**<인공지능 기초와 활용 미드텀 프로젝트 과제 보고서>**

산업공학과 ITM전공 22102009이지원

1. **요약**

본 과제는 Computer Vision분야에서 대표적인 데이터셋인 CIFAR-10 데이터셋을 이용해 직접 딥러닝 모델을 구성하고 수업에서 배운 다양한 기법을 이용해 효율적으로 분류 과업을 수행하는 모델을 만드는 것을 목표로 했습니다.

우선 저는 GPU와 CUDA 라이브러리를 통해 빠르고 효율적으로 학습을 진행할 수 있어 PC 환경에 맞는 GPU 사용 환경을 구축했습니다. 이어서 여러 데이터 전처리 기법과 가장 기본적인 Sequential 모델 파이프라인을 구축해 분류 모델을 학습했습니다. 50 epochs을 우선 돌려본 결과 80% 수준의 정확도에 머물렀습니다. 그래프 개형을 분석해보니 더 많은 epochs를 통해 성능 개선 여지가 보였지만 그 속도가 너무 느렸고, 전체 학습 과정에 소요시간이 크지 않아 더 깊고 복잡한 모델을 사용해 성능 향상 속도와 최고 성능을 더 높이자고 판단했습니다.

따라서 저는 VGG 기반의 더 깊고 복잡한 모델을 구성했으며 50epochs로 동일하게 학습을 진행한 결과 90%수준까지 더 빠르고 높게 성능이 향상되었음을 확인했습니다. 또한 여전히 성능이 우상향하는 모습을 보였기에 300epochs까지 늘려 Overfitting이 발생하는 지점을 파악해 최적의 학습지점을 찾았습니다.

그 결과 약 7~80 epochs지점부터는 training 데이터셋에 대한 정확도는 높아지지만 validation 데이터셋에 대한 성능 향상이 이루어지지 않아 모델이 유의미한 학습을 하지 못하고 있다고 판단했으며 최종적으로 test 데이터셋에 대해서 약 91%의 성능을 달성할 수 있었습니다.

1. **실행 환경**

과제 및 모델 학습에 사용한 PC의 주요 스펙은 다음과 같습니다.

* CPU: AMD Ryzen 9 5950X 16-Core
* GPU: NVIDIA GeForce RTX 3080
* RAM: DDR5 64GB
* OS: Windows 10

위 스펙에 맞춰 사용한 소프트웨어 사양은 다음과 같습니다.

* Python version: 3.10
* CUDA version: 11.2
* CuDnn version: 8.1.1

위의 서술된 환경 하에 GPU를 통해 빠르고 효율적으로 모델 학습을 진행했습니다.

1. **데이터 전처리**

y\_train = to\_categorical(y\_train, 10)  
y\_test = to\_categorical(y\_test, 10)  
  
train\_datagen = ImageDataGenerator(  
 rescale=1./255,  
 rotation\_range=15,  
 width\_shift\_range=0.1,  
 height\_shift\_range=0.1,  
 horizontal\_flip=True,  
 validation\_split=0.1  
)  
train\_generator = train\_datagen.flow(x\_train, y\_train, batch\_size=128, subset='training')  
val\_generator = train\_datagen.flow(x\_train, y\_train, batch\_size=128, subset='validation')  
  
test\_datagen = ImageDataGenerator(rescale=1./255)  
test\_generator = test\_datagen.flow(x\_test, y\_test, batch\_size=128)

과제에 명시된 대로 CIFAT-10 데이터셋을 불러왔으며, validation split 비율은 0.1, rescale비율은 1/255를 사용했습니다. 모델의 overfitting을 막고 robustness를 위해 이미지rotation, flip 등 여러 data augmentation 기법을 통해 모델이 다양한 이미지를 충분히 학습할 수 있도록 유도했습니다.

1. **1차 실험**

model = Sequential([  
 Conv2D(32, (3,3), activation='relu', padding='same', input\_shape=(32,32,3)),  
 BatchNormalization(),  
 Conv2D(32, (3,3), activation='relu', padding='same'),  
 MaxPooling2D(),  
 Dropout(0.25),  
  
 Conv2D(64, (3,3), activation='relu', padding='same'),  
 BatchNormalization(),  
 Conv2D(64, (3,3), activation='relu', padding='same'),  
 MaxPooling2D(),  
 Dropout(0.25),  
  
 Flatten(),  
 Dense(512, activation='relu'),  
 Dropout(0.3),  
 Dense(10, activation='softmax')  
])  
  
model.compile(optimizer=Adam(), loss='categorical\_crossentropy', metrics=['accuracy'])

1차 실험에 사용한 모델은 위와 같이 가장 기본적인 sequential 모델 구조를 참조해 구성했습니다. Conv > BN > Conv > Pooling > Dropout 구조를 2번 우선 적용한 뒤 충분히 특성을 파악한 이미지를 flatten하여 fully-connected layer에 입력으로 사용해 최종 분류를 수행하도록 했습니다. Dropout을 사용해 overfitting을 막고 robustness를 증가시켰으며 batch normalization을 통해 모델의 안정적인 학습을 도왔습니다.

1. **개선점 파악**

그 결과 <그림1>과 같은 그래프를 얻을 수 있었습니다. x축은 epochs 수이며 y축은 각각 Loss와 accuracy입니다. 흥미로운 점은 데이터셋 구성에 다양한 augmentation 기법과 dropout등과 같은 학습 보조 기법을 사용했기에 training시의 성능지표보다 validation시의 지표가 더 높은 경우가 존재한다는 점 입니다. 이를 통해 앞서 언급한 여러 기법들이 효과적으로 작동하고 있다는 것을 간접적을 확인했습니다. 또한 모델의 accuracy 그래프가 지속적으로 우상향 (loss그래프는 우하향) 하고 있기에 epochs수를 늘린다면 더 높은 성능을 달성할 수 있을 것이라 파악했습니다. 하지만 현재로도 한 epoch당 약 10초정도 소요되어 충분히 빠른 편이기에 더 성능이 좋고 무거운 모델을 사용해보자고 판단해 모델을 수정하여 2차 실험을 진행했습니다.

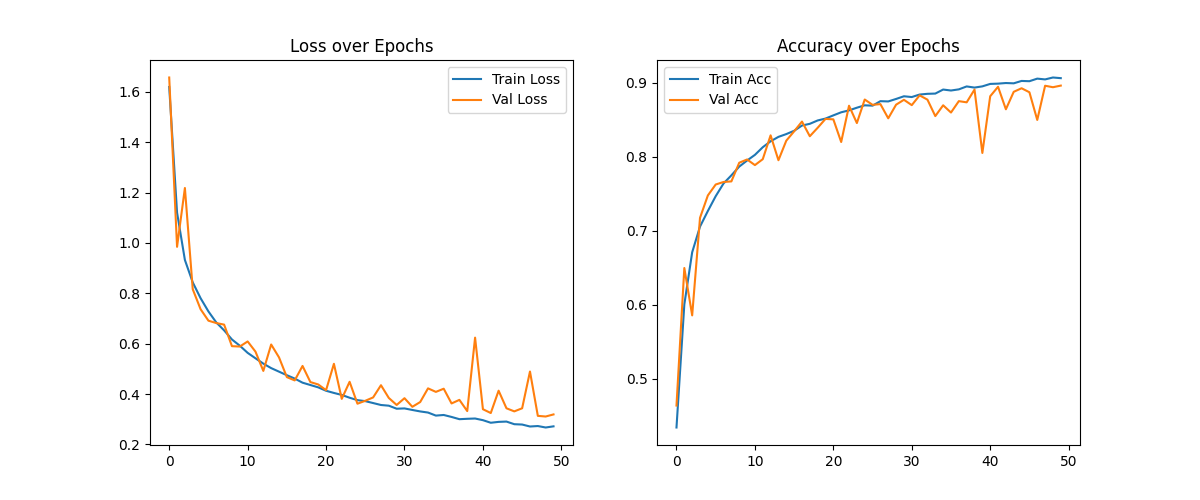
텍스트, 스크린샷, 도표, 그래프이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

<그림1> 1차 실험 결과 그래프

1. **2차 실험**
2. model = Sequential()  
     
   model.add(Conv2D(64, (3,3), padding='same', input\_shape=(32,32,3)))  
   model.add(BatchNormalization())  
   model.add(Activation('relu'))  
   model.add(Conv2D(64, (3,3), padding='same'))  
   model.add(BatchNormalization())  
   model.add(Activation('relu'))  
   model.add(MaxPooling2D(pool\_size=(2,2)))  
   model.add(Dropout(0.25))  
     
   model.add(Conv2D(128, (3,3), padding='same'))  
   model.add(BatchNormalization())  
   model.add(Activation('relu'))  
   model.add(Conv2D(128, (3,3), padding='same'))  
   model.add(BatchNormalization())  
   model.add(Activation('relu'))  
   model.add(MaxPooling2D(pool\_size=(2,2)))  
   model.add(Dropout(0.35))  
     
   model.add(Conv2D(256, (3,3), padding='same'))  
   model.add(BatchNormalization())  
   model.add(Activation('relu'))  
   model.add(Conv2D(256, (3,3), padding='same'))  
   model.add(BatchNormalization())  
   model.add(Activation('relu'))  
   model.add(MaxPooling2D(pool\_size=(2,2)))  
   model.add(Dropout(0.4))  
     
   model.add(Flatten())  
   model.add(Dense(512))  
   model.add(BatchNormalization())  
   model.add(Activation('relu'))  
   model.add(Dropout(0.5))  
     
   model.add(Dense(10, activation='softmax'))  
     
   model.compile(optimizer=Adam(learning\_rate=0.001),  
    loss='categorical\_crossentropy',  
    metrics=['accuracy'])

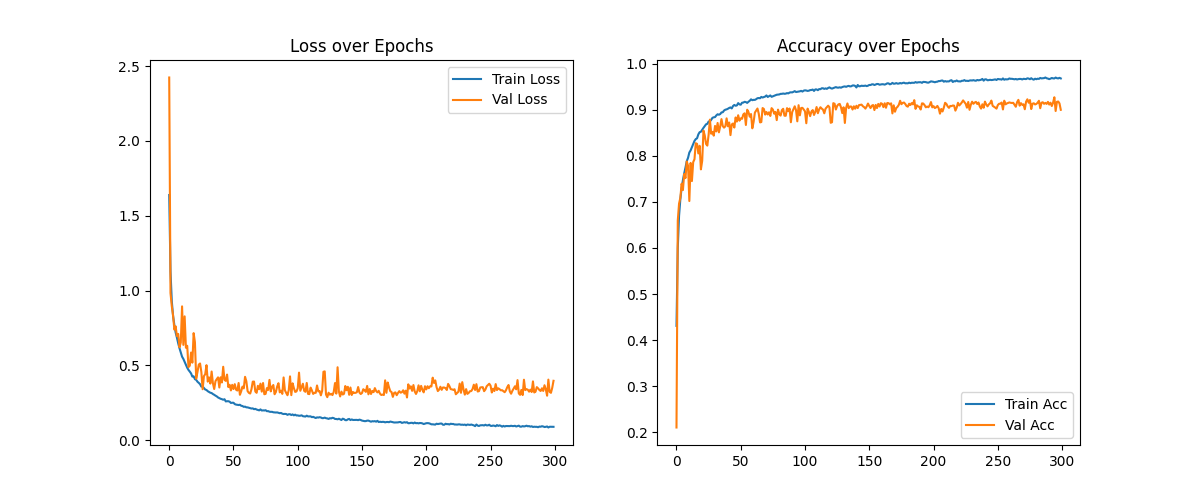
2차실험에서 개선된 모델은 위와 같습니다. 앞선 모델과 동일하게 Sequential 모델 구조이지만 Image classification 분야에 큰 발전을 보여준 VGG 모델 구조를 참고해 레이어를 더 깊고 복잡하게 구성했습니다. 결과적으로 블록 각각이 Conv > BN > Conv > BN > Pooling > Dropout 구조로 더 깊어 졌으며 블록수도 2개에서 3개로 늘렸습니다. 또한 마지막 fully-connected 블록에도 BN 레이어를 추가해서 높은 성능을 유도했습니다.



<그림2> 2차 실험 결과 그래프

그 결과1차 실험보다 더 빠르게 높은 성능을 달성할 수 있었습니다(약 89%). 하지만 여전히 지속적으로 accuracy 그래프가 우상향 (loss그래프는 우하향) 하고 있기에 최종적으로 epochs 수를 300까지 돌려보아 overfitting이 발생하는 지점을 파악해 해당 지점까지의 모델 최대 성능을 파악해봤습니다.

1. **최종 결과 보고**



<그림3> 최종 실험 결과 그래프

300 epochs까지 돌려본 결과입니다. 아쉽게도 이전과 동일하게 약 50 epochs 이후부터 validation에 대한 성능이 크게 개선되지 않았습니다. 대략 7~80 epochs 이후부턴 training 성능은 계속 높아지지만 validation 성능은 높아지지 않아 모델이 추가적인 학습을 하고 있지 못하고 있습니다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

<그림4> 최종 실험 결과 accuracy

최종적으로 validation 데이터셋에 대해선 92.74%의 accuracy를 달성했으며 test 데이터셋에는 약 91%의 accuracy를 달성할 수 있었습니다.

90%가 넘는 충분한 정확도를 달성했지만, 더 높은 데이터셋을 달성하기 위해서 1) 더 최신의 고도화된 모델 사용, 2) Data augmentation이나 optimizer, dropout등에 여러 hyper parameter를 통해 최적의 셋팅값 구성 등 다양한 방안을 통해 더 높은 성능을 달성할 수 있을 것이라 생각합니다.