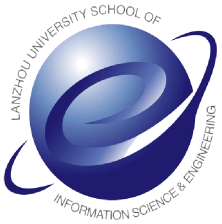

基于卷积神经网络的遥感图像分类

姓名：张恒顺；学号：320190940751；班级：2019 级数据科学 4 班；课程：机器学习

1 引言

卫星遥感技术是一项对地观测综合性技术，长期以来，应用于水利水文、地质、土壤、环境、农林业等领域。近年来，随着遥感运载工具与摄影测量技术的演进，空间技术与信息技术的发展，人类通过遥感技术获取地理空间信息的能力也得到了很大的提高。对遥感卫星图像进行自动化处理将大力助推社会发展。

本次工作，我将以遥感卫星数据集做引，回顾、练习、总结本学期课程中所学的深度学习知识方法。

2 方法与实现

本次工作通过卷积神经网络（CNN）完成。最初，我使用在课堂上学习的 CNN 模型进行测试，随后在此基础上运用所学知识对网络结构、评测方法等进行逐步修改优化，得到最终的模型。

2.1 数据及预处理

本次工作中使用的数据是来自 Kaggle 的公开数据集 Satellite Image Classification (https://www.kaggle.com/datasets/mahmoudreda55/satellite-image-classification)。此数据集由来自遥感系统和谷歌地图的截图构成，共有 5631 份数据，均为 JPG 图片格式。数据集由 4 个类别组成，分别表示不同的遥感图像。

表 1: 数据集样本分布

cloudy	**desert**	**green_area**	**water**
1500	1131	1500	1500

所有的数据按照自己的标签（类别）被存放在相应的文件夹。因此，我通过在读取数据文件的同时读取其所在的文件夹名称，实现对数据标签的同步标注。

为了便于对图片数据进行特征提取，我使用 open CV 库对图片进行加工，将其转化为三维数组。其中，前两维度表示图片的像素点，第 3 个维度表示该像素点的色彩。数据集中图片尺寸有 64×64 和 256×256 两种。为了统一数据维度，并简化计算，我将所有的图片转化为 64×64。这样，所有的数据都被预处理为 $64 \times 64 \times 3$ 的维度。

对图片的分类任务要求我们将数据分为用于训练和测试的两部分。我使用 SciKit-Learn 的 train_test_split 方法将数据集划分为 70% 的训练集和 30% 的测试集两部分。最后，为了方便 TensorFlow 进行输出，标签 y 被转化为 one-hot 编码。我使用 tensorflow.keras.utils.to_categorical 实现这一目的。最终，我们得到了 $5361 \times 64 \times 64 \times 3$ 的数据 X 和 5361×4 的标签 y。

2.2 模型概览

网络由卷积层、非线性映射层、池化层、卷积层、非线性映射层、池化层依次拼接，随后对接 2 层全连接层和输出层。

2.2.1 卷积层

卷积层是 CNN 用于卷积运算的层，由卷积单元构成，负责提取特征。在我的模型中，我采用 3×3 的卷积核，使用”same” 的 padding 方法，保证卷积运算后大小

不变。步长采用默认值 1. 对于输入卷积层，输入尺寸设置为数据的大小 $64 \times 64 \times 3$，filters 数量设置为 64. 对于第 2 个卷积层，filters 设置为 128，其他设置相同。

卷积层通过 TensorFlow 中 Keras 的 layers 实现。

```
model.add(Conv2D(filters=64, kernel_size=(3, 3), padding='same', input_shape=(64, 64, 3)))
model.add(Conv2D(filters=128, kernel_size=(3, 3), padding='same'))
```

2.2.2 池化层

池化层一般是为了提取一定区域的主要特征，并减少参数数量，防止模型过拟合，起到降维作用。本模型中的池化层均采用 2×2 的窗口，使用 2 作为步长。池化方法采用最大池化（Max Pooling），即采用池化窗口内最大值作为池化后该位置的值。两个池化层使用相同的设置。

池化层通过 TensorFlow 中 Keras 的 layers 实现。

```
model.add(MaxPool2D(pool_size=(2, 2), strides=(2, 2)))
```

2.2.3 全连接层

全连接层，是每一个结点都与上一层的所有结点相连，用来把前边提取到的特征综合起来。它所起到的作用是加深网络，增强训练效果。

本模型采用 2 个全连接层，其中一个作为输出层。第一个全连接层在数据被“压”为一维后进行全连接，采用 512 个单元。第二个全连接层作为输出层，输出维度为 4，与 y 的维度相对应。激活层使用 softmax 函数作为激活函数。

```
model.add(Dense(units=512))
model.add(Dense(units=4, activation='softmax'))
```

2.2.4 模型训练

模型采用 Adam 优化器，使用 10^{-4} 学习率，使用交叉熵损失函数，用 Accuracy 作为测量标准。

```
model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
```

此模型采用了 2 种回溯方法——model checkpoint 和 early stopping。设置 checkpoint 主要是为了将训练结果最好的模型参数保存起来，而 early stopping 可以在训练效果不再提高时“及时止损”，有效防止过拟合，将参数回设为效果最好的时候。

由于设置了 early stopping，所以本模型将迭代次数设置得较大，为 1000. 批大小设为 128，使用测试集作为验证集（尽管并不规范），同时通过 model checkpoint 和 early stopping 进行回溯。

```
checkpoint = ModelCheckpoint("model.hdf5", monitor='val_accuracy', verbose=1, save_best_only=True,
save_weights_only=True, mode='auto', period=1)
early = EarlyStopping(monitor='val_accuracy', min_delta=0, patience=20, verbose=1, mode='auto',
restore_best_weights=True)
callbacks_list = [checkpoint, early]
hist = model.fit(X_train, y_train, batch_size=128, epochs=1000, verbose=1, validation_data=(X_test,
y_test), callbacks=callbacks_list)
```

2.2.5 模型测试和评估

```
    y_true = y_test.argmax(-1)
y_pred = model.predict(X_test).argmax(-1)
```

同时打印 precision、recall、accuracy 三种评价指标。

```
print("Prec: "+ str(precision_score(y_true, y_pred, average='weighted')))
print("Recall: "+ str(recall_score(y_true, y_pred, average='weighted')))
print("Accuracy: "+ str(accuracy_score(y_true, y_pred)))
```

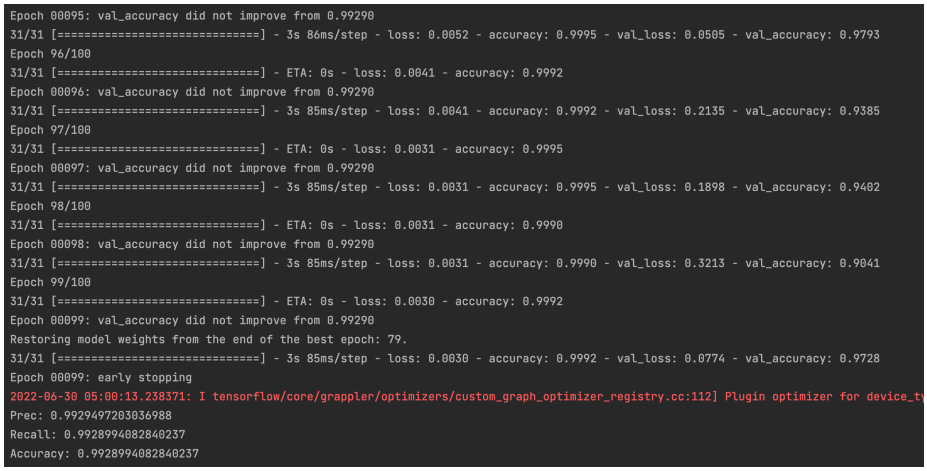

图 1: 最终结果

3 实验

3.1 评价方法

本工作中对于结果的评价同时计算 precision、recall、accuracy 三种常用的评价指标，以 accuracy 作为我们主要的评价指标。

$$precision = \frac{tp}{tp + fp}$$

$$recall = \frac{tp}{tp + fn}$$

$$accuracy = \frac{tp + tn}{tp + tn + fp + fn}$$

在神经网络的训练之中，损失函数也是及其重要的。我们使用交叉熵损失函数，以此为依据进行自动微分参数调整。

$$Loss = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

3.2 模型优化

本次工作中的最终模型是由课堂中所学基本 CNN 模型一步步优化而来的。在本部分中，我将记录我的模型的“进化史”。大体分为三步。

最初，我完成数据预处理后直接将数据集载入初始模型中，但效果并不理想。

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 64, 64, 64)        1792

max_pooling2d (MaxPooling2D  (None, 32, 32, 64)        0
)

conv2d_1 (Conv2D)            (None, 32, 32, 128)       73856

flatten (Flatten)            (None, 131072)            0

dense (Dense)                (None, 128)               16777344

dense_1 (Dense)              (None, 4)                 516

=================================================================
```

图 2: 模型 1

第一步，我首先尝试添加验证集。

```
model.fit(X_train, y_train, batch_size=128, epochs=30, verbose=1, validation_data=(X_test, y_test))
```

效果得到了较大长进。

第二步，我为 compile 过程添加了 callbacks 参数，即 model checkpoint 和 early stopping。

```
checkpoint = ModelCheckpoint("model.hdf5", monitor='val_accuracy', verbose=1, save_best_only=True,
                             save_weights_only=True, mode='auto', period=1)
early = EarlyStopping(monitor='val_accuracy', min_delta=0, patience=20, verbose=1, mode='auto',
                      restore_best_weights=True)
callbacks_list = [checkpoint, early]

model.fit(X_train, y_train, batch_size=128, epochs=1000, verbose=1, validation_data=(X_test, y_test),
          callbacks=callbacks_list)
```

图 3: callbacks

这时，我发现每次运行的结果相差都很大，甚至会有时出现梯度消失的情况。我通过查阅资料逐步添加了 Batch Normalization 层，使用学习课程资料添加了 LeakyReLU，降低了学习率。

三次实验的效果记录如表 2.

表 2: 数据集样本分布

Metrics	实验 1	实验 2	实验 3
Precesion	0.8855	0.9307	0.9929
Recall	0.8621	0.9296	0.9929
Accuracy	0.8621	0.9296	0.9929

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 64, 64, 64)        1792

batch_normalization_8 (Batc  (None, 64, 64, 64)        256
hNormalization)

leaky_re_lu_6 (LeakyReLU)    (None, 64, 64, 64)        0

max_pooling2d_4 (MaxPooling  (None, 32, 32, 64)        0
2D)

conv2d_5 (Conv2D)            (None, 32, 32, 128)       73856

batch_normalization_9 (Batc  (None, 32, 32, 128)       512
hNormalization)

leaky_re_lu_7 (LeakyReLU)    (None, 32, 32, 128)       0

batch_normalization_10 (Bat  (None, 32, 32, 128)       512
chNormalization)

max_pooling2d_5 (MaxPooling  (None, 16, 16, 128)       0
2D)

flatten_2 (Flatten)          (None, 32768)             0

dense_4 (Dense)              (None, 512)               16777728

batch_normalization_11 (Bat  (None, 512)               2048
chNormalization)

leaky_re_lu_8 (LeakyReLU)    (None, 512)               0

dense_5 (Dense)              (None, 4)                 2052

=================================================================
```

图 4: 最终模型

3.3 对比实验

为验证效果，我将本模型与经典机器学习模型支持向量机（SVM）和普通神经网络（NN）做了一个简单的对比实验。先介绍一下对比模型。

3.3.1 对比模型

支持向量机的实现借助了 SciKit-Learn 包。由于我们的数据有 4 个维度（4 个类），而 SciKit-Learn 的 SVC 分类器通常要求维度 ≤ 2. 所以我对数据做了一下变形。

```
train_n_samples, dim1, dim2, dim3 = X_train.shape
```

```
test_n_samples, _, _, _ = X_test.shape

X_train = X_train.reshape((train_n_samples, dim1 * dim2 * dim3))

X_test = X_test.reshape((test_n_samples, dim1 * dim2 * dim3))
```

我分别使用径向基、线性、多项式三种核函数进行测试。他们最后得到的 Accuracy 分别为 0.9083、0.7686、0.8651.

```
The score of rbf is : 0.908284
The score of linear is : 0.768639
The score of poly is : 0.865089
```

图 5: SVM 结果

我用 TensorFlow 实现了一个简单的深度神经网络，由 3 个全连接层和组成。

```
model = Sequential()

model.add(Dense(100, activation='sigmoid', input_shape=(64, 64, 3)))

model.add(Flatten())

model.add(Dense(200, activation='relu'))

model.add(Dense(units=4, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam')

model.fit(X_train, y_train, epochs=10)
```

模型的测试结果为 0.8917.

```
Epoch 9/10
124/124 [==============================] - 26s 213ms/step - loss: 0.6110
Epoch 10/10
124/124 [==============================] - 26s 213ms/step - loss: 1.1429
2022-07-01 02:20:01.999185: I tensorflow/core/grappler/optimizers/custom_gra
Prec: 0.9040588035580079
Recall: 0.8917159763313609
Accuracy: 0.8917159763313609
```

图 6: NN 结果

4 总结

本次工作使用了神经网络对遥感卫星图像进行分类，并在细节处进行优化，如回调（callbacks）等。结果表明，以卷积神经网络为代表的深度神经网络经过针对特定数据集的结构优化可以达到很高的准确率，对社会生产生活产生巨大的贡献，有极高的研究和应用价值。

在本课程中，我们学习了深度学习的基本思想、发展历程、基本概念、重要方法、代表性模型，动手实践了编程基础、神经网络任务等。通过本课程的学习，我加深了对于深度学习知识的系统性理解，在遇到实际问题的时候不再像无头苍蝇到处乱调参，而是根据其背后的数学原理有目的地针对性地进行调整。

今后，我将继续深耕于深度学习和人工智能领域，继续深入研究深度学习知识，争取为该领域的发展做出自己的贡献。