<인공지능 3차 과제 1>

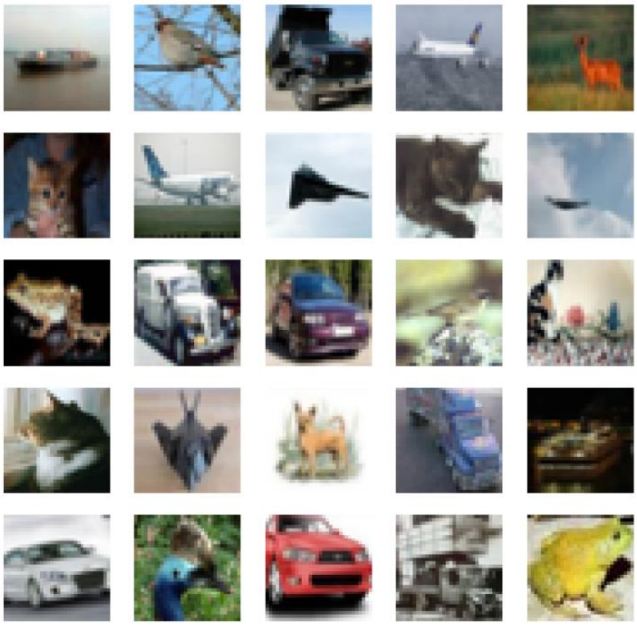
<과제 3.1: CIFAR-10 인식을 위한 Shallow Convolution Neural Network 설계>

1. 개요

이 과제에서는 CIFAR-10 데이터 셋에 포함된 사물과 동물을 분류하기 위한 Convolution Neural Network Classifier 와 VGGNet을 디자인한다. Python으로 구현하고, 사용가능한 라이브러리를 numpy, pillow, pickle, OrderedDict() 으로 제한한다.

2. CIFAR-10

비행기, 자동차, 새, 고양이, 사슴, 개, 개구리, 말, 배, 트럭 총 10가지의 이미지로 구성된 이미지 집합으로, 훈련 이미지 50,000장, 시험 이미지 10,000장으로 이루어져 있다. 각 이미지 데이터는 32(가로)x32(세로)x3(RGB)의 칼라 이미지로 이루어져 있고, 각 픽셀의 값은 0부터 1까지의 값을 취한다.(Normalized 되어 있다.) CIFAR10-Web으로 받은 이미지는 [데이터의 개수,(32*32(R)+32*32(G)+32*32(B))=즉 3072] 즉 한 줄로 되어있다.



3. 개요

Input Image 즉 CIFAR-10의 경우 5만개의 training data와 10000개의 test data로 이루어져 있으며, 각각의 데이터는 R 32*32, G32*32, B32*32의 데이터가 일렬로 이어진 3072개의 feature를 가지고 있다. 즉 train_data.shape = (50000,3072) test_data.shape = (10000, 3072) 이다. 우리는 이 data를 Shallow_CNN를 통해 Classifier를 하는 것이 목적이고, 그렇기에 이 과제에선 Shallow_CNN Architecture를 직접 구현해야 한다.

그림 1

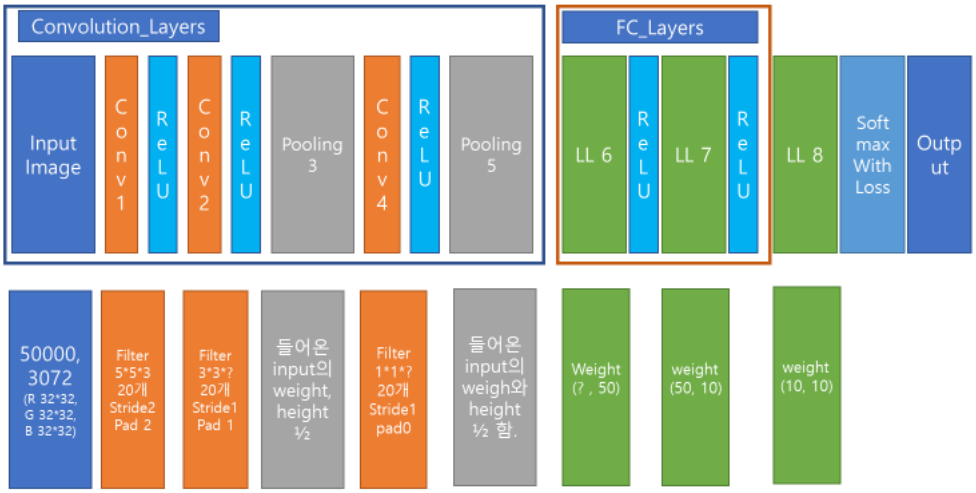


그림1은 1차 과제의 Shallow CNN의 전체적인 모습이다. Conv Layer와 Pooling Layer로 이루어져 있는 Convolution_Layers 와 Linear_Layer로 이루어져있는 FullyConnected_Layers 그리고 Softmax로 들어가기 전 class의 개수와 Score의 결과값을 맞춰주기 위한 LL8과 CrossEntropy를 사용하기 위한 클래스 SoftmaxWithLoss가 존재한다.

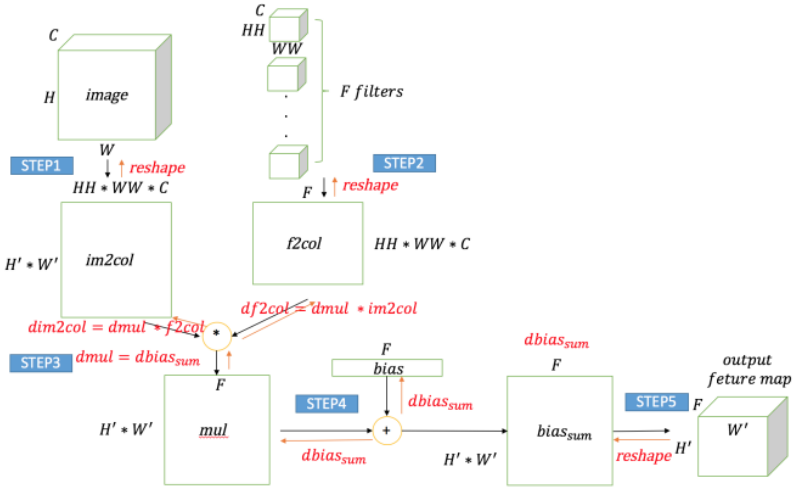
Convolution_Layers의 Conv1에서는 filter의 크기(size)는 5*5 그리고 Input Image의 채널(RGB)에 맞는 3개의 channel로 이루어져있고, filter가 20개 존재한다. Conv1 안에 있는 모든 필터는 stride=2, pad=2으로 동작한다. Conv1을 계산한 결과값을 Non-Linear화 하기 위해 Activate Function 클래스인 ReLU를 통과시켜 준다. Conv2에서는 filter의 크기(size)는 3*3을 사용한다. 또한 filter의 개수는 20개이다. stride와 pad는 각각 1로 동작한다. Conv2를 계산한 결과값을 Non-Linear화 하기 위해 클래스 ReLU를 통과시키고, ReLU에 통과된 data를 MaxPooling으로 Pooling해 가로와 세로를 절반으로 줄인다.(Pool3) Conv4는 1*1*?의 필터를 가지고 있고 stride=1, pad=0이다.

이후 그 결과값을 MaxPooling 방식으로 가로와 세로를 절반으로 줄인다.(Pool5). 여기까지가 Convolution_Layers이다.

그 후 FC_Layers와 Softmax Layer를 통해서 Classifier를 할 것이다. LL6의 경우 Conv_olution_Layers의 결과값으로 나온 4차원의 데이터(데이터의 개수, 채널, width, height)를 2차원의 data로 바꾸고, 그 것에 대해 FullyConnected 연산을 한다. LL1에서의 뉴런의 개수는 50개이다. 그 후 Non-Linear 화 하기 위해 ReLU를 사용한다. LL2의 뉴런의 개수는 10개이고, Non-Linear 화 하기 위해 ReLU를 사용하였고, 마지막 LL3는 Class를 맞추기 위해 그리고 Non-linear를 사용하지 않기 위해 따로 분류하였다. 즉 class의 개수인 10개의 뉴런을 사용하고 FullyConnected 연산만 한다.

데이터 5만개를 다 사용하는 것은 이번 과제에 큰 부담이 되므로, 데이터는 5000개만 학습 시킬 것이다. 또한 5000개를 한 번에 학습시키는 것이 아닌 minibatch(batch_size =100)를 통해 학습을 할 것이다. Backpropagation은 Stochastic Gradient Descent가 될 것이다. 이미 트레이닝은 해둔 상태로 Weight와 bias를 구해두었다.

그림2



이 것은 im2col 과 col2im에 대한 기본적인 설명이다. Im2col과 col2im을 사용하는 목적은 이미지 데이터를 2차원 벡터라이즈화 하여서 병렬연산(np.dot)을 하기 위함이다. Im2col은 4차원 image의 shape를 2차원의 shape(데이터, feature) 으로 바꾸는 것이고 col2im은 2차원 shape(데이터, feature)를 다시 4차원 데이터로 바꾸는 것이다.

4. 코드에 대한 설명

4.0 load_params

load_params 는 미리 트레이닝해둔 weight와 bias를 해당 ShallowCNN클래스에 넣어주는 함수이다. 인자로는 ShallowCNN 클래스와 "params3rd.pkl" 이라는 트레이닝된 데이터를 받고서 해당 클래스의 Weight와 bias에 업데이트 해준다.

4.1 LinearLayer class

FullConnected 연산을 하기위한 Layer로 forward와 backward가 구현 되어있다. forward 에서 전과 달라진 점은 4차원 데이터가 들어왔을 때에는 2차원으로 평탄화 시켜준다. backward에선 이전 데이터의 shape 형태로 backward 해준다.

4.2 ReLU class

Non-Linear 연산을 하는 Activate Function을 하는 클래스이다. 마찬가지로 Forward와 Backward가 구현되어 있다.

4.3 im2col

4차원의 이미지묶음(이미지의 개수, 가로, 세로, 채널)을 필터를 통과하는 연산을 빠르게 하기 위해 2차원화 하는 함수이다.(np.dot을 통한 병렬처리를 하려고 하는 것) filter_w와 filter_h를 통해서 out_height와 out_weight(filter를 거쳐서 나오는 데이터의 가로와 세로 크기)을 먼저 구한 후 아래의 연산을 진행한다.

4.4 col2im

2차원의 shape를 가진 연산을 다시 4차원 이미지로 바꾸는 함수이다.

4.5 Convolution

이미지 묶음(혹은 4차원 데이터)과 filter와 연산을 하기위해 만든 클래스이다. Convolution Layer에는 그 Layer에 해당하는 모든 filter가 Weight로 존재한다. 즉 Convolution의 W = (filter의 개수, filter의 channel, filter의 height, filter의 weight) 로 존재하게 된다. 각각 convolution 연산을 하기 위한 변수들이 존재하고 forward와 backward 함수가 구현되어 있다.

4.6 Pooling

이미지 묶음(혹은 4차원 데이터)의 weight와 height를 줄이기 위해 MaxPooling 하는 클래스이다. Forward와 backward에 어떻게 연산하는지에 대해 잘 구현되어 있다.

4.7 CNN_Layer_dim_set

Convolution_Layers[Convolution Layer들의 집합]를 setting하는 함수이다. input으로 list를 받으며, 이 list를 통해서 convolution_layers{Convolution과 Pooling이 존재}가 생성이 된다. 예시를 들어 설명한다면 layerlist= [['C', 30, 3, 3, 2, 1], ['P', 2, 2, 0]] 이고, 이 CNN_Layer_dim_set(layerlist)를 하면 Convolution Layer가 하나 Pooling Layer가 하나 순차적으로 생성이 된다.

List[0]은 Convolution이면 'C', Pooling이면 'P'이고, Convolution Layer의 경우 filter의 개수는 list[1]인 30개, filter의 size는 list[2]인 3개, 이 필터의 channel은 list[3]인 3개, 이 필터의 stride 는 list[4]인 2, 필터의 pad 는 list[5]인 1이 된다.

Pooling Layer의 경우 list[1]은 Pooling filter의 Size이고, list[2]는 그 filter의 stride, list[3]은 그 필터의 pad이다.

layerlist = [['C', 20, 5, 3, 2, 2], ['C', 20, 3, 20, 1, 1], ['P', 2,2,0], ['C', 20, 1, 20, 1, 0], ['P', 2, 2, 0]] 이라면 CNN_Layer_dim_set의 결과는 Conv1(filter 20개 filter_size = 5*5, filter channel = 3, stride=2, pad=2),

Conv2(filter 20개 filter_size = 3*3, filter channel = 20, stride=1, pad=1)

Pool3(filter_size =2*2, stride=2, pad=0)

Conv4(filter 20개 filter_size = 1*1, filter channel = 20, stride=1, pad=0)

Pool5(filter_size=2*2, stride=2, pad=0) 이 된다.

4.8 FullyConnected_Layers_set

위와 같이 FCLayers를 Setting하는 함수이다. Input으로 list를 받으며, 이 리스트를 통해서 FullyConnected Layer가 생성된다. 예를들어 [[160, 50], [50, 10]]이면 두개의 LinearLayer가 생성이 되고 LinearLayer 1의 Weight는 (160,50)의 shape를 가지고 Linear Layer2는 (50, 10)의 shape를 가지게 된다.

4.9 ShallowCNN

여러분들이 구현화 해야하는 클래스이다. 생성할 때 ConvLayerList와 FullLayerlist라는 두 개의 리스트를 인자로 받으며 그 2개의 리스트를 통해 자동으로 Convolution_Layer와 FC_Layer가 생성이 된다. 그 후 self.layers 라는 OrderedDict() 타입의 Dictionary 변수를 하나 만들고 거기에 순차적으로 저장한다. 저장할 때 Convolution Layer이거나 Linear Layer라면 뒤에 자동적으로 Acitviate Function을 쓰기 위한 ReLU 클래스를 추가하게 해두었다.

여러분이 구현해야 하는 것은 **Backpropagation으로 layers에 순차적으로 저장된 클래스를 backpropagation 하여 각각의 레이어에 Weight와 bias를 업데이트 하면 된다. 업데이트 방식은 2차과제와 동일하게 Gradient Descent(Ex: W = W - learning_rate * dW) 방법을 사용하면 된다.**

5. 과제 요구사항

1 ShallowCNN을 위의 함수를 사용해 구현해야 한다. Weight와 bias는 미리 training 되어있는 데이터(params.pkl)를 줄 것이기에 지정된 식으로 5번의 트레이닝을 한 후의 결과값이 아래와 같이 나오면 성공이다.

```
In [21]: ConvLayerlist =
         FullLayerlist =

         SCNN = ShallowCNN(ConvLayerlist, FullLayerlist)

         load_params(SCNN)
```

```
In [22]: train_input_x=train_images.reshape(50000, 3 ,32, 32)

         minibatch_data = train_input_x[:5000, :, :, :]
         minibatch_label = train_labels[:5000, :]
```

```
In [24]: test2_images =test_images.reshape(-1, 3, 32, 32)
         SCNN.accuracy(test2_images, test_labels)
Out[24]: 0.4345
```

```
In [25]: for i in range(5):
             loss = SCNN.forward(train_input_x[i*100:(i+1)*100], train_labels[i*100:(i+1)*100])
             grads = SCNN.backpropagation(0.1)
             print(i,"번의 Test Accuracy :", SCNN.accuracy(test2_images, test_labels))

         0 번의 Test Accuracy : 0.4406
         1 번의 Test Accuracy : 0.4414
         2 번의 Test Accuracy : 0.4399
         3 번의 Test Accuracy : 0.44
         4 번의 Test Accuracy : 0.4384
```

2 해당 과제를 리눅스에서 돌려보고 submit 해야한다. submit에 포함되어야 하는 데이터는 params.pkl과 hw3_submit.py이다. hw3_submit.ipynb 를 hw3_submit.py로 변환 후 submit 한다..

#주석을 보고 지워야 하는 부분 (matplotlib, !pip ~~)을 잘 지우고 테스트해보시길 바랍니다.

Submit 명령어는 submit K2019AIS hw3 이다.

3 해당 코드가 리눅스에서 돌아가고 제 코드의 값과 같으면 30점 아닐 시 0점 이다.

4 제출마감시간은 **6월 17일 월요일 01:00AM** 이다.

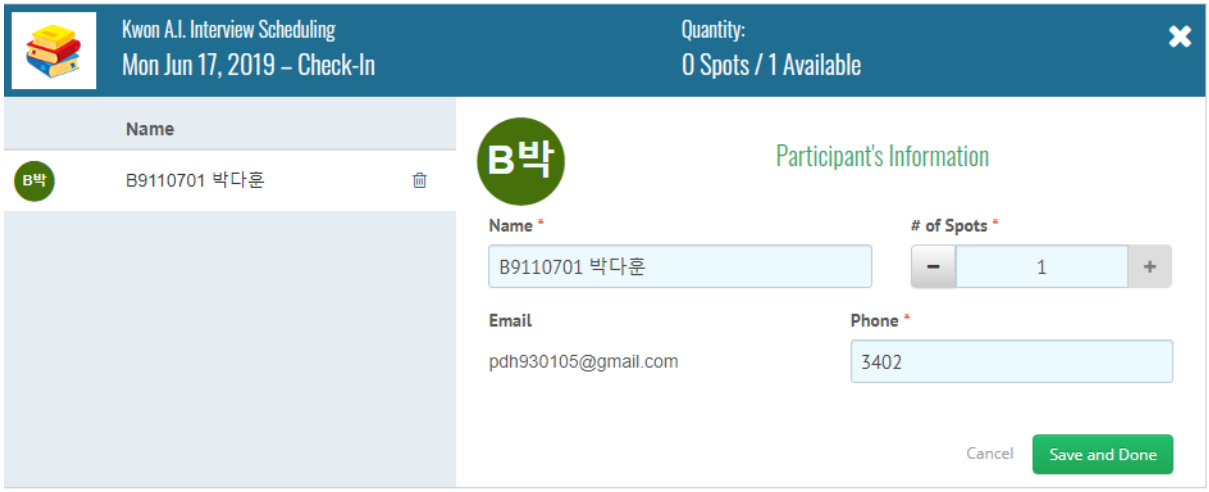
6 인터뷰(40점 + 틀릴 시 1차, 2차과제 감점)

3차과제에 대해서는 인터뷰가 존재합니다. 1차 2차에 대한 간단한 성적 매긴 이유와 감점 사유에 대한 이야기와 1차, 2차, 3차과제 + CNN 그리고 https://github.com/wonjaek36/sodeeps_project2

해당 부분의 src안의 optimizer.py와 actvation.py 또한 퀴즈의 범위입니다.

만약 1차 2차 과제에 해당하는 부분에 대한 대답을 못했을 시엔 감점이 존재합니다. 인터뷰 장소에 대해선 큰 이변이 없을 시 T719-A호에서 하겠습니다.(만약 옮겨진다면 공지하겠습니다.)

https://signup.com/client/invitation2/secure/2831225/true#/invitation



해당 링크에 자신의 학번과 이름으로 sign up을 해주세요.[휴대폰 번호는 아무거나 적으셔도 상관없습니다.] 만약 위에 해당하는 시간이 전부 안될 시에는 kwon2019ai@gmail.com으로 연락을 주시면 다른 시간대를 잡도록 하겠습니다. 시간이 지날수록 퀴즈 문제의 난이도가 더 어려워질 예정입니다.