教程 二 : 基于卷积神经网络的图像识别任务

提交日期：2024.08.03

提交人：张健豪

指导教师：秦科

目录

[1. 题目 4](#_Toc173601213)

[2. 算法原理描述 4](#_Toc173601214)

[2.1 数据预处理 4](#_Toc173601215)

[2.2 卷积神经网络模型 4](#_Toc173601216)

[2.3 数据增强 4](#_Toc173601217)

[2.4 模型训练和评估 5](#_Toc173601218)

[3. 实验结果与分析 5](#_Toc173601219)

[3.1实验数据 5](#_Toc173601220)

[3.2实验设置（参数、实验过程等说明） 5](#_Toc173601221)

[3.2.1 数据集与环境 5](#_Toc173601222)

[3.2.2数据预处理 5](#_Toc173601223)

[3.2.3模型架构 6](#_Toc173601224)

[3.2.4实验参数 6](#_Toc173601225)

[3.3 实验结果与分析（对比结果、结论、图、表等） 6](#_Toc173601226)

[4. 总结 7](#_Toc173601227)

[5. 参考文献 7](#_Toc173601228)

[6. 完整代码 7](#_Toc173601229)

# 题目

基于卷积神经网络的图像识别任务

# 算法原理描述

## 2.1 数据预处理

数据读取: 实验首先从CSV文件中读取图像标注信息，包括图像的ID和标签。使用load\_csv函数从CSV文件中提取这些数据，并将其分割为训练集和验证集。

数据集拆分: split\_dataset函数将数据集按照80%和20%的比例划分为训练集和验证集。这种拆分方法有助于评估模型在未见数据上的性能。

数据解压: extract\_zip函数将压缩包中的图像文件解压到指定目录，并列出解压缩后的文件列表，以确保文件的完整性。

图像加载和预处理: load\_image函数负责从文件系统中读取图像文件，将其标准化到0到1的范围，并将标签转换为对应的整数值。图像数据被转换为TensorFlow张量，以便于后续的模型输入。

## 2.2 卷积神经网络模型

模型构建: Vgg16net类定义了一个基于VGG16的卷积神经网络模型。VGG16是一个经典的卷积神经网络架构，其主要特点是使用多个卷积层和池化层来提取图像特征，然后通过全连接层进行分类。

VGG16基础模型: 该模型使用了预训练的VGG16模型（在ImageNet数据集上训练），并去掉了顶层全连接层。VGG16作为特征提取器，输出的特征图通过全局平均池化层（GlobalAveragePooling2D）被压缩为固定长度的向量。

自定义层: 在VGG16的基础上，模型添加了若干自定义层，包括：

全连接层（Dense）: 经过BatchNormalization、ReLU激活函数和Dropout层处理，以增加模型的非线性和鲁棒性。

分类层（Dense）: 最终的分类层使用softmax激活函数，以生成每个类别的概率分布。

## 2.3 数据增强

在训练过程中，数据增强操作被应用于图像数据集。包括：

随机水平翻转: tf.image.random\_flip\_left\_right。

随机对比度调整: tf.image.random\_contrast。

这些操作有助于提高模型的泛化能力，使其能够更好地处理不同的图像变换。

## 2.4 模型训练和评估

模型编译: 使用SGD优化器（tf.keras.optimizers.legacy.SGD）进行模型的训练，设置学习率和学习率衰减。损失函数为稀疏分类交叉熵（tf.keras.losses.sparse\_categorical\_crossentropy），评价指标为稀疏分类准确率（tf.metrics.SparseCategoricalAccuracy）。

训练过程: 使用model.fit函数训练模型，同时保存验证集上表现最好的模型权重。ModelCheckpoint回调函数用于保存最佳权重。

模型评估: 训练完成后，通过model.evaluate函数在验证集上评估模型的性能，输出验证集的准确率。

通过上述步骤，该实验实现了对CIFAR-10数据集的图像分类任务，展示了卷积神经网络在图像识别中的应用。

# 实验结果与分析

## 3.1实验数据

直接选择用了trainLabels.csv以及train文件夹中前80%作为训练集，后20%作为测试集，训练数据集共计40000张照片。

## 3.2实验设置（参数、实验过程等说明）

### 3.2.1 数据集与环境

数据集: CIFAR-10数据集，包含10个类别，每个类别6000张32x32像素的彩色图像。

环境: Linux操作系统，Python 3.x，TensorFlow 2.x，OpenCV，NumPy，Pandas。

### 3.2.2数据预处理

图像解压: 使用extract\_zip函数解压缩图像数据。

数据加载与划分: 从CSV文件中读取图像ID和标签，使用split\_dataset函数将数据集划分为训练集（80%）和验证集（20%）。

### 3.2.3模型架构

基础模型: 预训练VGG16模型（tf.keras.applications.VGG16），不包括顶层全连接层。

自定义层: 添加全局平均池化层、全连接层、BatchNormalization、ReLU激活函数、Dropout层和分类层（softmax）。

### 3.2.4实验参数

图像输入尺寸: (32, 32, 3)

类别数量: 10

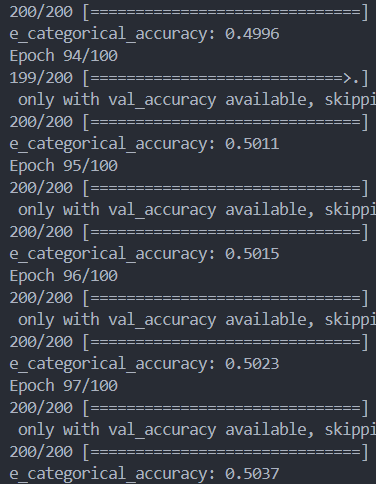
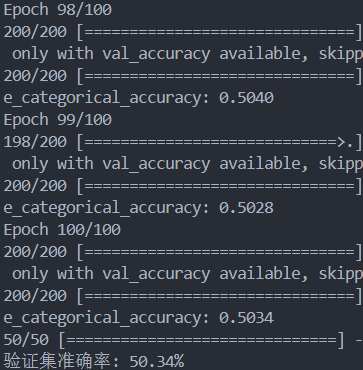
学习率: 1e-3

学习率衰减: 1e-4

批量大小: 200

训练轮次: 100

## 3.3 实验结果与分析（对比结果、结论、图、表等）

本实验使用基于VGG16的卷积神经网络对CIFAR-10数据集进行了图像分类任务。CIFAR-10数据集包含10个类别的图像，因此随机猜测的理论准确率应为10%。本模型在测试集上的最终准确率为50.34%，显著高于随机猜测准确率，这表明模型对图像分类具有一定的学习和泛化能力。

在训练过程中，模型的损失值和准确率指标逐渐改善，表明模型在学习过程中有效地优化了权重并提升了性能。具体地，模型在最后一个epoch（第100轮）达到了1.4275的验证集损失和50.34%的准确率，说明训练过程中的优化调整有助于提升分类性能。

与预期相比，模型的表现显著好于随机猜测，这表明卷积神经网络能够有效地捕捉图像中的特征，并将其映射到相应的类别。然而，模型的准确率距离理想状态还有提升空间，这可能与数据预处理、模型结构和训练参数的设置有关。进一步的优化和实验可以包括数据增强、调整模型结构或使用更高级的优化算法。

总之，本实验展示了基于VGG16的卷积神经网络在图像分类任务中的有效性，并为后续的模型改进和优化提供了基础。

# 总结

本实验通过使用基于VGG16的卷积神经网络对CIFAR-10数据集进行了图像分类任务。模型的最终验证集准确率为50.34%，显著高于随机猜测的10%准确率，验证了卷积神经网络在图像分类中的有效性。训练过程中，损失值逐步降低，准确率稳步提升，表明模型成功地学习了图像特征并优化了分类性能。尽管模型表现良好，但仍有进一步提升的空间，建议未来的研究可以探讨数据增强、模型结构调整或优化算法改进等策略。总体而言，本实验为卷积神经网络在图像分类任务中的应用提供了有价值的实践经验。

# 参考文献

1. 卷积神经网络：https://www.zybuluo.com/hanbingtao/note/485480

2. 图像分类的卷积神经网络：https://zhuanlan.zhihu.com/p/62077601

3. 参考论文-残差网络："Deep Residual Learning for Image Recognition"

# 6. 完整代码

import os

import random

import zipfile

import cv2

import numpy as np

import pandas as pd

import tensorflow as tf

from tensorflow.keras.layers import Dropout, Input, GlobalAveragePooling2D, Dense, BatchNormalization, Activation

from tensorflow.keras.callbacks import ModelCheckpoint

from tqdm import tqdm

class PATH:

csv\_path = '/home/nas/zhangjianhao/cifar-10/trainLabels.csv'

train\_zip = '/home/nas/zhangjianhao/cifar-10/train.zip'

temp\_train\_dir = '/tmp/train/train' # 更新为正确的解压路径

weight\_path = '/home/nas/zhangjianhao/cifar-10/cifar\_vgg16\_not\_trained\_20191216.h5' # 修改为Linux路径

class CONFIG:

image\_shape = (32, 32)

n\_class = 10

lr = 1e-3

decay\_lr = 1e-4

num\_epochs = 100

batch\_size = 200

label\_class = ["airplane", "automobile", "bird", "cat", "deer", "dog", "frog", "horse", "ship", "truck"]

def load\_csv(csv\_path):

"""读取csv文件"""

csv\_data = pd.read\_csv(csv\_path)

id\_list = list(csv\_data['id'])

id\_list = map(lambda x: str(x), id\_list)

label\_list = list(csv\_data['label'])

return list(zip(id\_list, label\_list))

def split\_dataset(data\_list, train\_perc=0.8):

"""将数据集分为训练集与验证集"""

random.seed(3)

train\_size = int(len(data\_list) \* train\_perc)

train\_list = data\_list[:train\_size]

val\_list = data\_list[train\_size:]

return train\_list, val\_list

def unzip\_data(data\_list):

"""将[(a1,b1), (a2, b2)]分解为[a1, a2], [b1, b2]"""

id\_list = [data[0] for data in data\_list]

label\_list = [data[1] for data in data\_list]

return tuple((id\_list, label\_list))

def load\_image(image\_id, label):

image\_path = os.path.join(PATH.temp\_train\_dir, str(image\_id.numpy(), 'utf-8') + ".png")

image = cv2.imread(image\_path)

if image is None:

raise FileNotFoundError(f"无法打开图像文件: {image\_path}")

image = image / 255.0

label\_class = CONFIG.label\_class

label = label\_class.index(label)

return tf.convert\_to\_tensor(image, dtype=tf.float32), tf.convert\_to\_tensor(label, dtype=tf.int32)

def set\_shape(image, label):

image.set\_shape([CONFIG.image\_shape[0], CONFIG.image\_shape[1], 3])

label.set\_shape([])

return image, label

def extract\_zip(zip\_path, extract\_to):

"""解压缩文件，并列出文件列表"""

with zipfile.ZipFile(zip\_path, 'r') as zip\_ref:

zip\_ref.extractall(extract\_to)

print(f"解压缩到: {extract\_to}")

print("解压缩的文件列表:")

for root, dirs, files in os.walk(extract\_to):

for file in files:

print(os.path.join(root, file))

class Vgg16net(tf.keras.Model):

def \_\_init\_\_(self, n\_class=2):

super().\_\_init\_\_()

self.n\_class = n\_class

self.vgg16\_model = self.load\_vgg16()

self.global\_pool = GlobalAveragePooling2D()

self.conv\_vgg = Dense(256, use\_bias=False, kernel\_initializer='uniform')

self.batch\_normalize = BatchNormalization()

self.relu = Activation("relu")

self.dropout\_1 = Dropout(0.2)

self.conv\_1 = Dense(64, use\_bias=False, kernel\_initializer='uniform')

self.batch\_normalize\_1 = BatchNormalization()

self.relu\_1 = Activation("relu")

self.dropout\_2 = Dropout(0.2)

self.classify = Dense(n\_class, kernel\_initializer='uniform', activation="softmax")

def call(self, input):

x = self.vgg16\_model(input)

x = self.global\_pool(x)

x = self.conv\_vgg(x)

x = self.batch\_normalize(x)

x = self.relu(x)

x = self.dropout\_1(x)

x = self.conv\_1(x)

x = self.batch\_normalize\_1(x)

x = self.relu\_1(x)

x = self.dropout\_2(x)

x = self.classify(x)

return x

def load\_vgg16(self):

vgg16 = tf.keras.applications.VGG16(weights='imagenet', include\_top=False, input\_tensor=Input(

shape=(CONFIG.image\_shape[0], CONFIG.image\_shape[1], 3)), classes=self.n\_class)

vgg16.trainable = False

return vgg16

def main():

# 解压缩训练数据

if not os.path.exists(PATH.temp\_train\_dir):

os.makedirs(PATH.temp\_train\_dir)

extract\_zip(PATH.train\_zip, '/tmp/train')

# 读取csv

csv\_data = load\_csv(PATH.csv\_path)

train\_list, val\_list = split\_dataset(csv\_data, train\_perc=0.8)

train\_list = unzip\_data(train\_list)

val\_list = unzip\_data(val\_list)

train\_dataset = tf.data.Dataset.from\_tensor\_slices(train\_list)

train\_dataset = train\_dataset.shuffle(len(train\_list)).map(lambda x, y: tf.py\_function(load\_image, [x, y], [tf.float32, tf.int32])).map(set\_shape).map(

lambda image, label: (tf.image.convert\_image\_dtype(image, tf.float32), label)

).cache().map(

lambda image, label: (tf.image.random\_flip\_left\_right(image), label)

).map(

lambda image, label: (tf.image.random\_contrast(image, lower=0.0, upper=1.0), label)

).batch(CONFIG.batch\_size)

val\_dataset = tf.data.Dataset.from\_tensor\_slices(val\_list)

val\_dataset = val\_dataset.shuffle(len(val\_list)).map(lambda x, y: tf.py\_function(load\_image, [x, y], [tf.float32, tf.int32])).map(set\_shape).batch(CONFIG.batch\_size)

model = Vgg16net(CONFIG.n\_class)

checkpoint\_callback = ModelCheckpoint(

PATH.weight\_path, monitor='val\_accuracy', verbose=1,

save\_best\_only=True, save\_weights\_only=True, save\_freq='epoch')

# 使用旧版优化器

optimizer = tf.keras.optimizers.legacy.SGD(learning\_rate=CONFIG.lr, decay=CONFIG.decay\_lr)

model.compile(

optimizer=optimizer,

loss=tf.keras.losses.sparse\_categorical\_crossentropy,

metrics=[tf.metrics.SparseCategoricalAccuracy()]

)

model.fit(train\_dataset, validation\_data=val\_dataset, epochs=CONFIG.num\_epochs, callbacks=[checkpoint\_callback])

# 评估模型

loss, accuracy = model.evaluate(val\_dataset)

print(f"验证集准确率: {accuracy \* 100:.2f}%")

if \_\_name\_\_ == "\_\_main\_\_":

main()