위성 이미지 분류 보고서

2022315965 문준원

---

1. 데이터 개요

위성 이미지 분류를 위한 EuroSAT 데이터셋을 활용한 프로젝트를 수행했습니다. 이 데이터셋은 다양한 지리적 특성을 포함하는 고해상도 위성 이미지로 구성되어 있으며, 총 27,000장의 이미지를 통해 10가지 서로 다른 지리 카테고리를 구분하고자 했습니다.

- 훈련용 데이터는 전체 이미지의 70%를 차지하여 모델의 학습에 중점을 둠
- 검증 세트는 20%로 구성되어 모델의 성능을 중간에 평가하고 최적화하는 데 활용
- 최종 테스트를 위한 데이터셋은 10%로 제한되어 모델의 일반화 성능을 객관적으로 평가

기술적 전처리 과정에서 모든 이미지는 64x64 픽셀 크기로 통일되었으며, 각 이미지는 RGB 컬러 채널 정보를 보존하여 풍부한 시각적 특징을 유지했습니다. 이를 통해 다양한 지리적 환경을 정확하게 식별하고 분류할 수도록 하였습니다.

---

2. 이미지 증강기법

이미지 증강을 통해 모델의 일반화 성능을 높이기 위해 여러 기법을 적용했습니다. 그러나 회색조 변환과 좌우 반전 등의 기법은 성능 저하를 일으킨 것을 실험적으로 확인하여 제외하였습니다.

1. 회전**:** 이미지를 90도 회전시켜 다양한 시점의 데이터를 학습

image = tf.image.rot90(image)

2. 밝기 조정**:** `max_delta=0.1`로 이미지를 랜덤하게 밝기를 조정하여 조명 변화에 대응.

image = tf.image.random_brightness(image, max_delta=0.1)

3. 랜덤 크롭**:** 이미지를 랜덤하게 자르기 위해 `random_crop`을 사용, 크롭된 영역을 원본 크기(64x64)로 재조정하여 공간적 다양성을 추가.

4. 정규화**:** 픽셀 값을 [0, 1]로 스케일링하여 학습 안정화. 이 모든 증강은 `AUTOTUNE` 설정을 사용하여 효율적으로 로드 및 전처리.

---

3. 모델

모델 구조는 4개의 컨볼루션 블록(conv_block)과 **Fully Connected** 레이어로 구성됩니다. 각 컨볼루션 블록은 Conv2D → BatchNormalization → ReLU → MaxPooling2D 순으로 구성되어 있으며, 각 블록의 필터 수는 64 → 128 → 256 → 512로 점차적으로 증가합니다.

이러한 구조는 모델이 점점 더 복잡한 특징을 학습할 수 있도록 도왔습니다.

최종적으로 **GlobalAveragePooling2D**를 사용하여 피처맵을 평균화하고, 이후 **Fully Connected** 레이어와 연결하여 최종적으로 10개의 클래스를 분류합니다. 출력 레이어에서는 **Softmax** 활성화 함수를 사용하여 각 클래스에 대한 확률 분포를 반환합니다.

모델 구조 변경 이유

모델을 **conv_block** 함수를 사용하여 모듈화한 이유는 코드의 재사용성과 가독성을 높이고, 모델의 확장성을 증가시키기 위해서였습니다. 이전 모델은 각 Conv2D와 MaxPooling2D가 고정된 방식으로 연결되어 있었고, 코드의 중복이 많았습니다. conv_block 함수는 컨볼루션 레이어와 배치 정규화(Batch Normalization), **ReLU** 활성화 함수, 맥스 풀링을 포함하여 동일한 작업을 반복할 수 있도록 만들었습니다.

변경된 모델 구조:

```python
# Conv Block 정의
def conv_block(filters, kernel_size=3, stride=1, padding="same"):
    return tf.keras.Sequential([
        layers.Conv2D(filters=filters, kernel_size=kernel_size, strides=stride, padding=padding),
        layers.BatchNormalization(),  # Batch Normalization
        layers.ReLU(),
        layers.MaxPooling2D(pool_size=2, strides=1)  # MaxPooling
    ])
```

```python
def build_model():
    model = tf.keras.Sequential([
        layers.Input(shape=(64, 64, 3)),  # 입력 크기 지정
        conv_block(64, 3),
        conv_block(128, 3),
        conv_block(256, 3),
        conv_block(512, 3),
        layers.GlobalAveragePooling2D(),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.3),
        layers.Dense(64, activation='relu'),
        layers.Dropout(0.3),
        layers.Dense(10, activation='softmax'),
    ])
    return model

# 모델 생성
model = build_model()
model.summary()

```

Model: "sequential_12"

Layer (type)	Output Shape	Param #
sequential_8 (Sequential)	(None, 63, 63, 64)	2,048
sequential_9 (Sequential)	(None, 62, 62, 128)	74,368
sequential_10 (Sequential)	(None, 61, 61, 256)	296,192
sequential_11 (Sequential)	(None, 60, 60, 512)	1,182,208
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 512)	0
dense_9 (Dense)	(None, 128)	65,664
dropout_3 (Dropout)	(None, 128)	0
dense_10 (Dense)	(None, 64)	8,256
dropout_4 (Dropout)	(None, 64)	0
dense_11 (Dense)	(None, 10)	650

Total params: 1,629,386 (6.22 MB)
Trainable params: 1,627,466 (6.21 MB)
Non-trainable params: 1,920 (7.50 KB)

모델 구조 설명

1. **Sequential Layers (**컨볼루션 블록**)**
    - **sequential_8 (**출력 크기**: (None, 63, 63, 64))**: 첫 번째 컨볼루션 블록으로, 필터 64개를 사용하며 Conv2D, BatchNormalization, ReLU, MaxPooling2D로 구성됩니다.
    - **sequential_9 (**출력 크기**: (None, 62, 62, 128))**: 두 번째 컨볼루션 블록으로, 필터 128개를 사용합니다.
    - **sequential_10 (**출력 크기**: (None, 61, 61, 256))**: 세 번째 컨볼루션 블록으로, 필터 256개를 사용합니다.
    - **sequential_11 (**출력 크기**: (None, 60, 60, 512))**: 네 번째 컨볼루션 블록으로, 필터 512개를 사용합니다.
2. **Global Average Pooling**
    - **global_average_pooling2d_1 (**출력 크기**: (None, 512))**: 피처맵의 평균값을 구하여 차원을 줄이고, 후속의 Dense 레이어와 연결됩니다. 이는 모델이 더 효율적으로 학습할 수 있도록 돕습니다.
3. **Fully Connected Layers**
    - **dense_9 (**출력 크기**: (None, 128))**: 첫 번째 Fully Connected 레이어로, 128개의 유닛을 사용하며 ReLU 활성화 함수가 적용됩니다.
    - **dropout_3 (**드롭아웃 레이어**)**: 드롭아웃 비율 0.3을 사용하여 과적합을 방지합니다.

- **dense_10 (**출력 크기**: (None, 64))**: 두 번째 Fully Connected 레이어로, 64개의 유닛을 사용합니다.
- **dropout_4 (**드롭아웃 레이어**)**: 두 번째 드롭아웃 레이어로, 과적합 방지를 위해 사용됩니다.
- **dense_11 (**출력 크기**: (None, 10))**: 출력 레이어로, 10개의 클래스를 분류하기 위해 **Softmax** 활성화 함수가 사용됩니다.

모델 파라미터 분석

- 전체 파라미터 수: 1,629,386 (약 6.22 MB)
  - **Trainable params**: 1,627,466 (학습 가능한 파라미터, 6.21 MB)
  - **Non-trainable params**: 1,920 (학습되지 않는 파라미터, 7.5 KB)

---

4. 최적화**,** 손실 함수

- 최적화 알고리즘: Adam Optimizer를 사용하여 가중치 업데이트.
- 손실 함수: Sparse Categorical Crossentropy를 사용하여 다중 클래스 분류를 최적화.
- 평가지표: 정확도(Accuracy)를 사용하여 성능 평가.

---

5. 스케줄러와 조기 종료

훈련을 안정화하고 과적합을 방지하기 위해 다음과 같은 콜백을 적용했습니다:

1. **ReduceLROnPlateau**: 3번의 에포크 동안 val_loss가 개선되지 않으면 학습률을 줄입니다
2. **EarlyStopping**: 검증 손실(val_loss)이 5번 연속 개선되지 않으면 훈련 종료.
3. **ModelCheckpoint**: 가장 낮은 val_loss를 기록한 모델을 .keras 파일로 저장.

---

6. 학습 파라미터들

- 배치 크기: 32,64 실험
- 에포크 수: 10,30 실험
- 입력 이미지 크기: 64x64x3 (RGB)
- 학습률 초기값: Adam Optimizer의 기본값 (0.001)

---

7. 결과 분석

훈련 및 테스트 과정에서의 성능 향상을 관찰할 수 있었습니다. 아래는 각 단계별로 성능 변화의 분석입니다.

7.1 기본 코드**(**교수님이 제시해주신 코드**)**

처음 모델을 실행했을 때, 기본 코드로 테스트셋에서 다음과 같은 결과를 얻었습니다:

- 테스트셋 정확도: 0.7589
- 테스트셋 손실: 1.5564

7.2 콜백 추가 **(ReduceLROnPlateau, EarlyStopping, ModelCheckpoint)**

훈련 안정화와 과적합 방지를 위해 **ReduceLROnPlateau**, **EarlyStopping**, 그리고 **ModelCheckpoint**를 추가하여 모델을 개선했습니다. 그 결과, 테스트셋 정확도와 손실은 다음과 같이 향상되었습니다:

- 최종 테스트 정확도: 0.7941
- 최종 테스트 손실: 1.4272

이 추가적인 콜백 설정은 학습률 감소와 조기 종료 기능을 통해 모델의 학습을 최적화하고, 과적합을 방지하여 성능을 더욱 향상시킨 것으로 분석됩니다.

7.3 이미지 증강 기법 추가 **(**밝기 조정**)**

이미지 증강 기법으로 밝기 조정을 추가했더니, 테스트셋 정확도는 다음과 같이 추가로 상승했습니다:

- 최종 테스트 정확도: 0.8081

- 최종 테스트 손실: 0.6638

밝기 조정은 조명 변화에 대한 모델의 강건성을 높여줬으며, 데이터 다양성을 증가시켜 더 나은 성능을 도출할 수 있었습니다.

7.4 모델 구조의 변경

모델 구조 변경의 핵심은 컨볼루션 블록의 모듈화와 블록마다 증가하는 필터 수를 사용한 것입니다. 이를 통해 모델의 성능을 향상시키고, 과적합을 방지할 수 있었습니다.

```
Epoch 23/30
591/591 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step - accuracy: 0.9319 - loss: 0.2159
Epoch 23: val_loss did not improve from 0.26648
591/591 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61s 103ms/step - accuracy: 0.9319 - loss: 0.2159 - val_accuracy: 0.8670 - val_loss: 0.4608 - learning_rate: 1.2500e-04
Epoch 24/30
591/591 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step - accuracy: 0.9313 - loss: 0.2201
Epoch 24: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.

Epoch 24: val_loss did not improve from 0.26648
591/591 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61s 103ms/step - accuracy: 0.9313 - loss: 0.2201 - val_accuracy: 0.8804 - val_loss: 0.3847 - learning_rate: 1.2500e-04
Epoch 25/30
591/591 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step - accuracy: 0.9332 - loss: 0.2078
Epoch 25: val_loss improved from 0.26648 to 0.26372, saving model to best_model.keras
591/591 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61s 103ms/step - accuracy: 0.9332 - loss: 0.2078 - val_accuracy: 0.9113 - val_loss: 0.2637 - learning_rate: 6.2500e-05
Epoch 26/30
591/591 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step - accuracy: 0.9351 - loss: 0.1987
Epoch 26: val_loss improved from 0.26372 to 0.26141, saving model to best_model.keras
591/591 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61s 103ms/step - accuracy: 0.9351 - loss: 0.1987 - val_accuracy: 0.9139 - val_loss: 0.2614 - learning_rate: 6.2500e-05
Epoch 27/30
591/591 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 95ms/step - accuracy: 0.9357 - loss: 0.2008
Epoch 27: val_loss did not improve from 0.26141
591/591 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61s 103ms/step - accuracy: 0.9357 - loss: 0.2008 - val_accuracy: 0.8993 - val_loss: 0.2978 - learning_rate: 6.2500e-05
Epoch 28/30
591/591 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step - accuracy: 0.9393 - loss: 0.1884
Epoch 28: val_loss improved from 0.26141 to 0.21627, saving model to best_model.keras
591/591 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61s 103ms/step - accuracy: 0.9393 - loss: 0.1884 - val_accuracy: 0.9274 - val_loss: 0.2163 - learning_rate: 6.2500e-05
Epoch 29/30
591/591 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 95ms/step - accuracy: 0.9382 - loss: 0.1943
Epoch 29: val_loss did not improve from 0.21627
591/591 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61s 103ms/step - accuracy: 0.9382 - loss: 0.1943 - val_accuracy: 0.8713 - val_loss: 0.4231 - learning_rate: 6.2500e-05
Epoch 30/30
591/591 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step - accuracy: 0.9392 - loss: 0.1915
Epoch 30: val_loss did not improve from 0.21627
591/591 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61s 103ms/step - accuracy: 0.9392 - loss: 0.1915 - val_accuracy: 0.9165 - val_loss: 0.2519 - learning_rate: 6.2500e-05
Restoring model weights from the end of the best epoch: 28.
```

```
1 loss, accuracy = model.evaluate(test)
2
3
```

```
85/85 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - accuracy: 0.9312 - loss: 0.2294
```

- 최종 테스트 정확도: 0.9312
- 최종 테스트 손실: 0.2294

최종적으로, 모델은 약 168%의 정확도 향상과 함께 손실을 대폭 줄였습니다. 이는 구조 변경, 이미지 증강 기법 추가, 그리고 콜백 설정을 통해 모델 성능을 효율적으로 개선했기 때문입니다.

---

8. 느낀점 및 참고사항

본 프로젝트를 통해 다음을 배웠습니다:

1. 데이터 증강과 정규화가 모델의 일반화 성능을 향상시키는 데 효과적임을 확인.

사전 학습된 모델을 사용하면 더 높은 정확도를 얻을 수 있다는 점을 알게 되었습니다. 특히, 논문에서 **ResNet** 모델을 사용하면 정확도가 98%까지 상승한다는 것을 확인했지만, 기본 모델 구조 변경에 더욱 신경을 써서 정확도 상승을 목표로 실험을 진행했습니다. 여러 실험을 통해 가능한 한 다양한 실험적인 결과를 확인하려고 노력했습니다.

---

9. 참고자료 및 문헌

1. TensorFlow 공식 문서: https://www.tensorflow.org/
2. EuroSAT 데이터셋: https://knowyourdata-tfds.withgoogle.com/
3. Keras API 문서: https://keras.io/api/
4. Deep Transfer Learning for Land Use and Land Cover Classification: A Comparative Study, December 2021, Sensors 21(23):8083, DOI: 10.3390/s21238083, License: CC BY 4.0

5. IMPROVING LULC CLASSIFICATION FROM SATELLITE IMAGERY USING DEEP LEARNING - EUROSAT DATASET
6. EuroSAT: A Novel Dataset and Deep Learning Benchmark for Land Use and Land Cover Classification
7. ChatGPT (OpenAI). Available at: https://openai.com/chatgpt