**과목명: 시스템프로그래밍**

**1 분반**

**<<Project #4>>**

**서강대학교 [컴퓨터 공학과]**

**[20171653]**

**[신나현]**

**목차**

1. **프로그램 개요**
2. **프로그램 설명**
   1. 프로그램 실행 방법
   2. 모델의 구성 방법 및 향상 방법
   3. confusion matrix
   4. Loss function
   5. 학습 및 테스트 데이터 Rank 4
3. **프로그램 개요**

기본적인 이미지 분류 예제로 유명한 MNIST dataset을 가지고 이미지 분류를 하는 머신러닝 모델을 개발하는 방법을 살펴본다.

MNIST 예제를 바탕으로 프로젝트 4에서는 Google Colab에서 Keras를 이용하여 이미지 분류를 하는 머신 러닝 모델을 개발한다. CIFAR-10이라는 dataset을 이용하여 이미지 분류 성능을 측정하고 직접 모델의 파라미터들을 바꿔가며 테스트 셋에 대한 정확도 80% 이상을 얻는 것을 목표로 한다.

1. **프로그램 설명**
   1. **프로그램 실행 방법**

20171653.ipynb파일을 구글 드라이브에 올린 뒤 google colab에서‘파일’탭을 누른 뒤‘드라이브에서 찾기’를 클릭해준다. 그러고 나서 나의 구글 드라이브에서 20171653.ipynb파일을 찾아 클릭해주고‘Google Colaboratory로 열기’를 클릭해주어 해당 항목을 연다.

해당 파일이 google Colab에서 열리면‘런타임’탭의 ‘모두 실행’ 버튼을 눌러 실행해 주면 된다.

만약 실행 속도가 너무 느리다면‘수정’탭을 눌러‘노트 설정’버튼을 누른 뒤 하드웨어 가속기가 GPU로 설정되어 있는지 확인한다.

* 1. **모델의 구성 방법 및 향상 방법**

스크린샷이(가) 표시된 사진

자동 생성된 설명

우선 32개의 3x3 kernel size의 convolution filters가 있는 layer를 추가해준다. 여기서 activation function은 relu를 사용한다. input\_shape는 32x32크기와 컬러 값으로 3이되어 (32, 32, 3)의 값을 가진다. 이 layer를 거치고 나면 (30, 30, 32)의 output이 된다.

그런 뒤 64개의 3x3 kernel size의 filters가 있는 layer를 추가해 준다. 이 layer를 거치고 나면 (28, 28, 64)의 output이 된다.

2x2 kernel size의 MaxPooling하는 layer를 추가하여 주요 값만 뽑아 작은 출력 값을 만들어준다. pooling을 거치고 나면 (14, 14, 64)가 output이 된다.

여기에 overfitting을 예방하기 위해 Drpout(0.25)를 적용해준다.

다시 여기에 128개의 3x3 kernel size의 filters가 있는 layer를 추가해 주고 이 layer를 거치고 나면 (12, 12, 128)의 output이 된다.

그런 뒤 256개의 2x2 kernel size의 filters가 있는 layer를 추가해 준다. 이 layer를 거치고 나면 (11, 11, 256)의 output이 된다.

위와 동일하게 2x2 kernel size의 MaxPooling하는 layer를 추가하여 작은 출력값을 만들어준다. MaxPooling을 거치고 나면 (5, 5, 256)이 output이 된다. 그런 뒤 다시 Dropout(0.25)를 적용해 준다.

여기에 또 한 번 512개의 2x2 kernel size의 filters가 있는 layer를 추가해 준다. 이 layer를 거치고 나면 (4, 4, 512)의 output이 되고 MaxPooing을 해주면 (2, 2, 512)의 output을 가진다. Dropout(0.25)를 적용해 준다.

여기까지의 과정은 convolution과 pooling layer로 feature를 추출하는 단계였다.

MaxPooling과 Convolution은 주로 2차원을 다루기 때문에 Flatten()을 이용해 1차원으로 바꿔준 뒤 fully connected layer에 넣어준다. 1차원으로 바꾼 결과는 2\*2\*512 = 2048이 된다.

다음 단계는 fully connected layer를 이용해 이는 어떤 class에 속하는지를 판단해 주는 것이다.

relu를 activation function으로 사용하는 2048 to 256인 fully connected layer를 사용해준다. 그 결과로 나온 output은 256이 된다.

여기서 한 층의 fully connected layer를 더 추가해 주는데 여기서 추가하는 layer는 relu를 activation function으로 사용하며 256 to 128인 layer이다. 여기서의 output은 128이 된다.

그런 뒤 다시 dropout(0.5)를 통해 overfitting을 예방해 준다.

마지막으로 activation function으로 softmax를 사용하는 128 to numclass 즉 128 to 10인 fully connected layer를 추가해주어 결과적으로 output이 10이 나오도록 해준다.

스크린샷이(가) 표시된 사진

자동 생성된 설명스크린샷이(가) 표시된 사진

자동 생성된 설명

또한 optimizer도 바꿔서 적용해 주었다.

나이프이(가) 표시된 사진

자동 생성된 설명

스크린샷이(가) 표시된 사진

자동 생성된 설명

여기서는 optimizer로 Adam을 사용해 주었고 learning rate는 0.0005로 설정해 주었다.

Adam은 Adagrad와 RMSProp의 장점을 섞어놓은 것인데 step size가 gradient의 rescaling에 영향 받지 않는다. gradient가 커져도 step size는 묶여 있어서 어떤 objective 함수를 사용한다 하더라도 안정적으로 최적화를 위한 하강이 가능하다는 장점이 있다.

이렇게 layer들을 추가해 주고 optimiser와 learning rate를 변경시켜 80퍼센트 이상의 정확도를 얻을 수 있었다.

* 1. **confusion matrix**

다음은 학습으로 훈련된 분류 알고리즘의 성능을 시각화 할 수 있는 표인 confusion matrix이다.

키보드, 컴퓨터이(가) 표시된 사진

자동 생성된 설명

* 1. **Loss function**

스크린샷, 나이프이(가) 표시된 사진

자동 생성된 설명

loss function으로는 categorial\_crossentropy를 사용해 주었다. 여기서 class가 10이기 때문에 binary가 아닌 categorial을 사용해준 것이다.

여기서 crossentropy loss function을 사용하면 출력 레이어에서 활성화 함수의 도함수에 의한 영향을 제거할 수 있다. 또한 이는 낮은 확률로 예측해서 맞추거나 높은 확률로 예측해서 틀리는 경우 loss가 더 크다.

* 1. **학습 및 테스트 데이터 Rank 4**

TensorFlow 시스템에서, tensor는 rank라는 차원 단위로 표현된다. 이Tensor rank는 행렬의 rank와 다르다. 이는 tensor의 차원을 의미한다. 이 프로젝트에서 사용한 학습 및 테스트 데이터는 이미지이다. 따라서 [batch, 이미지 높이, 이미지 너비, 이미지 채널] 이렇게 4개의 rank를 갖는 tensor이다. 그러므로 rank를 4로 정하는 것이다.