北京航空航天大學
BEIHANG UNIVERSITY

模式识别与机器学习第二次大作业

CIFAR-10 数据集相关实验

学　院　名　称　前沿科学技术创新研究院

专　业　名　称　电子信息

学　生　姓　名　崔译文

学　　号　ZY2343223

指　导　教　师　郑建英、郭玉柱

2024 年 6 月

目录

1. 引言

1.1 项目背景

深度学习技术在图像分类领域取得了显著的进展，特别是卷积神经网络（CNN）的应用，使得图像识别和分类的准确率大幅提高。CIFAR-10 数据集是一个常用的图像分类基准数据集，包含了 10 个不同类别的小图像，广泛用于评估图像分类算法的性能。本报告基于 pycharm 开发环境，代码公开 github 链接为 ccuiyiwen/CIFAR-10_classification_cyw (github.com)。

1.2 CIFAR-10 数据集简介

CIFAR-10 数据集由 60000 张 32x32 像素的彩色图像组成，这些图像分为 10 个类别，每个类别包含 6000 张图像。数据集分为 50000 张训练图像和 10000 张测试图像。

该数据集分为五个训练批次和一个测试批次，每个批次包含 10000 张图像。测试批次包含每个类别的 1000 张随机选择的图像。训练批次包含剩余的图像，这些图像按随机顺序排列，但某些训练批次可能包含来自某一类别的图像比其他类别更多。总体而言，训练批次中每个类别的图像数量都是 5000 张。

CIFAR-10 数据集中的类别及其对应的示例如图 1 所示，这些类别完全是互斥的，汽车和卡车之间没有重叠。“automobile” 类别包括轿车、SUV 等类型，不包括卡车。“truck” 类别仅包括大型卡车，不包括皮卡。通过这个数据集，可以对图像分类算法进行评估和比较，从而验证算法的性能和效果。

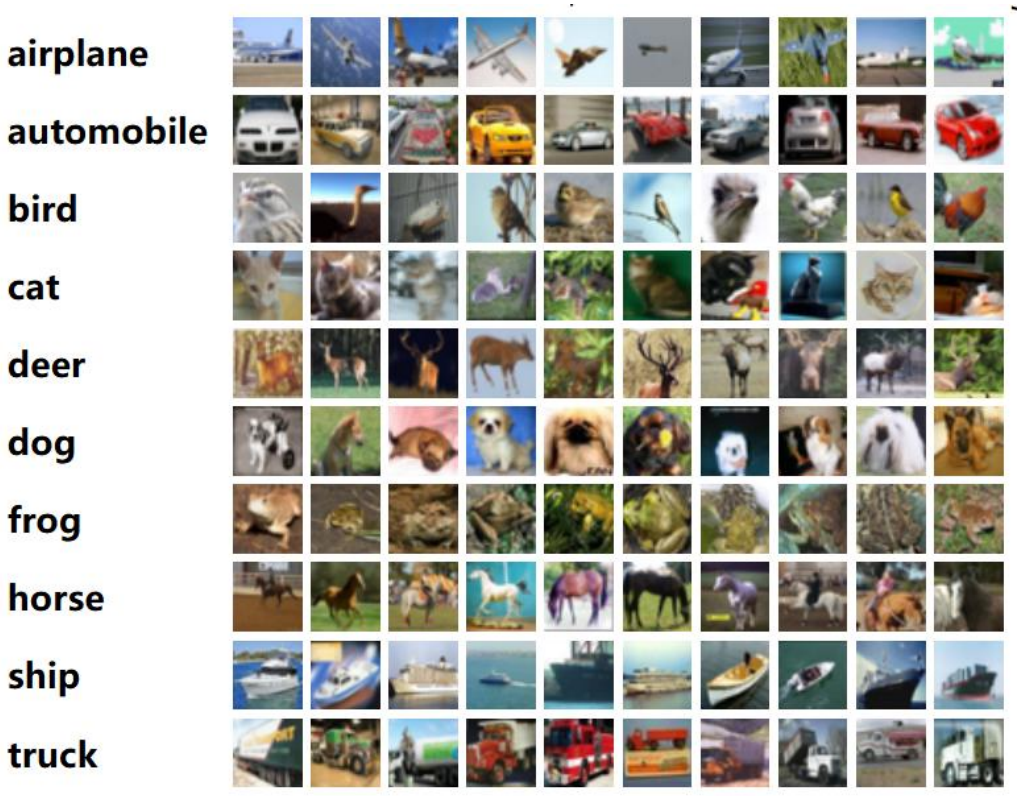

图 1 CIFAR-10 数据集中的类别及其对应的示例图

1.3 项目目标与任务描述

本项目的目标是使用深度学习技术实现一个图像分类器，能够将 CIFAR-10 数据集中的图像正确分类为预定义的 10 个类别。具体任务包括：

（1） 数据预处理：加载数据集并进行必要的预处理，如归一化和图像大小调整。
（2） 模型选择与搭建：设计并实现全连接前馈神经网络（FCNN）和卷积神经网络（CNN）模型。
（3） 模型训练：使用选定的模型对数据集进行训练，选择适当的损失函数和优化器，并确定合适的训练迭代次数。
（4） 模型评估：使用测试集对训练好的模型进行评估，计算准确率和其他评价指标来衡量模型性能。
（5） 结果可视化：可视化模型的训练过程和评估结果，通过绘制损失曲线和准确率曲线展示结果。
（6） 结果分析：分析模型的可解释性，进行超参数实验和数据增强实验，并比较不同设置对模型性能的影响。

2. 数据预处理

在图像分类任务中，数据预处理是至关重要的一步。通过有效的数据预处理，可以提升模型的性能和泛化能力。本文的预处理步骤包括加载数据集、数据归一化、图像大小调整、数据集分割和数据增强技术的应用。

2.1 加载数据集

首先，我们使用 TensorFlow 加载 CIFAR-10 数据集。CIFAR-10 是一个包含 60000 张 32x32 彩色图像的标准数据集，分为 10 个类别，每个类别有 6000 张图像。数据集分为训练集（50000 张图像）和测试集（10000 张图像）。

```
from tensorflow.keras import datasets

(x_train, y_train), (x_test, y_test) = datasets.cifar10.load_data()
```

2.2 数据归一化

为了加速模型的训练过程并提升模型性能，我们将图像的像素值从 0-255 归一化到 0-1 之间。归一化可以使得输入数据具有相同的量纲，减少模型训练的复杂度。

```
x_train, x_test = x_train / 255.0, x_test / 255.0
```

2.3 图像大小调整

CIFAR-10 数据集中的图像已经是标准的 32x32 大小，因此无需进一步调整

图像大小。然而，在其他数据集中，可能需要将图像调整到特定大小，以适应神经网络的输入要求。

2.4 数据集分割

在加载数据集后，我们将数据集分为训练集和测试集。训练集用于训练模型，测试集用于评估模型的性能。CIFAR-10 数据集已经按比例分割为训练集和测试集，我们直接使用即可。

```
# 数据集已经按比例分割为训练集和测试集
print(f'x_train shape: {x_train.shape}')
print(f'y_train shape: {y_train.shape}')
print(f'x_test shape: {x_test.shape}')
print(f'y_test shape: {y_test.shape}')
```

2.5 数据增强技术介绍

数据增强技术是通过对原始图像进行各种变换（如旋转、缩放、裁剪等），生成更多的训练样本，从而提升模型的泛化能力。在训练神经网络时，数据增强可以有效防止过拟合，尤其在数据量较少的情况下。

2.5.1 使用数据增强技术的必要性

在图像分类任务中，数据增强技术可以增加训练数据的多样性，帮助模型学习到更加鲁棒的特征，提升模型在未见过数据上的表现。对于 CIFAR-10 这样的标准数据集，数据增强同样有助于提高模型的准确率。

2.5.2 数据增强技术的实现方法

在本文中，我们使用 TensorFlow 和 Keras 的 ImageDataGenerator 类实现数据增强。数据增强的常用方法是随机旋转图像、随机水平翻转图像、随机缩放图像和随机平移图像等。

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True
)

# 通过 datagen.flow() 生成增强后的训练数据
datagen.fit(x_train)
```

3. 模型选择与搭建

3.1 全连接前馈神经网络（FCNN）

全连接前馈神经网络（Fully Connected Neural Network，FCNN）是一种基本的神经网络结构，所有神经元在相邻层之间完全连接。它适用于处理结构化数据，但在图像分类任务中，表现可能不如卷积神经网络。为了提升 FCNN 的性能，我们可以引入集成学习技术，如 Adaboost。

3.1.1 模型结构设计

（1） 基础全连接前馈神经网络（FCNN）

基础的全连接前馈神经网络由若干层组成，每层的神经元与下一层的所有神经元相连。网络通常包括输入层、隐藏层和输出层。输入层的节点数等于输入数据的特征数，输出层的节点数等于分类任务的类别数。

```python
def build_fcnn():
    model = models.Sequential([
        layers.Flatten(input_shape=(32, 32, 3)),  # 展开输入
        layers.Dense(1024, activation='relu'),    # 全连接层
        layers.Dropout(0.5),                      # Dropout 层，防止过拟合
        layers.Dense(512, activation='relu'),     # 全连接层
        layers.Dropout(0.5),                      # Dropout 层，防止过拟合
        layers.Dense(256, activation='relu'),     # 全连接层
        layers.Dense(128, activation='relu'),     # 全连接层
        layers.Dense(10, activation='softmax')    # 输出层
    ])
    return model
```

（2） 基于 Adaboost 优化的全连接前馈神经网络（FCNN+Adaboost）

Adaboost 是一种集成学习算法，通过组合多个弱分类器来提升模型的整体性能。在 FCNN+Adaboost 模型中，Adaboost 将多个基础的全连接前馈神经网络组合起来，从而提高分类准确率。

```python
# 自定义 Keras 分类器以便在 scikit-learn 中使用
class KerasClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self, build_fn, epochs=20, batch_size=64, verbose=0):
        self.build_fn = build_fn
        self.epochs = epochs
        self.batch_size = batch_size
        self.verbose = verbose
        self.model = None
        self.classes_ = None
        self.history = None  # 用于存储训练历史记录
    def fit(self, X, y, sample_weight=None):
        self.model = self.build_fn()
        self.model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
        if sample_weight is not None:
            sample_weight = sample_weight.astype(np.float32)
            self.history = self.model.fit(X, y, epochs=self.epochs, batch_size=self.batch_size, verbose=self.verbose,
                                          sample_weight=sample_weight)
        else:
            self.history = self.model.fit(X, y, epochs=self.epochs, batch_size=self.batch_size, verbose=self.verbose)
        self.classes_ = np.unique(y)
        return self
    def predict(self, X):
        y_pred = self.model.predict(X)
        return np.argmax(y_pred, axis=1)
    def predict_proba(self, X):
        return self.model.predict(X)
```

3.1.2 层数和神经元数量选择

在设计全连接前馈神经网络时，层数和每层的神经元数量是需要重点考虑的两个参数。这些参数的选择通常基于经验或通过超参数优化技术进行调整。

(1) 层数：增加隐藏层的数量可以提高模型的表达能力，但也会增加计算复杂度和过拟合风险。通常可以选择 2 到 3 层隐藏层。

(2) 神经元数量：每层的神经元数量需要根据任务复杂度进行选择。对于图像分类任务，通常选择 512 到 1024 个神经元。

3.1.3 激活函数选择

激活函数在神经网络中引入非线性，常用的激活函数有 ReLU、Sigmoid 和 Tanh。ReLU（Rectified Linear Unit）是目前使用最广泛的激活函数，因为它在训练深层网络时能够有效缓解梯度消失问题。

```
layers.Dense(1024, activation='relu')
```

3.1.4 Dropout 层的使用

Dropout 是一种防止神经网络过拟合的技术，通过在训练过程中随机丢弃一部分神经元，使得网络在每次训练时都会学习不同的特征。Dropout 层通常放置在全连接层之后。

```
layers.Dropout(0.5)
```

在训练过程中，每次迭代都会随机“关闭”一部分神经元，使得网络在每次训练时都会学习不同的特征，增强模型的泛化能力。

3.2 卷积神经网络（CNN）

卷积神经网络（CNN）是深度学习中用于图像分类和识别的常用模型结构。与传统的全连接神经网络相比，卷积神经网络通过卷积操作和池化操作能够更好地捕捉图像的局部特征，同时减少参数数量，提升计算效率。

3.2.1 模型结构设计

一个典型的卷积神经网络由多个卷积层、池化层和全连接层组成。每个卷积层和池化层的输出被作为下一个卷积层或池化层的输入，最后通过一个或多个全连接层完成分类任务。在设计 CNN 模型时，通常会依次堆叠以下几种层：

（1） 卷积层（Convolutional Layer）：用于提取图像的局部特征，通过多个滤波器进行卷积操作。

（2） 激活层（Activation Layer）：在每个卷积层后面添加激活函数，以增加网络的非线性。

（3） 池化层（Pooling Layer）：用于降低特征图的维度，减少参数量和计算量。

（4） 全连接层（Fully Connected Layer）：将前面的特征映射到最终的分类结

果。

```
from tensorflow.keras import models, layers

def build_cnn():
    model = models.Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
    return model
```

3.2.2 卷积层和池化层的设计

卷积层的主要参数包括滤波器数量、滤波器大小和步幅等。滤波器的数量决定了输出特征图的通道数，滤波器的大小通常选择 3x3 或 5x5。步幅决定了滤波器在图像上移动的步长，一般设为 1。

```
layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3))
```

池化层通过最大池化或平均池化操作对特征图进行降采样，减少特征图的尺寸。常用的池化大小为 2x2，步幅通常设置为 2。

```
layers.MaxPooling2D((2, 2))
```

3.2.3 激活函数选择

激活函数用于引入网络的非线性特性，使得网络可以更好地拟合复杂的函数关系。

```
layers.Conv2D(32, (3, 3), activation='relu')
```

在卷积神经网络中，常用的激活函数是 ReLU（Rectified Linear Unit），其公式为（式 1）。ReLU 激活函数的优点是计算简单，并且在实际应用中表现良好。

$$ReLU(x) = \max(0, x) \quad \text{（式 1）}$$

3.2.4 Dropout 层的使用

Dropout 是一种防止过拟合的技术，通过在训练过程中随机丢弃一部分神经元，使得网络不会过度依赖某些局部特征。Dropout 层通常放置在全连接层之后。

```
layers.Dropout(0.5)
```

在训练过程中，每次迭代都会随机“关闭”一部分神经元，使得网络在每次训练时都会学习不同的特征，增强模型的泛化能力。

```
def build_cnn_with_dropout():
    model = models.Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(10, activation='softmax')
    ])
    return model
```

3.3 模型编译

模型编译是模型训练的前置步骤，涉及到损失函数、优化器和评估指标的选择。每个选择都会影响模型的训练效果和性能表现。本文将分别介绍并对比各个选择，并给出最终的选择原因。

3.3.1 损失函数选择

损失函数是模型优化的目标，通过最小化损失函数，模型的预测结果与实际标签之间的差异逐渐减小。对于分类问题，常用的损失函数包括：

（1） 稀疏分类交叉熵损失（Sparse Categorical Crossentropy）：适用于多类别分类任务，标签是整数形式。

（2） 分类交叉熵损失（Categorical Crossentropy）：适用于多类别分类任务，标签是独热编码形式。

CIFAR-10 数据集的标签是整数形式，因此我们选择稀疏分类交叉熵损失，这样可以直接使用原始标签，避免额外的独热编码处理。

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

3.3.2 优化器选择

优化器用于更新模型的权重参数，使得损失函数逐渐收敛。常用的优化器

包括：

（1） SGD（随机梯度下降）：最经典的优化器，简单高效，但对学习率敏感。

（2） Adam：结合了动量优化和 RMSProp 优化的优点，能够自动调整学习率，具有较好的收敛性。

（3） RMSProp：适合处理非平稳目标的优化器，通过对梯度的平方进行指数加权移动平均，调节学习率。

Adam 优化器结合了动量优化和 RMSProp 优化的优点，能够自动调整学习率，在大多数情况下表现优越。因此，我们选择 Adam 优化器来训练模型。

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

3.3.3 评估指标选择

评估指标用于衡量模型在验证集或测试集上的表现，表 1 总结了各个评估指标的定义和适用场景，并给出了相应的公式。其中：TP (True Positive) 表示正确预测为正类的样本数，TN (True Negative) 表示正确预测为负类的样本数，FP (False Positive) 表示错误预测为正类的样本数，FN (False Negative) 表示错误预测为负类的样本数。

表 1 各评价指标的定义、使用场景及公式

评估指标	定义	适用场景	公式
准确率 (Accuracy)	分类正确的样本数占总样本数的比例，适用于均衡数据集。	适用于均衡数据集	$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$
精确率 (Precision)	分类正确的正样本数占预测为正样本数的比例，适用于数据集不均衡时关注正样本的情况。	适用于数据集不均衡时关注正样本的情况	$\text{Precision} = \frac{TP}{TP+FP}$
召回率 (Recall)	分类正确的正样本数占实际正样本数的比例，适用于数据集不均衡时关注召回的情况。	适用于数据集不均衡时关注召回的情况	$\text{Recall} = \frac{TP}{TP+FN}$
F1-score	精确率和召回率的调和平均数，综合评估分类模型的性能。	综合评估分类模型的性能	$\text{F1-score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision}+\text{Recall}}$

4. 模型训练

4.1 训练参数设置

4.1.1 训练迭代次数（epochs）

CNN 和 FCNN 的训练迭代次数：为了确保模型有足够的时间学习训练数据的特征，我们通常设置较高的迭代次数。例如，对于 FCNN 和 CNN 模型，迭代次数分别设置为 20 和 10 次。

较高的迭代次数能够使模型在训练集上不断优化，从而提高在验证集上的表现。然而，过多的迭代次数可能会导致过拟合，因此需要根据验证集的表现

动态调整。

4.1.2 批量大小（batch size）

CNN 和 FCNN 的批量大小：常见的选择是 32、64 或 128。批量大小较小时，训练更加稳定，但时间较长；批量较大时，训练速度加快，但需要更多内存。

批量大小的选择平衡了训练时间和模型稳定性。对于 CIFAR-10 数据集，64 是一个合理的选择，能够在速度和稳定性之间取得平衡。

4.2 训练过程

4.2.1 训练和验证数据的损失和准确率曲线绘制

在实现过程中，我们通过绘制训练和验证数据的损失曲线和准确率曲线，直观地观察模型的训练过程和性能。具体来说，损失曲线展示了训练损失和验证损失随迭代次数的变化情况，而准确率曲线则展示了训练准确率和验证准确率随迭代次数的变化情况。通过分析这些曲线，我们可以判断模型的优化情况：如果验证损失不断下降，说明模型在不断优化；如果验证损失上升，则可能出现过拟合现象。

```
import matplotlib.pyplot as plt

def plot_fcnn_training_history(history):
    plt.figure(figsize=(12, 4))

    # 绘制损失曲线
    plt.subplot(1, 2, 1)
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend(loc='upper right')
    plt.title('Training and Validation Loss')

    # 绘制准确率曲线
    plt.subplot(1, 2, 2)
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend(loc='lower right')
    plt.title('Training and Validation Accuracy')

    plt.show()

# 假设 history 是模型训练的历史记录
plot_fcnn_training_history(history)
```

4.2.2 模型训练的可视化展示

通过可视化展示模型在训练过程中的表现，能够帮助我们更好地理解模型的学习情况和发现潜在的问题。具体实现方面，通过绘制混淆矩阵，可以展示模型在测试集上的分类效果；对于 CNN 模型，还可以可视化卷积层的特征图，观察不同层次的特征提取效果。这样不仅能直观地了解模型的性能，还能帮助我们进行深入分析和改进。

5. 模型评估

5.1 使用测试集进行模型评估

在训练完成后，使用独立的测试集对模型进行评估是验证其泛化能力的重要步骤。测试集的数据从未用于训练，能够有效衡量模型在未知数据上的表现。

5.2 准确率计算

模型的准确率是分类任务中最常用的评估指标之一。准确率（Accuracy）是指模型预测正确的样本数量占总样本数量的比例。计算公式如（式 2），该指标适用于数据集类别均衡的情况，但在类别不均衡的数据集中，可能会导致评价偏差。

$$Accuracy = \frac{Number\ of\ Correct\ PredictionsTotal}{Number\ of\ Predictions} \quad （式 2）$$

5.3 混淆矩阵绘制与分析

混淆矩阵（Confusion Matrix）是评估分类模型性能的常用工具，它能够详细显示模型的分类结果，包括真正类（TP，True Positive）、假正类（TN，True Negative）、假负类（FP，False Positive）和真负类（FN，False Negative）的数量。通过绘制混淆矩阵，可以直观地观察模型在各个类别上的分类效果。

5.4 其他评价指标（如精确度、召回率、F1 分数等）

除了准确率，还可以使用以下几个常见的评价指标来全面评估分类模型的性能。

（1） 精确率（Precision）：指模型预测为正样本中实际为正样本的比例。适用于关注预测结果为正样本的场景。

$$Precision = \frac{TP}{TP+FP} \quad （式 3）$$

（2） 召回率（Recall）：指模型实际为正样本中被正确预测为正样本的比例。适用于关注实际正样本的召回情况。

$$Recall = \frac{TP}{TP+FN} \quad （式 4）$$

（3） F1 分数（F1-score）：精确率和召回率的调和平均数，用于综合评估分类模型的性能，特别是在类分布不均衡的情况下。

$$F1-Score=\frac{2\times Precision\times Recall}{Precision+Recall} \quad \text{（式 5）}$$

6. 结果可视化

6.1 模型训练过程的损失曲线和准确率曲线

在模型训练过程中，我们记录了每一轮训练和验证数据的损失及准确率变化情况，并绘制了相应的曲线，以便直观了解模型的收敛情况。

（1） 损失曲线：在每一轮训练和验证过程中，我们分别记录了训练集和验证集的损失值，并绘制出损失曲线。通过观察损失曲线，可以看出模型在训练过程中的收敛速度和稳定性。如图所示，训练损失和验证损失随训练轮数逐渐下降，表明模型在不断优化，图 2-4 分别展示了 FCNN, FCNN+Adboost, CNN 的训练过程和验证过程的损失函数。

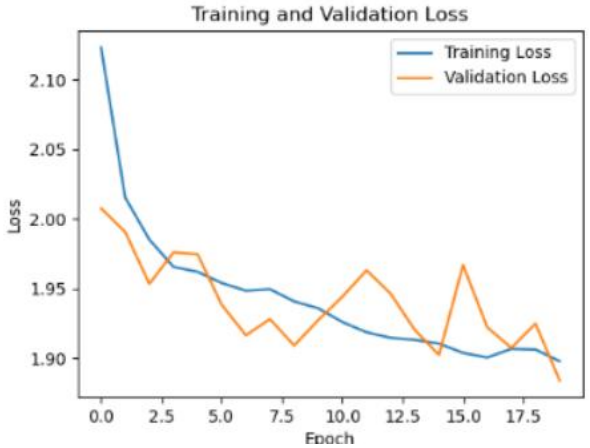

图 2 FCNN 分类器训练集和验证集的损失函数

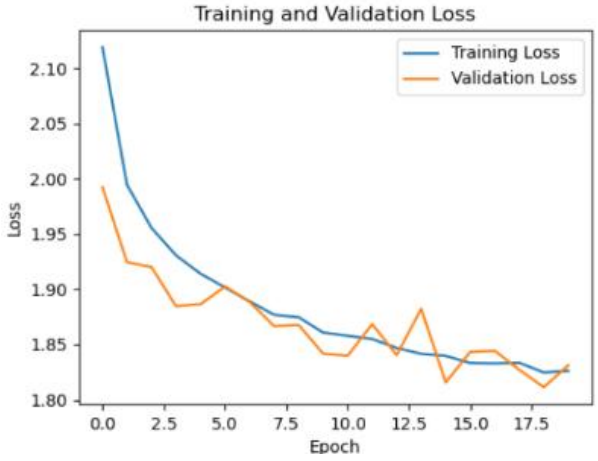

图 3 FCNN+Adboost 集成分类器训练集和验证集的损失函数

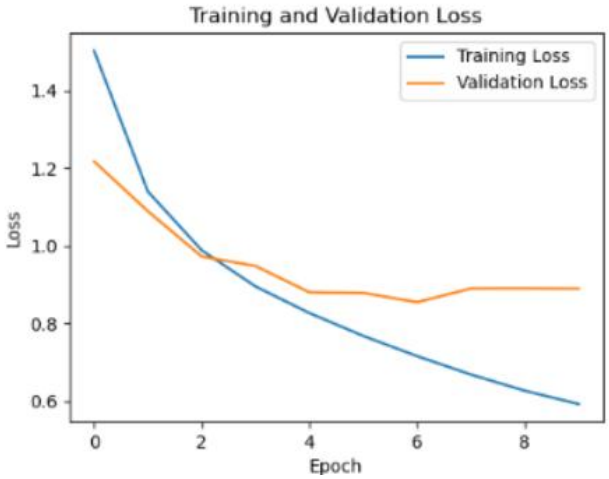

图 4 CNN 分类器训练集和验证集的损失函数

（2） 准确率曲线：同样地，我们记录了每一轮训练和验证的准确率，并绘制出准确率曲线。通过观察准确率曲线，可以了解模型在训练集和验证集上的表现。如图所示，训练准确率和验证准确率逐渐上升，表明模型的分类性能在不断提高。图 5-7 分别展示了 FCNN, FCNN+Adboost, CNN 的训练过程和验证过程的损失函数。

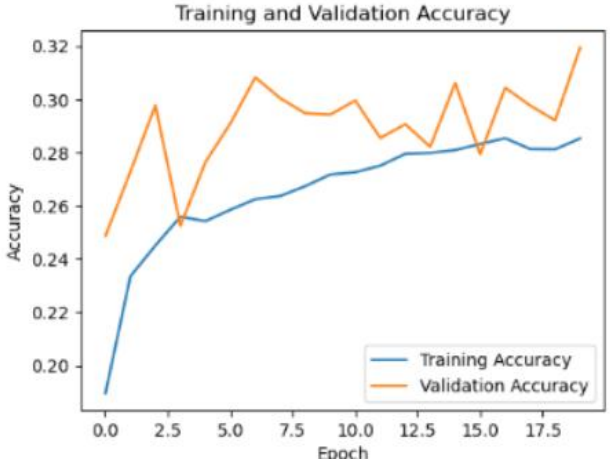

图 5 FCNN 分类器训练集和验证集的准确率

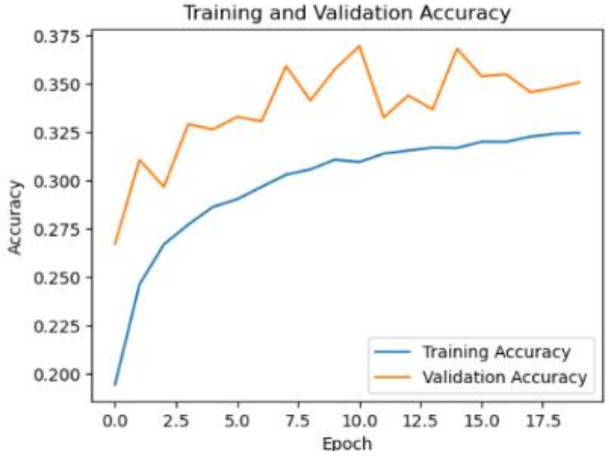

图 6 FCNN+Adboost 分类器训练集和验证集的准确率

图 6 CNN 分类器训练集和验证集的准确率

6.2 测试集上的预测结果可视化

为了更好地理解模型的分类效果，我们对测试集上的预测结果进行了可视化分析。我们随机选取了一些测试集中被正确分类和错误分类的样本，并展示它们的图像及其对应的预测标签。通过观察这些样本，可以直观地看到模型在处理不同类别图像时的表现。展示了模型在处理各种图像时的正确分类和错误分类结果，有助于理解模型的优势和不足之处，从而进一步改进模型。图 7-9 分别展示了 FCNN, FCNN+Adboost, CNN 三种分类器的预测结果可视化图。

图 7 FCNN 预测结果可视化

图 8 CNN+Adboost 预测结果可视化

图 9 CNN 预测结果可视化

6.3 混淆矩阵分析

混淆矩阵是评价分类模型性能的重要工具之一。它可以直观地展示模型在各个类别上的分类准确率和错误分布情况。通过分析混淆矩阵，可以发现模型在哪些类别上表现较好，哪些类别上容易出现分类错误，从而针对性地进行模型优化。

实现方法： 通过绘制混淆矩阵，可以直观地观察模型在测试集上的分类效果。具体来说，混淆矩阵的对角线元素表示模型正确分类的样本数量，非对角线元素表示模型错误分类的样本数量。

分析：

（1） **正确分类：**混淆矩阵的对角线元素表示模型正确分类的样本数量。每个类别的对角线元素越大，说明模型在该类别上的分类准确率越高。

（2） **错误分类：**混淆矩阵的非对角线元素表示模型错误分类的样本数量。通过分析这些元素，可以发现模型在某些类别上的分类混淆情况，例如，某些类别容易被误分类为其他特定类别。

通过对混淆矩阵的详细分析，我们可以了解模型在不同类别上的表现，并采取相应措施改进模型。图 10-12 分别展示了 FCNN, FCNN+Adboost, CNN 三种分类器的混淆矩阵图。

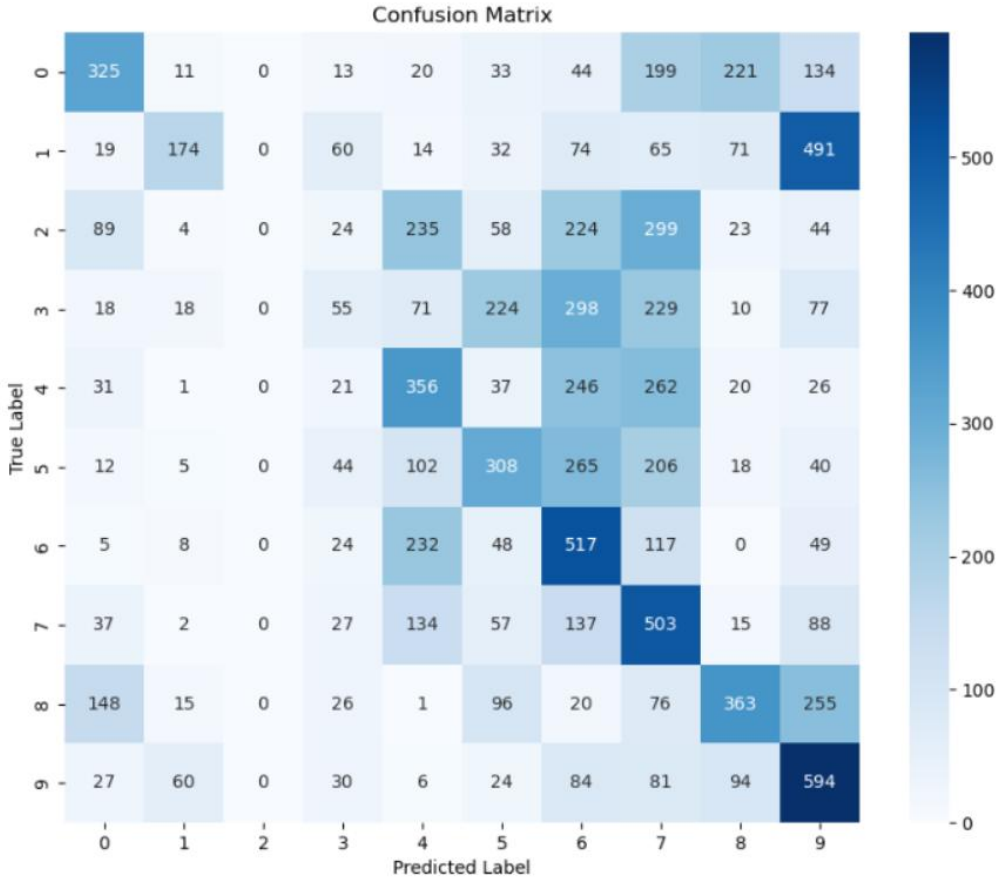

图 10 FCNN 预测结果混淆矩阵图

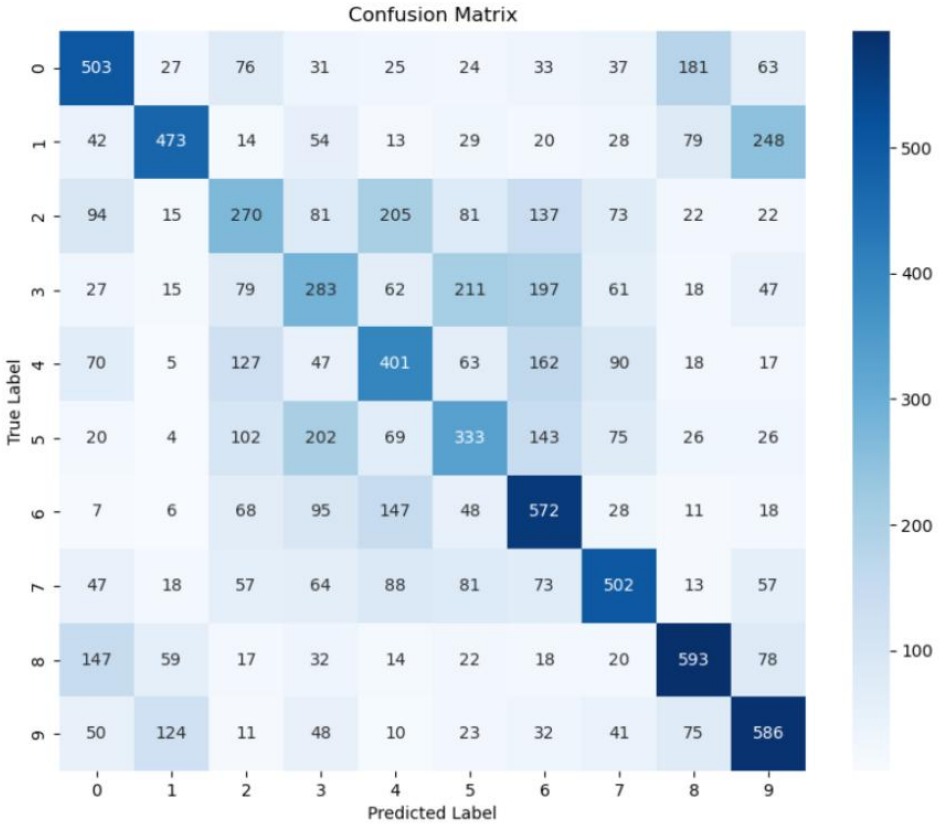

图 10 FCNN+Adboost 预测结果混淆矩阵图

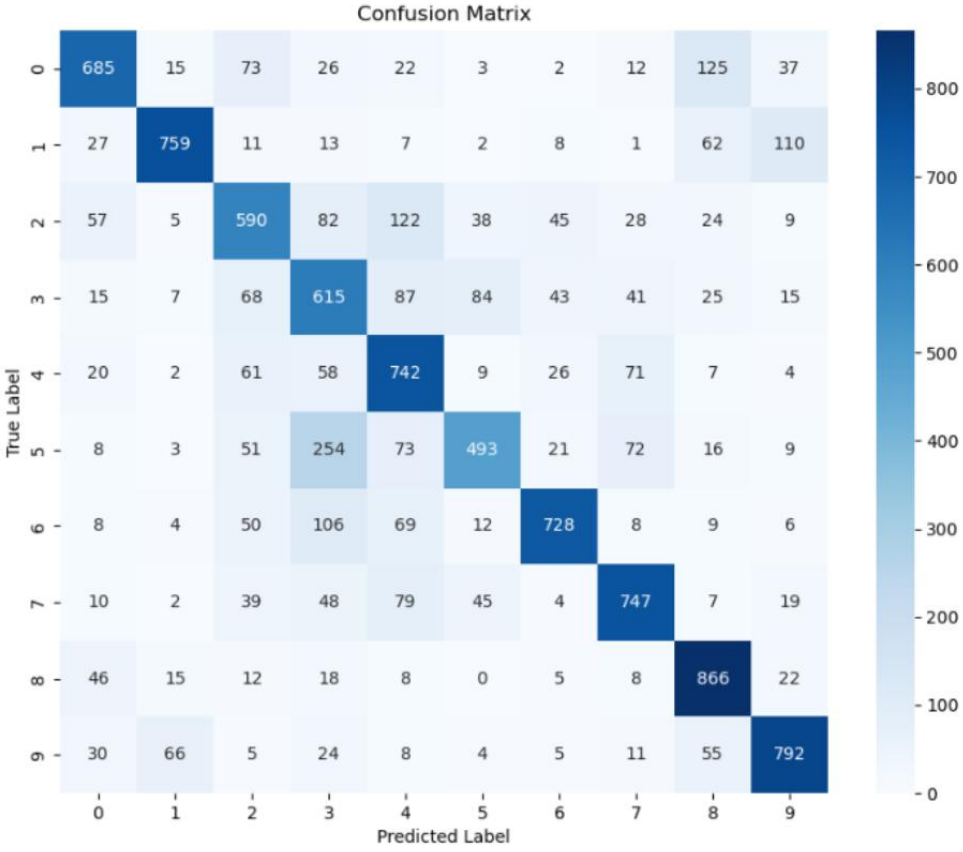

图 11 CNN 预测结果混淆矩阵图

7. 结果分析

7.1 模型的可解释性分析、

7.1.1 可视化卷积层的特征图

在卷积神经网络（CNN）中，卷积层的特征图反映了网络在不同层次上对输入图像特征的提取效果。通过可视化这些特征图，可以帮助我们理解模型在处理图像时的行为和逻辑。

实现方法：通过对训练好的 CNN 模型的卷积层输出进行提取和可视化，展示模型在不同层次上对图像的特征提取效果。

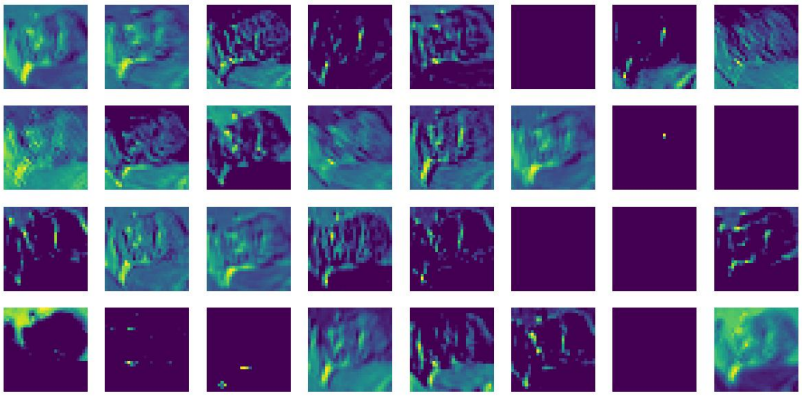

图 11 第一卷积层特征图

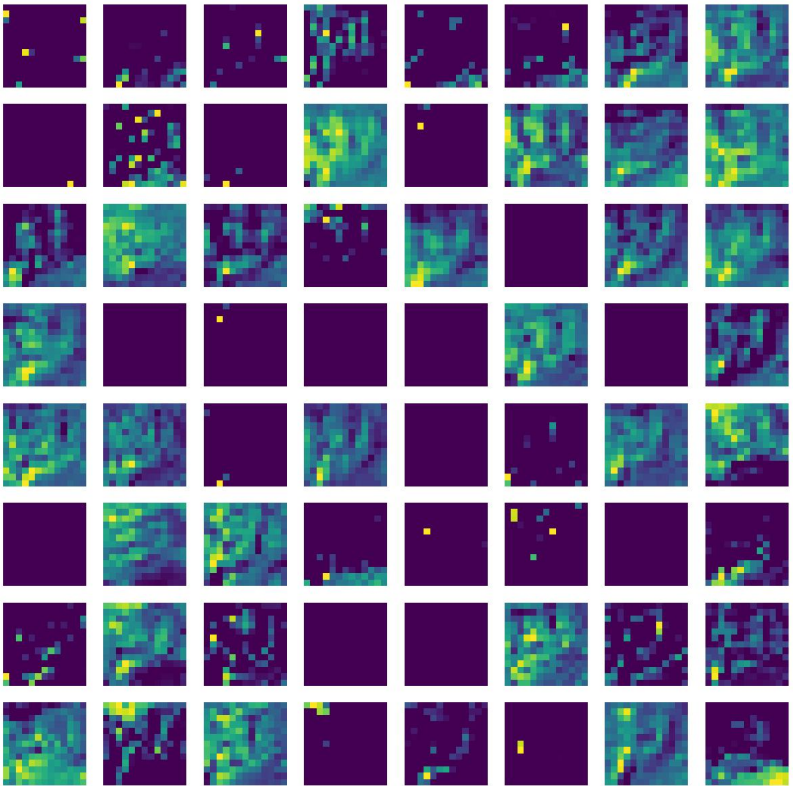

图 12 第二卷积层特征图

分析：通过观察卷积层的特征图，可以看到模型在前几层提取了一些低级特征（如边缘、纹理等），而在后面的层次逐渐提取出更高级的语义特征（如物体的形状、部件等）。这些特征图可以帮助我们理解模型如何一步步地从输入图

像中提取有用的信息。图 12 和 13 分别为第一和第二卷积层的特征图，第一卷积层主要提取了图像的边缘和简单纹理特征，这些特征图显示了模型如何在最初的层次上捕捉到输入图像中的基本结构；第二卷积层在前一层特征的基础上进一步提取更复杂的模式和细节，可以看到，特征图中开始出现更丰富的纹理和形状信息。

7.1.2 分析不同卷积层的特征提取效果

实现方法：通过对比不同卷积层的特征图，可以发现模型在不同层次上对输入图像的特征提取效果。

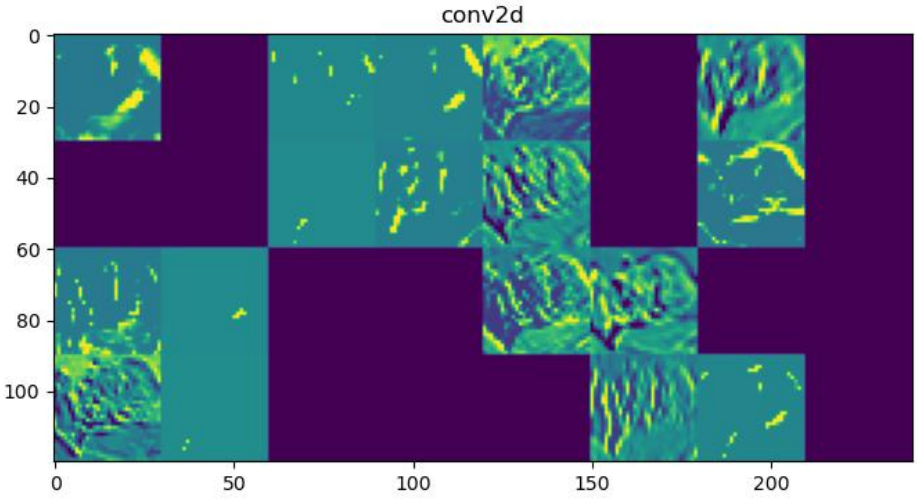

图 13 第一卷积层特征图

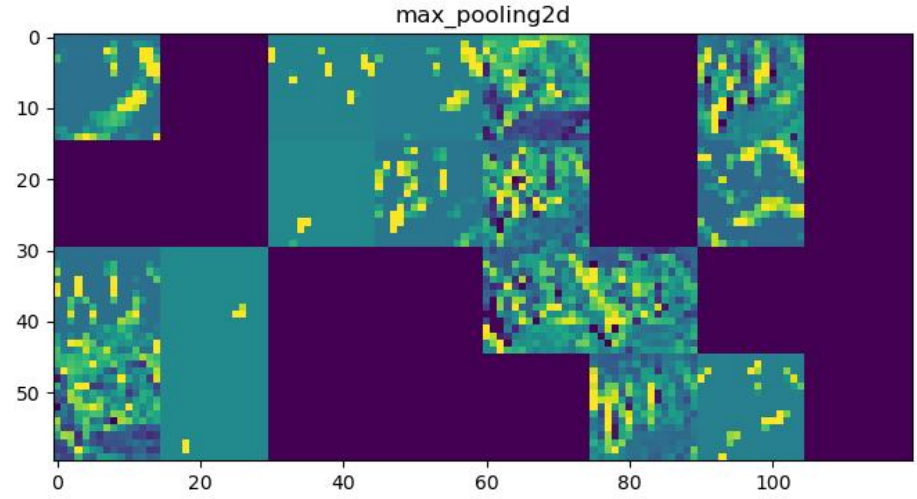

图 14 最大池化层特征图

分析：卷积层越深，提取的特征越抽象。前几层主要关注图像的边缘和简单纹理，而后几层则逐渐捕捉到更高级的语义信息，如物体的形状和类别。通过这些分析，可以帮助我们调整模型结构，以便更好地提取有用特征。第一卷积层特征图（图 13）显示了模型在前期对图像简单纹理和边缘特征的提取效果；最大池化层特征图（图 14）展示了经过池化操作后的特征图，池化层主要用于降

低特征图的尺寸，同时保留重要特征；第二卷积层特征图（图 15）显示了模型在更深层次对图像更复杂特征的提取效果，这些特征包含更多的语义信息。通过这些可视化特征图，我们可以直观地观察到卷积神经网络在处理图像时，不同层次特征提取的行为和逻辑。观察卷积层的特征图，可以看到模型在前几层提取了一些低级特征（如边缘、纹理等），而在后面的层次逐渐提取出更高级的语义特征（如物体的形状、部件等）。这些特征图可以帮助我们理解模型如何一步步地从输入图像中提取有用的信息。

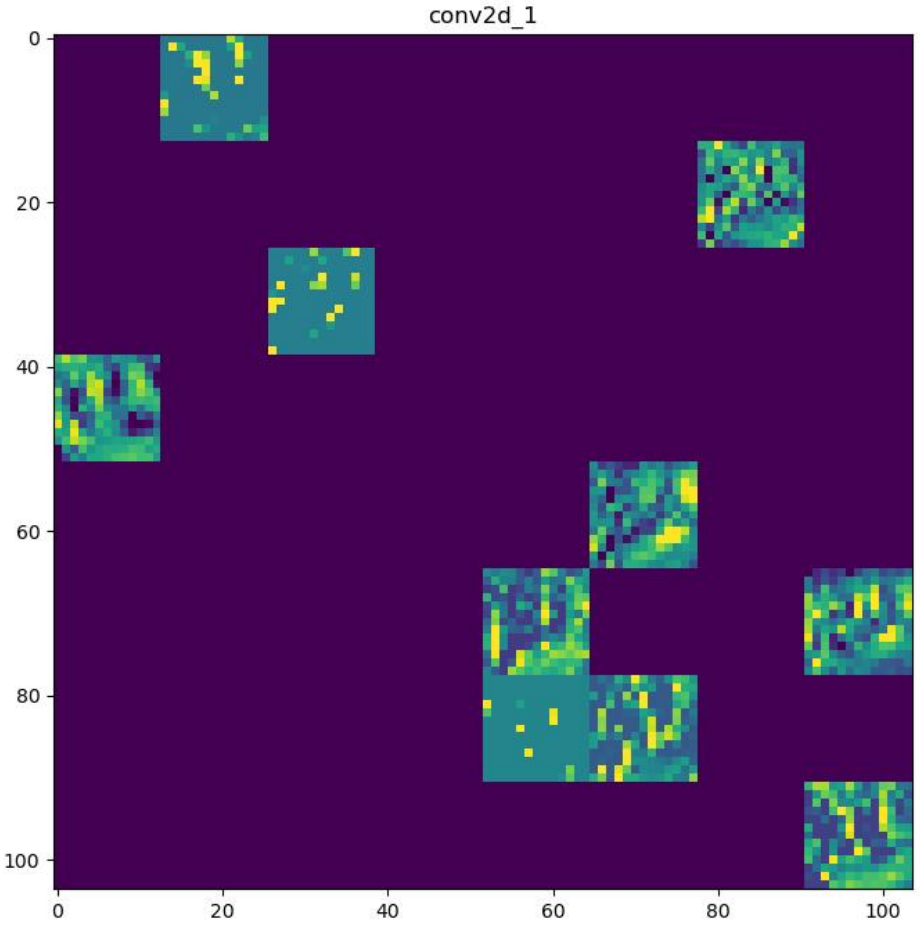

图 15 最大池化层特征图

7.2 超参数实验

7.2.1 不同超参数设置的实验结果比较

在模型训练过程中，超参数的选择对模型性能有着重要影响。我们对不同的超参数设置进行了实验，并比较了它们对模型性能的影响。不同超参数设置的实验结果比较如图 16 所示。

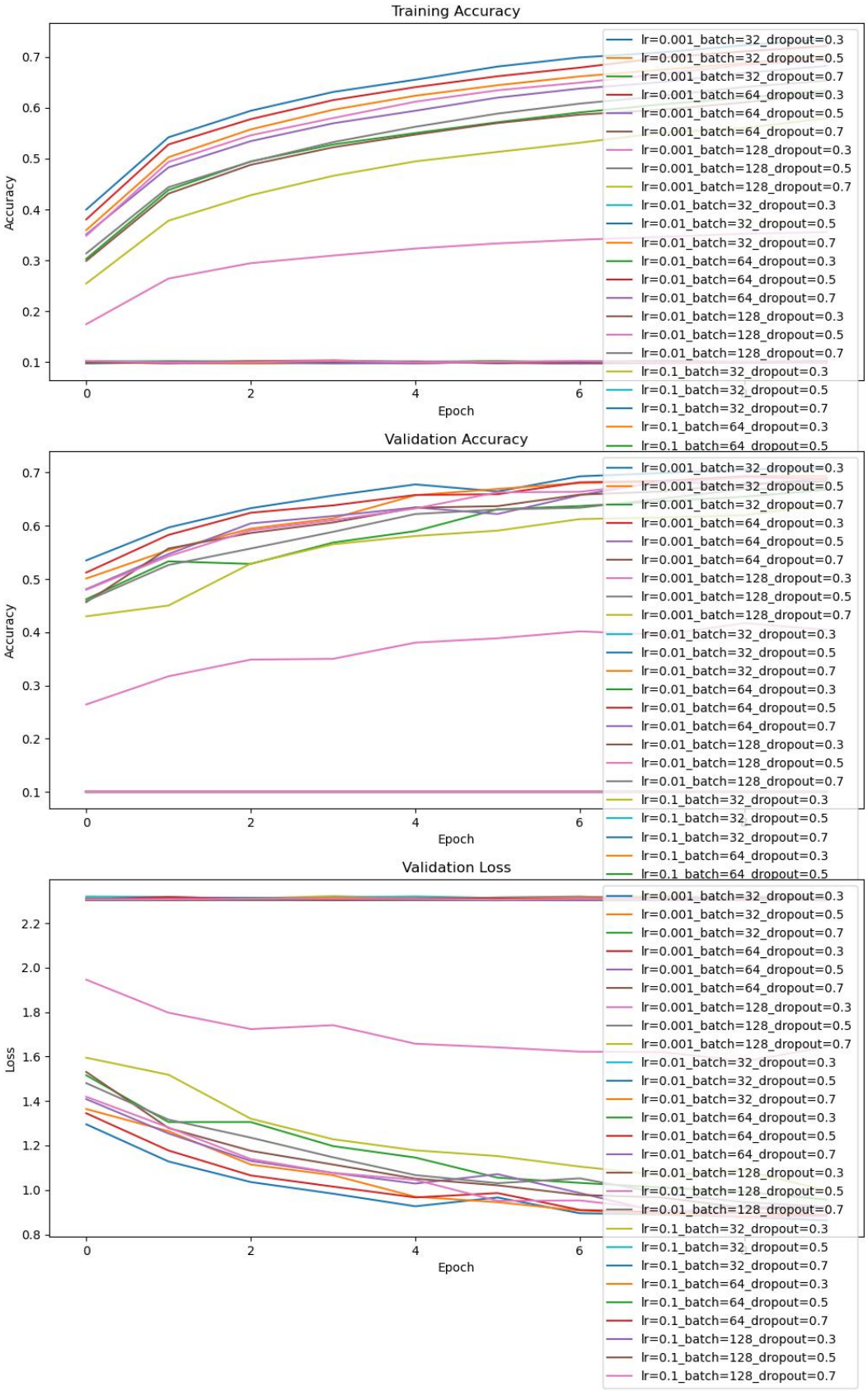

图 16 不同超参数设置的实验结果比较图

7.2.2 学习率、批量大小、Dropout 比例等对模型性能的影响

实现方法：通过实验对比不同的学习率、批量大小、Dropout 比例等超参数设置对模型性能的影响，绘制出不同设置下的模型性能曲线。

分析：

（1） 训练准确率（Training Accuracy）：

- 从训练准确率的曲线图中可以看到，不同超参数设置下模型的收敛速度和最终准确率都有显著差异。
- 学习率过大会导致模型震荡甚至不收敛（如粉色曲线所示），过小则收敛速度较慢。
- 批量大小的选择也需要平衡计算效率和模型性能，较小的批量大小有助于模型更细致地调整参数，但训练时间较长；较大的批量大小可以加快训练速度，但可能会忽略一些细微特征。
- 适当的 Dropout 比例（如 0.5）可以有效防止过拟合，提高模型的泛化能力。

（2） 验证准确率（Validation Accuracy）：

- 从验证准确率的曲线图中可以看出，不同超参数设置对模型在验证集上的表现也有显著影响。
- 学习率过大或过小都会导致模型在验证集上的表现不佳。适当的学习率可以平衡训练速度和验证准确率。
- Dropout 比例过高（如 0.7）会导致模型在验证集上的表现不稳定，过低则不能有效防止过拟合。
- 适当的批量大小有助于模型在验证集上取得更好的表现。

（3） 验证损失（Validation Loss）：

- 从验证损失的曲线图中可以看到，不同超参数设置对模型在验证集上的损失也有显著影响。
- 学习率过大会导致验证损失不稳定，模型难以收敛到较低的损失值。
- Dropout 比例适中（如 0.5）时，模型的验证损失下降较快且较为稳定。

批量大小的选择也对验证损失有影响，适当的批量大小可以帮助模型更好地学习特征，从而降低验证损失。

7.3 数据增强实验

7.3.1 数据增强技术的使用效果

在训练深度学习模型时，数据增强技术可以通过增加训练数据的多样性，提高模型的泛化能力。我们对比了使用和不使用数据增强技术的模型性能。

实现方法：在训练数据集上应用各种数据增强技术，如随机裁剪、旋转、平移等，训练模型并记录其性能。

分析：

（1） 学习率： 学习率过大会导致模型震荡甚至不收敛，过小则收敛速度慢。合适的学习率可以加快收敛速度，提高模型准确率。

（2） 批量大小： 批量大小的选择也需要平衡计算效率和模型性能。批量大小较小时，训练更加稳定，但时间较长；批量较大时，训练速度加快，但需要更多内存。适当的批量大小可以提高训练效率和模型性能。

（3） Dropout 比例： Dropout 是一种防止过拟合的技术。适当的 Dropout 比例可以有效防止过拟合，提高模型的泛化能力，但过高的 Dropout 比例会

导致模型欠拟合。

7.3.2 增强前后的模型性能比较

在训练深度学习模型时，数据增强技术可以通过增加训练数据的多样性，提高模型的泛化能力。我们对比了使用和不使用数据增强技术的模型性能。
实现方法：通过对比应用数据增强技术前后模型的训练和验证准确率、损失值等性能指标，评估数据增强技术的效果。图 17 和 18 分别展示了数据增强前后的准确率和损失函数随着迭代次数变化的折线图，具体数据见表 2。

表 2 数据增强对模型性能的影响

Metric	使用数据增强均值	不使用数据增强均值
Train Accuracy	0.7546	0.1000
Validation Accuracy	0.6981	0.1037
Train Loss	0.6874	1.0995
Validation Loss	0.9970	0.9840

分析： 数据增强技术在本次实验中显著提高了模型的性能，建议在实际应用中使用数据增强技术来提升模型的泛化能力和稳定性。

（1） 使用数据增强：

- 训练准确率和验证准确率均值显著提高，表明数据增强技术有效提高了模型的学习能力和泛化能力
- 训练损失均值较低，验证损失略高，但总体表现更为稳定。

（2） 不使用数据增强：

- 训练准确率和验证准确率均值较低，表明模型在未使用数据增强时的学习和泛化能力有限。
- 训练损失和验证损失均值较高，且损失值在训练后期的变化较大，可能存在过拟合问题。

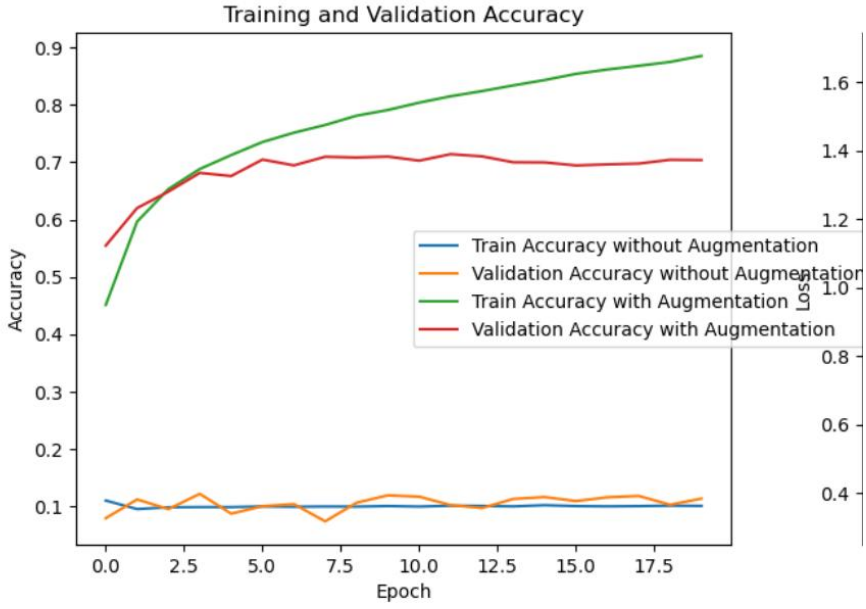

图 17 数据增强前后的准确率随着迭代次数变化的折线图

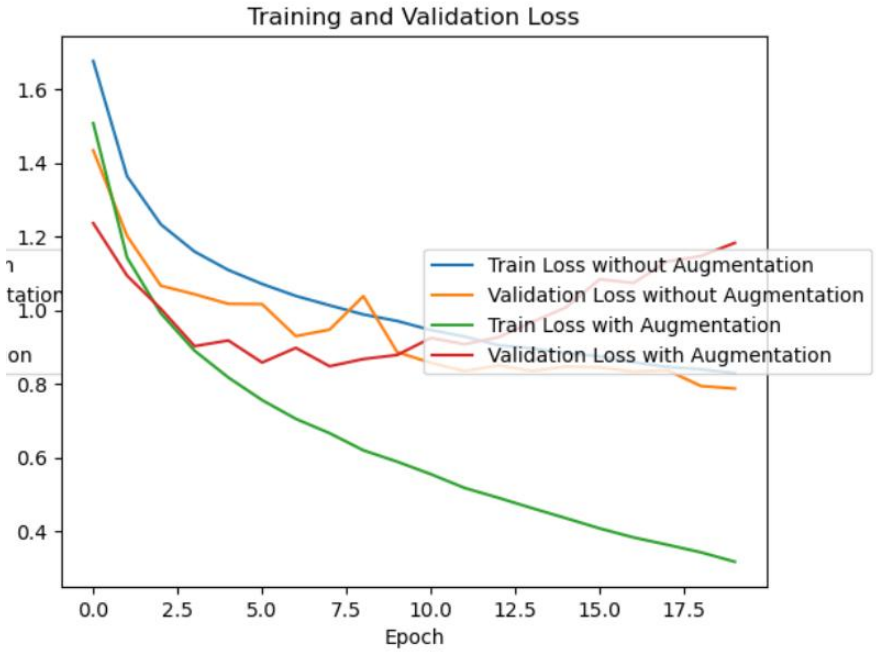

图 18 数据增强前后的损失函数随着迭代次数变化的折线图

8. 结论与展望

8.1 总结工作内容与结果

在本次实验中，我们针对 CIFAR-10 数据集进行了图像分类任务，通过使用全连接前馈神经网络（FCNN）、卷积神经网络（CNN）以及结合 AdaBoost 的 FCNN 模型，详细分析了不同模型在图像分类任务中的表现。以下是工作内容与结果的总结：

（1） 数据预处理：对 CIFAR-10 数据集进行了归一化处理，并将标签进行了扁平化处理，以便于模型训练和测试。

（2） 模型选择与搭建：

- 全连接前馈神经网络（FCNN）：构建了一个包含多个全连接层和 Dropout 层的 FCNN 模型。
- 卷积神经网络（CNN）：设计了一个包含卷积层、池化层和全连接层的 CNN 模型。
- 结合 AdaBoost 的 FCNN 模型：在 FCNN 模型的基础上，结合 AdaBoost 算法，提升模型的分类性能。

（3） 模型训练与评估：

- 对各模型进行了训练，并使用测试集进行了评估。
- 绘制了训练过程中损失值和准确率的变化曲线，展示了模型的收敛情况和性能表现。

（4） 结果可视化：

- 使用混淆矩阵对模型的分类效果进行了详细分析。
- 可视化了测试集上正确分类和错误分类的样本，帮助理解模型的分类效果和不足之处。

（5） 特征图可视化：对 CNN 模型的卷积层特征图进行了可视化，展示了不同层次上提取的图像特征。
（6） 超参数实验：通过调整学习率、批量大小和 Dropout 比例，分析了不同超参数设置对模型性能的影响。
（7） 数据增强实验：应用了数据增强技术，比较了使用数据增强技术前后的模型性能，证明了数据增强技术在提升模型泛化能力方面的有效性。

8.2 提出改进方案与未来工作方向

在本次实验的基础上，我们可以进一步优化和改进模型，提升图像分类任务的性能和准确率。以下是一些改进方案和未来工作方向：

（1） 模型优化：
- 超参数调优：可以使用网格搜索或贝叶斯优化等方法，进一步调整模型的超参数，以找到最佳参数组合，提升模型性能。
- 更复杂的模型结构：尝试使用更复杂的深度学习模型，如 ResNet、DenseNet 等，这些模型在图像分类任务中表现优异。

（2） 数据增强与预处理：
- 高级数据增强技术：引入更多的数据增强技术，如 Cutout、Mixup 等，进一步增加训练数据的多样性，提高模型的泛化能力。
- 数据预处理：尝试使用图像标准化、白化等高级预处理方法，提升模型对输入数据的适应性。

（3） 集成学习：
- 多模型融合：将不同模型的预测结果进行融合，如使用加权平均、投票等方法，提升分类准确率。
- 堆叠模型：构建堆叠模型（Stacking），将多个基础模型的预测结果作为输入，再训练一个上层模型，进一步提升分类性能。

（4） 模型解释性与可视化：
- Grad-CAM：使用 Grad-CAM 等可视化技术，生成模型对特定输入的关注区域图，帮助理解模型的决策过程。
- 特征重要性分析：分析模型对不同输入特征的敏感性，找出对分类结果影响最大的特征。

（5） 迁移学习与预训练模型：
- 迁移学习：使用在大规模数据集（如 ImageNet）上预训练的模型，并对其进行微调，以提升在 CIFAR-10 数据集上的分类性能。
- 半监督学习：利用未标注数据进行训练，结合半监督学习方法，进一步提升模型的泛化能力。

通过这些改进方案和未来工作方向，我们可以不断优化图像分类模型，提升其在 CIFAR-10 数据集及其他图像分类任务中的表现，为实际应用奠定基础。

附录

9.1 代码实现

本报告基于 pycharm 开发环境，代码公开 github 链接为 ccuiyiwen/CIFAR-10_classification_cyw (github.com)。

9.2 参考文献

[1] Yun, S., Han, D., Oh, S. J., Chun, S., Choe, J., & Yoo, Y. (2019). CutMix: Regularization strategy to train strong classifiers with localizable features. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 6023-6032). https://doi.org/10.1109/ICCV.2019.00613

[2] Zhang, H., Cisse, M., Dauphin, Y. N., & Lopez-Paz, D. (2018). mixup: Beyond empirical risk minimization. arXiv preprint arXiv:1710.09412. https://arxiv.org/abs/1710.09412

[3] Cubuk, E. D., Zoph, B., Shlens, J., & Le, Q. V. (2019). RandAugment: Practical automated data augmentation with a reduced search space. In Advances in Neural Information Processing Systems (Vol. 32). https://arxiv.org/abs/1909.13719

[4] Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., & Le, Q. V. (2019). AutoAugment: Learning augmentation policies from data. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 113-123). https://doi.org/10.1109/CVPR.2019.00020

[5] Hendrycks, D., Mu, N., Cubuk, E. D., Zoph, B., Gilmer, J., & Lakshminarayanan, B. (2020). AugMix: A simple data processing method to improve robustness and uncertainty. In International Conference on Learning Representations. https://openreview.net/forum?id=S1gmrxHFvB

[6] DeVries, T., & Taylor, G. W. (2017). Improved regularization of convolutional neural networks with cutout. arXiv preprint arXiv:1708.04552. https://arxiv.org/abs/1708.04552

[7] Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. Journal of Big Data, 6(1), 1-48. https://doi.org/10.1186/s40537-019-0197-0

[8] Sohn, K., Berthelot, D., Carlini, N., Zhang, Z., Zhang, H., Raffel, C. A., ... & Li, C. L. (2020). FixMatch: Simplifying semi-supervised learning with consistency and confidence. arXiv preprint arXiv:2001.07685. https://arxiv.org/abs/2001.07685

[9] Xie, Q., Dai, Z., Hovy, E., Luong, M. T., & Le, Q. V. (2020). Unsupervised data augmentation for consistency training. In Advances in Neural Information Processing Systems (Vol. 33). https://arxiv.org/abs/1904.12848

[10] Chen, T., Kornblith, S., Norouzi, M., & Hinton, G. (2020). A simple framework for contrastive learning of visual representations. In International Conference on Machine Learning (pp. 1597-1607). PMLR. https://proceedings.mlr.press/v119/chen20j.html