**과목명: 시스템프로그래밍**

**1분반**

**<<Project #4>>**

**서강대학교 [컴퓨터공학과]**

**[20181617]**

**[김채연]**

목 차

1. **프로그램 개요**
2. **프로그램 설명**
3. **모델 설명**

1. 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.

2. 프로그램 설명

1) 프로그램 실행방법

- 20181617.ipynb 파일을 다운로드 한다.

- Google Colab 사이트(colab.research.google.com)에 접속한다.

- Google 계정을 로그인한다.

- 파일-노트열기-업로드를 순서대로 클릭한 뒤 다운받은 20181617.ipynb를 선택한다.

- 런타임-모두실행을 클릭한 뒤 프로그램을 실행한다.

3. 모델 설명

1) 모델 개발하기

Google Colab에서 Keras를 이용하여 이미지 분류를 하는 머신 러닝 모델을 개발한다. Keras를 사용하면 data set을 직접 다운받지 않아도 쉽게 가지고 올 수 있다. Keras는 기본적으로 Python으로 작성되어 있다. Keras에서 레이어를 구성하는 방법으로 사용되는 자료구조형은 Model이다.

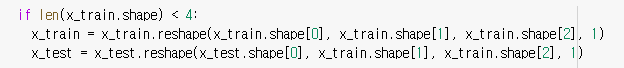
레이어는 간단하게 .add()를 사용하여 추가할 수 있다. 모델이 구성되면 .compile()을 통해 학습 방법을 설정할 수 있다. 그 뒤에 .fit()을 통해 모델에 직접 학습 데이터를 넣어 학습시킬 수 있다.

-loss function

loss function은 손실함수라고 하며 하나의 모델을 컴파일 하기 위해 필요한 두개의 매개변수 중 하나이다. 모델은 랜덤한 prediction을 가지게 되는데 모델을 학습시키며 가중치들을 수정해나가고 prediction을 정확하게 개선해야한다. 이때 loss function을 이용한다.

이 모델에서는 categorical\_crossentropy를 사용하였다. 이 함수는 모델이 리턴한 예측값과 실제 label을 비교하여 이 모델이 이미지를 잘 분류하고 있는지를 판단한다. prediction값과 label이 일치한다면 낮은 손실값을 리턴하고 다르다면 높은 손실값을 리턴한다. 따라서 모델이 낮은 loss값을 리턴해야 prediction이 정확한 것이므로 낮은 loss 값을 리턴하도록 모델의 인자값들을 변경 및 수정하는 과정을 거친다.

-rank



위의 코드를 보면 if문을 통해 전처리 과정에서 학습 및 테스트 데이터의 rank 를 4로 맞추어 주는 것을 확인 할 수 있다. 가장 기초적인 데이터는 텐서라고 부르는 다차원 배열 데이터에서 시작하는데 이 텐서의 축 개수를 rank라고 부른다. 이 축의 개수를 4로 맞추어 주는 것이다.

2) 모델 구성하기

model 중 가장 간단한 형태의 model인 Sequential Model을 사용한다. Sequential Model은 레이어들을 선형으로 쌓는 것이다. 아래의 그림을 보면 layer들이 순차적으로 쌓여져 있는 것을 확인 할 수 있다.

- model.add(Conv2D(64, kernel\_size=(3, 3), activation='relu', input\_shape=x\_train.shape[1:]))

Activation function으로 ReLU를 사용하는 64개의 3\*3 Kernel size의 convolution filters가 있는 layer를 추가한다. 첫번째 layer이기 때문에 input\_shape 인자를 넘겨준다. 이 코드의 output으로 (None, 30, 30, 64)가 나온다.

- model.add(MaxPooling2D(pool\_size=(2, 2)))

2\*2 Kernel size의 max pooling하는 layer를 추가한다. 이 코드의 input으로 (None, 30, 30, 64)가 들어오고 output으로 (None, 15, 15, 64)가 나온다.

- model.add(Dropout(0.15))

layer의 모든 노드에서 나가는 activation을 0.15의 확률로 지운다. 이 코드의 input으로 (None, 15, 15, 64)가 들어오고 output으로 (None, 15, 15, 64)가 나온다.

- model.add(Conv2D(128, (3, 3), activation='relu'))

Activation function으로 ReLU를 사용하는 128개의 3\*3 Kernel size의 convolution filters가 있는 layer를 추가한다. 이 코드의 input으로 (None, 15, 15, 64)가 들어오고 output으로 (None, 13, 13, 128)가 나온다.

- model.add(MaxPooling2D(pool\_size=(2, 2)))

2\*2 Kernel size의 max pooling하는 layer를 추가한다. 이 코드의 input으로 (None, 13, 13, 128)가 들어오고 output으로 (None, 6, 6, 128)가 나온다.

- model.add(Dropout(0.25))

layer의 모든 노드에서 나가는 activation을 0.25의 확률로 지운다. 이 코드의 input으로 (None, 6, 6, 128)가 들어오고 output으로 (None, 6, 6, 128)가 나온다.

- model.add(Conv2D(256, (3, 3), activation='relu'))

Activation function으로 ReLU를 사용하는 256개의 3\*3 Kernel size의 convolution filters가 있는 layer를 추가한다. 이 코드의 input으로 (None, 6, 6, 128)가 들어오고 output으로 (None, 4, 4, 256)가 나온다.

- model.add(MaxPooling2D(pool\_size=(2, 2)))

2\*2 Kernel size의 max pooling하는 layer를 추가한다. 이 코드의 input으로 (None, 4, 4, 256)가 들어오고 output으로 (None, 2, 2, 256)가 나온다.

- model.add(Dropout(0.35))

layer의 모든 노드에서 나가는 activation을 0.35의 확률로 지운다. 이 코드의 input으로 (None, 2, 2, 256)가 들어오고 output으로 (None, 2, 2, 256)가 나온다.

- model.add(Flatten())

1차원으로 바꿔준다. 이 코드의 input으로 (None, 2, 2, 256)가 들어오고 output으로 (None, 1024)가 나온다.

- model.add(Dense(512, activation='relu'))

Activation function으로 ReLU를 사용하는 K to 512인 fully connected layer를 사용하는 것을 의미한다. 이 코드의 input으로 (None, 1024)가 들어오고 output으로 (None, 512)가 나온다.

- model.add(Dense(1024, activation='relu'))

Activation function으로 ReLU를 사용하는 K to 1024인 fully connected layer를 사용하는 것을 의미한다. 이 코드의 input으로 (None, 512)가 들어오고 output으로 (None, 1024)가 나온다.

- model.add(Dropout(0.45))

layer의 모든 노드에서 나가는 activation을 0.45의 확률로 지운다. 이 코드의 input으로 (None, 1024)가 들어오고 output으로 (None, 1024)가 나온다.

- model.add(Dense(num\_classes, activation='softplus'))

Activation function으로 SoftPlus를 사용하는 K to num\_classes인 fully connected layer를 사용하는 것을 의미한다. num\_classes는 앞에서 설정된 10이다. 이 코드의 input으로 (None, 1024)가 들어오고 output으로 (None, 10)가 나온다.

아래는 구성한 모델을 가시화 한것이다. 위의 코드에서 layer를 추가한 순서대로 모델이 구성되어 있는 것을 확인할 수 있다.

|  |
| --- |
|  |

3) 모델 향상 방법

초기 주어진 코드에서 .add를 통해 layer를 더 추가하였다. layer의 filter 개수도 각 64, 128, …으로 각각 다르게 하였다.

Droupout 확률을 filter 개수에 비례하여 0.15, 0.25, … 순으로 다르게 해주었다.

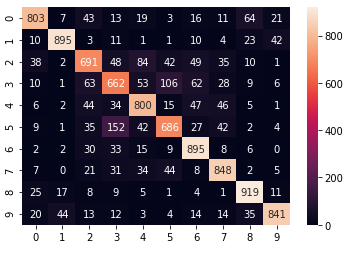
이렇게 Convolution filter가 있는 layer를 추가하고 maxpooling하고 이를 dropout하는 과정을 3번 반복하여 정확도를 높였다.

그리고 flatten한뒤 크기가 다른 Dense layer도 추가해 주었다.

마지막으로 신경망의 출력을 결정하는 Activation function도 여러가지 activation function을 해본 결과 softplus가 가장 정확도가 높게 나와 softmax에서 softplus로 변경하였다.

optimizer는 최적의 가중치를 검색하는데 사용되는 최적화 알고리즘인데 학습률을 학습진행에 따라 서서히 낮추어 가며 학습하는 방법인 adagrad가 마찬가지로 가장 정확도가 높아서 변경해주었다.

4) Confusion matrix



분류 결과를 표로 확인할 수 있다. 정답을 맞춘 경우는 왼쪽 모서리부터 대각 방향에 위치한다. 맞힌 개수가 많을수록 왼쪽 막대의 상단의 색깔에 가깝다. 대각선이 주로 밝은 색 위주로 되어있으므로 잘 작동을 하는 모델으로 판단 할 수 있다.