卷积神经网络-猫狗识别

• 姓名: 刘文博

班级:数据科学四班学号:320200931121

实验背景

图像分类一直是机器学习和计算机视觉领域的重要问题之一。随着大量图像数据的产生和普及,如何有效地对图像进行分类和识别成为了一个具有挑战性的任务。在这个实验中,我们将使用卷积神经网络(Convolutional Neural Network,简称CNN)来解决一个经典的图像识别问题,即猫狗图像的分类任务。

Kaggle是一个著名的数据科学竞赛平台,吸引了来自全球的数据科学家和机器学习工程师。其中,猫狗图像分类数据集是一个广泛应用于图像识别领域的经典数据集。该数据集包含大量已经标记好的猫和狗的图像,我们的目标是通过训练一个CNN模型,能够对新的猫狗图像进行准确的分类判断。

卷积神经网络(CNN)是一种专门用于处理图像数据的深度学习算法。与传统的神经网络相比, CNN在图像处理任务上具有更强的表现力和适应能力。它能够自动从原始图像中提取特征,并通过学习从而实现对图像的准确分类。

在该实验中,我们将使用Python编程语言和深度学习框架(TensorFlow)来实现卷积神经网络。 我们的目标是构建一个能够准确分类猫和狗图像的CNN模型。为了达到这个目标,我们将按照以 下步骤进行实验:

- 1. 数据集准备:我们将首先从Kaggle网站下载猫狗图像分类数据集,并对数据进行预处理和划分,以便用于训练、验证和测试。
- 2. 模型设计: 我们将设计一个包含卷积层、池化层和全连接层的CNN模型。我们将解释每一层的作用和原理,并说明如何选择适当的激活函数和损失函数。
- 3. 模型训练:我们将使用训练集的图像数据来训练CNN模型,并通过反向传播算法和梯度下降 优化来更新模型的权重参数。
- 4. 模型评估:我们将使用验证集的图像数据来评估已训练好的模型的性能。我们将选择适当的评估指标,如准确率。
- 5. 模型优化与调整:根据模型在验证集上的表现,我们将进行模型的优化与调整,以提高模型的性能和泛化能力。

通过完成这个实验,我们将能够学习和掌握如何使用卷积神经网络来解决图像识别问题。这将为我们日后在其他图像分类任务中应用CNN提供基础和思路。

数据集介绍

Kaggle的猫狗数据集是一个常用的图像分类数据集,用于训练和测试机器学习模型来区分猫和狗的图像。这个数据集包含了25,000张猫和狗的彩色图像(已标记)用于训练和12500张猫和狗的彩色图像(未标记)用于测试。训练集中包含12,500张狗的图像和12,500张猫的图像,测试集中包含6250张狗的图像和6250张猫的图像。

数据集中的图像是真实世界中的宠物猫和狗的照片,拍摄角度、光照条件、尺寸和品种各不相同,使得数据集更贴近真实场景。每个图像都被命名为"dog.1.jpg"、"cat.2.jpg"这样的方式,以区分狗和猫的类别。

实验方法

1 数据准备:

- 下载Kaggle的猫狗数据集,并解压缩。
- 将数据集分成训练集、验证集和测试集,可以使用60%的数据作为训练集,20%的数据 作为验证集,20%的数据作为测试集。
- 对图像进行预处理,以便于模型的训练和测试。

2. 模型设计:

- 选择卷积神经网络(CNN)作为模型的基础架构,由于CNN在图像分类任务上表现出色。
- 设计模型的层数、卷积核大小、池化层等结构,可以参考先前的经验和相关文献。
- 添加全连接层和激活函数以加入非线性特性,并进行分类。

3. 模型训练:

- 使用训练集进行模型的训练。将图像输入到模型中,并根据实际分类标签来计算损失函数。
- 选择适当的优化算法,如随机梯度下降(SGD)或Adam来更新模型的参数,最小化损失函数。
- 设置训练的超参数,如学习率、批次大小和训练的迭代次数,以获得最佳的训练效果。

4. 模型评估:

- 使用验证集对训练好的模型进行评估, 计算模型的准确率。
- 根据评估结果,对模型进行调整和优化,如调整超参数、增加训练数据量等,以提高模型的性能。

5. 模型优化与调整:

- 观察模型在训练和验证集上的表现,进行模型调整和优化,以提高模型的泛化能力和准确率。
- 可以尝试使用数据增强技术,如旋转、平移、镜像等操作来扩充训练数据集,提高模型的鲁棒性。

在进行实验时,注意合理设置实验环境,如选择适当的计算设备(如GPU)、使用合适的深度学习框架(如TensorFlow)和相关的库和工具(如NumPy、Matplotlib等)来支持实验的进行和结果分析。

实验过程

数据预处理

原始数据集位于./data/original目录下。将原始数据集中的训练集按照6:2:2的比例划分成训练集、验证集和测试集保存在./data/new目录下,在保存时将猫和狗的数据分开存放。数据预处理的完整代码如下。

```
import os
import shutil
# 原始数据集路径
original train dir = '../data/original/train'
# 新数据集路径
new train dir = '../data/new/train'
new_validation_dir = '../data/new/validation'
new_test_dir = '../data/new/test'
new_dir_lst = [new_train_dir, new_validation_dir, new_test_dir]
for i in range(12500):
    src_cat = os.path.join(original_train_dir, f'cat.{i}.jpg')
    src_dog = os.path.join(original_train_dir, f'dog.{i}.jpg')
    if i < 7500:
        dst_cat = os.path.join(new_train_dir, 'cats', f'cat.{i}.jpg')
        dst_dog = os.path.join(new_train_dir, 'dogs', f'dog.{i}.jpg')
        shutil.copyfile(src_cat, dst_cat)
        shutil.copyfile(src_dog, dst_dog)
    elif i < 10000:
        dst_cat = os.path.join(new_validation_dir, 'cats', f'cat.{i}.jpg')
        dst_dog = os.path.join(new_validation_dir, 'dogs', f'dog.{i}.jpg')
        shutil.copyfile(src_cat, dst_cat)
        shutil.copyfile(src_dog, dst_dog)
    else:
        dst_cat = os.path.join(new_test_dir, 'cats', f'cat.{i}.jpg')
        dst_dog = os.path.join(new_test_dir, 'dogs', f'dog.{i}.jpg')
        shutil.copyfile(src cat, dst cat)
        shutil.copyfile(src_dog, dst_dog)
print('total training cat images:', len(os.listdir('../data/new/train/cats')))
print('total training dog images:', len(os.listdir('../data/new/train/dogs')))
```

```
print('total validation cat images:',
len(os.listdir('../data/new/validation/cats')))
print('total validation dog images:',
len(os.listdir('../data/new/validation/dogs')))
print('total test cat images:', len(os.listdir('../data/new/test/cats')))
print('total test dog images:', len(os.listdir('../data/new/test/dogs')))
```

预处理结果如下:

```
total training cat images: 7500
total training dog images: 7500
total validation cat images: 2500
total validation dog images: 2500
total test cat images: 2500
total test dog images: 2500
```

模型设计

设计卷积神经网络,其中包含卷积层、池化层和全连接层。使用relu函数作为中间层激活函数, sigmoid函数作为输出层激活函数进行二分类。

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)))
model.add(tf.keras.layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu'))
model.add(tf.keras.layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.Conv2D(128, (3, 3), activation='relu'))
model.add(tf.keras.layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.Conv2D(128, (3, 3), activation='relu'))
model.add(tf.keras.layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.Dense(1024, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='relu'))
```

模型训练

配置模型时,采用Adam优化算法来更新模型的参数,最小化损失函数。采用Binary Cross-Entropy作为损失函数。采用准确率作为评价函数用于评估当前模型的性能,评价函数的结果不会用于训练过程中。

```
metrics=['accuracy'])
```

Keras中的ImageDataGenerator是一个用于数据增强的工具,它可以生成增加样本多样性的图像数据。我们将其用于训练集和验证集的图像归一化以及训练集的数据增强。在数据增强的过程中,使用rotation_range控制图像随机旋转的角度范围,width_shift_range控制图像水平方向上的随机平移范围,height_shift_range控制图像垂直方向上的随机平移范围,shear_range控制剪切强度的范围,zoom_range控制图像的随机缩放范围,horizontal_flip控制是否随机对图像进行水平翻转。

flow_from_directory是Keras中一个用于从文件目录中读取图像数据并进行批量生成的函数。它具有以下功能:

- 从文件目录中读取图像数据:它会自动遍历指定的目录,并加载目录中的图像文件。每个子目录一般对应一个类别。
- 将图像数据进行预处理:它可以对图像进行一些预处理操作,如缩放、归一化、旋转、翻转等,以增强模型的泛化能力和抵抗过拟合。
- 批量生成图像数据:它会自动将加载的图像数据按照指定的批次大小进行划分,并在模型训练和评估时逐批提供给模型。这样可以避免一次性将所有图像数据加载到内存中,从而节省内存资源。
- 自动标记类别: 它会自动为加载的图像数据分配对应的类别标签, 方便模型学习和预测。

在这里我将batch size设置为128,这是为了加快模型的训练。

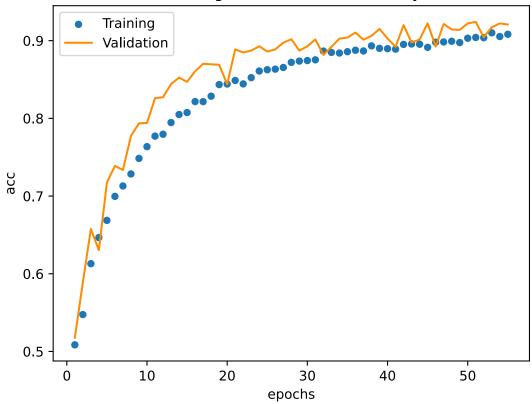
在正式训练模型前设置早停策略。当验证集上的loss值在5次epoch中没有明显变化时结束训练。 然后开始训练模型并将训练的模型保存在./model目录下。

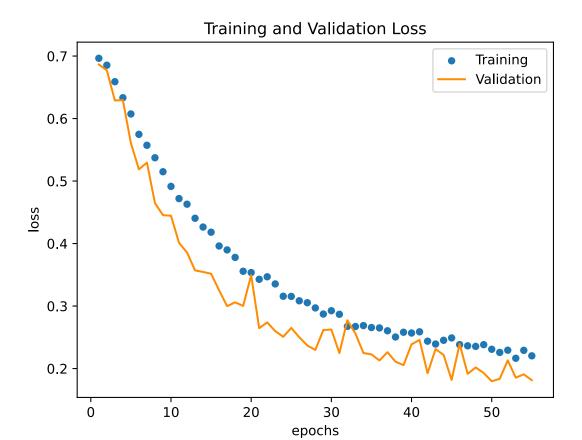
模型的部分训练过程如下,可以看到在迭代到55次时早停策略发挥作用,模型结束训练。

模型评估

在模型训练过程中,accuracy和loss的变化如下图所示。可以看到模型的准确率达到90%,且模型在训练集和验证集上的结果十分接近,说明没有产生过拟合的问题。模型的总体效果较好。

Training and Validation Accuracy





模型优化与调整

该模型是经过多次优化后的最终结果,模型的优化部分为:

• 优化器: 使用Adam代替随机梯度下降 (SGD) 作为优化算法;

• 早停策略:设置早停策略;

• 优化batch大小: 采用更大的batch size;

实验结果及分析

将保存的模型加载进新的python程序,使用测试集中的数据对模型性能进行测试。模型测试的完整代码如下。

```
import tensorflow as tf
# 加载模型
model = tf.keras.models.load_model('../model/dogs_and_cats.h5')
# 设置数据集路径
test_dir = '../data/new/test'
# 加载数据
test_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1. / 255)
test_generator = test_datagen.flow_from_directory(test_dir,
                                                 target_size=(150, 150),
                                                 batch_size=128,
                                                 class_mode='binary')
# 评估模型准确度
results = model.evaluate(test_generator, verbose=1)
# 输出结果
print(f'Test Loss: {results[0]}')
print(f'Test Accuracy: {results[1]}')
```

程序运行结果如下。我们可以看到模型在测试集上的准确度为92.6%, loss值为0.18。

从给出的测试结果来看,模型在测试集上具有较好的表现:

- 损失 (Loss): 损失值为 0.1821, 这是一个相对较低的数值。损失函数通常用于衡量模型预测与实际标签之间的差异,较低的损失值通常表示模型在测试数据上的预测较为准确。
- 准确度 (Accuracy): 准确度为 92.62%。这表示,大约 92.62% 的测试样本被模型正确分类。 在许多情况下,这被视为一个很好的结果。

结论:

- 模型的准确度很高,表明它在分类任务上表现良好,大多数测试样本都被正确分类。
- 损失值较低,这进一步证实了模型的预测相当接近实际标签。

问题及解决方法

- 1. 训练过程中loss无法下降:
 - 使用 binary_crossentropy 作为损失函数,而不是 categorical_crossentropy;
 - 增大学习率,使用0.001 (1e-3)代替0.0001 (1e-4)
- 2. 训练时间长:
 - 使用Adam代替SGD;
 - 增大batch大小;
 - 设置早停策略;
- 3. 验证集准确度低,模型过拟合:
 - 使用数据增强;