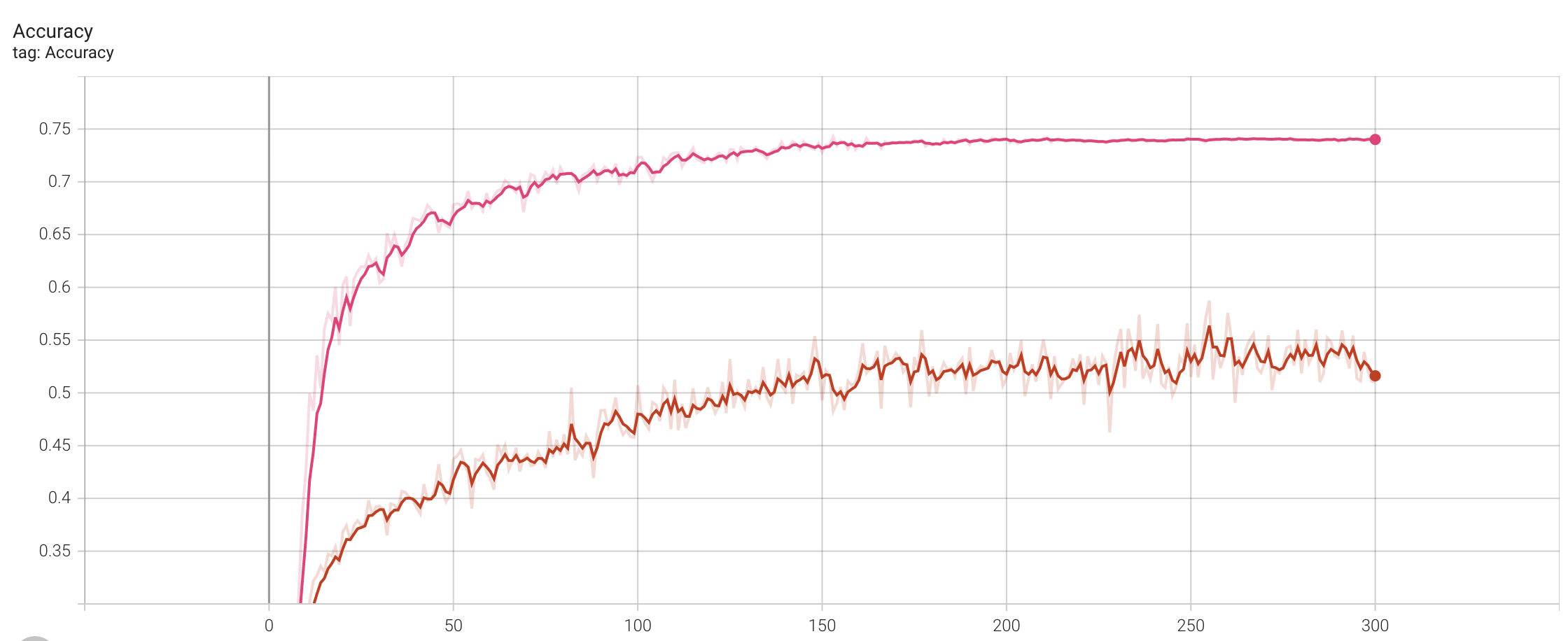


Figure 20 优化后ResNet18在FER2013上准确率图



Figure 21 优化后ResNet18在FER2013上损失图

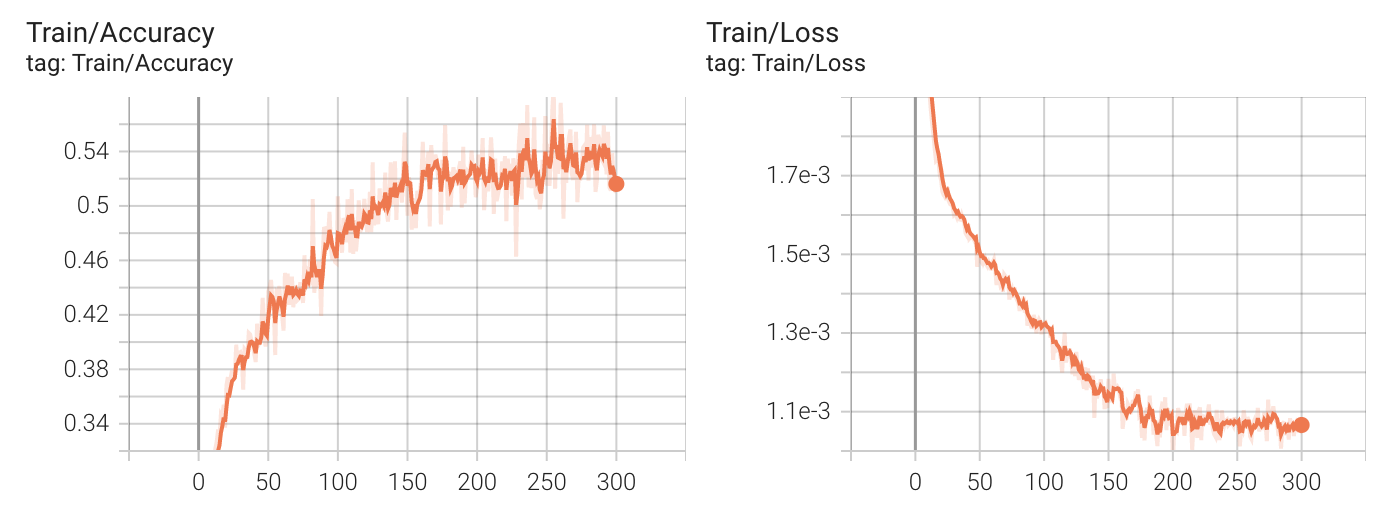


Figure 22 优化后ResNet18在FER2013上训练集训练图

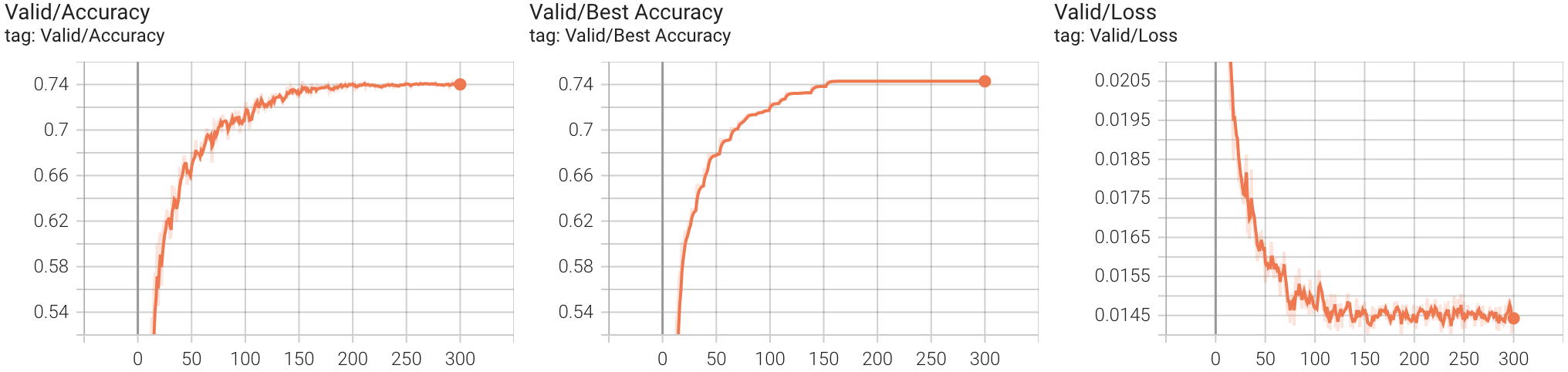
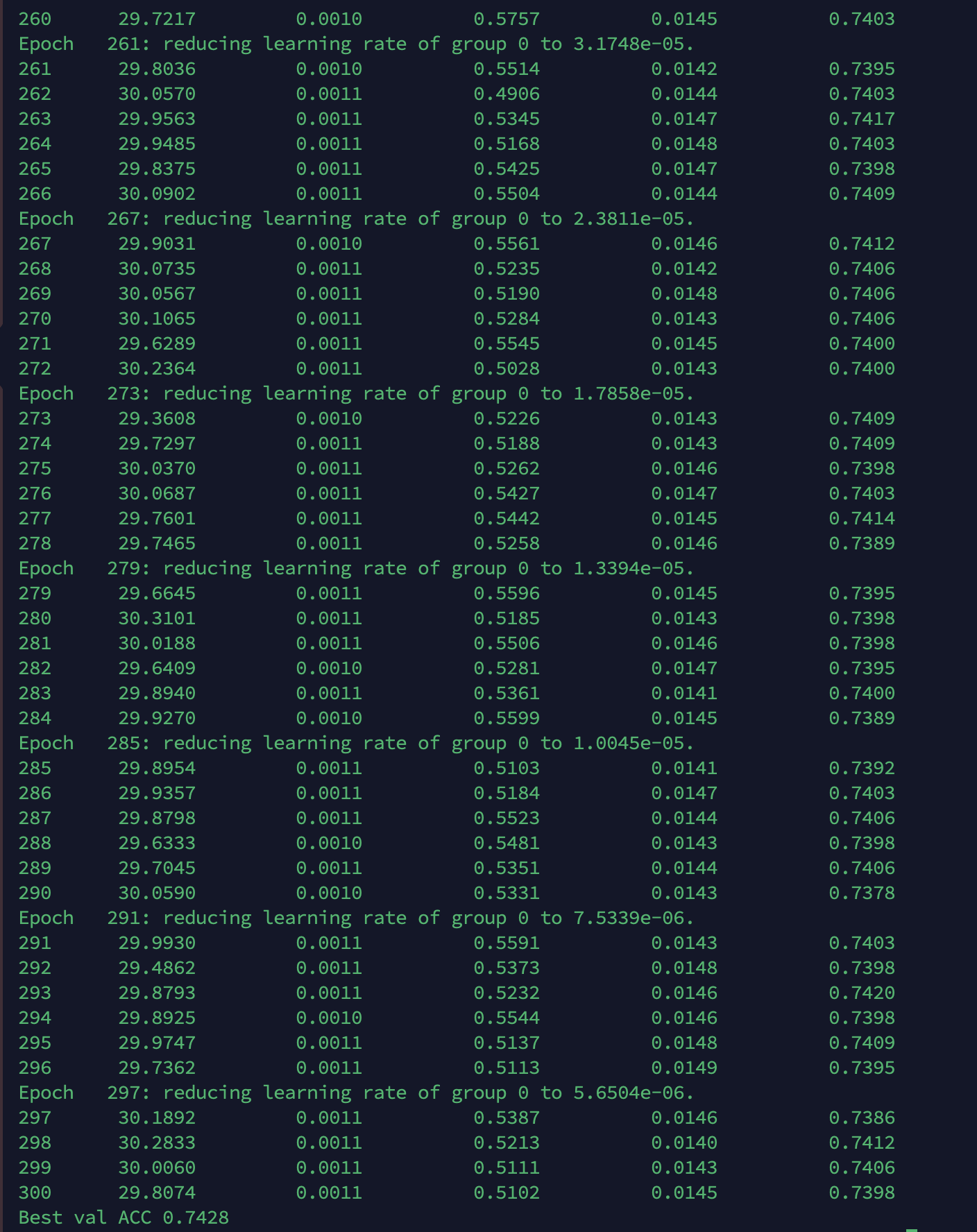


Figure 23 优化后ResNet18在FER2013上测试集训练图



# 第六章 结束语

# 附录

## Resnet18 结构

ResNet(

(conv1): Conv2d(1, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(layer1): Sequential(

(0): BasicBlock(

(conv1): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,track\_running\_stats=True)

(conv2): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,track\_running\_stats=True)

(shortcut): Sequential())

(1): BasicBlock(

(conv1): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,track\_running\_stats=True)

(conv2): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,track\_running\_stats=True)

(shortcut): Sequential())

)

(layer2): Sequential(

(0): BasicBlock(

(conv1): Conv2d(64, 128, kernel\_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(128, eps=1e-05,momentum=0.1,affine=True,track\_running\_stats=True)

(conv2): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(128, eps=1e-05,momentum=0.1,affine=True,track\_running\_stats=True)

(shortcut): Sequential(

(0): Conv2d(64, 128, kernel\_size=(1, 1), stride=(2, 2), bias=False)

(1): BatchNorm2d(128, eps=1e-05,momentum=0.1,affine=True,track\_running\_stats=True)))

(1): BasicBlock(

(conv1): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(128, eps=1e-05,momentum=0.1,affine=True,track\_running\_stats=True)

(conv2): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(128, eps=1e-05,momentum=0.1,affine=True,track\_running\_stats=True)

(shortcut): Sequential())

)

(layer3): Sequential(

(0): BasicBlock(

(conv1): Conv2d(128, 256, kernel\_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(256, eps=1e-05,momentum=0.1,affine=True,track\_running\_stats=True)

(conv2): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(256, eps=1e-05,momentum=0.1,affine=True,track\_running\_stats=True)

(shortcut): Sequential(

(0): Conv2d(128, 256, kernel\_size=(1, 1), stride=(2, 2), bias=False)

(1): BatchNorm2d(256, eps=1e-05,momentum=0.1,affine=True,track\_running\_stats=True)))

(1): BasicBlock(

(conv1): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(256, eps=1e-05,momentum=0.1,affine=True,track\_running\_stats=True)

(conv2): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(256, eps=1e-05,momentum=0.1,affine=True,track\_running\_stats=True)

(shortcut): Sequential())

)

(layer4): Sequential(

(0): BasicBlock(

(conv1): Conv2d(256, 512, kernel\_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(512, eps=1e-05,momentum=0.1,affine=True,track\_running\_stats=True)

(conv2): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(512, eps=1e-05,momentum=0.1,affine=True,track\_running\_stats=True)

(shortcut): Sequential(

(0): Conv2d(256, 512, kernel\_size=(1, 1), stride=(2, 2), bias=False)

(1): BatchNorm2d(512, eps=1e-05,momentum=0.1,affine=True,track\_running\_stats=True)))

(1): BasicBlock(

(conv1): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(512, eps=1e-05,momentum=0.1,affine=True,track\_running\_stats=True)

(conv2): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(512, eps=1e-05,momentum=0.1,affine=True,track\_running\_stats=True)

(shortcut): Sequential())

)

(linear): Linear(in\_features=512, out\_features=7, bias=True))