**과목명: 시스템프로그래밍**

**담당 교수 명: 김 지 환**

**2분반**

**<<Project #4>>**

**서강대학교 컴퓨터학과**

**20181618**

**김하늘**

목 차

1. **프로그램 개요**
2. **프로그램 실행방법**
3. 프로그램 설명
   1. 모델 구성
   2. 전처리
   3. Confusion matrix
   4. Loss function
4. 프로그램 개요

이 프로그램은 프로젝트4로, CIFAR-10 데이터 셋에 대해 이미지 분류를 하는 머신 러닝 모델을 개발하는 것이다. epoch 수는 50으로 고정되며, 기존의 53.86%의 정확도에서 80% 이상의 정확도를 나오도록 성능 향상을 하는 것이 개발의 목표이다.

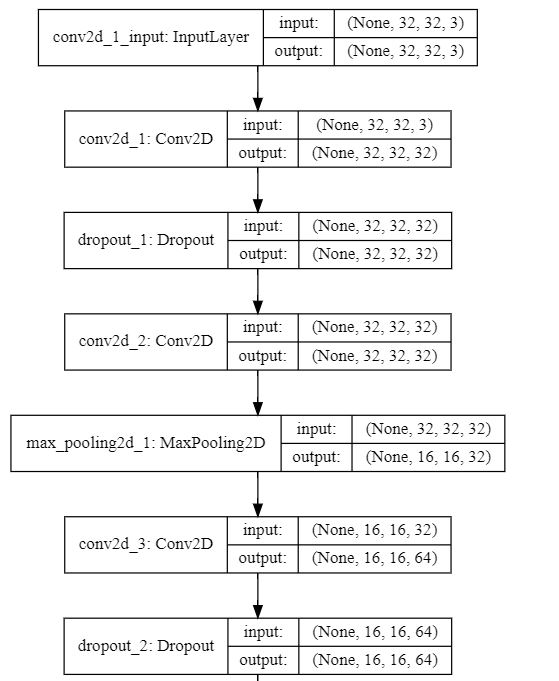
1. 프로그램 실행방법

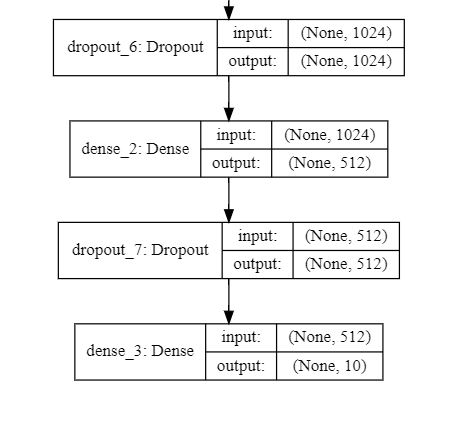
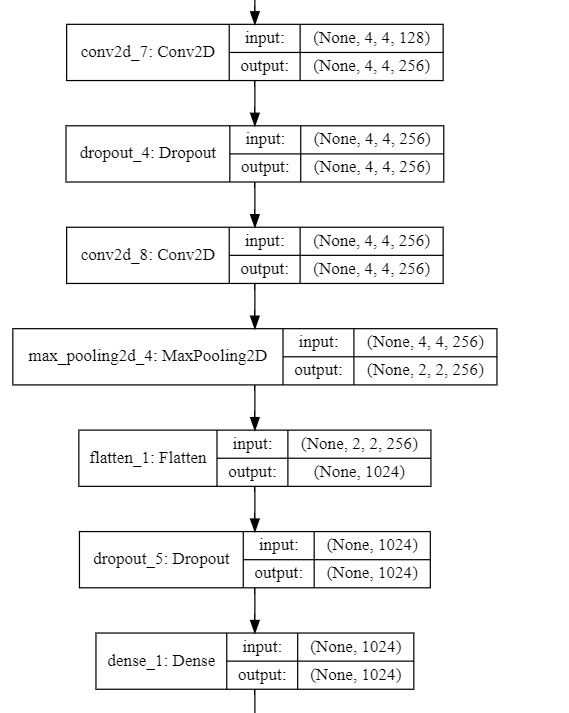
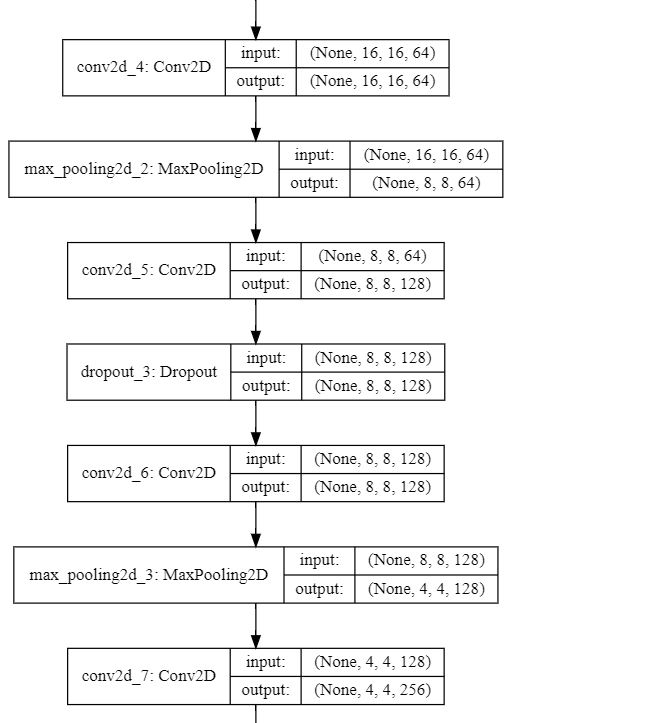
colab환경에서 파일-노드열기-업로드영역에 .ipynb 파일을 연다. 이후 런타임-모두실행을 통해 프로그램을 실행킨다.

1. 프로그램 설명
   1. 모델 구성

기존에 주어진 코드에서 추가적인 레이어를 많이 쌓았고, 거기에 따라 Dropout의 비율을 줄였다. 또한 기존의 activation function인 Relu에서 ELU로 바꾸었다. elu는Relu의 특성을 공유하지만, 그래디언트 값이 죽지 않아 출력값이 zero-centered에 가깝다는 장점이 있다.

<Layer 구조>





**- model.add(Conv2D(32, (3, 3), input\_shape=x\_train.shape[1:], activation='elu', padding='same'))**

입력층을 생성한다. 우선 input shape에 들어가는 x\_train의 shape를 살펴보면 (50000, 32, 32, 3)이므로 [1:]로 인덱싱하여 (32, 32, 3)의 값을 가져왔다. 또한 Activation function으로 ELU를 사용하는 32개의 3x3 kernel size의 convolution filters가 있는 layer를 추가하였고, padding에 same값을 넣어서 입출력시 데이터의 행, 열 크기는 같다.

**- model.add(Dropout(0.15))**

앞의 hidden layer에 배치되어있는 노드 중 15%를 임의로 꺼놓고 나머지 비율의 노드에 대해 학습을 진행한다.

**- model.add(Conv2D(32, (3, 3), activation='elu', padding='same'))**

Activation function으로 ELU를 사용하는 32개의 3x3 kernel size의 convolution filters가 있는 layer를 추가하였고, padding에 same값을 넣어서 입출력시 데이터의 행, 열 크기는 같다.

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

2x2 크기로 Max Polling을 한다.

**- model.add(Conv2D(64, (3, 3), activation='elu', padding='same'))**

Activation function으로 ELU를 사용하는 64개의 3x3 kernel size의 convolution filters가 있는 layer를 추가하였고, padding에 same값을 넣어서 입출력시 데이터의 행, 열 크기는 같다.

**- model.add(Dropout(0.15))**

앞의 hidden layer에 배치되어있는 노드 중 15%를 임의로 꺼놓고 나머지 비율의 노드에 대해 학습을 진행한다.

**- model.add(Conv2D(64, (3, 3), activation='elu', padding='same'))**

Activation function으로 ELU를 사용하는 64개의 3x3 kernel size의 convolution filters가 있는 layer를 추가하였고, padding에 same값을 넣어서 입출력시 데이터의 행, 열 크기는 같다.

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

2x2 크기로 Max Polling을 한다.

**- model.add(Conv2D(128, (3, 3), activation='elu', padding='same'))**

Activation function으로 ELU를 사용하는 128개의 3x3 kernel size의 convolution filters가 있는 layer를 추가하였고, padding에 same값을 넣어서 입출력시 데이터의 행, 열 크기는 같다.

**- model.add(Dropout(0.15))**

앞의 hidden layer에 배치되어있는 노드 중 15%를 임의로 꺼놓고 나머지 비율의 노드에 대해 학습을 진행한다.

**- model.add(Conv2D(128, (3, 3), activation='elu', padding='same'))**

Activation function으로 ELU를 사용하는 128개의 3x3 kernel size의 convolution filters가 있는 layer를 추가하였고, padding에 same값을 넣어서 입출력시 데이터의 행, 열 크기는 같다.

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

2x2 크기로 Max Polling을 한다.

**- model.add(Conv2D(256, (3, 3), activation='elu', padding='same'))**

Activation function으로 ELU를 사용하는 256개의 3x3 kernel size의 convolution filters가 있는 layer를 추가하였고, padding에 same값을 넣어서 입출력시 데이터의 행, 열 크기는 같다.

**-model.add(Dropout(0.15))**

앞의 hidden layer에 배치되어있는 노드 중 15%를 임의로 꺼놓고 나머지 비율의 노드에 대해 학습을 진행한다.

**- model.add(Conv2D(256, (3, 3), activation='elu', padding='same'))**

Activation function으로 ELU를 사용하는 256개의 3x3 kernel size의 convolution filters가 있는 layer를 추가하였고, padding에 same값을 넣어서 입출력시 데이터의 행, 열 크기는 같다.

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

2x2 크기로 Max Polling을 한다.

**- model.add(Flatten())**

전결합층에 전달해 학습하기 위해, 2차원에서 1차원 배열로 만들어준다.

**-model.add(Dropout(0.3))**

앞의 hidden layer에 배치되어있는 노드 중 30%를 임의로 꺼놓고 나머지 비율의 노드에 대해 학습을 진행한다.

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

1024개의 노드를 가진 full connected layer를 추가한다. 출럭 뉴련이 1024개가 된다.

**- model.add(Dropout(0.3))**

앞의 hidden layer에 배치되어있는 노드 중 30%를 임의로 꺼놓고 나머지 비율의 노드에 대해 학습을 진행한다.

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

512개의 노드를 가진 full connected layer를 추가한다. 출럭 뉴련이 1024개가 된다.

**- model.add(Dropout(0.5))**

앞의 hidden layer에 배치되어있는 노드 중 50%를 임의로 꺼놓고 나머지 비율의 노드에 대해 학습을 진행한다.

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

출력층을 생성한다.

* 1. 전처리

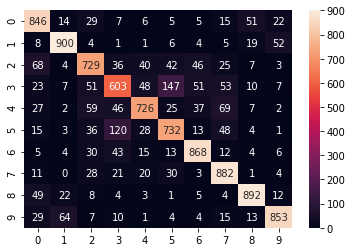
**if len(x\_train.shape) < 4:**

**x\_train = x\_train.reshape(x\_train.shape[0], x\_train.shape[1], x\_train.shape[2], 1)**

**x\_test = x\_test.reshape(x\_test.shape[0], x\_train.shape[1], x\_train.shape[2], 1)**

CIFAR-10 이미지를 전처리 하기위한 단계이다. CNN과 같은 이미지 데이터를 다룰 때는 입력 데이터로 4차원 데이터를 다루는데, (image수, channel 수, height, width)의 구조를 가진다. x\_train의 shape를 살펴보면 본 프로그램에선 미리 (50000, 32, 32, 3)로 초기화가 되어있는데, 만약 shape의 길이가 4가 되지 않을 경우 3번째 인덱스에 1값을 추가하여 4차원으로 만든다.

* 1. Confusion matrix



* 1. Loss function

이미지를 얼마나 학습했는지(epoch)에 따라 손실의 정도를 체크할 필요가 있다. Optimization을 통해 loss function에서의 손실이 최소가 되는 부분을 찾을 수 있는데, 이 프로그램에선 Cross entropy를 사용하였다.

로 정의될 수 있으며, ti는 목표 확률, si는 각 클래스 i에 대한, 현재 학습한 확률값이다. 어떤 si를 학습하고 있을때, ti에 가까워질수록 CE의 값은 작아지게 된다.

현재 프로그램의 Categorical Cross Entropy loss는 softmax activation 뒤에 cross entropy loss를 붙인 형태이다. 이미지를 분류하는 작업의 경우 주로 사용된다.