西安交通大學

XI'AN JIAOTONG UNIVERSITY

人工智能大作业

图像分类 CNN

课程名称：	人工智能
姓名：	计算机 001 班曾锦程
学院：	电信学部
专业：	计算机科学与技术
学号：	2203613040
指导老师：	相明

2023 年 2 月 10 日

一、 问题描述

1. 图像分类问题背景

当今互联网和互联网技术正高速发展，随着微信、陌陌等新型娱乐工具的兴起以及平板电脑、智能手机等配备数字摄像头的手持终端设备的广泛推广普及，网络上图像数据急剧增加。这些图像覆盖了人类生活的各个方面，传播了大量有用的信息。近年来，如何快速有效地提取和分析这些图像所包含的语义信息并运用到实际问题中已经成为图像分类和识别、图像搜索、图像理解和分析等领域的研究重点。

随着图像研究的深入，模糊集方法、决策树分类法、基于知识的分类方法、机器学习方法等智能图像分类算法不断涌现。模糊集方法是通过经验得到的，虽然可以很好地处理一些比较模糊的问题，但存在一定的不确定性与主观性。决策树分类法是一种比较好的分类方法，通过效仿人类思想而得出，但也存在依赖度大、分类决策规则与专家系统不易结合、不能充分利用分类对象的空间特征等缺点。基于知识的分类方法不具备自适应能力，当经验和知识受到外界因素干扰时，该方法的分类效果较差。因此，相较于上述三种方法，机器学习的方法因其理论深厚、效果显著，日益受到学者们的关注。而深度学习算法作为机器学习一个重要的分支，其优异的分析建模能力也为图像分类技术的研究提供了新方向。

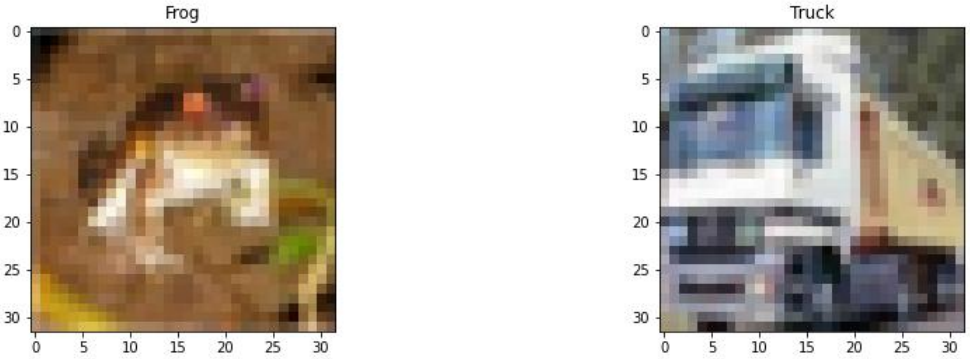


图 1: Cifar10 图像分类

2. 深度学习方法概述

随着互联网技术的发展，海量的无标签的图像数据涌现在网络之上，而深度学习算法利用多层的非线性变换可以从这些无标签数据中提取出抽象的特征表示用于图像分类，因此，深度学习算法已经成为了图像处理领域的研究热点。

在卷积提出以前，主要是通过将图片的所有像素点数据压成一维向量来训练神经网络，但是这样会导致参数过多，容易形成过拟合，并且训练速度非常慢。

最近几年随着深度学习神经网络飞速发展，许多基于卷积计算的神经网络涌现，通过不断改进，网络深度甚至达到了 152 层，有效减少了梯度消失和梯度爆炸的问题，准确率大大提高。

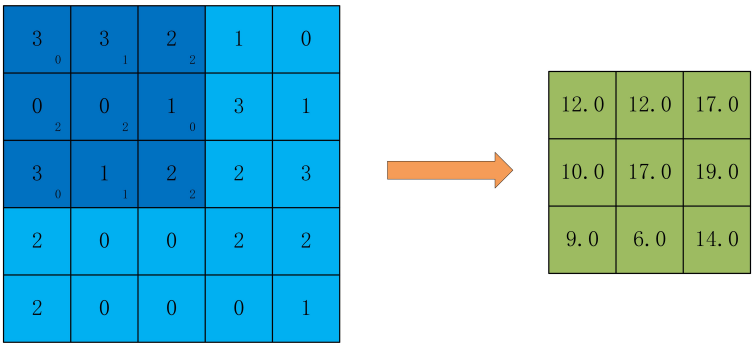


图 2: 卷积计算

二、 算法描述

1. LeNet5

LeNet5 的结构是：
(1) 第一、二层均为卷积 + sigmoid + 最大池化
(2) 第三、四层均为全连接层
(3) 最后一层为 softmax 输出

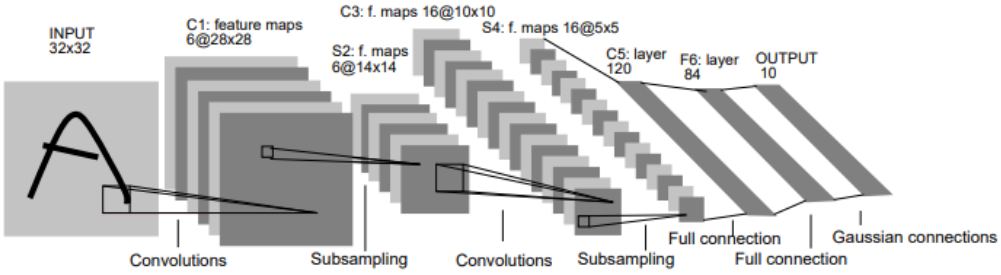


图 3: LeNet5

激活函数使用的是 Sigmoid 函数:

$$S(x) = \frac{1}{1+e^{-x}} \tag{1}$$

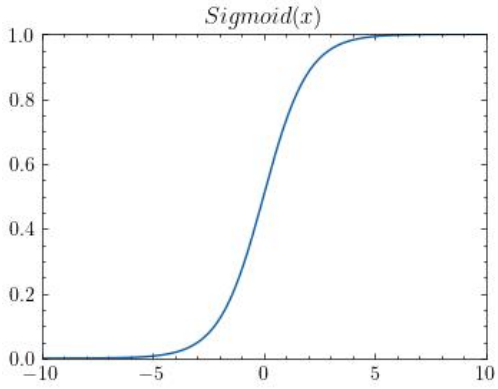


图 4: Sigmoid 函数

2. AlexNet8

AlexNet8 的结构是：
(1) 第一、二层均为卷积 +LRN(现用 BatchNormalization 代替)+relu+ 最大池化
(2) 第三、四层均为卷积 +relu
(3) 第五层为卷积 +relu+ 最大池化
(4) 第六、七层均为全连接 +relu+dropout
(5) 第八层为 softmax 输出

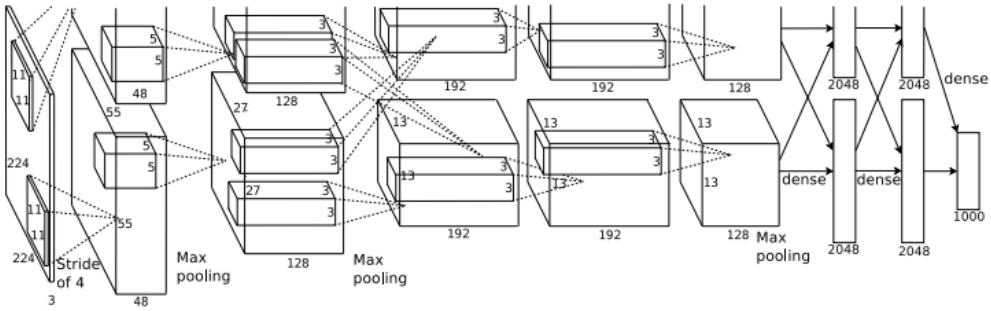


图 5: AlexNet8

AlexNet 的特点主要是：

(1)　使用了非线性激活函数：ReLU

传统的神经网络普遍使用 Sigmoid 或者 tanh 等非线性函数作为激励函数，然而它们容易出现梯度弥散或梯度饱和的情况。以 Sigmoid 函数为例，如图 4 所示，当输入的值非常大或者非常小的时候，这些神经元的梯度接近于 0（梯度饱和现象），如果输入的初始值很大的话，梯度在反向传播时因为需要乘上一个 Sigmoid 导数，会造成梯度越来越小，导致网络变的很难学习。

AlexNet 使用的激活函数是 ReLU:

$$F(x) = max(0, x) \tag{2}$$

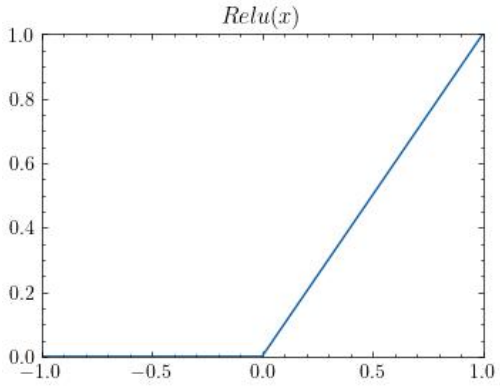
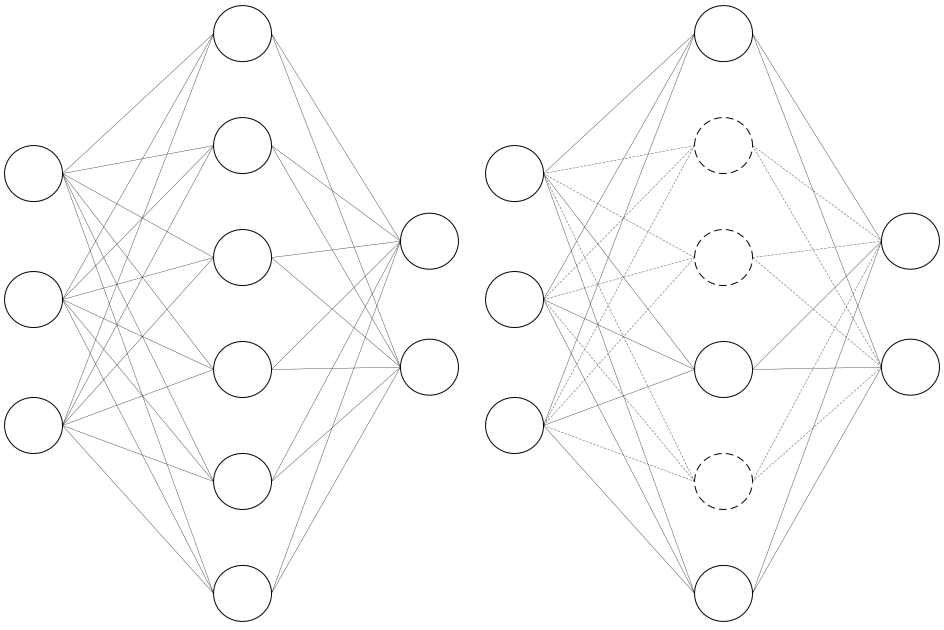


图 6: Relu 函数

可以看出 Relu 函数很好的解决了梯度饱和的现象，但是在 x 为负值时，由于梯度值为 0，也容易出现难以学习的情况。

(2)　随机失活：Dropout

引入 Dropout 主要是为了防止网络在训练过程中由于参数冗余出现的过拟合现象。在神经网络中 Dropout 通过修改神经网络本身结构来实现，对于某一层的神经元，通过定义的概率将神经元置为 0，这个神经元就不参与前向和后向传播，就如同在网络中被删除了一样，同时保持输入层与输出层神经元的个数不变，然后按照神经网络的学习方法进行参数更新。在下一次迭代中，又重新随机删除一些神经元（置为 0），直至训练结束。



(a) before　　(b) after

图 7: Dropout 前后

(3)　数据扩充：Data augmentation

由于神经网络算法是基于数据驱动的，因此，有一种观点认为神经网络是靠数据喂出来的，如果能够增加训练数据，提供海量数据进行训练，则能够有效提升算法的准确率，因为这样可以避免过拟合，从而可以进一步增大、加深网络结构。而当训练数据有限时，可以通过一些变换从已有的训练数据集中生成一些新的数据，以快速地扩充训练数据。其中，最简单、通用的图像数据变形的方式：水平翻转图像，从原始图像中随机裁剪、平移变换，颜色、光照变换。

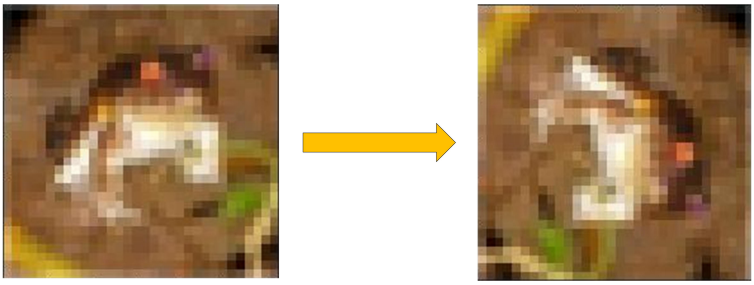


图 8: 旋转

3. VGG16

VGG16 的结构是 (将卷积 +BatchNormalization+relu 看作 CBA)：
(1) 第一、二层和三、四层均为 CBA+CBA+ 最大池化 +Dropout
(2) 中间层 (3 为单位) 为 CBA+CBA+CBA+ 最大池化 +Dropout
(3) 倒数二、三层为全连接 +Dropout
(4) 最后一层为 softmax 输出

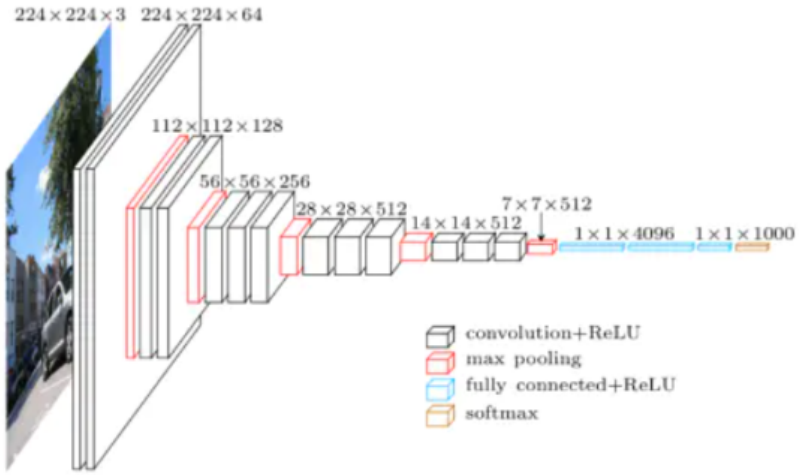


图 9: VGG16

VGG16 的特点主要是：

(1) 结构简洁

VGG 由 5 层卷积层、3 层全连接层、softmax 输出层构成，层与层之间使用 max-pooling（最大化池）分开，所有隐层的激活单元都采用 ReLU 函数。

(2) 小卷积核和多卷积子层

VGG 使用多个较小卷积核（3x3）的卷积层的堆叠代替一个卷积核较大的卷积层。这样在保证感受野的同时，一方面可以减少参数，另一方面相当于进行了更多的非线性映射，可以增加网络的拟合，表达，特征提取能力。

小卷积核是 VGG 的一个重要特点，虽然 VGG 是在模仿 AlexNet 的网络结构，但没有采用 AlexNet 中比较大的卷积核尺寸（如 7x7），而是通过降低卷积核的大小（3x3），增加卷积子层数来达到同样的性能

VGG 的作者认为两个 3x3 的卷积堆叠获得的感受野大小，相当一个 5x5 的卷积；而 3 个 3x3 卷积的堆叠获取到的感受野相当于一个 7x7 的卷积。这样可以增加非线性映射，也能很好地减少参数（例如 7x7 的参数为 49 个，而 3 个 3x3 的参数为 27）

(3) 小池化核

相比 AlexNet 的 3x3 的池化核，VGG 全部采用 2x2 的池化核。

(4) 通道数多

VGG 网络第一层的通道数为 64，后面每层都进行了翻倍，最多到 512 个通道，通道数的增加，使得更多的信息可以被提取出来。

(5) 层数更深、特征图更宽

由于卷积核专注于扩大通道数、池化专注于缩小宽和高，使得模型架构上更深更宽的同时，控制了计算量的增加规模

4. Inception10/GooLeNet

Inception10 的结构是:
(1) 第一层为 CBA
(2) 中间八层为四个 InceptionBlock
(3) 第九、十层之间用 GlobalAveragePooling 代替全连接层
(4) 第十层为 softmax 输出
Inception10 的主要特点是:

(1) InceptionBlock

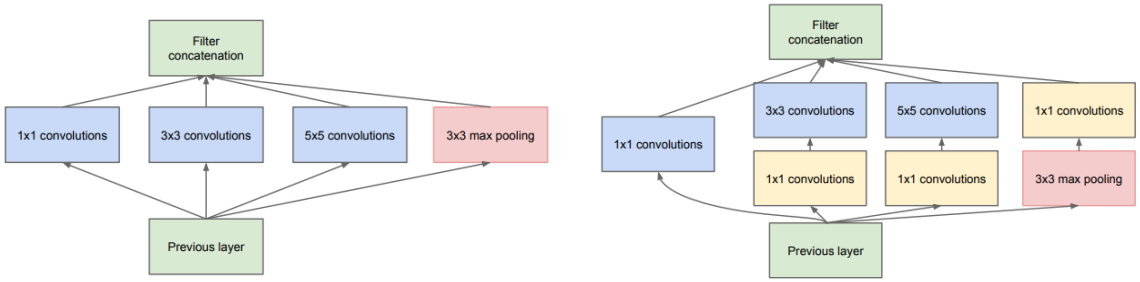
对于图 10 中 (a)，解释如下:
(1) 卷积核的大小在神经网络里是一种超参数，没有一种严格的数学理论证明那种尺寸的卷积核更适合提取特征，因此 Inception10 选择使用多种卷积核所得结果拼接。

(2) 采用不同大小的卷积核意味着不同大小的感受野，最后拼接意味着不同尺度特征的融合。之所以卷积核大小采用 1、3 和 5，主要是为了方便对齐。设定卷积步长 stride=1 之后，只要分别设定 pad=0、1、2，那么卷积之后便可以得到相同维度的特征，然后这些特征就可以直接拼接在一起了。

(3) 其中加入了池化层，原论文多种实验测试表明池化层十分有效。

对于图 10 中 (b)，则是对图 10 中 (a) 进行的降维处理:

例如：上一层的输出为 100x100x128，经过具有 256 个输出的 5x5 卷积层之后 (stride=1，pad=2)，输出数据为 100x100x256。其中，卷积层的参数为 128x5x5x256。假如上一层输出先经过具有 32 个输出的 1x1 卷积层，再经过具有 256 个输出的 5x5 卷积层，那么最终的输出数据仍为为 100x100x256，但卷积参数量已经减少为 128x1x1x32 + 32x5x5x256，大约减少了 4 倍。



(a) Inception module, naive version　　(b) Inception module with dimension reductions

图 10: InceptionBlock

5. ResNet18

ResNet18 的结构是:

(1) 第一层为 CBA

(2) 中间层由 8 个残差块组成

(3) 倒数两层用 GlobalAveragePooling 代替全连接层

(4) 最后一层为 softmax 输出

ResNet18 的主要特点是：

(1) Residual Learning

想要提高模型的效率，提高网络深度是一个有效手段，但是由于层数不断增多，很容易导致梯度爆炸 (输入的模值大于 1) 或者梯度消失 (输入的模值小于 1) 的现象，然而 Resnet 使用了如图 11 的残差块，就可以把问题转化为学习一个残差函数 $F(x)$，也相当于把浅层网络的值传向深层网络，降低随网络深度增加而导致的退化。

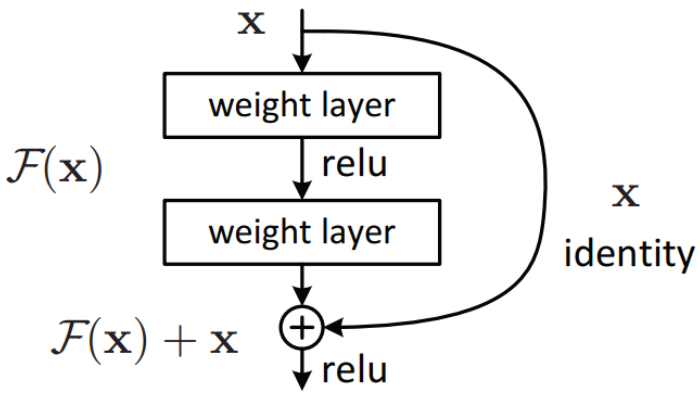


图 11: Residual Block

三、 不同算法性能对比

1. Cifar10 数据集介绍

CIFAR-10 是由 Hinton 的学生 Alex Krizhevsky 和 Ilya Sutskever 整理的一个用于识别普适物体的小型数据集。一共包含 10 个类别的 RGB 彩色图片：飞机（airplane）、汽车（automobile）、鸟类（bird）、猫（cat）、鹿（deer）、狗（dog）、蛙类（frog）、马（horse）、船（ship）和卡车（truck）。图片的尺寸为 32×32 ，数据集中一共有 50000 张训练图片和 10000 张测试图片。

与 MNIST 数据集中目比，CIFAR-10 具有以下不同点：

(1) CIFAR-10 是 3 通道的彩色 RGB 图像，而 MNIST 是灰度图像。

(2) CIFAR-10 的图片尺寸为 32×32，而 MNIST 的图片尺寸为 28×28，比 MNIST 稍大。

(3) 相比于手写字符，CIFAR-10 含有的是现实世界中真实的物体，不仅噪声很大，而且物体的比例、特征都不尽相同，这为识别带来很大困难。直接的线性模型如 Softmax 在 CIFAR-10 上表现得很差。

2. 准确率和 Loss 对比

将五种网络使用 Cifar10 数据集训练, 调用 Tensorflow 中的 **`model.compile()`** 和 **`model.fit()`** 进行训练

```
model.compile(optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
metrics=['sparse_categorical_accuracy'])
model.fit(x_train, y_train, batch_size=32, epochs=5, validation_data=(x_test, y_test),
    validation_freq=1)
```

得到以下结果：

Net	LeNet5	AlexNet8	VGG16	Inception10	Resnet18
loss	1.4818	1.0084	0.8670	0.8606	0.4046
acc	0.4575	0.6571	0.7089	0.6940	0.8712

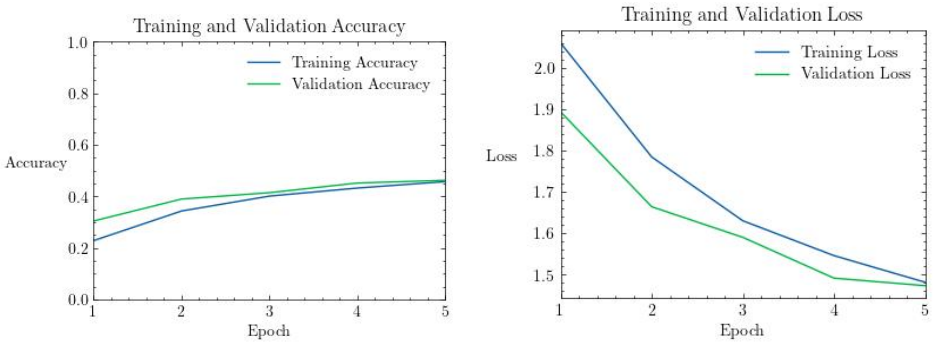
表 1: 训练集 acc 和 loss

Net	LeNet5	AlexNet8	VGG16	Inception10	Resnet18
loss	1.4734	1.0627	1.0066	0.8953	0.6218
acc	0.4627	0.6351	0.6791	0.6835	0.8026

表 2: 测试集 acc 和 loss

3. 训练集 acc 和 loss 变化曲线与测试集 acc 和 loss 变化曲线

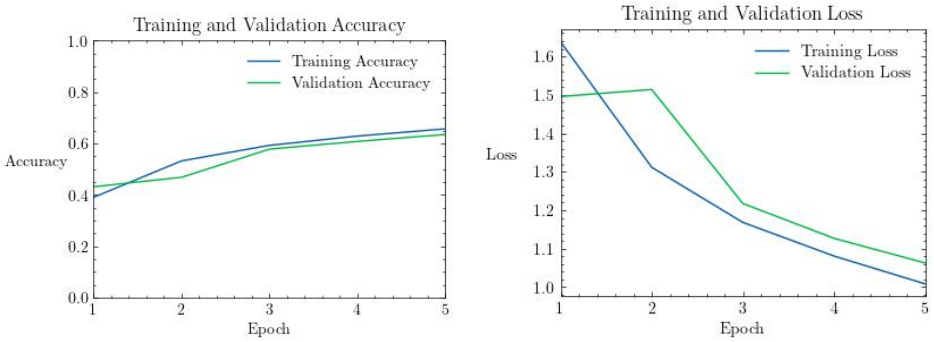
(1) LeNet5



(a) acc　　(b) Loss

图 12: LeNet5

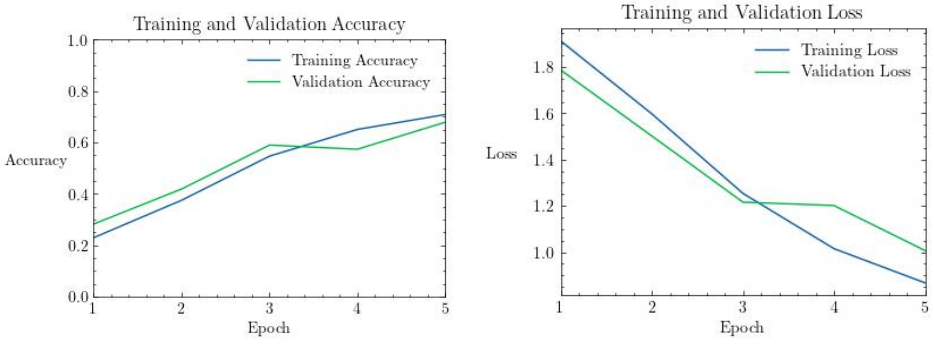
(2) AlexNet8



(a) acc　　(b) loss

图 13: AlexNet8

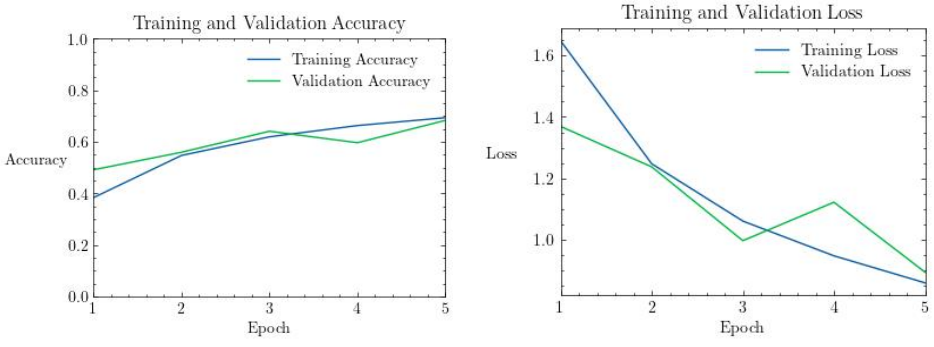
(3) VGG16



(a) acc　　(b) loss

图 14: VGG16

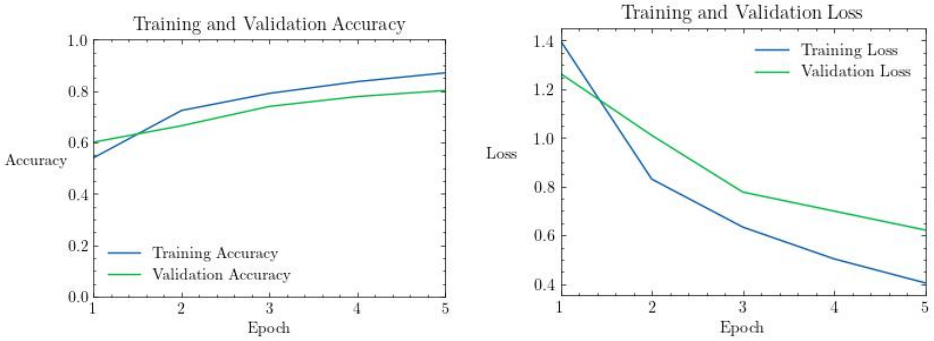
(4) Inception10



(a) acc　　(b) loss

图 15: Inception10

(5) Resnet18



(a) acc　　(b) loss

图 16: Resnet18

4. 训练时间对比

本次训练使用 Tensorflow2-cpu 版本进行训练，cpu 型号为 R7-4800H，运行时间如下表：

Net	LeNet5	AlexNet8	VGG16	Inception10	Resnet18
Epoch1	15s	375s	1213s	148s	1332s
Epoch2	14s	346s	1132s	141s	1332s
Epoch3	14s	351s	1097s	139s	1322s
Epoch4	14s	348s	1252s	138s	1343s
Epoch5	14s	349s	1210s	141s	1341s

表 3: 测试集 acc 和 loss

5. 总结

(1) 从 acc 与 loss 变化曲线来看

训练集和测试集的 acc 总的趋势上都是不断提高，训练集和测试集的 loss 总的趋势上都是不断减少，其中部分网络会有抖动现象，如果 Epoch 过高，则可能出现 acc 峰值和 loss 谷值，再增加则会导致过拟合问题。

(2) 从 Cifar10 数据集训练效果来看

$$Resnet18 > Inception10 > VGG16 > Alexnet8 > LeNet5$$

(3) 从训练速度上来看

可以看出 Inception10 训练速度很突出，说明 InceptionBlock 降维的方法十分有效，将 Inception 与 Residual Block 结合将会获得更好的效果！

四、 源代码

Image Classification

February 9, 2023

0.0.1 Tensorflow

[1]:
```
import tensorflow as tf
import os
import numpy as np
from matplotlib import pyplot as plt
from tensorflow.keras.layers import Conv2D, BatchNormalization, Activation,␣
↪MaxPool2D, Dropout, Flatten, Dense
from tensorflow.keras import Model
import scienceplots
```

0.0.2 Cifar10

[2]:
```
cifar10 = tf.keras.datasets.cifar10
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

0.0.3 Model1:LeNet

[3]:
```
class LeNet5(Model):
    def __init__(self):
        super(LeNet5, self).__init__()
        self.c1 = Conv2D(filters=6, kernel_size=(5, 5),
                         activation='sigmoid')
        self.p1 = MaxPool2D(pool_size=(2, 2), strides=2)

        self.c2 = Conv2D(filters=16, kernel_size=(5, 5),
                         activation='sigmoid')
        self.p2 = MaxPool2D(pool_size=(2, 2), strides=2)

        self.flatten = Flatten()
        self.f1 = Dense(120, activation='sigmoid')
        self.f2 = Dense(84, activation='sigmoid')
        self.f3 = Dense(10, activation='softmax')

    def call(self, x):
        x = self.c1(x)
        x = self.p1(x)
```

```
        x = self.c2(x)
        x = self.p2(x)

        x = self.flatten(x)
        x = self.f1(x)
        x = self.f2(x)
        y = self.f3(x)
        return y
```

[4]:
```
model = LeNet5()
model.compile(optimizer='adam',
              loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=False),
              metrics=['sparse_categorical_accuracy'])
```

[5]:
```
history = model.fit(x_train, y_train, batch_size=32, epochs=5,␣
 ↪validation_data=(x_test, y_test), validation_freq=1)
```

```
Epoch 1/5
1563/1563 [==============================] - 15s 9ms/step - loss: 2.0601 -
sparse_categorical_accuracy: 0.2284 - val_loss: 1.8937 -
val_sparse_categorical_accuracy: 0.3046
Epoch 2/5
1563/1563 [==============================] - 14s 9ms/step - loss: 1.7846 -
sparse_categorical_accuracy: 0.3435 - val_loss: 1.6643 -
val_sparse_categorical_accuracy: 0.3899
Epoch 3/5
1563/1563 [==============================] - 14s 9ms/step - loss: 1.6304 -
sparse_categorical_accuracy: 0.4013 - val_loss: 1.5903 -
val_sparse_categorical_accuracy: 0.4145
Epoch 4/5
1563/1563 [==============================] - 14s 9ms/step - loss: 1.5462 -
sparse_categorical_accuracy: 0.4326 - val_loss: 1.4918 -
val_sparse_categorical_accuracy: 0.4520
Epoch 5/5
1563/1563 [==============================] - 14s 9ms/step - loss: 1.4818 -
sparse_categorical_accuracy: 0.4575 - val_loss: 1.4734 -
val_sparse_categorical_accuracy: 0.4627
```

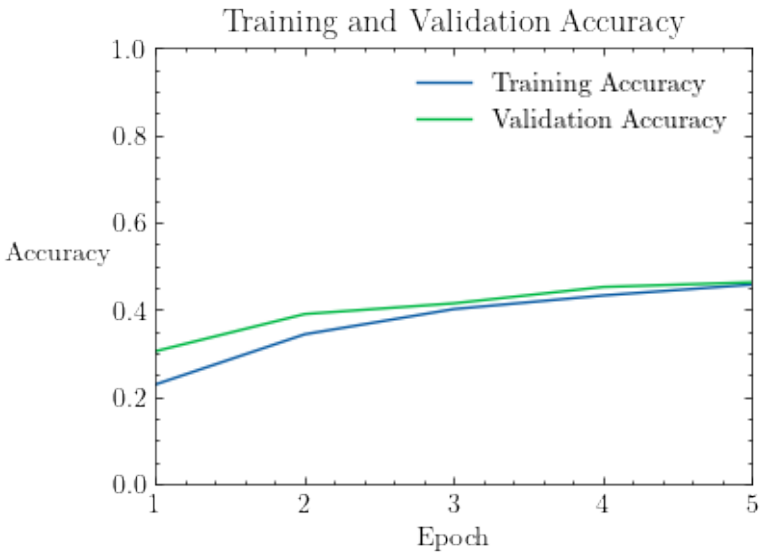
[6]:
```
model.summary()
```

```
Model: "le_net5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             multiple                  456
```

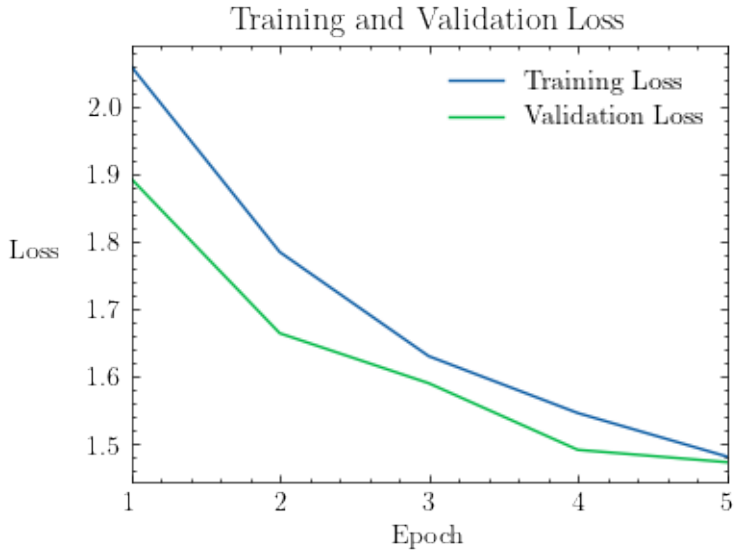
```
max_pooling2d (MaxPooling2D  multiple                  0
)

conv2d_1 (Conv2D)            multiple                  2416

max_pooling2d_1 (MaxPooling  multiple                  0
2D)

flatten (Flatten)            multiple                  0

dense (Dense)                multiple                  48120

dense_1 (Dense)              multiple                  10164

dense_2 (Dense)              multiple                  850

=================================================================
Total params: 62,006
Trainable params: 62,006
Non-trainable params: 0
_________________________________________________________________
```

[7]:
```
acc = history.history['sparse_categorical_accuracy']
val_acc = history.history['val_sparse_categorical_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
```

[8]:
```
a = [0,1,2,3,4]
labels = ['1', '2', '3', '4', '5']
plt.figure(figsize=(4,3),dpi=100)
plt.style.use('science')
plt.ylim(0,1)
plt.xlim(0,4)
plt.xticks(a,labels)
plt.ylabel('Accuracy',rotation=0,labelpad=20)
plt.xlabel('Epoch')
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.savefig('C:/Users/Lenovo/Desktop/  -ResNet /figures/1.jpg')
plt.show()
```



[9]:
```
plt.figure(figsize=(4,3),dpi=100)
plt.style.use('science')
plt.xlim(0,4)
plt.ylabel('Loss',rotation=0,labelpad=20)
plt.xlabel('Epoch')
plt.xticks(a,labels)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.savefig('C:/Users/Lenovo/Desktop/  -ResNet /figures/2.jpg')
plt.show()
```



0.0.4 Model2:AlexNet8

[10]:
```
class AlexNet8(Model):
    def __init__(self):
        super(AlexNet8, self).__init__()
        self.c1 = Conv2D(filters=96, kernel_size=(3, 3))
        self.b1 = BatchNormalization()
        self.a1 = Activation('relu')
        self.p1 = MaxPool2D(pool_size=(3, 3), strides=2)

        self.c2 = Conv2D(filters=256, kernel_size=(3, 3))
        self.b2 = BatchNormalization()
        self.a2 = Activation('relu')
        self.p2 = MaxPool2D(pool_size=(3, 3), strides=2)

        self.c3 = Conv2D(filters=384, kernel_size=(3, 3), padding='same',
                         activation='relu')

        self.c4 = Conv2D(filters=384, kernel_size=(3, 3), padding='same',
                         activation='relu')

        self.c5 = Conv2D(filters=256, kernel_size=(3, 3), padding='same',
                         activation='relu')
        self.p3 = MaxPool2D(pool_size=(3, 3), strides=2)

        self.flatten = Flatten()
        self.f1 = Dense(2048, activation='relu')
        self.d1 = Dropout(0.5)
        self.f2 = Dense(2048, activation='relu')
        self.d2 = Dropout(0.5)
        self.f3 = Dense(10, activation='softmax')

    def call(self, x):
        x = self.c1(x)
        x = self.b1(x)
        x = self.a1(x)
        x = self.p1(x)

        x = self.c2(x)
        x = self.b2(x)
        x = self.a2(x)
        x = self.p2(x)

        x = self.c3(x)
```

```
        x = self.c4(x)

        x = self.c5(x)
        x = self.p3(x)

        x = self.flatten(x)
        x = self.f1(x)
        x = self.d1(x)
        x = self.f2(x)
        x = self.d2(x)
        y = self.f3(x)
        return y
```

[11]:
```
model = AlexNet8()
```

[12]:
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=False),
              metrics=['sparse_categorical_accuracy'])
```

[13]:
```
history = model.fit(x_train, y_train, batch_size=32, epochs=5,␣
 ↪validation_data=(x_test, y_test), validation_freq=1,)
```

```
Epoch 1/5
1563/1563 [==============================] - 375s 239ms/step - loss: 1.6376 -
sparse_categorical_accuracy: 0.3905 - val_loss: 1.4955 -
val_sparse_categorical_accuracy: 0.4317
Epoch 2/5
1563/1563 [==============================] - 346s 221ms/step - loss: 1.3114 -
sparse_categorical_accuracy: 0.5323 - val_loss: 1.5138 -
val_sparse_categorical_accuracy: 0.4684
Epoch 3/5
1563/1563 [==============================] - 351s 225ms/step - loss: 1.1682 -
sparse_categorical_accuracy: 0.5928 - val_loss: 1.2174 -
val_sparse_categorical_accuracy: 0.5781
Epoch 4/5
1563/1563 [==============================] - 348s 223ms/step - loss: 1.0808 -
sparse_categorical_accuracy: 0.6289 - val_loss: 1.1269 -
val_sparse_categorical_accuracy: 0.6082
Epoch 5/5
1563/1563 [==============================] - 349s 223ms/step - loss: 1.0084 -
sparse_categorical_accuracy: 0.6571 - val_loss: 1.0627 -
val_sparse_categorical_accuracy: 0.6351
```

[14]:
```
model.summary()
```

```
Model: "alex_net8"
```

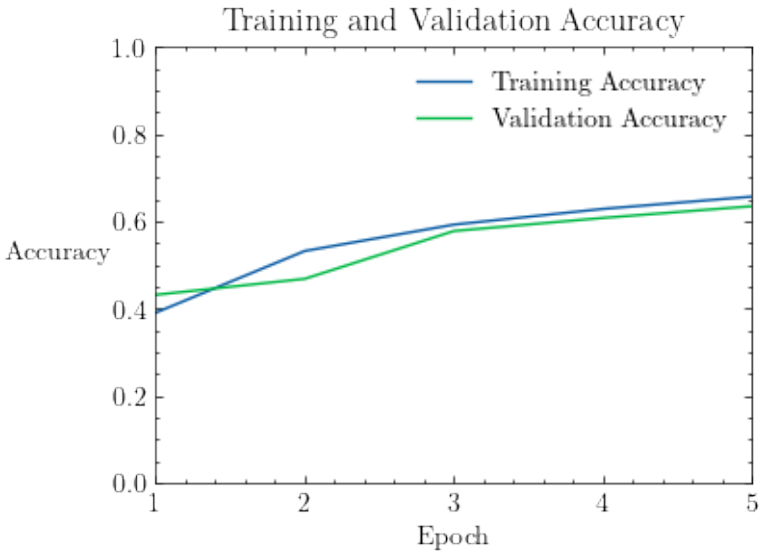
Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	multiple	2688
batch_normalization (BatchNormalization)	multiple	384
activation (Activation)	multiple	0
max_pooling2d_2 (MaxPooling2D)	multiple	0
conv2d_3 (Conv2D)	multiple	221440
batch_normalization_1 (BatchNormalization)	multiple	1024
activation_1 (Activation)	multiple	0
max_pooling2d_3 (MaxPooling2D)	multiple	0
conv2d_4 (Conv2D)	multiple	885120
conv2d_5 (Conv2D)	multiple	1327488
conv2d_6 (Conv2D)	multiple	884992
max_pooling2d_4 (MaxPooling2D)	multiple	0
flatten_1 (Flatten)	multiple	0
dense_3 (Dense)	multiple	2099200
dropout (Dropout)	multiple	0
dense_4 (Dense)	multiple	4196352
dropout_1 (Dropout)	multiple	0
dense_5 (Dense)	multiple	20490

Total params: 9,639,178
Trainable params: 9,638,474
Non-trainable params: 704

---

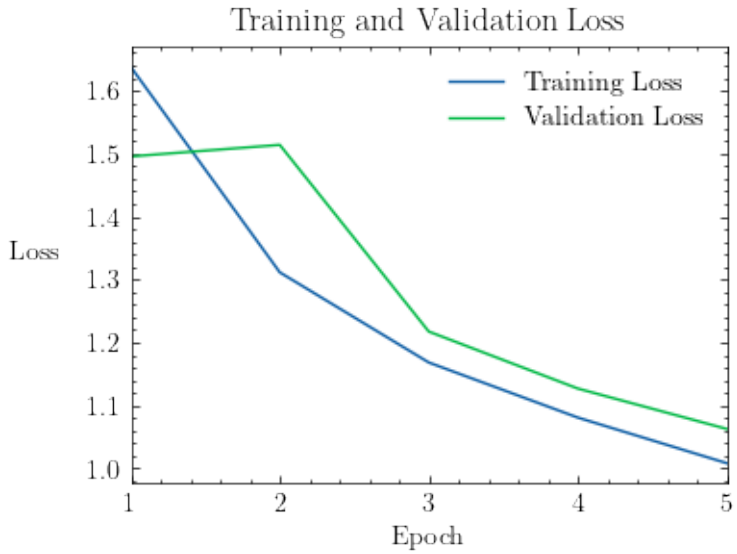
[15]:
```
acc = history.history['sparse_categorical_accuracy']
val_acc = history.history['val_sparse_categorical_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
```

[16]:
```
a = [0,1,2,3,4]
labels = ['1', '2', '3', '4', '5']
plt.figure(figsize=(4,3),dpi=100)
plt.style.use('science')
plt.ylim(0,1)
plt.xlim(0,4)
plt.xticks(a,labels)
plt.ylabel('Accuracy',rotation=0,labelpad=20)
plt.xlabel('Epoch')
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.savefig('C:/Users/Lenovo/Desktop/  -ResNet /figures/3.jpg')
plt.show()
```



[17]:
```
plt.figure(figsize=(4,3),dpi=100)
plt.style.use('science')
plt.xlim(0,4)
plt.ylabel('Loss',rotation=0,labelpad=20)
plt.xlabel('Epoch')
plt.xticks(a,labels)
```

```
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.savefig('C:/Users/Lenovo/Desktop/  -ResNet /figures/4.jpg')
plt.show()
```



0.0.5 Model3:VGG16

[18]:
```
class VGG16(Model):
    def __init__(self):
        super(VGG16, self).__init__()
        self.c1 = Conv2D(filters=64, kernel_size=(3, 3), padding='same')  #  1
        self.b1 = BatchNormalization()  # BN 1
        self.a1 = Activation('relu')  #  1
        self.c2 = Conv2D(filters=64, kernel_size=(3, 3), padding='same', )
        self.b2 = BatchNormalization()  # BN 1
        self.a2 = Activation('relu')  #  1
        self.p1 = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')
        self.d1 = Dropout(0.2)  # dropout

        self.c3 = Conv2D(filters=128, kernel_size=(3, 3), padding='same')
        self.b3 = BatchNormalization()  # BN 1
        self.a3 = Activation('relu')  #  1
        self.c4 = Conv2D(filters=128, kernel_size=(3, 3), padding='same')
        self.b4 = BatchNormalization()  # BN 1
        self.a4 = Activation('relu')  #  1
        self.p2 = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')
```

```
        self.d2 = Dropout(0.2)  # dropout

        self.c5 = Conv2D(filters=256, kernel_size=(3, 3), padding='same')
        self.b5 = BatchNormalization()  # BN 1
        self.a5 = Activation('relu')  #  1
        self.c6 = Conv2D(filters=256, kernel_size=(3, 3), padding='same')
        self.b6 = BatchNormalization()  # BN 1
        self.a6 = Activation('relu')  #  1
        self.c7 = Conv2D(filters=256, kernel_size=(3, 3), padding='same')
        self.b7 = BatchNormalization()
        self.a7 = Activation('relu')
        self.p3 = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')
        self.d3 = Dropout(0.2)

        self.c8 = Conv2D(filters=512, kernel_size=(3, 3), padding='same')
        self.b8 = BatchNormalization()  # BN 1
        self.a8 = Activation('relu')  #  1
        self.c9 = Conv2D(filters=512, kernel_size=(3, 3), padding='same')
        self.b9 = BatchNormalization()  # BN 1
        self.a9 = Activation('relu')  #  1
        self.c10 = Conv2D(filters=512, kernel_size=(3, 3), padding='same')
        self.b10 = BatchNormalization()
        self.a10 = Activation('relu')
        self.p4 = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')
        self.d4 = Dropout(0.2)

        self.c11 = Conv2D(filters=512, kernel_size=(3, 3), padding='same')
        self.b11 = BatchNormalization()  # BN 1
        self.a11 = Activation('relu')  #  1
        self.c12 = Conv2D(filters=512, kernel_size=(3, 3), padding='same')
        self.b12 = BatchNormalization()  # BN 1
        self.a12 = Activation('relu')  #  1
        self.c13 = Conv2D(filters=512, kernel_size=(3, 3), padding='same')
        self.b13 = BatchNormalization()
        self.a13 = Activation('relu')
        self.p5 = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')
        self.d5 = Dropout(0.2)

        self.flatten = Flatten()
        self.f1 = Dense(512, activation='relu')
        self.d6 = Dropout(0.2)
        self.f2 = Dense(512, activation='relu')
        self.d7 = Dropout(0.2)
        self.f3 = Dense(10, activation='softmax')

    def call(self, x):
        x = self.c1(x)
```

```
        x = self.b1(x)
        x = self.a1(x)
        x = self.c2(x)
        x = self.b2(x)
        x = self.a2(x)
        x = self.p1(x)
        x = self.d1(x)

        x = self.c3(x)
        x = self.b3(x)
        x = self.a3(x)
        x = self.c4(x)
        x = self.b4(x)
        x = self.a4(x)
        x = self.p2(x)
        x = self.d2(x)

        x = self.c5(x)
        x = self.b5(x)
        x = self.a5(x)
        x = self.c6(x)
        x = self.b6(x)
        x = self.a6(x)
        x = self.c7(x)
        x = self.b7(x)
        x = self.a7(x)
        x = self.p3(x)
        x = self.d3(x)

        x = self.c8(x)
        x = self.b8(x)
        x = self.a8(x)
        x = self.c9(x)
        x = self.b9(x)
        x = self.a9(x)
        x = self.c10(x)
        x = self.b10(x)
        x = self.a10(x)
        x = self.p4(x)
        x = self.d4(x)

        x = self.c11(x)
        x = self.b11(x)
        x = self.a11(x)
        x = self.c12(x)
        x = self.b12(x)
        x = self.a12(x)
```

```
        x = self.c13(x)
        x = self.b13(x)
        x = self.a13(x)
        x = self.p5(x)
        x = self.d5(x)

        x = self.flatten(x)
        x = self.f1(x)
        x = self.d6(x)
        x = self.f2(x)
        x = self.d7(x)
        y = self.f3(x)
        return y
```

[19]:
```
model = VGG16()
```

[20]:
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=False),
              metrics=['sparse_categorical_accuracy'])
```

[21]:
```
history = model.fit(x_train, y_train, batch_size=32, epochs=5,␣
 ↪validation_data=(x_test, y_test), validation_freq=1)
```

```
Epoch 1/5
1563/1563 [==============================] - 1213s 774ms/step - loss: 1.9133 -
sparse_categorical_accuracy: 0.2293 - val_loss: 1.7877 -
val_sparse_categorical_accuracy: 0.2825
Epoch 2/5
1563/1563 [==============================] - 1132s 724ms/step - loss: 1.5982 -
sparse_categorical_accuracy: 0.3751 - val_loss: 1.5024 -
val_sparse_categorical_accuracy: 0.4188
Epoch 3/5
1563/1563 [==============================] - 1097s 702ms/step - loss: 1.2542 -
sparse_categorical_accuracy: 0.5468 - val_loss: 1.2162 -
val_sparse_categorical_accuracy: 0.5899
Epoch 4/5
1563/1563 [==============================] - 1252s 801ms/step - loss: 1.0151 -
sparse_categorical_accuracy: 0.6509 - val_loss: 1.2014 -
val_sparse_categorical_accuracy: 0.5738
Epoch 5/5
1563/1563 [==============================] - 1210s 774ms/step - loss: 0.8670 -
sparse_categorical_accuracy: 0.7089 - val_loss: 1.0066 -
val_sparse_categorical_accuracy: 0.6791
```

[22]:
```
model.summary()
```

```
Model: "vgg16"
```

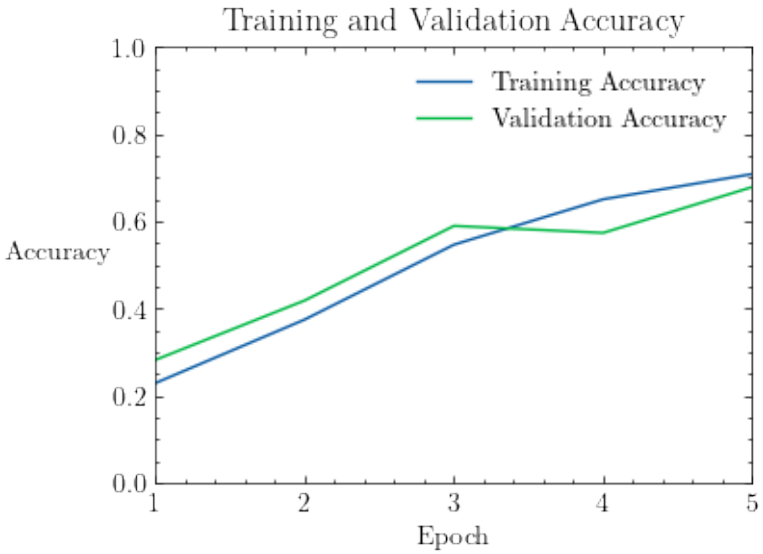
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            multiple                  1792

batch_normalization_2 (Batc  multiple                  256
hNormalization)

activation_2 (Activation)    multiple                  0

conv2d_8 (Conv2D)            multiple                  36928

batch_normalization_3 (Batc  multiple                  256
hNormalization)

activation_3 (Activation)    multiple                  0

max_pooling2d_5 (MaxPooling  multiple                  0
2D)

dropout_2 (Dropout)          multiple                  0

conv2d_9 (Conv2D)            multiple                  73856

batch_normalization_4 (Batc  multiple                  512
hNormalization)

activation_4 (Activation)    multiple                  0

conv2d_10 (Conv2D)           multiple                  147584

batch_normalization_5 (Batc  multiple                  512
hNormalization)

activation_5 (Activation)    multiple                  0

max_pooling2d_6 (MaxPooling  multiple                  0
2D)

dropout_3 (Dropout)          multiple                  0

conv2d_11 (Conv2D)           multiple                  295168

batch_normalization_6 (Batc  multiple                  1024
hNormalization)

activation_6 (Activation)    multiple                  0
```

```
conv2d_12 (Conv2D)          multiple                  590080

batch_normalization_7 (Batc  multiple                 1024
hNormalization)

activation_7 (Activation)   multiple                  0

conv2d_13 (Conv2D)          multiple                  590080

batch_normalization_8 (Batc  multiple                 1024
hNormalization)

activation_8 (Activation)   multiple                  0

max_pooling2d_7 (MaxPooling  multiple                 0
2D)

dropout_4 (Dropout)         multiple                  0

conv2d_14 (Conv2D)          multiple                  1180160

batch_normalization_9 (Batc  multiple                 2048
hNormalization)

activation_9 (Activation)   multiple                  0

conv2d_15 (Conv2D)          multiple                  2359808

batch_normalization_10 (Bat  multiple                 2048
chNormalization)

activation_10 (Activation)  multiple                  0

conv2d_16 (Conv2D)          multiple                  2359808

batch_normalization_11 (Bat  multiple                 2048
chNormalization)

activation_11 (Activation)  multiple                  0

max_pooling2d_8 (MaxPooling  multiple                 0
2D)

dropout_5 (Dropout)         multiple                  0

conv2d_17 (Conv2D)          multiple                  2359808

batch_normalization_12 (Bat  multiple                 2048
```

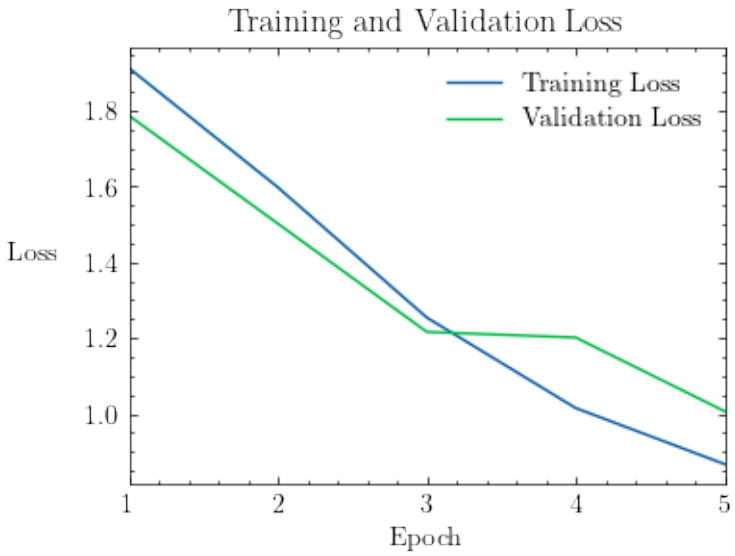
```
chNormalization)

activation_12 (Activation)  multiple                  0

conv2d_18 (Conv2D)          multiple                  2359808

batch_normalization_13 (Bat  multiple                 2048
chNormalization)

activation_13 (Activation)  multiple                  0

conv2d_19 (Conv2D)          multiple                  2359808

batch_normalization_14 (Bat  multiple                 2048
chNormalization)

activation_14 (Activation)  multiple                  0

max_pooling2d_9 (MaxPooling  multiple                 0
2D)

dropout_6 (Dropout)         multiple                  0

flatten_2 (Flatten)         multiple                  0

dense_6 (Dense)             multiple                  262656

dropout_7 (Dropout)         multiple                  0

dense_7 (Dense)             multiple                  262656

dropout_8 (Dropout)         multiple                  0

dense_8 (Dense)             multiple                  5130

=================================================================
Total params: 15,262,026
Trainable params: 15,253,578
Non-trainable params: 8,448
_________________________________________________________________
```

[23]:
```
acc = history.history['sparse_categorical_accuracy']
val_acc = history.history['val_sparse_categorical_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
```

[24]:
```
a = [0,1,2,3,4]
labels = ['1', '2', '3', '4', '5']
plt.figure(figsize=(4,3),dpi=100)
plt.style.use('science')
plt.ylim(0,1)
plt.xlim(0,4)
plt.xticks(a,labels)
plt.ylabel('Accuracy',rotation=0,labelpad=20)
plt.xlabel('Epoch')
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.savefig('C:/Users/Lenovo/Desktop/  -ResNet /figures/5.jpg')
plt.show()
```



[25]:
```
plt.figure(figsize=(4,3),dpi=100)
plt.style.use('science')
plt.xlim(0,4)
plt.ylabel('Loss',rotation=0,labelpad=20)
plt.xlabel('Epoch')
plt.xticks(a,labels)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.savefig('C:/Users/Lenovo/Desktop/  -ResNet /figures/6.jpg')
plt.show()
```



0.0.6 Model4:Inception10

[28]:
```
from tensorflow.keras.layers import GlobalAveragePooling2D
```

[29]:
```
class ConvBNRelu(Model):
    def __init__(self, ch, kernelsz=3, strides=1, padding='same'):
        super(ConvBNRelu, self).__init__()
        self.model = tf.keras.models.Sequential([
            Conv2D(ch, kernelsz, strides=strides, padding=padding),
            BatchNormalization(),
            Activation('relu')
        ])

    def call(self, x):
        x = self.model(x, training=False)␣
 ↪# training=False BN           training=True   batch          training=False
        return x


class InceptionBlk(Model):
    def __init__(self, ch, strides=1):
        super(InceptionBlk, self).__init__()
        self.ch = ch
        self.strides = strides
        self.c1 = ConvBNRelu(ch, kernelsz=1, strides=strides)
        self.c2_1 = ConvBNRelu(ch, kernelsz=1, strides=strides)
        self.c2_2 = ConvBNRelu(ch, kernelsz=3, strides=1)
        self.c3_1 = ConvBNRelu(ch, kernelsz=1, strides=strides)
```

```
        self.c3_2 = ConvBNRelu(ch, kernelsz=5, strides=1)
        self.p4_1 = MaxPool2D(3, strides=1, padding='same')
        self.c4_2 = ConvBNRelu(ch, kernelsz=1, strides=strides)

    def call(self, x):
        x1 = self.c1(x)
        x2_1 = self.c2_1(x)
        x2_2 = self.c2_2(x2_1)
        x3_1 = self.c3_1(x)
        x3_2 = self.c3_2(x3_1)
        x4_1 = self.p4_1(x)
        x4_2 = self.c4_2(x4_1)
        # concat along axis=channel
        x = tf.concat([x1, x2_2, x3_2, x4_2], axis=3)
        return x


class Inception10(Model):
    def __init__(self, num_blocks, num_classes, init_ch=16, **kwargs):
        super(Inception10, self).__init__(**kwargs)
        self.in_channels = init_ch
        self.out_channels = init_ch
        self.num_blocks = num_blocks
        self.init_ch = init_ch
        self.c1 = ConvBNRelu(init_ch)
        self.blocks = tf.keras.models.Sequential()
        for block_id in range(num_blocks):
            for layer_id in range(2):
                if layer_id == 0:
                    block = InceptionBlk(self.out_channels, strides=2)
                else:
                    block = InceptionBlk(self.out_channels, strides=1)
                self.blocks.add(block)
            # enlarger out_channels per block
            self.out_channels *= 2
        self.p1 = GlobalAveragePooling2D()
        self.f1 = Dense(num_classes, activation='softmax')

    def call(self, x):
        x = self.c1(x)
        x = self.blocks(x)
        x = self.p1(x)
        y = self.f1(x)
        return y
```

[30]:
```
model = Inception10(num_blocks=2, num_classes=10)
```

[31]:
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=False),
              metrics=['sparse_categorical_accuracy'])
```

[32]:
```
history = model.fit(x_train, y_train, batch_size=32, epochs=5,␣
 ↪validation_data=(x_test, y_test), validation_freq=1)
```

```
Epoch 1/5
1563/1563 [==============================] - 148s 92ms/step - loss: 1.6471 -
sparse_categorical_accuracy: 0.3839 - val_loss: 1.3691 -
val_sparse_categorical_accuracy: 0.4918
Epoch 2/5
1563/1563 [==============================] - 141s 90ms/step - loss: 1.2476 -
sparse_categorical_accuracy: 0.5476 - val_loss: 1.2376 -
val_sparse_categorical_accuracy: 0.5599
Epoch 3/5
1563/1563 [==============================] - 139s 89ms/step - loss: 1.0615 -
sparse_categorical_accuracy: 0.6198 - val_loss: 0.9977 -
val_sparse_categorical_accuracy: 0.6413
Epoch 4/5
1563/1563 [==============================] - 138s 88ms/step - loss: 0.9485 -
sparse_categorical_accuracy: 0.6634 - val_loss: 1.1226 -
val_sparse_categorical_accuracy: 0.5968
Epoch 5/5
1563/1563 [==============================] - 141s 90ms/step - loss: 0.8606 -
sparse_categorical_accuracy: 0.6940 - val_loss: 0.8953 -
val_sparse_categorical_accuracy: 0.6835
```

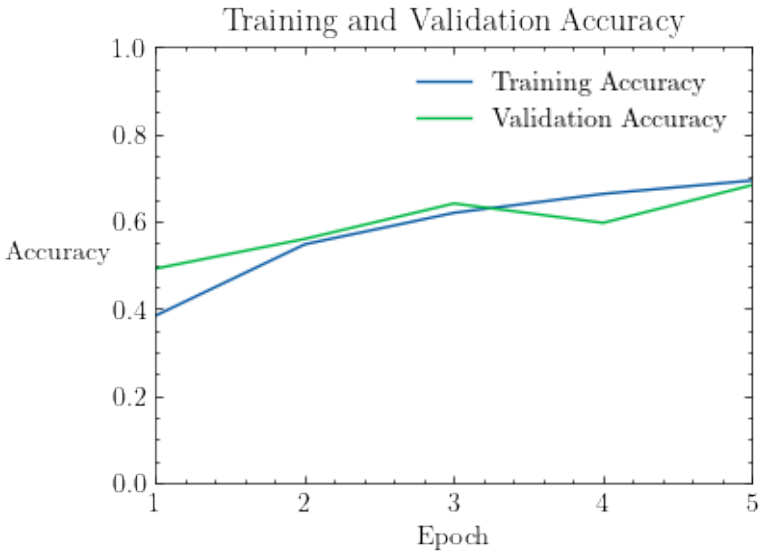
[33]:
```
model.summary()
```

```
Model: "inception10_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv_bn_relu_25 (ConvBNRelu  multiple                 512
 )

 sequential_27 (Sequential)  (None, 8, 8, 128)         119616

 global_average_pooling2d (G  multiple                 0
 lobalAveragePooling2D)

 dense_9 (Dense)             multiple                  1290

=================================================================
Total params: 121,418
Trainable params: 120,234
```

```
Non-trainable params: 1,184
_________________________________________________________________
```

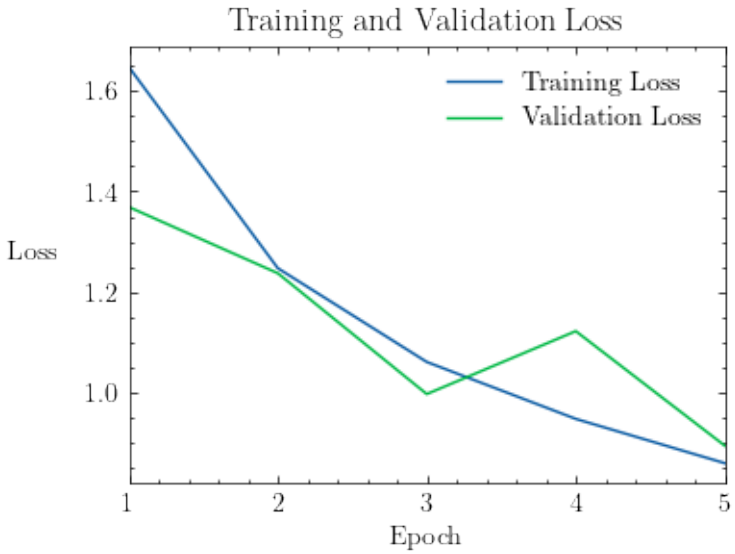
[34]:
```
acc = history.history['sparse_categorical_accuracy']
val_acc = history.history['val_sparse_categorical_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
```

[35]:
```
a = [0,1,2,3,4]
labels = ['1', '2', '3', '4', '5']
plt.figure(figsize=(4,3),dpi=100)
plt.style.use('science')
plt.ylim(0,1)
plt.xlim(0,4)
plt.xticks(a,labels)
plt.ylabel('Accuracy',rotation=0,labelpad=20)
plt.xlabel('Epoch')
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.savefig('C:/Users/Lenovo/Desktop/  -ResNet /figures/7.jpg')
plt.show()
```



[36]:
```
plt.figure(figsize=(4,3),dpi=100)
plt.style.use('science')
plt.xlim(0,4)
plt.ylabel('Loss',rotation=0,labelpad=20)
plt.xlabel('Epoch')
```

```
plt.xticks(a,labels)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.savefig('C:/Users/Lenovo/Desktop/  -ResNet /figures/8.jpg')
plt.show()
```



0.0.7 Model5:ResNet18

[37]:
```
class ResnetBlock(Model):

    def __init__(self, filters, strides=1, residual_path=False):
        super(ResnetBlock, self).__init__()
        self.filters = filters
        self.strides = strides
        self.residual_path = residual_path

        self.c1 = Conv2D(filters, (3, 3), strides=strides, padding='same',␣
↪use_bias=False)
        self.b1 = BatchNormalization()
        self.a1 = Activation('relu')

        self.c2 = Conv2D(filters, (3, 3), strides=1, padding='same',␣
↪use_bias=False)
        self.b2 = BatchNormalization()

        # residual_path True        1x1        x F(x)
```

```
        if residual_path:
            self.down_c1 = Conv2D(filters, (1, 1), strides=strides,␣
↪padding='same', use_bias=False)
            self.down_b1 = BatchNormalization()

        self.a2 = Activation('relu')

    def call(self, inputs):
        residual = inputs  # residual     residual=x
        #     BN    F(x)
        x = self.c1(inputs)
        x = self.b1(x)
        x = self.a1(x)

        x = self.c2(x)
        y = self.b2(x)

        if self.residual_path:
            residual = self.down_c1(inputs)
            residual = self.down_b1(residual)

        out = self.a2(y + residual)  #        F(x)+x F(x)+Wx,
        return out


class ResNet18(Model):

    def __init__(self, block_list, initial_filters=64):  # block_list  block
        super(ResNet18, self).__init__()
        self.num_blocks = len(block_list)  #   block
        self.block_list = block_list
        self.out_filters = initial_filters
        self.c1 = Conv2D(self.out_filters, (3, 3), strides=1, padding='same',␣
↪use_bias=False)
        self.b1 = BatchNormalization()
        self.a1 = Activation('relu')
        self.blocks = tf.keras.models.Sequential()
        #  ResNet
        for block_id in range(len(block_list)):  #   resnet block
            for layer_id in range(block_list[block_id]):  #

                if block_id != 0 and layer_id == 0:  #    block   block
                    block = ResnetBlock(self.out_filters, strides=2,␣
↪residual_path=True)
                else:
                    block = ResnetBlock(self.out_filters, residual_path=False)
                self.blocks.add(block)  #    block resnet
```

```
            self.out_filters *= 2  #   block     block 2
        self.p1 = tf.keras.layers.GlobalAveragePooling2D()
        self.f1 = tf.keras.layers.Dense(10, activation='softmax',␣
↪kernel_regularizer=tf.keras.regularizers.l2())

    def call(self, inputs):
        x = self.c1(inputs)
        x = self.b1(x)
        x = self.a1(x)
        x = self.blocks(x)
        x = self.p1(x)
        y = self.f1(x)
        return y
```

[38]:
```
model = ResNet18([2, 2, 2, 2])
```

[39]:
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.
↪SparseCategoricalCrossentropy(from_logits=False),
              metrics=['sparse_categorical_accuracy'])
```

[40]:
```
history = model.fit(x_train, y_train, batch_size=32, epochs=5,␣
↪validation_data=(x_test, y_test), validation_freq=1)
```

```
Epoch 1/5
1563/1563 [==============================] - 1332s 851ms/step - loss: 1.3984 -
sparse_categorical_accuracy: 0.5407 - val_loss: 1.2639 -
val_sparse_categorical_accuracy: 0.6014
Epoch 2/5
1563/1563 [==============================] - 1332s 852ms/step - loss: 0.8307 -
sparse_categorical_accuracy: 0.7248 - val_loss: 1.0105 -
val_sparse_categorical_accuracy: 0.6653
Epoch 3/5
1563/1563 [==============================] - 1322s 846ms/step - loss: 0.6336 -
sparse_categorical_accuracy: 0.7914 - val_loss: 0.7774 -
val_sparse_categorical_accuracy: 0.7407
Epoch 4/5
1563/1563 [==============================] - 1343s 859ms/step - loss: 0.5033 -
sparse_categorical_accuracy: 0.8371 - val_loss: 0.6995 -
val_sparse_categorical_accuracy: 0.7789
Epoch 5/5
1563/1563 [==============================] - 1341s 858ms/step - loss: 0.4046 -
sparse_categorical_accuracy: 0.8712 - val_loss: 0.6218 -
val_sparse_categorical_accuracy: 0.8026
```

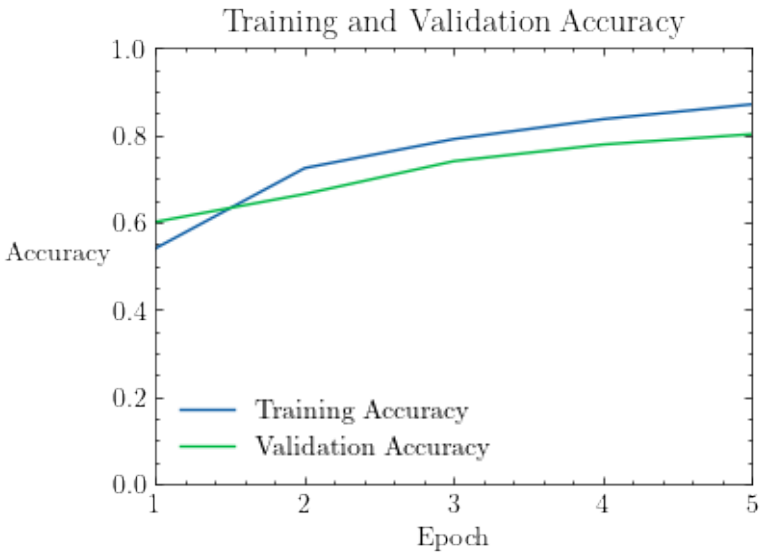
[41]:
```
model.summary()
```

```
Model: "res_net18"
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_70 (Conv2D)          multiple                  1728

 batch_normalization_65 (Bat  multiple                 256
 chNormalization)

 activation_65 (Activation)  multiple                  0

 sequential_52 (Sequential)  (None, 4, 4, 512)         11176448

 global_average_pooling2d_1   multiple                 0
 (GlobalAveragePooling2D)

 dense_10 (Dense)            multiple                  5130

=================================================================
Total params: 11,183,562
Trainable params: 11,173,962
Non-trainable params: 9,600
_________________________________________________________________
```

[42]:
```
acc = history.history['sparse_categorical_accuracy']
val_acc = history.history['val_sparse_categorical_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
```

[43]:
```
a = [0,1,2,3,4]
labels = ['1', '2', '3', '4', '5']
plt.figure(figsize=(4,3),dpi=100)
plt.style.use('science')
plt.ylim(0,1)
plt.xlim(0,4)
plt.xticks(a,labels)
plt.ylabel('Accuracy',rotation=0,labelpad=20)
plt.xlabel('Epoch')
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.savefig('C:/Users/Lenovo/Desktop/  -ResNet /figures/9.jpg')
plt.show()
```



[44]:
```
plt.figure(figsize=(4,3),dpi=100)
plt.style.use('science')
plt.xlim(0,4)
plt.ylabel('Loss',rotation=0,labelpad=20)
plt.xlabel('Epoch')
plt.xticks(a,labels)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.savefig('C:/Users/Lenovo/Desktop/  -ResNet /figures/10.jpg')
plt.show()
```