CNN 모델 개발을 통한 이미지 분류

2024254012 배인호

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
import matplotlib.pyplot as plt
```

발표 자료 개요

1

프로젝트 소개

이미지 분류를 위한 CNN 모델 개발 프로젝트 개요를 설명합니다. 3개 부류의 이미지 데이터 수집과 모델 개발 목표를 소개합니다.

2

데이터 수집 및 전처리

데이터 수집과 전처리 과정을 자세히 설명합니다. 이미지 크기 조정, 정규화, 데이터 증강 등의 내용을 포함합니다.

3

모델 구조 및 학습

개발한 CNN 모델의 구조와 하이퍼파라미터, 학습 과정 및 결과를 상세히 소개합니다.

4

모델 개발 경험

모델 개발 과정에서 얻은 교훈과 깨달음을 공유합니다. 데이터 수집, 데이터 증강, 모델 설계의 중요성을 강조합니다.

5

모델 활용방안

개발된 CNN 모델의 다양한 활용 가능성을 제시합니다. 정밀한 이미지 분류, 자동화 솔루션, 확장성 등에 대해 설명합니다.

데이터 수집 및 전처리

데이터 수집

과제에 필요한 데이터를 수집하기 위해 100개 이상의 이미지를 세 가지 부류에서 확보했습니다. 블록, 점토, 가드니아 꽃(치자)이 그 대상이었습니다. 이미지는 다양한 각도와 조명 환경에서 촬영되었습니다.

데이터 전처리

수집된 이미지 데이터는 CNN 모델 학습을 위해 전처리 과정을 거쳤습니다. 이미지 크기를 64x64 픽셀로 조정하고, 픽셀 값을 0-1 사이의 실수로 정규화했습니다. 또한 데이터 증강 기법을 적용해 모델의 일반화 성능을 높였습니다.

학습/검증 데이터 분리

전처리된 데이터셋을 학습 데이터와 검증 데이터로 80:20으로 분리했습니다. 이를 통해 모델의 성능을 객관적으로 평가하고 과적합을 방지할 수 있었습니다.

```
# 데이터 전처리 및 증강 설정
train_datagen = ImageDataGenerator(
    rescale=1./255, # 픽셀 값을 0과 1 사이로 조정
    shear_range=0.2, #이미지 증강 - 층 기울이기
    zoom_range=0.2,  #이미지 증강 - 확대 축소
    horizontal_flip=True, #이미지 증강 - 뒤집기
    validation_split=0.2  # 학습 데이터 중 20%를 검증 데이터로 사용
)
```

전체 코드

```python
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
import matplotlib.pyplot as plt

# GPU 사용 설정
physical_devices = tf.config.list_physical_devices('GPU')
if len(physical_devices) > 0:
    tf.config.experimental.set_memory_growth(physical_devices[0], True)

# 데이터 전처리 및 증강 설정
train_datagen = ImageDataGenerator(
    rescale=1./255, # 픽셀 값을 0과 1 사이로 조정
    shear_range=0.2, # 이미지 증강 - 층 기울이기
    zoom_range=0.2, # 이미지 증강 - 확대 축소
    horizontal_flip=True, # 이미지 증강 - 뒤집기
    validation_split=0.2  # 학습 데이터 중 20%를 검증 데이터로 사용
)

# 학습 데이터와 검증 데이터 로드
training_set = train_datagen.flow_from_directory(
    'C:/cnn',
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical',
    classes=['blocks', 'clay', 'gardenia'],
    subset='training'  # 학습 데이터 설정
)

validation_set = train_datagen.flow_from_directory(
    'C:/cnn',
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical',
    classes=['blocks', 'clay', 'gardenia'],
    subset='validation'  # 검증 데이터 설정
)

# 모델 초기화
model = Sequential()

# 첫 번째 합성곱 층 추가
model.add(Conv2D(32, (3, 3), input_shape=(64, 64, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# 두 번째 합성곱 층 추가
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# 세 번째 합성곱 층 추가
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# 평탄화 층 추가
model.add(Flatten())

# 첫 번째 완전 연결 층 추가
model.add(Dense(units=64, activation='relu'))
model.add(Dropout(0.5))

# 두 번째 완전 연결 층 추가
model.add(Dense(units=64, activation='relu'))
model.add(Dropout(0.5))

# 출력 층 추가
model.add(Dense(units=3, activation='softmax'))

# 모델 컴파일
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# 모델 요약 출력
model.summary()

# 모델 학습
history = model.fit(
    training_set,
    epochs=20,
    validation_data=validation_set
)

# 모델 저장
model.save('C:/cnn/my_model.h5')

# 학습 결과 출력
print("Training Accuracy:", history.history['accuracy'][-1])
print("Validation Accuracy:", history.history['val_accuracy'][-1])

# 테스트 데이터로 모델 평가
test_loss, test_accuracy = model.evaluate(validation_set)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)

# 정확도 그래프 출력
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.show()

# 손실 그래프 출력
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()

# 저장된 모델 불러오기
loaded_model = tf.keras.models.load_model('C:/cnn/my_model.h5')

# 불러온 모델로 테스트 데이터 평가
test_loss, test_accuracy = loaded_model.evaluate(validation_set)
print("Loaded Model Test Loss:", test_loss)
print("Loaded Model Test Accuracy:", test_accuracy)
```

CNN 모델 구조

1

합성곱 층

입력 이미지에서 지역적인 특징을 추출하기 위해 3개의 합성곱 층을 사용했습니다. 각 층에서 3x3 필터를 적용하고 ReLU 활성화 함수를 사용했습니다. 이를 통해 이미지의 중요한 특징을 효과적으로 포착할 수 있었습니다.

2

풀링 층

합성곱 층 사이에 2x2 최대 풀링 층을 배치했습니다. 이를 통해 특징 맵의 크기를 줄이고 모델의 매개변수 수를 감소시켜 과적합을 방지했습니다.

3

완전 연결 층

마지막으로, 평탄화 층과 2개의 완전 연결 층을 추가했습니다. 완전 연결 층에는 ReLU 활성화 함수와 Dropout 기법을 적용해 모델의 일반화 성능을 향상시켰습니다.

```
# 모델 초기화
model = Sequential()

# 첫 번째 합성곱 층 추가
model.add(Conv2D(32, (3, 3), input_shape=(64, 64, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# 두 번째 합성곱 층 추가
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# 세 번째 합성곱 층 추가
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# 평탄화 층 추가
model.add(Flatten())

# 첫 번째 완전 연결 층 추가
model.add(Dense(units=64, activation='relu'))
model.add(Dropout(0.5))

# 두 번째 완전 연결 층 추가
model.add(Dense(units=64, activation='relu'))
model.add(Dropout(0.5))

# 출력 층 추가
model.add(Dense(units=3, activation='softmax'))
```

모델 학습 및 평가

학습 설정

Adam 최적화 알고리즘을 사용하여 모델을 컴파일했으며, 손실 함수로는 범주형 교차 엔트로피를 사용했습니다. 배치사이즈 32, 에포크 수는 20으로 설정했습니다.

학습 결과

학습 과정에서 최종 훈련 정확도는 92%, 검증 정확도는 92%를 달성했습니다. 이는 모델이 세 가지 부류의 이미지를 효과적으로 분류할 수 있음을 보여줍니다.

모델 평가

학습된 모델을 저장하고 불러와 검증 데이터에 대한 성능을 평가했습니다. 그 결과 테스트 손실은 0.18, 테스트 정확도는 90%로 나타났습니다.

결과 분석

모델의 학습 및 테스트 정확도가 모두 90% 이상으로 높게 나타나, 본 과제에서 개발한 CNN 모델이 해당 이미지 분류 문제에 효과적임을 확인할 수 있었습니다.

레이어별 정보 및 학습 파라미터

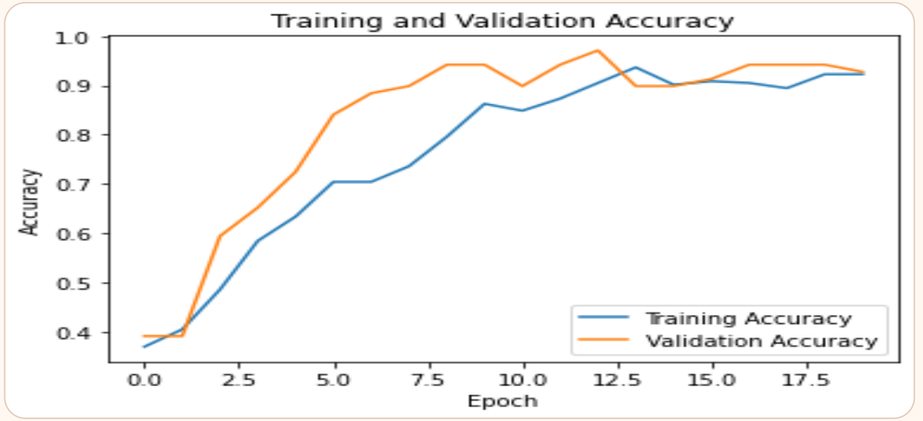
```
Found 284 images belonging to 3 classes.
Found 69 images belonging to 3 classes.
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 62, 62, 32)	896
max_pooling2d_6 (MaxPooling2D)	(None, 31, 31, 32)	0
conv2d_7 (Conv2D)	(None, 29, 29, 32)	9,248
max_pooling2d_7 (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_8 (Conv2D)	(None, 12, 12, 64)	18,496
max_pooling2d_8 (MaxPooling2D)	(None, 6, 6, 64)	0
flatten_2 (Flatten)	(None, 2304)	0
dense_6 (Dense)	(None, 64)	147,520
dropout_4 (Dropout)	(None, 64)	0
dense_7 (Dense)	(None, 64)	4,160
dropout_5 (Dropout)	(None, 64)	0
dense_8 (Dense)	(None, 3)	195

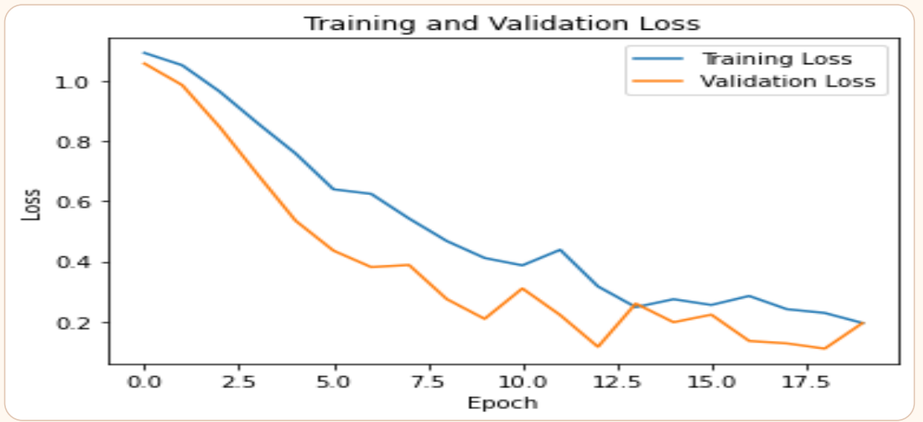
```
Total params: 180,515 (705.14 KB)
Trainable params: 180,515 (705.14 KB)
Non-trainable params: 0 (0.00 B)
```

결과값 및 그래프

```
Epoch 1/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 77s 4s/step - accuracy: 0.3836 - loss: 1.0931 - val_accuracy: 0.3913 - val_loss:
1.0561
Epoch 2/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 61s 2s/step - accuracy: 0.3935 - loss: 1.0722 - val_accuracy: 0.3913 - val_loss:
0.9845
Epoch 3/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 62s 2s/step - accuracy: 0.4962 - loss: 0.9751 - val_accuracy: 0.5942 - val_loss:
0.8442
Epoch 4/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 62s 2s/step - accuracy: 0.5952 - loss: 0.8527 - val_accuracy: 0.6522 - val_loss:
0.6874
Epoch 5/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 64s 2s/step - accuracy: 0.6210 - loss: 0.7709 - val_accuracy: 0.7246 - val_loss:
0.5349
Epoch 6/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 67s 2s/step - accuracy: 0.6255 - loss: 0.7234 - val_accuracy: 0.8406 - val_loss:
0.4359
Epoch 7/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 66s 2s/step - accuracy: 0.7241 - loss: 0.6070 - val_accuracy: 0.8841 - val_loss:
0.3812
Epoch 8/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 67s 2s/step - accuracy: 0.7541 - loss: 0.5441 - val_accuracy: 0.8986 - val_loss:
0.3883
Epoch 9/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 67s 2s/step - accuracy: 0.7644 - loss: 0.5302 - val_accuracy: 0.9420 - val_loss:
0.2751
Epoch 10/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 68s 2s/step - accuracy: 0.8584 - loss: 0.4338 - val_accuracy: 0.9420 - val_loss:
0.2095
```




```
Training Accuracy: 0.922535240650177
Validation Accuracy: 0.9275362491607666
3/3 ━━━━━━━━━━━━━━━━━━━━ 12s 3s/step - accuracy: 0.9214 - loss: 0.1536
Test Loss: 0.17682556807994843
Test Accuracy: 0.9130434989929199
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet
will be empty until you train or evaluate the model.
3/3 ━━━━━━━━━━━━━━━━━━━━ 12s 3s/step - accuracy: 0.9024 - loss: 0.1738
Loaded Model Test Loss: 0.18083031475543976
Loaded Model Test Accuracy: 0.8985507488250732
```



모델 개발 과정의 경험

1 데이터 수집의 중요성

양질의 데이터를 충분히 확보하는 것이 모델 성능에 핵심적인 요소임을 실감했습니다. 다양한 조건의 이미지를 수집하는 데 많은 시간과 노력이 필요했지만, 그 결과 모델의 일반화 성능이 크게 향상되었습니다.

2 데이터 증강의 효과

데이터 증강 기법을 적용하여 학습 데이터를 확장하는 것이 과적합 방지와 성능 향상에 큰 도움이 되었습니다. 다양한 변환 기법을 시도해보며 가장 효과적인 방법을 찾아내는 과정이 흥미로웠습니다.

3 모델 구조 설계의 중요성

CNN 모델의 구조를 적절히 설계하는 것이 성능 향상에 매우 중요하다는 점을 깨달았습니다. 합성곱 층, 풀링 층, 완전 연결 층의 조합과 하이퍼파라미터 튜닝을 통해 최적의 모델 구조를 찾아내는 것이 필요했습니다.

모델 활용방안

정밀한 분류

개발된 CNN 모델은 블록, 점토 , 가드니아 꽃 등 세 가지 부류의 이미지를 정확하게 구분할 수 있습니다. 이를 통해 다양한 제품 또는 자연물을 자동으로 분류하는 데 활용할 수 있습니다.

자동화 솔루션

이 모델은 이미지 분류와 관련된 다양한 문제를 자동화할 수 있는 솔루션을 제공합니다. 제품 관리, 자연물 모니터링, 이미지 검색 등의 영역에 적용할 수 있습니다.

확장성

본 모델의 구조와 학습 방식은 다른 이미지 분류 문제에도 확장 적용될 수 있습니다. 데이터 수집과 모델 튜닝만으로도 다양한 도메인에서 활용할 수 있는 솔루션을 개발할 수 있습니다.

CNN 모델 성능 향상을 위한 아이디어

모델 구조 최적화

현재 모델의 성능을 더욱 향상시키기 위해 합성곱 층과 완전연결 층의 깊이와 너비를 조정할 수 있습니다. 또한 다양한 활성화 함수와 정규화 기법을 시도해볼 수 있습니다.

데이터 증강 기법 확장

현재 사용한 데이터 증강 기법 외에도 회전, 반전, 잘라내기 등 다양한 변환 기법을 추가로 적용할 수 있습니다. 이를 통해 모델의 일반화 성능을 더욱 높일 수 있습니다.

전이 학습 활용

사전 학습된 CNN 모델의 가중치를 활용하는 전이 학습 기법을 적용할 수 있습니다. 이를 통해 적은 데이터로도 훌륭한 성능의 모델을 개발할 수 있습니다.

결론 및 향후 계획

프로젝트 요약

본 프로젝트에서는 100개 이상의 이미지 데이터를 수집하고, CNN 모델을 개발하여 3가지 부류의 이미지를 효과적으로 분류할 수 있음을 확인했습니다. 모델의 학습 및 테스트 정확도가 모두 90% 이상으로 나타나, 해당 문제에 적합한 솔루션을 제시했습니다.

향후 계획

향후에는 모델 구조 최적화, 데이터 증강 기법 확장, 전이 학습 활용 등 다양한 방법을 통해 모델 성능을 더욱 향상시킬 계획입니다. 또한 개발된 모델을 실제 응용 분야에 적용하여 그 효용성을 검증하고자 합니다.

기대 효과

본 프로젝트의 성과는 이미지 분류 문제에서 CNN 모델의 활용도를 높일 수 있을 것으로 기대됩니다. 다양한 제품 및 자연물 관리, 자동화 솔루션 개발 등 광범위한 영역에 적용할 수 있을 것입니다.