
COSE362-2019F: Final Project Report

CNN을 이용한 고려대학교 길고양이 분류 및 배경 학습을 통한 성능 개선

고주연 공인호 김상엽 염형빈

Abstract

본 프로젝트에서는 AlexNet을 기반으로, 총 869 장의 학습 데이터를 가지고 고려대학교 이과캠퍼스에 서식하는 네 마리의 고양이에 대해 분류 해주는 CNN 모델을 설계하였다. 추가로 모델의 성능을 개선시키기 위해, ‘배경’이라는 새로운 class를 도입하여 242장의 고양이 서식지 사진을 학습시켰고, 이에 모델이 고양이의 특징과 배경의 특징을 구분하여 학습할 수 있도록 하였다. 100장의 Testset에 대해 10번의 Test를 해본 결과, 기존 53%의 정답률이 배경 학습을 통해 P_3 과 P_4 기준 각각 57%, 55%로 증가하여 모델의 성능 개선을 확인할 수 있었다.

1. Introduction

1.1. Motivation

최근 교내 고양이 개체수 증가에 따라, 이에 대한 학생들의 관심이 증가하고 있다. 하지만 고양이들의 구별 또한 어려워지고 있어, 고양이의 관리에 대한 우려 또한 증가하고 있는 실정이다. 이러한 상황에서, 고양이들을 정확히 구별해 이름과 매칭하는 작업은 교내 고양이들의 관리와 학생들의 주의를 위해서도 굉장히 중요한 일이다. 이를 위해 고양이 사진만으로 어떤 고양이인지 구별할 수 있는 모델을 지도학습과 합성곱신경망을 이용하여 구현해보고자 한다.

1.2. Problem definition

본 프로젝트에서 다룬 문제는 고려대학교에 서식하는 네 마리 고양이 뾰또, 베찌, 찰리, 채플린의 사진을 입력으로 주었을 때, 이를 분류하는 모델을 설계하는 것이다.

고양이 사진은 336*336 크기의 bmp 파일로 주어지고, 해당 입력을 받은 모델은 각 class에 대한 확률을 출력해야 한다. 모델의 평가는, test 데이터의 정답률이 높은 모델이 더 우수하다고 판단한다.

1.3. Challenges

고양이들은 자신만의 활동하는 영역이 있다. 어떤 고양이들은 활동 영역(Section 8.1 참고)이 서로 다른 반면, 항상 같이 다녀 활동 영역이 완전히 겹치는 고양이들도 존재한다. 이는 모델학습에 안좋은 영향을 끼칠 수 있다. 그 예로,

항상 비슷한 위치에서 사진이 찍힌 고양이는 고양이가 아닌 배경의 특징들이 학습될 가능성이 있으며, 이는 활동 영역을 공유하는 다른 고양이와 구분하는 데에 있어 문제가 생길 수 있다. 모델의 output으로 배경사진이라는 새로운 class를 도입해 학습시킴으로써 이를 해결하였다.

구현에 대한 문제도 존재했다. 본 프로젝트를 진행하면서, CNN을 완벽히 이해하고 그 알고리즘을 개선하자는 목표를 가지고, C++언어를 사용하여 별도의 라이브러리 없이 이를 구현하였다. 하지만 CPU 환경에서 900장의 사진 데이터를 학습시키는 경우 약 15시간이 소모된다. 뿐만 아니라, 좌우 대칭과 회전, 부분적으로 자른 사진도 학습을 시키려면 이에 8배 이상의 시간이 소모된다. 이는 CUDA와 GPU 병렬 연산처리를 통해 해결하였다. 기존에 60초 이상 걸리던 iteration 시간이 2초 미만으로 줄면서, 학습에 필요한 시간을 대폭 줄일 수 있었다. (Section 8.5 참고)

2. Image Data

2.1. 사진 수집 방법

본 프로젝트에서는 세 가지 방법을 통해 사진 데이터를 수집하였다. 첫 번째로 교내 동아리 ”고려대학교 고양이 쉼터“에 연락하여 고양이 사진을 요청하였다. 두 번째로, 학교 커뮤니티인 ‘고파스’와 ‘에브리타임’에 게시글을 검색하여 사진을 다운받았다. 사진의 다양성을 확보하고 편향을 막기 위해서는 모든 데이터를 위와 같은 방법으로 수집해야 하나, 사람들의 선호도 차이에 의해 고양이별 사진 수가 불균등하여 추가로 직접 사진을 찍어 데이터를 수집하였다. 결국 우리는 뾰또, 베찌, 찰리, 채플린에 대하여 각각 294, 232, 201, 222 장의 사진을 얻었다.

고양이들의 활동 영역 사진을 따로 학습시키기 위해 배경 사진 데이터도 필요하였다. 배경 사진은 고양이 사진에서 배경을 자르거나, 직접 사진을 찍어 데이터를 수집했다. 최종적으로 배경 사진 총 262장을 수집하였다.

2.2. Data set

위에서 수집한 뾰또, 베찌, 찰리, 채플린의 사진 중 각각 274, 212, 181, 202장을 dataset으로 설정하였다. 하지만 data의 개수가 부족하다고 판단하여, Alex Net과 마찬가지로 사진을 자르고 회전하여 더 많은 data를 확보하였다. 결과적으로 뾰또 2192장, 베찌 1800장, 찰리 1448장, 채플린 1688장을 dataset으로 사용하였다. 수집한 배경 사진

262장 중 testset을 제외한 242장을 같은 방법으로 편집하여, 총 1136장을 확보하였다.

2.3. Test set

수집한 사진 데이터 중에서 dataset으로 사용할 사진들을 제외하고 뾰뜨, 벼찌, 칠리, 채플린 그리고 배경 사진 각각 20장씩을 testset으로 사용하여 모델을 평가했다.

3. Model

본 프로젝트에서는 AlexNet을 baseline으로 설정하여 모델을 설계하였다. 우리의 모델에는 5개의 convolutional layer(이하 컨볼루션 층)와 2개의 fully connected layer(이하 fc층), 총 7개의 layer가 있으며, 각 layer에서 필터의 크기, stride 크기, zero padding, 활성화 함수, pooling 크기 등을 AlexNet과 동일하게 설정하여 진행하였다. (Krizhevsky et al., 2012)

첫번째 컨볼루션 층은 $336 \times 336 \times 3$ 의 input 사진을 96개의 $11 \times 11 \times 3$ 크기의 필터를 이용하여 stride 4로 필터링을 진행한다. 이후 활성화 함수와 stride 2의 3×3 max pooling을 거친다. 두번째 컨볼루션 층은 첫번째 컨볼루션 층의 output을 input으로 받아 256개의 $5 \times 5 \times 96$ 필터를 이용하여 필터링 후 활성화 함수와 pooling을 진행한다. 세번째 컨볼루션 층은 384개의 $3 \times 3 \times 256$ 필터를 이용하여 필터링하고 활성화 함수를 거친다. 네번째 컨볼루션 층은 384개의 $3 \times 3 \times 384$ 필터로 필터링하고 활성화 함수를 거친다. 다섯번째 컨볼루션 층은 256개의 $3 \times 3 \times 384$ 필터로 필터링 후 활성화 함수와 pooling을 진행한다. 이후 fc층에 4096개의 뉴런이 있고, 마지막 layer에서 4개의 출력값인 softmax 활성화함수를 거친다.

3.1. Baseline: AlexNet

AlexNet은 ImageNet LSVRC-2010 contest의 120만개의 이미지를 1000개의 클래스로 분류한 CNN이다. AlexNet은 5개의 컨볼루션 층과 3개의 fc층을 갖는다. (각 층에 대한 크기 정보는 Section 8.4의 표 참고)

3.1.1. ALEXNET과의 차이점

먼저 본 모델은 input 사진 크기가 $336 \times 336 \times 3$ 인 반면, AlexNet은 $224 \times 224 \times 3$ 으로 다르다. 그리고 본 모델은 fc층이 2개이나 AlexNet은 3개를 이용하였다. 또한 첫번째, 두번째 층에서 Local response normalization을 이용한 AlexNet과 달리, 본 모델에서는 정규화를 진행하지 않고 hyper parameter로 설정하여 이를 처리하였다. 그리고 AlexNet은 학습을 진행하기 위해 각 층을 두개로 분리하여 2개의 GPU로 각각 학습시켰으나, 본 모델은 GPU 1개만을 이용하여 층을 분리하지 않았다.

또한 본 프로젝트는 학습을 위하여 batch size 1인 Stochastic Gradient Descent(이하 SGD)를 이용하나, AlexNet은 batch size 128인 mini batch SGD를 이용한다. 그리고 AlexNet은 오버피팅을 막기 위하여 dropout을 이용하여

일부 뉴런을 0으로 만드는 작업을 하나 본 모델에서는 dropout을 이용하지 않았다. Learning rate 역시 본 프로젝트에서는 [0.000015, 0.000005]로 설정한 반면 AlexNet은 [0.01, 0.001]로 설정하였다.

3.1.2. SOTA : NOISY STUDENT

본 프로젝트에서도 다루고 있는 image 분류 분야에 있어 SOTA는 Noisy Student이다.(Xie et al., 2019) 이전의 ImageNet model들이 지도학습의 데이터로 labeled data만을 이용한 반면, 이 모델은 unlabeled data를 사용하여 정답률을 87.4%까지 올렸다. 본 프로젝트의 모델 또한 labeled data만을 이용하므로, unlabeled data를 접목시킨다면 훙미로운 연구주제가 될 것이라고 생각한다.

3.2. 배경 학습

고양이 사진 데이터를 분석한 결과, 영역동물의 특징으로 인해 각 고양이가 특정 공간에서 사진이 자주 찍힌 것을 확인하였다. 이로 인해 모델이 고양이가 아닌 배경의 특징으로 학습될 위험이 있다. 이에 본 프로젝트에서는 배경을 새로운 class로 추가하여, 모델에게 배경의 특징과 고양이의 특징을 구분하도록 학습시켰다. 그리고 최종 output vector의 크기가 1×5 가 되도록 모델을 수정하였다.

3.3. Initialization and regularization

He Uniform initialization을 사용하여(Section 8.3 참고)(He et al., 2015) 가중치의 초기화를 진행하였고, 모델에 사진을 입력하기 이전에 사진의 RGB값을 hyperparameter인 256으로 나누어 정규화를 진행하였다.

4. Details of Learning

4.1. 학습 개요

본 프로젝트에서는 모델 학습과 optimization에 있어서 SGD를 사용하였다. Batch size는 1로 정하여 하나의 데이터가 입력될 때마다 모델이 학습되도록 하였다. 앞으로의 서술에 있어 총 학습 횟수를 epoch로, 한 번의 학습과정을 iteration이라고 하겠다. Learning rate(ϵ)는 0.000015부터 시작하여, 총 10번에 걸쳐 감소시켜 학습 종료 시 0.000005가 된다.

본 프로젝트에서 정의한 문제가 multiclass 분류문제이며, 모델의 결과값이 활성화함수 softmax에 의해 확률로 나오는 점을 고려하여 손실함수로 cross entropy를 사용했다.

$$L = CE = - \sum_j y_j \log p_j \quad (y_j : \text{answer}, p_j : \text{output})$$

배경 학습의 효과를 확인하기 위하여, 배경 이미지를 제외한 Dataset으로 첫번째 모델을 학습시켰고, 두번째 모델에서는 배경 이미지를 추가한 데이터셋으로 학습을 진행했다. 각 모델에서 class당 2000번의 iteration을 하였고, 이에 epoch는 8000과 10000으로 설정했다. 학습은 google colaboratory의 gpu(Tesla K80 GPU)로 진행했다.

4.2. Back-propagation을 위한 수학적 기반 및 사전 정의

본 프로젝트의 모델들은 5층의 컨볼루션 층과 2층의 fc층 총 7개의 층으로 이루어지고, 이 결과가 softmax function으로 활성화 된다. 하지만 간결한 서술을 위해 마지막 활성화 단계를 softmax layer라고 하여, 총 8개의 층으로 간주하겠다. n번째 layer의 input, output, weight을 다음과 같이 표현하고,

$$input^n, output^n, W^n \quad (1 \leq n \leq 8)$$

weight와 input의 gradient를 다음과 같이 정의한다.

$$\begin{aligned} W_g^n &= \frac{\partial CE}{\partial W^n} \\ input_g^n &= \frac{\partial CE}{\partial input^n} \end{aligned}$$

또한 다음과 같은 함수를 정의한다. (Section 8.2 참고)

- $flip(W)$: W 를 180도 회전시킨 결과를 반환한다.
- $conv^{-1}(W, g, n)$: W 와 g 를 layer n 에 따라 각 단면에 대해 Frobenius inner product한 결과를 반환한다.
- $dup(g, i, n)$: g 의 i 번째 단면만을 잘라와 layer n 에 맞게 duplication 한 결과를 반환한다.
- $W \otimes g$: $conv^{-1}(W, dup(g, i, n), n)$

4.3. Back-propagation and Update

n 번째 layer에서 input의 gradient와 weight의 gradient는 chain rule에 의해 다음과 같이 정의된다.

$$\begin{aligned} input_g^n &= \frac{\partial CE}{\partial input^n} \\ &= \frac{\partial CE}{\partial input^{n+1}} \frac{\partial input^{n+1}}{\partial input^n} \\ &= \frac{\partial CE}{\partial input^{n+1}} \frac{\partial input^{n+1}}{\partial output^n} \frac{\partial output^n}{\partial input^n} \end{aligned} \quad (1)$$

$$\begin{aligned} W_g^n &= \frac{\partial CE}{\partial W^n} \\ &= \frac{\partial CE}{\partial input^{n+1}} \frac{\partial input^{n+1}}{\partial W^n} \\ &= \frac{\partial CE}{\partial input^{n+1}} \frac{\partial input^{n+1}}{\partial output^n} \frac{\partial output^n}{\partial W^n} \end{aligned} \quad (2)$$

여기서 $input^{n+1} = ReLU(output^n)$ 이므로, $\frac{\partial input^{n+1}}{\partial output^n}$ 은 $output^n$ 의 값에 따라 1 또는 0이다. 이는 $output^n$ 이 양수인 경우에만 $input^{n+1}$ 의 gradient가 전파됨을 의미한다. 몇몇 layer에서는 $output^n$ 과 $input^{n+1}$ 사이에 max-pooling이 이루어진 경우가 있는데, 이 때는 $output^n$ 이 최대값으로 선택된 경우에만 이에 대응하는 $input^{n+1}$ 의 gradient가 전파된다. 이처럼, $\frac{\partial input^{n+1}}{\partial output^n}$ 는 복잡한 연산 없이 단순히 값의 비교만을 통해 계산이 가능하므로, 모든

layer에서의 gradient 전파와 관련하여 이를 고려하였음을 명시하고 앞으로의 서술에서는 이에 대한 언급을 생략하도록 하겠다.

n 번째 layer에서, $output^n$ 은 $input^n$ 과 W^n 과의 연산에 의해 계산된다. 따라서, $\frac{\partial output^n}{\partial input^n}$ 과 $\frac{\partial output^n}{\partial W^n}$ 은 n 번째 layer에서 어떤 연산을 하느냐에 따라 달라진다. 이를 고려한 i 번째 iteration에서의 역전파 알고리즘은 다음과 같다.

Algorithm 1 Back-propagation algorithm

```

procedure BACKPRO(y)                                ▷ y : answer
    input_g_i^8 ← output_i^8 - y                  ▷ softmax layer
    for n = 7 → 6 do                            ▷ fully connected layer
        input_g_i^n ← (W_i^n)^T input_g_i^{n+1}
        W_g_i^n ← input_g_i^{n+1} (input_i^n)^T
    for n = 5 → 1 do                            ▷ convolution layer
        j ← num                                     ▷ number of filter
        input_g_i^n ← sum_j flip((W_j)_i^n) ⊗ input_g_i^{n+1}
        for k = 1 → j do
            (W_k)_i^n ← input_i^n ⊗ input_g_i^{n+1}
    return W_g_i                                     ▷ The total weight gradient

```

한 iteration에서 back-propagation이 끝나면, 다음의 update rule에 따라 weight와 bias를 갱신한다.

Algorithm 2 Update Rule for weight W

```

procedure UPDATE(W_g_i)
    g_i ← -(W_g_i) × ε                           ▷ gradient
    w_{i+1} ← w_i + g_i                          ▷ weight
    bias_{i+1} ← bias_i + g_i                     ▷ bias
    return

```

총 epoch번의 iteration이 이루어지면 학습을 종료한다.

5. Results

본 프로젝트는 모델의 평가를 위하여 두 Testset과 통계량을 다음과 같이 정의하였다.

- Testset 1: 배경사진이 포함되어 있는 Testset
- Testset 2: 배경사진이 포함되어 있지 않은 Testset
- P_1 : 모델을 Testset 1로 평가했을 때의 정답률
- P_2 : 모델을 Testset 2로 평가했을 때의 정답률
- P_3 : 모델을 Testset 2로 평가했을 때, 고양이사진을 배경이라고 답한 Test케이스를 제외한 경우의 정답률
- P_4 : 모델을 Testset 2로 평가했을 때, 고양이사진을 배경이라고 답한 경우, 해당 케이스를 단순히 제외하는 것이 아니라 배경 다음으로 높은 확률이 나오는 class를 답으로 취하였을 때 정답률

두 모델의 성능을 비교하는 데에 있어 parameter의 초기값의 영향을 최소화하기 위해, 두 모델을 같은 값으로 초기화한 후 모델 학습을 진행하였다. 초기값을 매번 다르게 설정하여 10번의 test를 진행한 결과는 다음과 같다(Table 1). 여기서 A는 배경을 고려하지 않은 모델, B는 배경 사진을 학습시킨 모델이다. A는 출력값으로 배경을 선택할 수 없기 때문에, P_2 가 모델의 정확도를 표현하는 유일한 통계량이다. 이에 반해 B는 P_1, P_2, P_3, P_4 네 통계량을 통해 모델의 정확도를 나타낼 수 있어, 어떤 통계량이 정확도를 대표할 수 있는지에 대한 논의가 필요하다.

Table 1. Test 결과: 10회의 Tests

	A, P_2	B, P_1	B, P_2	B, P_3	B, P_4
Test 1	0.5375	0.5600	0.5875	0.5950	0.6000
Test 2	0.5500	0.5300	0.5500	0.5789	0.5750
Test 3	0.5750	0.5300	0.5250	0.5915	0.5500
Test 4	0.5250	0.4800	0.5250	0.5455	0.5500
Test 5	0.4500	0.4700	0.5000	0.5405	0.5250
Test 6	0.5750	0.5100	0.5625	0.6164	0.5750
Test 7	0.5000	0.5000	0.4875	0.5270	0.5000
Test 8	0.5250	0.4800	0.4875	0.5652	0.5250
Test 9	0.6000	0.5700	0.5875	0.6438	0.6375
Test 10	0.4625	0.4500	0.4125	0.4783	0.4625
Avg.	0.5300	0.5080	0.5225	0.5682	0.5500

5.1. Analysis

A의 P_2 와 B의 P_1 을 기준으로 두 모델을 비교했을 때, A가 B보다 정답률이 높은 경우가 더 많은 것을 확인할 수 있다. 하지만 본 프로젝트의 Problem은 배경 사진에 대한 정확도를 요구하지 않는다. 따라서 이를 정답률 계산에서 배제할 수 있으며, 이것은 P_2 와 같다. A의 P_2 와, B의 P_2 를 비교했을 때 A가 극소하게 더 높은 정답률을 보였다.

하지만 여전히 P_2 에는 고양이 사진을 배경이라고 판단한 경우가 있어, 모델의 정답률을 낮추고 있다. 이런 경우를 배제하여 계산한 것이 P_3 이며, 모든 Test에서 B의 P_3 가 A의 P_2 보다 높은 것을 확인할 수 있다. 여기서 P_3 는 구별이 어려운 고양이 사진에 대해 판단을 보류했을 때의 정답률이라고 해석이 가능하다. 만약 모델의 결과값을 고양이들 중 하나로 특정지어야 하는 상황이라면 네 고양이 중 가장 확률이 높은 class를 선택해야 하고, 이에 대한 정확도는 P_4 와 같다. 이를 기준으로 A의 P_2 와 비교했을 때, 10번 중 9번의 test에서 B의 P_4 가 크거나 같았다.

본 프로젝트에서는 배경에 대한 정확도를 고려하지 않아도 되므로, 통계량 P_3 과 P_4 가 B의 정확도를 대표한다고 볼 수 있다. 이는 앞선 분석에 의해 모델 B가 모델 A보다 성능이 좋음을 의미한다.

5.2. Review

본 프로젝트는 모델의 결과값으로 배경 class를 추가하고, 배경 사진을 추가적으로 학습시킴으로써 그 성능을 향상 시킬 수 있음을 보여주었다. 특히, 서로 다른 두 고양이를

훈동하였을 때 받는 페널티가 매우 커 이러한 상황을 반드시 피해야 하는 경우, B의 P_3 값은 매우 유의미한 결과라고 생각한다.

본 프로젝트의 test 결과는 작은 batch size로 인해 발생하는 noise의 관측뿐이라는 의견이 있을 수 있다. 하지만 본 프로젝트는 여러 번의 test를 거쳐 모델을 검증하였고, 그 결과가 일관성을 보이기 때문에, 이에 대한 문제는 없다고 생각한다.

6. Future Direction

이번 프로젝트에서 시간 및 자원의 부족으로 적용하지 못한 개선 방안은 다음과 같다.

첫 번째는 모델 평가의 개선 방안으로, 배경을 지운 고양이 사진을 testset으로 사용하는 것이다. 본 프로젝트의 testset으로는 모델이 고양이의 특징을 학습하여 정답률이 올라간 것인지 배경의 특징을 학습하여 정답률이 올라간 것인지 확인할 수 없다. 이에 배경을 지운 고양이 사진으로 test하여 기존의 정답률과 비교하면 모델이 어떤 특징을 바탕으로 결과값을 도출하는지 해석할 수 있을 것이다.

두 번째는 더 많은 데이터를 수집하는 것이다. 본 프로젝트의 dataset과 testset은 총 1211장뿐이다. 만약 더 많은 데이터를 수집하여 dataset과 testset의 개수를 늘릴 수 있다면, 현재보다 더 정확한 학습을 시킬 수 있을 것이고, 모델의 평가 또한 정확해질 것이다.

세 번째 방안은 batch size를 늘리는 것이다. 본 프로젝트의 개발 환경에서는 최대 12시간동안만 GPU를 사용 가능했기 때문에, batch size가 1인 SGD를 사용할 수 밖에 없었다. 만약 이에 대한 제약이 없다면 batch size를 충분히 늘려 더 정확한 학습을 시킬 수 있을 것이다.

마지막으로는 배경 학습을 세분화하는 것이다. 본 프로젝트에서는 고양이의 서식지 사진을 배경이라는 하나의 class로 두고 학습을 진행하였다. 하지만 공통점이 전혀 없는 나무나 건물, 바닥 등을 하나의 class로 학습시키는 것은 효과적이지 못하다. 이에 배경이라는 class를 여러 개의 subclass로 세분화한다면 더 정확한 배경 학습이 가능할 것이다. 뿐만 아니라 학습 데이터에도 고양이 이름에 대한 label뿐만 아니라 배경에 대한 label을 추가해줌으로써 배경 학습을 통한 성능 개선 효과를 극대화 시킬 수 있을 것이다.

7. Roles

Table 2. 팀원 학번, 이름, 역할

학번	이름	역할
2018320222	고주연	모델 설계 및 보고서 편집 담당
2018320207	공인호	코딩 및 모델 학습 담당
2018320212	김상엽	역전파 계산 및 수식 담당
2018320226	염형빈	데이터 수집 및 편집
-	공통	보고서 작성

8. Appendix

8.1. 각 고양이 사진 및 활동 영역



Figure 1. 壬또, 생명대 서관



Figure 2. 벼찌, 과학도서관 뒤 및 공학관

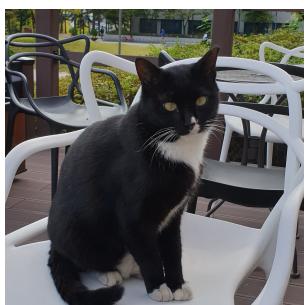
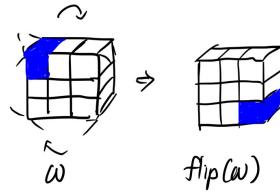
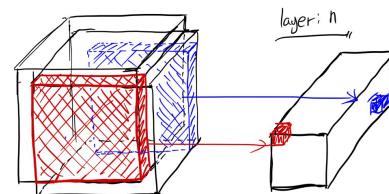
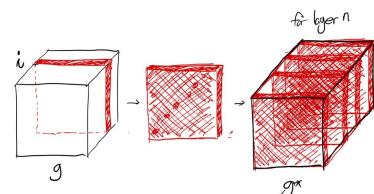
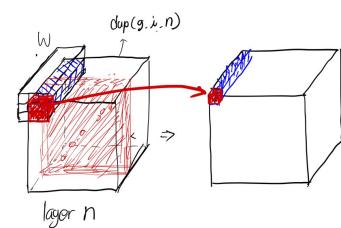


Figure 3. 찰리, 애기능 진리마루



Figure 4. 채플린, 애기능 진리마루

8.2. Section 4.2에서 정의한 함수들

Figure 5. $flip(W)$ Figure 6. $conv^{-1}(W, g, n)$ Figure 7. $dup(g, i, n)$ Figure 8. $W \otimes g$

8.3. He Uniform Initialization

$$W \sim \mathcal{U}\left(-\sqrt{\frac{6}{n_{in}}}, +\sqrt{\frac{6}{n_{in}}}\right)$$

n_{in} = 이전 layer(input)의 노드 수

8.4. AlexNet과 우리의 모델 크기 비교 표

Table 3. 모델 크기 비교: 첫번째 층 (convolutional layer)

AlexNet		우리의 모델
224*224*3	Input 크기	336*336*3
11*11*3	필터 크기	11*11*3
96	필터 개수	96
4	Stride	4
0	Zero padding	0
55*55*96	Resulted feature map size	81*81*96
ReLU	활성화 함수	ReLU
3*3, in stride 2	Max-pooling	3*3, in stride 2
27*27*96	최종 feature map	40*40*96

Table 4. 모델 크기 비교: 두번째 층 (convolutional layer)

AlexNet		우리의 모델
27*27*48	Input 크기	40*40*96
5*5*48	필터 크기	5*5*96
256	필터 개수	256
1	Stride	1
2	Zero padding	2
27*27*256	Resulted feature map size	40*40*256
ReLU	활성화 함수	ReLU
3*3, in stride 2	Max-pooling	3*3, in stride 2
13*13*256	최종 feature map	20*20*256

Table 5. 모델 크기 비교: 세번째 층 (convolutional layer)

AlexNet		우리의 모델
13*13*256	Input 크기	20*20*256
3*3*256	필터 크기	3*3*256
384	필터 개수	384
1	Stride	1
1	Zero padding	1
13*13*384	Resulted feature map size	20*20*384
ReLU	활성화 함수	ReLU
-	Max-pooling	-
13*13*384	최종 feature map	20*20*384

Table 6. 모델 크기 비교: 네번째 층 (convolutional layer)

AlexNet		우리의 모델
13*13*384	Input 크기	20*20*384
3*3*192	필터 크기	3*3*284
384	필터 개수	384
1	Stride	1
1	Zero padding	1
13*13*384	Resulted feature map size	20*20*384
ReLU	활성화 함수	ReLU
-	Max-pooling	-
13*13*384	최종 feature map	20*20*384

Table 7. 모델 크기 비교: 다섯번째 층 (convolutional layer)

AlexNet		우리의 모델
13*13*384	Input 크기	20*20*384
3*3*192	필터 크기	3*3*284
256	필터 개수	256
1	Stride	1
1	Zero padding	1
13*13*256	Resulted feature map size	20*20*256
ReLU	활성화 함수	ReLU
3*3, in stride 2	Max-pooling	3*3, in stride 2
6*6*256	최종 feature map	10*10*256

Table 8. 모델 크기 비교: 여섯번째 층 (fully connected layer)

AlexNet		우리의 모델
6*6*256	Input 크기	10*10*256
ReLU	활성화 함수	ReLU
1*4096	최종 output vector	1*4096

Table 9. 모델 크기 비교: 일곱번째 층 (fully connected layer)

AlexNet		우리의 모델
1*4096	Input 크기	1*4096
Softmax	활성화 함수	Softmax
1*1000	최종 output vector	1*4

8.5. Part of source code

```
// definition of Fdot3()
__global__
void Fdot(float* conv_filter, float* conv_result_g,
          float* conv_input_g, int i, int I, int J,
          int K, int A, int B, int C) {
    __shared__ float sdata[1024];
    unsigned int tid = threadIdx.x;
    unsigned int j = (blockIdx.x*K)%J;
    unsigned int k = blockIdx.x%K;
    sdata[tid] = 0.0;
    float init = 0.0;
    for(int a1=- (A/2); a1<= (A/2); a1++) {
        for(int b1=- (B/2); b1<= (B/2); b1++) {
            __syncthreads();
            if((i+a1)>=0 && (i+a1)<I && (j+b1)>=0 && (j+b1)<J) {
                init += conv_filter[B*K*C*((A/2)-a1)
                                    +K*C*((B/2)-b1)+C*k+tid]
                       *conv_result_g[J*C*(i+a1)
                                      +C*(j+b1)+tid];
            }
            __syncthreads();
        }
    }
    __syncthreads();
    sdata[tid] = init;
```

```

__syncthreads();
for(unsigned int s=blockDim.x/2;
    s>0; s>>=1) {
    if (tid < s) {
        sdata[tid] += sdata[tid + s];
    }
    __syncthreads();
}
__syncthreads();
if(tid==0) conv_input_g[i*J*K+blockIdx.x]
    = sdata[tid];
__syncthreads();
}

// implementation
void backpropagation(int ans, float *result){
...
for(int i=0; i<20; i++) {
    Fdot<<<7680,256>> (conv5_filter,
        conv5_result_g, conv5_input_g,
        i ,20,20,384, 3, 3, 256);
    cudaDeviceSynchronize();
}
...
}

```

Back-propagation 과정 중 5번째 convolution layer^o $input_g^5$ 를 계산하는 코드이다. 각 layer에 대해 Forward 와 Back-propagation을 위한 위와 같은 함수를 정의하였다.

References

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV ’15, pp. 1026–1034, Washington, DC, USA, 2015. IEEE Computer Society. ISBN 978-1-4673-8391-2. doi: 10.1109/ICCV.2015.123. URL <http://dx.doi.org/10.1109/ICCV.2015.123>.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. Curran Associates, Inc., 2012. URL <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>.

Xie, Q., Hovy, E., Luong, M.-T., and Le, Q. V. Self-training with noisy student improves imagenet classification, 2019.