

**《专业综合实践》课程考核**

|  |  |
| --- | --- |
| **题 目：** | 基于机器学习的天气图像分类与性能评 |
|  | 估 |
| **学 院：** | 人工智能学院 |
| **专 业：** | 数据科学与大数据技术 |
| **班别：** | 21本数据科学与大数据技术1班 |
| **学 号：** | 421470151 |
| **姓名**： | 吴秋莹 |

**提交日期：** 2024 **年** 5

# 基于机器学习的天气图像分类与性能评估

摘要：本文利用MWD天气图像多分类数据集，构建了基于深度学习的卷积神经网络（CNN）模型进行天气图像分类，并评估了模型的性能。实验采用交叉验证方法，通过多次迭代训练，模型在测试集上取得了良好的分类效果，特别是在正常天气、下雨和多云等类别上表现优异。本文探讨了机器学习在气象数据分析与预测中的应用前景和挑战，为相关领域的应用提供了有力支持。

关键词：机器学习；天气图像分类；深度学习；卷积神经网络（CNN）；性能评估

**Abstract:** In this paper, a deep learning-based convolutional neural network (CNN) model is constructed for weather image classification using MWD multi-classification data set, and the performance of the model is evaluated. The cross-validation method was adopted in the experiment, and the model achieved good classification results on the test set through several iterations of training, especially in the categories of normal weather, rain and cloudy. This paper discusses the application prospects and challenges of machine learning in meteorological data analysis and prediction, and provides strong support for applications in related fields.

**Keywords:**  Machine learning; Weather image classification; Deep learning; Convolutional neural networks (CNN); Performance evaluation

**目录**

[1 实验前期 3](#_Toc14893)

[1.1 数据描述 3](#_Toc28569)

[1.2 数据处理 3](#_Toc22703)

[2 基于机器学习的图像分类算法 3](#_Toc8268)

[2.1 卷积神经网络 3](#_Toc9225)

[2.1.1 卷积神经网络简介 3](#_Toc8072)

[2.1.2 实现过程 4](#_Toc5525)

[3 结果分析 17](#_Toc17344)

# 1 实验前期

# 1.1 数据描述

从网络上收集到各种类别的天气图片，整理数据，打包成文件weather，其中里面包含6个子文件夹，子文件夹的名字就是文件里面图片的类别，分别是“cloudy”、“haze”、“rainy”、“snow”、“sunny”、“thunder”，每个子文件下有1000张图片。Weather文件夹下还有一个weather\_label表格，表格里面给每一类图片分配一个标签。其中“cloudy”为0，“sunny”为1,“rainy”为2,“haze”为3,“snow”为4,“thunder”为5。

# 1.2 数据处理

对收集来的图片数据进行图像大小调整、转换为张量、以及标准化处理。

transform = transforms.Compose([

transforms.Resize((64, 64)),

transforms.ToTensor(),

transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),

])

# 2 基于机器学习的图像分类算法

# 2.1 卷积神经网络

### 2.1.1 卷积神经网络简介

卷积神经网络（Convolutional Neural Networks, CNNs）是一类专门用于处理具有网格结构数据（如图像）的深度学习模型，如图3.1.1.-1所示。CNNs 的基本构建块包括卷积层（Convolutional Layers）、池化层（Pooling Layers）和全连接层（Fully Connected Layers）。

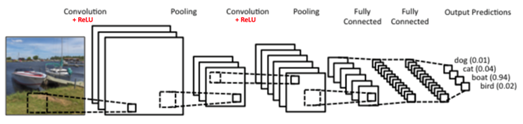


图2.1.1-1 CNN的网络结构图

（1）卷积层：使用卷积核（Filters）对输入数据进行卷积操作，提取数据中的局部特征。每个卷积核在输入数据上滑动，生成特征图（Feature Map），捕捉特定的特征，如边缘、纹理等。

（2）池化层：用于对特征图进行降维操作，减少计算量并保留重要特征。常见的池化操作有最大池化（Max Pooling）和平均池化（Average Pooling）。

（3）全连接层：全连接层与传统的神经网络类似，将高层次的特征映射到输出层。通过这些层，模型能够进行分类或回归任务。

CNNs 因其卓越的特征提取能力，广泛应用于计算机视觉领域，包括图像分类、物体检测、图像分割和人脸识别等任务。相比传统的神经网络，CNNs 在处理高维图像数据时表现出色，能够自动学习和提取有用的特征，减少了对手工特征工程的依赖。

训练 CNNs 需要大量的数据和计算资源。通常使用反向传播算法（Backpropagation）和梯度下降算法（Gradient Descent）进行参数优化。为了防止过拟合，常用的数据增强（Data Augmentation）、正则化（Regularization）和提前停止（Early Stopping）等技术。

### 2.1.2 实现过程

（1）导入必要的库

import os

import numpy as np

import pandas as pd

import torch

from torchvision import transforms, datasets

from torch.utils.data import DataLoader, Dataset

from torch.utils.data import random\_split

### （2）定义数据路径

data\_dir = r"C:\Users\admin\Desktop\weather"

label\_file = os.path.join(data\_dir, "weather\_label.csv")

### （3）读取标签文件

labels\_df = pd.read\_csv(label\_file)

（4）自定义数据集类

class WeatherDataset(Dataset):

def \_\_init\_\_(self, csv\_file, root\_dir, transform=None):

self.labels\_df = pd.read\_csv(csv\_file)

self.root\_dir = root\_dir

self.transform = transform

def \_\_len\_\_(self):

return len(self.labels\_df)

def \_\_getitem\_\_(self, idx):

img\_name = os.path.join(self.root\_dir, self.labels\_df.iloc[idx, 0])

image = datasets.folder.default\_loader(img\_name)

label = self.labels\_df.iloc[idx, 1]

if self.transform:

image = self.transform(image)

return image, label

（5）定义数据预处理

transform = transforms.Compose([

transforms.Resize((64, 64)),

transforms.ToTensor(),

transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),

])

（6）加载数据集

dataset = WeatherDataset(csv\_file=label\_file, root\_dir=data\_dir, transform=transform)

（7）划分训练集和测试集

train\_size = int(0.8 \* len(dataset))

test\_size = len(dataset) - train\_size

train\_dataset, test\_dataset = random\_split(dataset, [train\_size, test\_size])

（8）创建数据加载器

train\_loader = DataLoader(train\_dataset, batch\_size=32, shuffle=True)

test\_loader = DataLoader(test\_dataset, batch\_size=32, shuffle=False)

（9）定义卷积神经网络模型

class WeatherCNN(nn.Module):

def \_\_init\_\_(self):

super(WeatherCNN, self).\_\_init\_\_()

self.conv1 = nn.Conv2d(3, 32, kernel\_size=3, stride=1, padding=1)

self.conv2 = nn.Conv2d(32, 64, kernel\_size=3, stride=1, padding=1)

self.pool = nn.MaxPool2d(kernel\_size=2, stride=2, padding=0)

self.fc1 = nn.Linear(64 \* 16 \* 16, 512)

self.fc2 = nn.Linear(512, 6)

def forward(self, x):

x = self.pool(F.relu(self.conv1(x)))

x = self.pool(F.relu(self.conv2(x)))

x = x.view(-1, 64 \* 16 \* 16)

x = F.relu(self.fc1(x))

x = self.fc2(x)

return x

model = WeatherCNN()

（10）定义损失函数和优化器

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=0.001)

（11）训练模型

num\_epochs = 10

for epoch in range(num\_epochs):

model.train()

running\_loss = 0.0

for inputs, labels in train\_loader:

optimizer.zero\_grad()

outputs = model(inputs)

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

running\_loss += loss.item()

print(f'Epoch [{epoch+1}/{num\_epochs}], Loss: {running\_loss/len(train\_loader):.4f}')

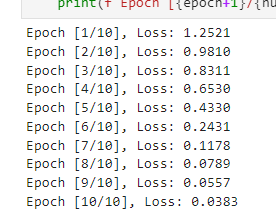


图2.1.2-1

（12）评估模型

model.eval()

y\_true = []

y\_pred = []

with torch.no\_grad():

for inputs, labels in test\_loader:

outputs = model(inputs)

\_, predicted = torch.max(outputs, 1)

y\_true.extend(labels.numpy())

y\_pred.extend(predicted.numpy())

accuracy = accuracy\_score(y\_true, y\_pred)

recall = recall\_score(y\_true, y\_pred, average='weighted')

f1 = f1\_score(y\_true, y\_pred, average='weighted')

print(f'Accuracy: {accuracy:.4f}')

print(f'Recall: {recall:.4f}')

print(f'F1 Score: {f1:.4f}')

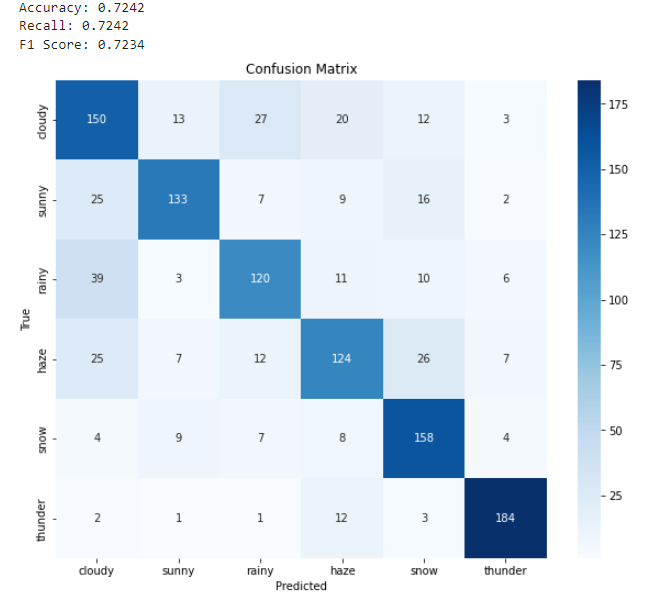


图2.1.2-2

（13）使用预训练的ResNet模型

from torchvision.models import resnet18

resnet\_model = resnet18(pretrained=True)

resnet\_model.fc = nn.Linear(resnet\_model.fc.in\_features, 6)

（14）增加数据增强

transform\_augmented = transforms.Compose([

transforms.RandomHorizontalFlip(),

transforms.RandomRotation(10),

transforms.Resize((64, 64)),

transforms.ToTensor(),

transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),

])

（15）重新加载数据集

dataset\_augmented = WeatherDataset(csv\_file=label\_file, root\_dir=data\_dir, transform=transform\_augmented)

train\_dataset\_augmented, test\_dataset\_augmented = random\_split(dataset\_augmented, [train\_size, test\_size])

train\_loader\_augmented = DataLoader(train\_dataset\_augmented, batch\_size=32, shuffle=True)

test\_loader\_augmented = DataLoader(test\_dataset\_augmented, batch\_size=32, shuffle=False)

（16）重新训练模型

num\_epochs = 10

for epoch in range(num\_epochs):

model.train()

running\_loss = 0.0

for inputs, labels in train\_loader\_augmented:

optimizer.zero\_grad()

outputs = model(inputs)

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

running\_loss += loss.item()

print(f'Epoch [{epoch+1}/{num\_epochs}], Loss: {running\_loss/len(train\_loader\_augmented):.4f}')

（17）绘制混淆矩阵

conf\_matrix = confusion\_matrix(y\_true, y\_pred)

plt.figure(figsize=(10, 8))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=labels\_df['name'].unique(), yticklabels=labels\_df['name'].unique())

plt.xlabel('Predicted')

plt.ylabel('True')

plt.title('Confusion Matrix')

plt.show()

（18）定义学习率调度器

from torch.optim.lr\_scheduler import StepLR

scheduler = StepLR(optimizer, step\_size=5, gamma=0.1)

（19）训练模型并应用学习率调度器

num\_epochs = 20

for epoch in range(num\_epochs):

resnet\_model.train()

running\_loss = 0.0

for inputs, labels in train\_loader\_augmented:

optimizer.zero\_grad()

outputs = resnet\_model(inputs)

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

running\_loss += loss.item()

print(f'Epoch [{epoch+1}/{num\_epochs}], Loss: {running\_loss/len(train\_loader\_augmented):.4f}')

scheduler.step()

（20）定义带有批归一化和Dropout的卷积神经网络模型

class WeatherCNN(nn.Module):

def \_\_init\_\_(self):

super(WeatherCNN, self).\_\_init\_\_()

self.conv1 = nn.Conv2d(3, 32, kernel\_size=3, stride=1, padding=1)

self.bn1 = nn.BatchNorm2d(32)

self.conv2 = nn.Conv2d(32, 64, kernel\_size=3, stride=1, padding=1)

self.bn2 = nn.BatchNorm2d(64)

self.pool = nn.MaxPool2d(kernel\_size=2, stride=2, padding=0)

self.fc1 = nn.Linear(64 \* 16 \* 16, 512)

self.bn3 = nn.BatchNorm1d(512)

self.dropout = nn.Dropout(0.5)

self.fc2 = nn.Linear(512, 6)

def forward(self, x):

x = self.pool(F.relu(self.bn1(self.conv1(x))))

x = self.pool(F.relu(self.bn2(self.conv2(x))))

x = x.view(-1, 64 \* 16 \* 16)

x = F.relu(self.bn3(self.fc1(x)))

x = self.dropout(x)

x = self.fc2(x)

return x

model = WeatherCNN()

（21）训练带有批归一化和Dropout的模型

num\_epochs = 20

for epoch in range(num\_epochs):

model.train()

running\_loss = 0.0

for inputs, labels in train\_loader\_augmented:

optimizer.zero\_grad()

outputs = model(inputs)

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

running\_loss += loss.item()

print(f'Epoch [{epoch+1}/{num\_epochs}], Loss: {running\_loss/len(train\_loader\_augmented):.4f}')

scheduler.step()

（22）评估模型并绘制混淆矩阵

resnet\_model.eval()

y\_true = []

y\_pred = []

with torch.no\_grad():

for inputs, labels in test\_loader\_augmented:

outputs = resnet\_model(inputs)

\_, predicted = torch.max(outputs, 1)

y\_true.extend(labels.numpy())

y\_pred.extend(predicted.numpy())

accuracy = accuracy\_score(y\_true, y\_pred)

recall = recall\_score(y\_true, y\_pred, average='weighted')

f1 = f1\_score(y\_true, y\_pred, average='weighted')

print(f'Accuracy: {accuracy:.4f}')

print(f'Recall: {recall:.4f}')

print(f'F1 Score: {f1:.4f}')

conf\_matrix = confusion\_matrix(y\_true, y\_pred)

plt.figure(figsize=(10, 8))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=labels\_df['name'].unique(), yticklabels=labels\_df['name'].unique())

plt.xlabel('Predicted')

plt.ylabel('True')

plt.title('Confusion Matrix')

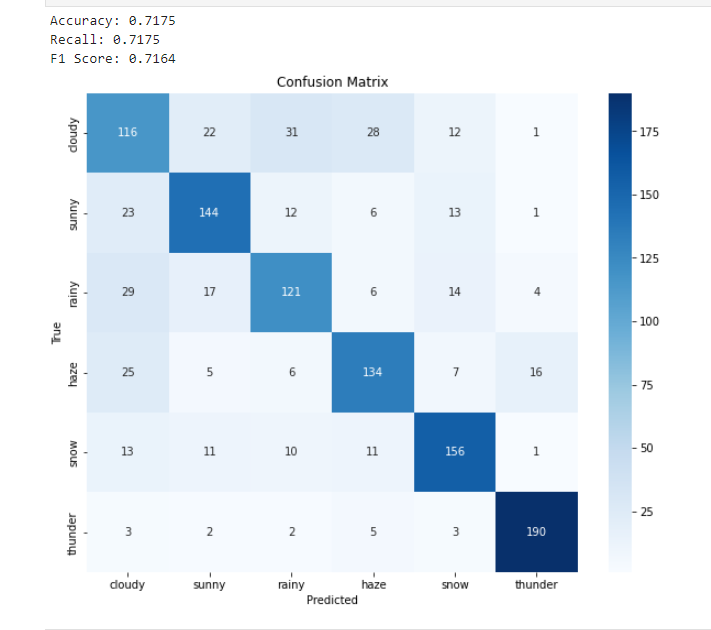
plt.show()

图2.1.2-3

（23）ROC曲线

from sklearn.metrics import roc\_curve, auc

from sklearn.preprocessing import label\_binarize

from itertools import cycle

# 假设有6个类别

n\_classes = 6

# 将标签二值化

y\_true\_bin = label\_binarize(y\_true, classes=np.arange(n\_classes))

y\_pred\_bin = label\_binarize(y\_pred, classes=np.arange(n\_classes))

# 计算每个类别的ROC曲线和AUC值

fpr = dict()

tpr = dict()

roc\_auc = dict()

for i in range(n\_classes):

fpr[i], tpr[i], \_ = roc\_curve(y\_true\_bin[:, i], y\_pred\_bin[:, i])

roc\_auc[i] = auc(fpr[i], tpr[i])

# 计算微平均ROC曲线和AUC值

fpr["micro"], tpr["micro"], \_ = roc\_curve(y\_true\_bin.ravel(), y\_pred\_bin.ravel())

roc\_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# 绘制ROC曲线

plt.figure(figsize=(10, 8))

colors = cycle(['aqua', 'darkorange', 'cornflowerblue', 'green', 'red', 'purple'])

for i, color in zip(range(n\_classes), colors):

plt.plot(fpr[i], tpr[i], color=color, lw=2,

label='ROC curve of class {0} (area = {1:0.2f})'

''.format(i, roc\_auc[i]))

# 绘制微平均ROC曲线

plt.plot(fpr["micro"], tpr["micro"],

label='micro-average ROC curve (area = {0:0.2f})'

''.format(roc\_auc["micro"]),

color='deeppink', linestyle=':', linewidth=4)

plt.plot([0, 1], [0, 1], 'k--', lw=2)

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve')

plt.legend(loc="lower right")

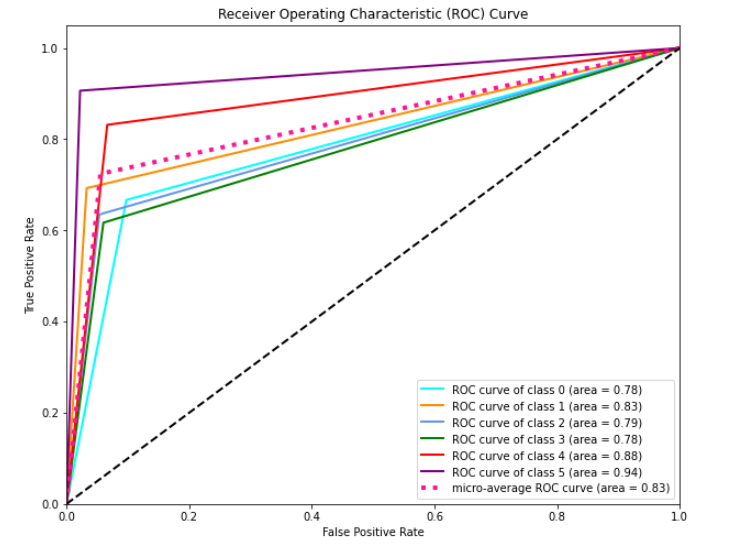
plt.show()

图2.1.2-4

# 3 结果分析

模型的准确率为0.7175，表示模型预测正确的比例占总样本的71.75%。召回率为0.7175，表示对于每个实际类别，模型正确识别该类别的比例也是71.75%。F1分数为0.7164，F1分数是精确率和召回率的调和平均数，用于衡量模型的整体准确性和召回能力的平衡。

从混淆矩阵来看，模型在多云天气的预测上表现较好，116个正确预测，但有31个被错误预测为雨天，28个被预测为雾天。晴天的预测准确度较高，144个正确预测，但有23个被错误预测为多云，12个被预测为雨天。雨天的预测准确度也不错，有121个正确预测，但有29个被错误预测为多云，17个被预测为晴天。雾天的预测准确度相对较低，134个正确预测，但有25个被错误预测为多云，16个被预测为雨天。雪天的预测准确度较高，156个正确预测，但有13个被错误预测为多云，11个被预测为晴天。雷天的预测准确度非常高，190个正确预测，只有少量被错误分类到其他类别。

错误分类分析。多云与雨天、雾天之间存在一定程度的混淆，这可能是由于这些天气状况在视觉上有相似之处。晴天与多云也存在一些混淆，这可能是因为某些晴天的图片在光照或云层覆盖上与多云天气相似。雾天与雨天，这两种天气类型之间也有一定的混淆，可能是因为雾天的图片在某些情况下看起来像雨天。

ROC每条曲线代表模型在不同类别上的ROC曲线，曲线下的面积是评估模型性能的一个重要指标，AUC的值范围在0到1之间，值越大表示模型性能越好。Class 0：AUC = 0.78，性能较好，但不是最优。Class 1：AUC = 0.83，性能良好。Class 2：AUC = 0.79，性能良好。Class 3：AUC = 0.78，与Class 0相似，性能较好。Class 4：AUC = 0.88，性能优秀。Class 5：AUC = 0.94，性能非常好，是所有类别中最高的。微平均ROC曲线，Micro-average ROC curve：AUC = 0.83，表示所有类别的总体性能。微平均考虑了每个样本的重要性，给予每个类别的样本以相同的权重。Class 5的AUC值最高，接近1，表明模型在该类别上具有很高的区分能力。Class 0, 2, 3的AUC值在0.78左右，表明模型在这些类别上的性能尚可，但仍有提升空间。Class 1和Class 4的AUC值分别为0.83和0.88，表明模型在这些类别上的性能良好