제 129회 석사학위 논문
지도교수 강 훈

이미지 컨텐츠 검색을 위한 CNN의 구조

A Structure of Convolutional Neural Networks
for Image Contents Search

중앙대학교 대학원
전자전기공학부 제어 및 시스템 전공
이 현 수
2018년 8월

이미지 컨텐츠 검색을 위한 CNN의 구조

A Structure of Convolutional Neural Networks
for Image Contents Search

이 논문을 석사학위논문으로 제출함

20018년 8월

중앙대학교 대학원
전자전기공학부 제어 및 시스템 전공
이 현 수

이현수의 석사학위 논문으로 인정함

심 사 위 원 장 전 홍 태 ㊞

심 사 위 원 이 홍 기 ㊞

심 사 위 원 강 훈 ㊞

중앙대학교 대학원

2018년 8월

목 차

그 림 / 표 목 차

제 1 장. 서 론

최근 몇 년 동안 Hinton 교수의 'Deep Belief Network'[1] 논문이 발표된 이후에 4차 산업혁명의 AI분야라고 할 수 있는 딥 러닝(Deep Learning)[2]이 많은 분야에서 다양한 알고리즘과 함께 개발되어왔다. 이미지 인식, 음성 인식, 문장 예측 등의 다양한 분야에서 딥 러닝을 사용하지 않은 곳이 없다. 그 중 딥 러닝 연구 대상의 기본이라고 할 수 있는 이미지에 대해 본 논문은 다루려고 한다.

현재 이미지 인식(Recognition) 및 분류(Classification) 분야에서 가장 각광을 받고 있는 알고리즘은 LeCun 교수가 오래전에 고안했던 딥 러닝의 일종인 CNN(Convolution Neural Network)이다.[3] CNN을 통하여 이미지 특징 인식 및 분류가 엄청난 발전을 이루어왔고, 컴퓨터 비전 영역에 있어 엄청난 퍼포먼스를 보여주고 있다.

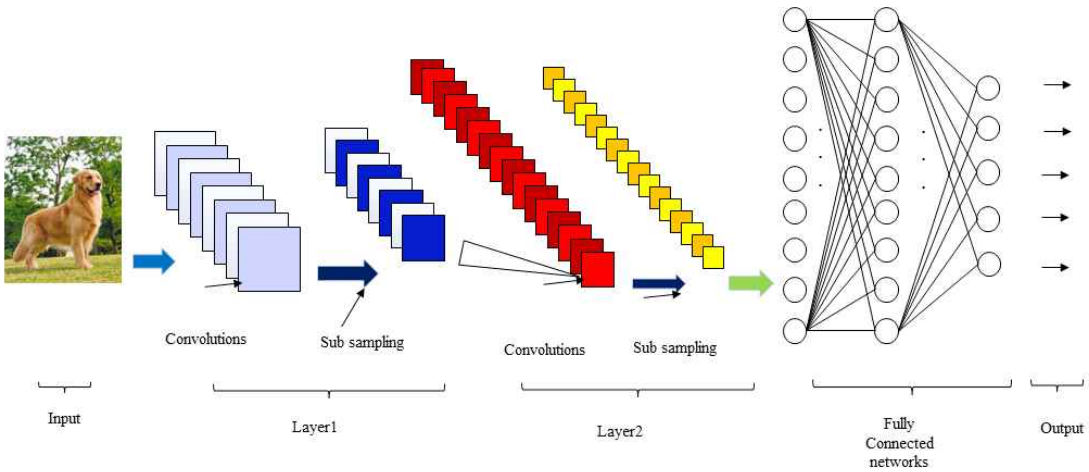


Fig. 1.1 Convolution Neural Network의 일반적인 구조

CNN의 기본 구조는 위의 Fig.1.1과 같이 이미지를 Convolution Layer의 필터와 Pooling Layer를 반복적인 통과를 하여 기존의 Fully Connected Network단인 Multi-Layer Perceptron과 Softmax 알고리즘을 통해 이미지를 분류 해준다. 이미 Google에서 제공하는 Tensorflow[4]나 Keras[5]

를 통하여 개발자가 아닌 개인들도 딥 러닝과 CNN을 이용한 이미지 인식과 이미지 분류를 오픈된 알고리즘을 통해 실험할 수가 있다.

지금까지 CNN은 깊은 층을 통하여 이미지의 복잡한 특징 중 중요한 부분을 특출해 내는 것을 바탕으로 정확도를 높여 나아갔다. 본 논문에서는 보다 깊지 않은 상태에서 많은 개수의 필터를 사용하였다. 또한 현재까지 나온 이미지 대상의 딥 러닝의 여러 알고리즘을 합쳐서 최고 수준의 정확도는 아니지만, 어느 정도 이상의 높은 정확도를 보임으로써 매우 깊지 않은 층을 가지고도 많은 개수의 필터로 정확도가 높을 수 있다는 것을 확인하였다.

본 논문의 CNN의 Convolution Layer는 6개의 층이 있으며 Pooling Layer는 3개의 층 그리고 Multi-Layer Perceptron[6]의 일종인 Fully Connected Layer를 통하여 Output 값을 계산해주며 이를 Backpropagation[7] 과정을 통하여 학습을 시켜주었고, 필터의 개수를 기존의 CNN 논문들과 달리 처음 Layer부터 많이 쌓아갔다. 이 과정을 통하여 준수한 결과를 내였다.

제 2 장 :영상 컨텐츠 검색을 위한 CNN의구조

2.1 Convolution Layer

다음 절에서는 CNN의 Layer 중 Convolutional Layer에 대해서 살펴보도록 하겠다. Convolutional Layer은 먼저 입력인 이미지의 값에 필터(다른 말로 커널이라고 한다.)를 씌워 이를 통해 출력을 계산한 값을 보여준다. 먼저 Convolution의 계산식을 통해 이미지가 어떻게 처리 되는지를 보려고 한다. 2D 이미지의 Convolution식은 다음과 같다.[8]

$$C(x,y) = \sum_{a=0}^{n-1}\sum_{b=0}^{n-1} I(x,y)\, W(a-x,b-y) \tag{2.1}$$

$C(x,y)$는 입력된 이미지$I(x,y)$와 커널 필터$W(x,y)$의 Convolution 값이다. Convolution 1D 일 때 식이

$$C(x) = \sum_{a=o}^{n-1} I(x)w(a-x) \tag{2.2}$$

위와 같이 한쪽 함수를 역전시켜 Convolution을 해주는 것을 생각해 볼 때, 실제적으로 2D 이미지에서의 Convolution은 이미지 필터 값인 $W(x,y)$를 180° 회전시켜 곱해 줌을 알 수가 있다. 하지만 우리가 CNN에서 Convolution 필터를 씌웠을 때 다음과 같은 예시의 과정으로 설명한다.

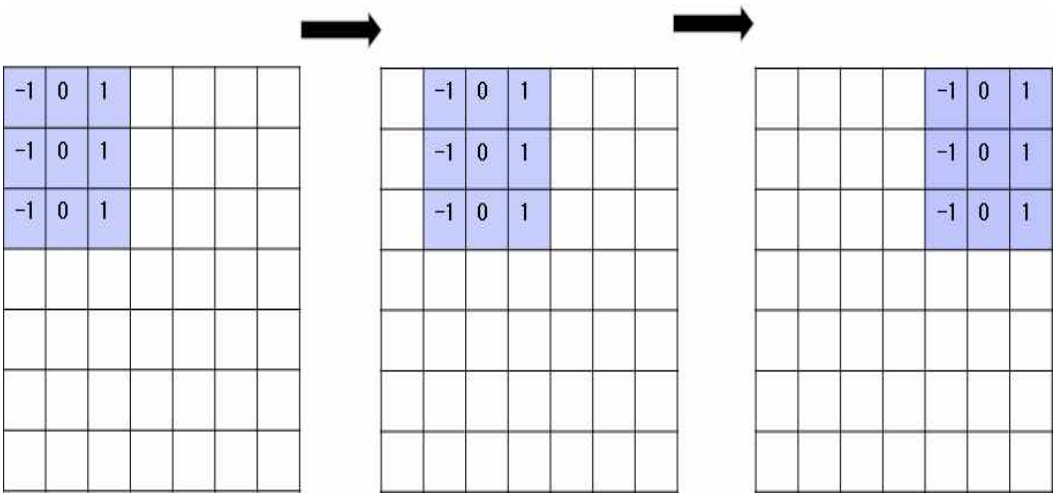


Fig. 2.1 Convolution 필터 계산 과정의 예시

Fig. 2.1과 같이 7X7이미지에 3X3필터를 씌웠을 때 Sobel 필터를 씌웠을 때 180° 회전된 상태에서 계산하는 것이 아니라 정 방향으로 계산을 한다. 이 과정을 Cross-Correlation 과정이다. Cross-Correlation의 식은 다음과 같다.[9]

$$C(x,y) = \sum_{a=0}^{n-1} \sum_{b=0}^{n-1} I(x,y)\, W(a+x, b+y) \tag{2.3}$$

식(2.1)과 식(2.3)의 다른 점은 W안의 x,y값의 부호의 차이이다. 이를 이미지를 두고 생각해보면 이미지가 180° 회전된 것 이다. 그런데 실제로 결과가 다른지 계산해보면 두 계산의 이미지 결과 값이 같게 나와 기존의 Convolution 필터를 씌우는 과정을 설명할 때 결과가 같고 좀 더 보기 쉬운 Cross-Correlation 과정으로 설명을 한다.

다음은 Convolution 필터를 통과한 이미지의 크기가 줄어드는 과정이다.

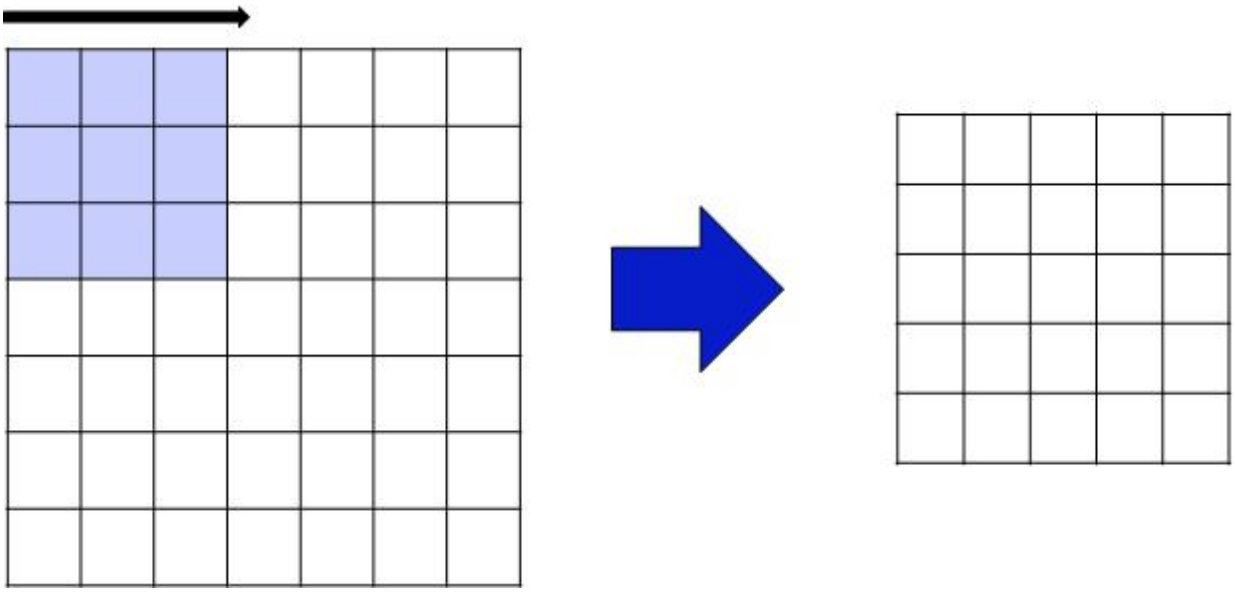


Fig. 2.2 이미지에 Convolution 필터 씌움으로 크기가 주는 과정

위의 Fig. 2.2은 7X7 이미지를 3X3 필터를 Convolution을 한 결과이다. Fig. 2.1처럼 필터를 한 칸씩 이동하는 것을 Stride 값이 1이라고 한다. 이 과정을 걸치면 Fig. 2.2의 결과와 같이 이미지의 사이즈가 줄어든다. Stride가 클수록 사이즈가 더 줄어드는 것을 확인할 수 있겠다. Convolution필터를 씌울수록 이미지의 사이즈가 줄어들기 때문에 Padding이라는 기법을 통해 원래 이미지가 줄어들지 않게 한다. Padding은 다음 Fig. 2.3과 같이 이미지 주변에 값이 0인 픽셀들을 주위에 붙여준다.

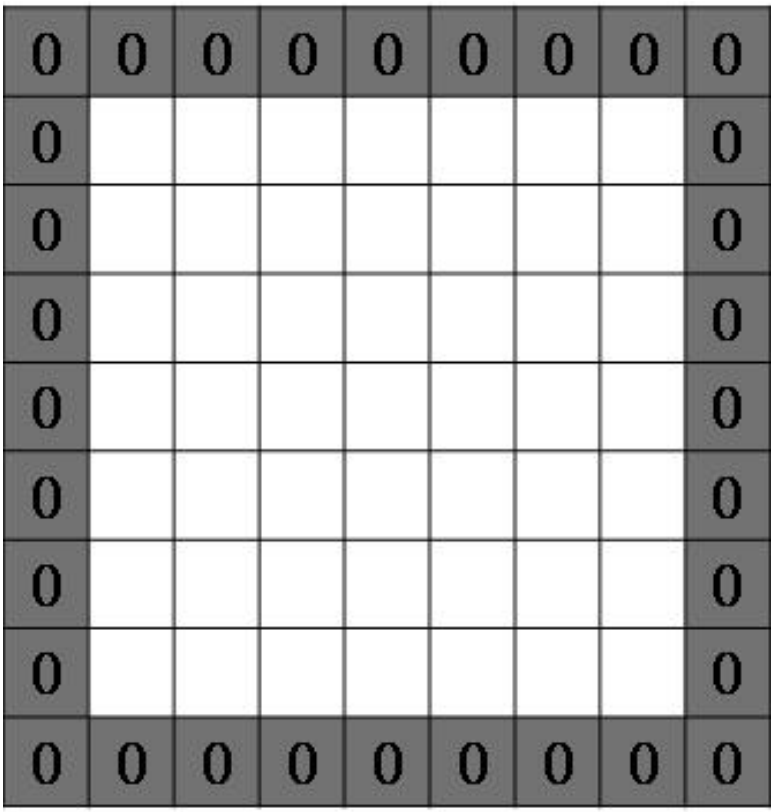


Fig. 2.3 Zero Padding의 예시

Padding을 씌우면 원래 이미지보다 사이즈는 커지고 원래 이미지의 특징 값들은 그대로 뽑아낼 수 있다. 또, 이미지의 사이즈는 줄어들지 않기 때문에 Convolution Layer에서 중요한 역할을 한다. Padding의 유무, Stride의 개수, 필터의 크기에 의해 이미지의 사이즈를 다음과 같이 계산할 수 있다.

$$\frac{I-W+2\times P}{S}+1 \tag{2.4}$$

I는 이미지의 크기, W는 필터의 크기, P는 Padding의 유(1), 무(0)의 값을 대입, 그리고 S는 Stride의 크기이다. Fig. 2.2의 과정을 식(2.4)에 넣어 계산하면

$$\frac{7-3+2\times 0}{1}+1=5$$

으로 Convolution 필터를 통과한 뒤의 이미지 사이즈를 쉽게 구할 수 가

있다. 지금까지 일련의 과정을 통하여 이미지 픽셀 값들 중에서 중요한 값들을 추출하는 것이 Convolution Layer의 목적이다.[10]

2.2 Pooling Layer

Convolutional Layer를 통해 추출된 특징들을 분류하는 과정을 하는 Layer를 Pooling Layer라고 할 수 있다. Pooling Layer은 Max Pooling, Average Pooling, Min Pooling, Fraction Pooling 등등이 있다. 다음은 Pooling 과정의 예시들이다.[11]

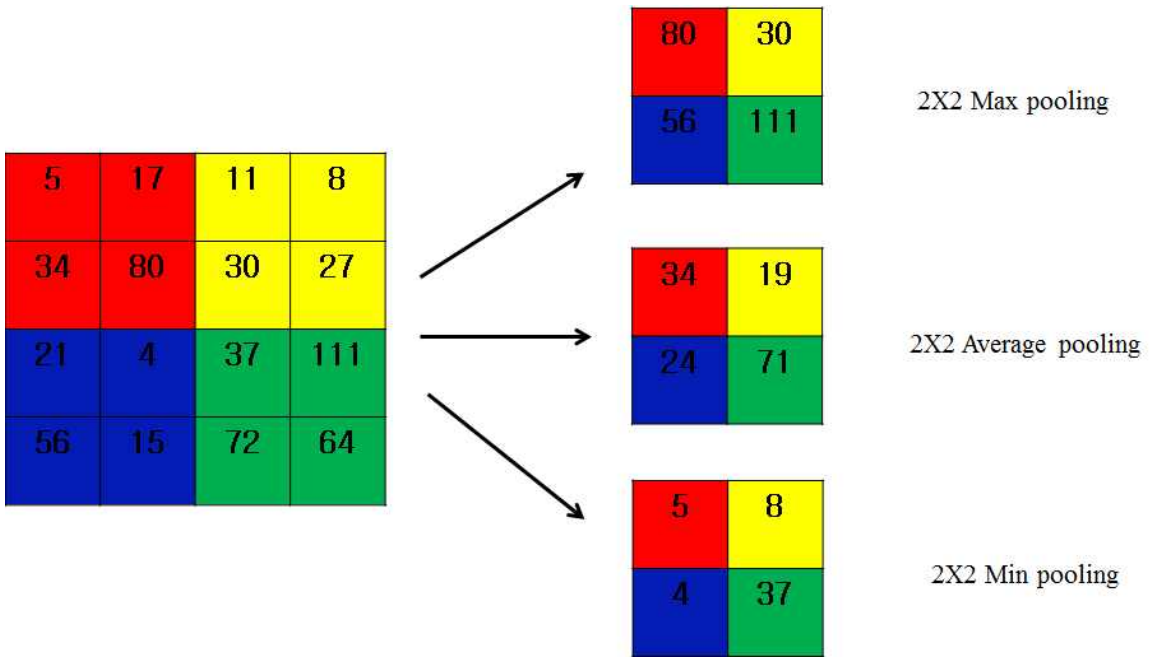


Fig. 2.4 여러 가지 Pooling의 예시

위의 그림은 2x2 필터를 Stride2로 한 Pooling들이다. 이 중에 주로 Max Pooling을 많이 사용하고 Subsampling이라고도 불린다. Pooling을 통하여 Convolution 필터를 통해 얻은 이미지의 특징들 중 더 중요한 값을 추출해 낼 수 있다. 하지만, 필터를 통해 축소된 이미지의 픽셀을 반으로 줄여 깊은 층의 학습을 시키고 싶을 때, Pooling을 씌우는 것이 작은

크기의 이미지로는 깊은 층의 학습을 시킬 수가 없을 수가 있다.

이를 보완하기위해 Graham, Benjamin이 제안한 "Fractional Max-Pooling"이란 알고리즘을 통해 기존의 Pooling의 정수 배로 이미지의 크기가 주는 것과 달리 실수 배로 이미지의 크기가 줄어들기 때문에 Pooling Layer를 통과하더라도 이미지의 크기가 천천히 줄어들 수가 있다. 이를 통해 아무리 작은 크기의 이미지라도 깊은 층의 CNN을 학습시킬 수 있다.[12]

2.3 Activation Functions

신경회로망에서 중요한 역할을 하는 것 중에 하나는 활성화 함수(Activation Function)이다. 이 함수는 신경회로망에서 각각의 뉴런들에 들어오는 입력신호의 합을 출력신호로 변환시키는 함수이다. 활성화 함수에는 선형함수와 비선형 함수가 있다. 이 절에서는 활성화 함수의 종류와 그 쓰임새를 보겠다. [13]

2.3.1 Sigmoid Function

Sigmoid 함수는 Logistic 함수라고도 불린다. 수식은 다음과 같다.

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.5}$$

$$\sigma'(x) = \sigma(x)(1-\sigma(x)) \tag{2.6}$$

Sigmoid 함수는 신경회로망에서 많이 쓰이는 함수이다.

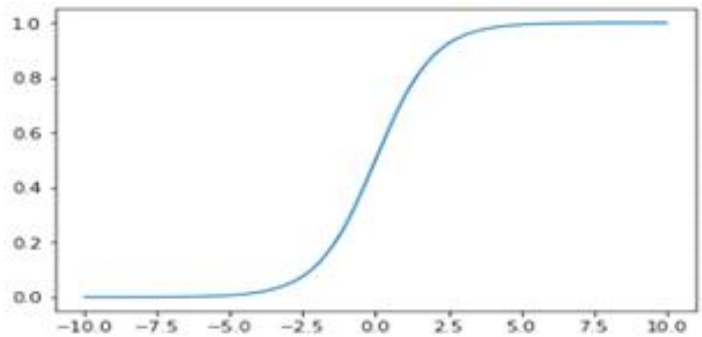


Fig. 2.5 Sigmoid 함수의 그래프

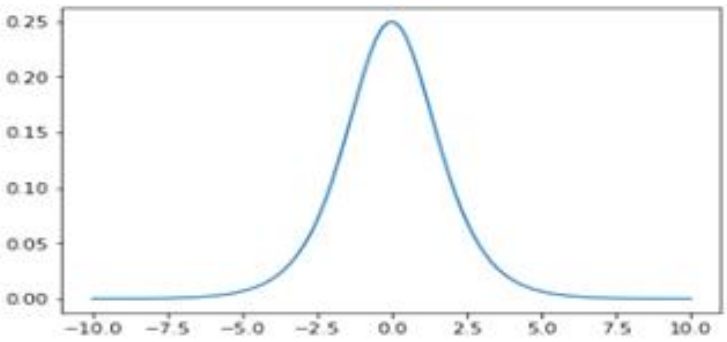


Fig. 2.6 Sigmoid 미분 함수의 그래프

하지만 어느 정도 숫자이상이거나 이하일 경우 Backpropagation을 할 때, gradient 값이 작아지는 현상이 나타나 학습률이 떨어지고, Fig.2.6 과 같이 미분의 수치가 작아 계산이 복잡하게 되어 학습이 느려지고 Global Minimum으로 수렴을 하기가 어려워 질수 가 있다.[14] 이를 Gradient Vanishing 현상이라고 한다.[15] 이로 인해 깊은 층의 신경회로망을 만들 경우 Sigmoid만으로 형성하기 어렵다. 현재는 Softmax 같은 분류과정에서 사용되거나 적은 층의 Multi-Layer Perceptron에 사용된다.

2.3.2 Hyperbolic Tangent Function

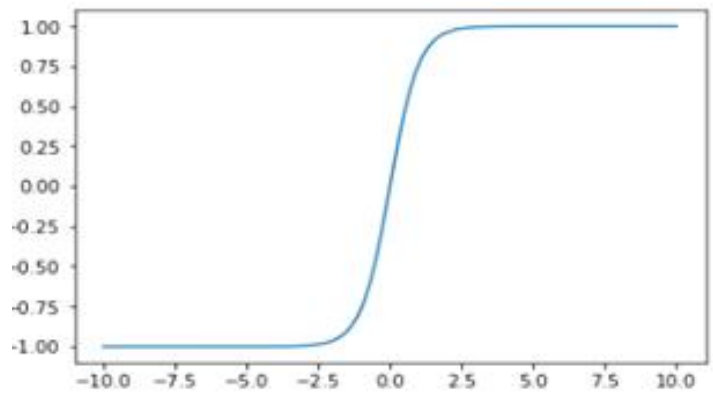


Fig. 2.7 Hyperbolic Tangent Function의 그래프

Hyperbolic Tangent Function은 Fig. 2.7과 같이 Sigmoid 함수에서 크기와 위치가 조절되었다고 볼 수 있다. 수식과 미분식은 다음과 같다.

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.7}$$

$$\tanh'(x) = 1 - \tanh^{2}(x) \tag{2.8}$$

Hyperbolic Tangent Function은 Sigmoid 함수 보다 Activation Function으로 쓰였을 때 학습 수렴 속도가 더 빠르다는 장점을 가지고 있다. 그러나 여전히 Sigmoid 함수가 갖는 단점을 가지고 있다.

2.3.3 Rectified Linear Unit(ReLU)

ReLU 함수는 최근에 신경회로망에서 가장 많이 쓰이는 활성화 함수 이다. 밑의 Fig. 2.8과 같이 음의 값에서는 0 양의 값에서는 x값을 갖는다. 다음은 ReLU 함수의 수식이다.[16]

$$ReLU(x) = \max(0, x) \tag{2.9}$$

ReLU 함수는 Fully Connected Layer에서 뿐만 아니라 Convolution Layer에서 사용된다. Convolution Layer에서 각각의 픽셀들을 뉴런들로 볼 때 이 뉴런들을 ReLU라는 함수를 통과 시켜준다.

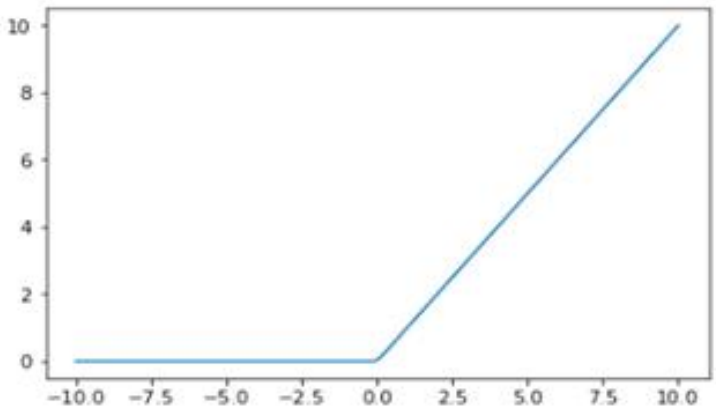


Fig. 2.8 Rectified Linear Unit의 그래프

ReLU 함수의 장점은 Gradient Vanishing 현상이 없다는 것과 계산이 매우 효율적이며 수렴하는 속도가 Sigmoid함수에 비해 약 6배 정도 빠르다는 것이다. 하지만 중심 값이 0이 아니라는 점이다. 그리고 음의 값을 항

상 0으로 만들기 때문에 음의 값을 갖는 뉴런은 업데이트가 안 될 가능성도 있다. 하지만 칼라 이미지는 거의 양의 값을 가지기 때문에 이 단점이 큰 문제가 되지는 않는다. 또, 지금까지의 활성화함수의 단점인 파라미터 수치들이 업데이트가 안 된다는 장점을 공통으로 가지고 있다. 이를 보완하기 위해 Leaky ReLU라 하여 ReLU의 변형 함수와 마찬가지로 ReLU에서 변현된 ELU라는 가장 최근의 활성화 함수가 있다.[17][18]

2.4 CNN의 Backpropagation Learning

위의 장의 모든 과정을 걸친 다음 Fully Connected Layer을 통과 하면 일련의 Feed Forward 계산 과정을 마치게 된다. 다음으로는 모든 신경회로망에서 가장 중요한 Backpropagation 과정이다. CNN의 Backpropgation과정은 기존 Neural Network인 Multi-Layer Perceptron 보다 복잡하다. CNN의 Backpropagation 과정 중 Fully Connected Layer은 기존의 Backpropagtion 과정을 걸친다. 다음은 기존의 신경회로망에서 노드의 함수를 통과 했을 때의 식과 에너지 식이다.

$$y_{i,j} = \sum_a \sum_b w_{a,b} x_{i+a,j+b} + b \tag{2.10}$$

$$E = \frac{1}{2} \sum_i \sum_j (\overline{y^d_{1+i,1+j}} - \overline{y_{1+i,1+j}})^2 \tag{2.11}$$

w능 웨이트의 값 b는 bias, $\bar{y}$는 활성화 함수를 통과한 y이다.

이를 바탕으로Pooling Layer와 Convolution Layer가 각각 어떻게 Backpropagtion을 하는지 살펴보도록 한다.

2.4.1 Pooling layer의 Backpropagation Learning

먼저 Backpropagation의 과정은 원래 과정을 역으로 하기 때문에 Pooling Layer를 살펴보도록 하겠다. 본 논문에서는 Max-Pooling을 사용

하기 때문에 Max-Pooling과정을 보려한다. Max-Pooling은 필터를 통과한 각 영역들의 최댓값을 통과하기 때문에 Backpropagation 과정을 통해 흘러 들어온 Gradient 값들이 최댓값이면 1을 곱하고 아니면 0을 곱한다. 그럼으로 Max-Pooling의 Backpropagation과정은 다음 Fig. 2.9과 같이 간단한 과정을 거친다.[19]

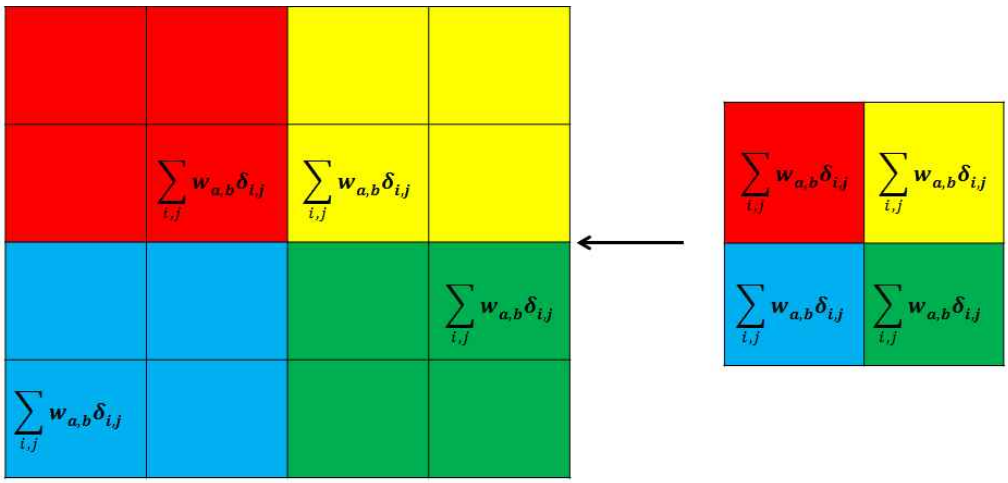


Fig. 2.9 Max-Pooling의 Backpropagation 과정

Fig. 2.9에 보이는 것과 같이 흘러들어오는 Gradient 값이 원래의 픽셀의 최댓값 자리로 들어가는 것을 보게 된다.

2.4.2 Convolution layer의 Backpropagation Learning

Convolution Layer의 Backpropagation을 이해하기 위해서는 먼저 Convolution 필터를 씌울 때 각 픽셀을 일종의 노드로 본다. 이해를 돕기 위해 다음 Fig2.10의 간단 예시가 있다.

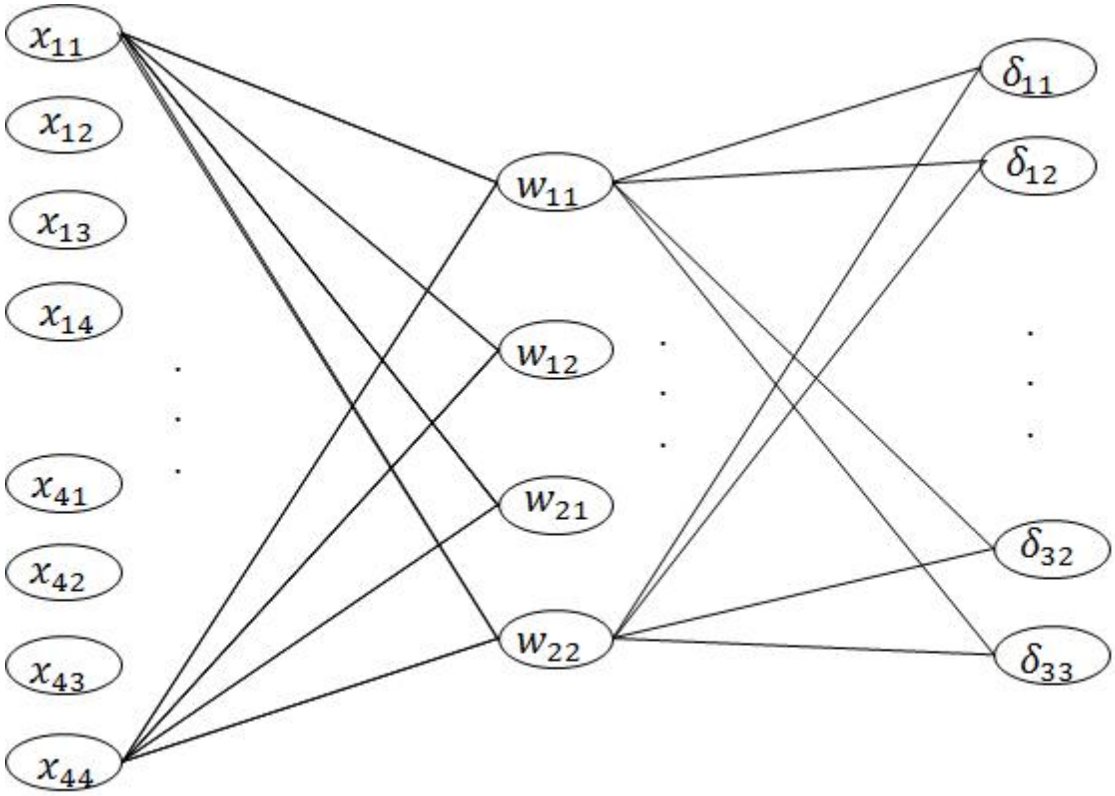


Fig. 2.10 Convolution의 Backpropagation 과정

위의 Fig. 2.10 은 4x4에 2x2 필터를 씌웠을 때 역으로 Gradient인 δ가 들어왔을 때를 이미지를 노드로 정렬했을 때 그림이다. 위의 그림과 변형시켜 기존의 신경회로망처럼 Backpropagation을 계산해주면 다음 식과 같이 $\triangle w \triangle b$를 식(2.10)과 (2.11)을 통해서 구하면 다음과 계산을 할 수가 있다.

$$-\mu\frac{\partial E}{\partial w_{a,b}} = \mu\sum_i\sum_j \delta_{i,j}\frac{\partial \overline{y_{i,j}}}{\partial w_{a,b}} \tag{2.12}$$

$$= \mu\sum_i\sum_j \delta_{i,j}\frac{\partial ReLU(y_{i,j})}{\partial y_{i,j}}\frac{\partial y_{i,j}}{\partial w_{a,b}}$$

$$\therefore -\mu\frac{\partial E}{\partial w_{a,b}} = \mu\sum_i\sum_j \delta_{i,j}\frac{\partial ReLU(y_{i,j})}{\partial y_{i,j}} x_{i+a,j+b} \tag{2.13}$$

$$\triangle w_{a,b} = \mu\sum_i\sum_j \frac{\partial ReLU(y_{i,j})}{\partial y_{i,j}}\delta_{i,j} x_{i+a,j+b} \tag{2.14}$$

$$\triangle b = \mu \sum_i \sum_j \frac{\partial ReLU(y_{i,j})}{\partial y_{i,j}} \delta_{i,j} \tag{2.15}$$

위의 식(2.12)과정을 통해 식(2.13),(2.14)(2.15)의 결과를 얻을 수가 있다. 위의 식에서 보여주듯이 Convolution Layer의 Backpropagation 과정에서 가장 중요한 것은 Notation이다. 밑의 첨자들이 이미지와 필터의 행과 열의 위치를 알려주기 때문에 이것을 통해 이미지와 필터의 크기도 알 수가 있고 각 노드, 즉, 픽셀의 위치도 알 수가 있다.

제 3 장 : CNN 실험

3.1 실험의 컴퓨터 환경

본 연구에서는 학습을 위해 아래 스펙의 컴퓨터를 사용하였다.

CPU: Intel i7-8700 3.2GHz
RAM: 16GB
GPU: GTX 1060i 6G
SSD: 512GB

RAM 과 GPU의 성능을 높여 학습의 계산량과 속도를 높였다.
여기에 사용된 OS 는 Window7 Ultimate K이고, Window 환경에서 가상 Linux 환경인 Anaconda Prompt를 사용하였다.[20] 가상환경에서 Python언어를 사용 하였고, Keras, Tensorflow, 그리고 Python에 사용되는 여러 라이브러리 툴을 통하여 실험에 사용될 코딩을 하였다. Tensorflow는 딥러닝을 쉽게 코딩 할 수 있게 하는 유용한 툴이다. Keras는 Tensorflow같은 여러 라이브러리의 상위 단계에 속한 것으로 Tensorflow외에 여러 툴을 여러 가지를 사용할 수 있게 도와준다. 또한, Tensorflow에서 제공 되는 Tensorboard를 통하여 사용자가 설계한 신경회로망과 각각 layer의 loss 및 이미지의 모양을 볼 수가 있다. 지금까지의 환경을 통하여 본 연구에서 원하는 CNN을 설계할 수 있게 되었다.

3.2 Benchmark Database

본 연구는 기존의 Benchmark Database을 대상으로 실험을 하였다.

Fig. 3.1 MNIST Benchmark Database 예시

먼저 설계한 CNN의 코드가 동작하는지 보기위해 비교적 쉬운 대상인 MNIST Benchmark Database을 대상으로 먼저 실험해보았다.[21] MNIST는 28x28 크기의 흑백 이미지이고, 6만장의 학습데이터와 1만장의 검증데이터로 구성되어있다. 실험을 통해 학습데이터는 99.9%일 정확도를 보였고, 검증데이터는 99.7% 정확도를 보였다.

다음은 본 연구의 메인 실험이라고 할 수 있는 실험데이터인 Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton의해 수집 되어 토론토대학에서 제공한 CIFAR-10 Benchmark Database이다.[22] CIFAR-10은 비행기, 자동차, 새, 고양이, 사슴, 개, 개구리, 말, 배, 트러 등의 10개 클라스로 되어 있는 Benchmark Database이다. 각 클래스 당 5천장의 학습데이터와 1천장의 검증데이터로 구성되어, 총 5만장의 학습데이터와 1만장의 검증데이터로 구성되어있다. CIFAR-10는 32x32 크기의 칼라 이미지이다.

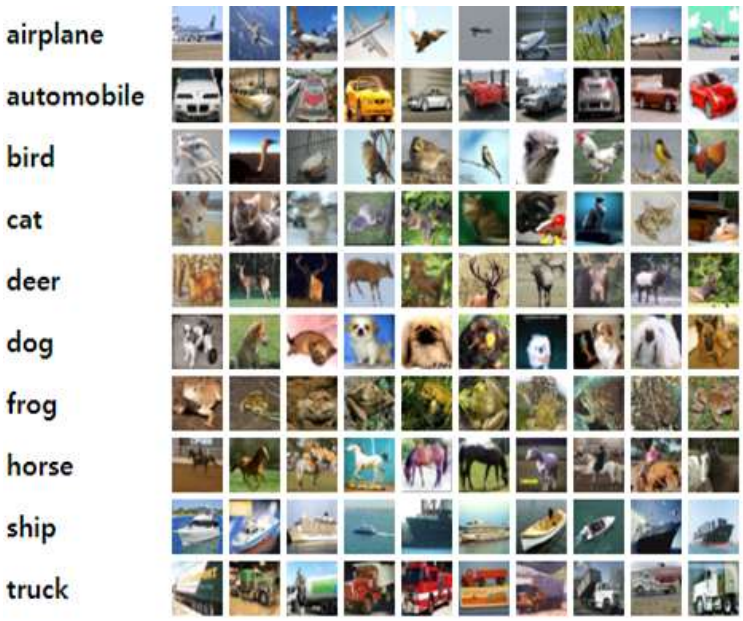


Fig. 3.2 CIFAR-10 Benchmark Database 예시

Fig. 3.3 CIFAR-100 Benchmark Database 예시

또 다른 대상으로는 같은 토론토 대학에서 제공한 CIFAR-100 Benchmark Databaset이다.[23] CIFAR-100의 이미지 특징은 CIFAR-10과 같다. CIFAR-100은 상위 20 클래스 밑에 각각 5개의 클래스가 있다. 클래스의 구성은 다음과 같다.

상위 클래스	**클래스**
수생 포유 동물	비버, 돌고래, 수달, 물개, 고래
물고기	수족관 물고기, 넙치, 레이, 상어, 송어
꽃	난초, 양귀비, 장미, 해바라기, 튤립
식품 용기	병, 그릇, 캔, 컵, 접시
과일 및 야채	사과, 버섯, 오렌지,배, 달콤한 고추
가정용 전기 장치	시계, 컴퓨터 키보드, 램프, 전화, 텔레비전
가정용 가구	침대, 의자, 소파, 테이블, 옷장
곤충	꿀벌, 딱정벌레, 나비, 애벌레, 바퀴벌레
대형 육식 동물	곰, 표범, 사자, 호랑이, 늑대
큰 인공 옥외 것들	다리, 성, 집, 도로, 초고층 빌딩
대형 자연 야외 경관	구름, 삼림, 산, 평야, 바다
대형 잡식 동물과 초식 동물	낙타, 소, 침팬지, 코끼리, 캥거루
중형 포유류	여우, 고슴도치, 좀도둑, 너구리, 스컹크
무 곤충 무척추 동물	게, 가재, 달팽이, 거미, 웜
사람	아기, 소년, 여자, 남자, 여자
파충류	악어, 공룡, 도마뱀, 뱀, 거북
작은 포유류	햄스터, 마우스, 토끼, 말다툼, 다람쥐
나무	단풍 나무, 참나무, 종려 나무, 소나무, 버드 나무
탈 것 1	자전거, 버스, 오토바이, 픽업 트럭, 기차
탈 것 2	잔디 깎는 기계, 로켓, 전차, 탱크, 트랙터

Table.3.1 CIFAR-100의 클래스 구성

각 클래스 당 500개의 학습 이미지 100개의 검증 이미지로 구성되어있다.

Fig. 3.4 Caltech 101 Benchmark Database 예시

마지막 실험대상인 Caltech 101 Benchmark Database이다.[24] Caltech 101은 101개의 클래스로 되어있으며 클래스마다 이미지의 개수와 사이즈가 다르다. 평균 40~60개의 이미지가 있으며 많은 것은 800개가 있다. 크기도 다르기 때문에 실험할 때 모든 이미지사이즈를 128x128로 맞춰주어 실험을 하였다.

3.3 다중 필터, 소수 층 구조의 CNN 설계

다음은 본 논문에서 설계하여 주로 사용한 CNN의 구조이며, CIFAR-10의 32x32 이미지를 예시로 하였다.

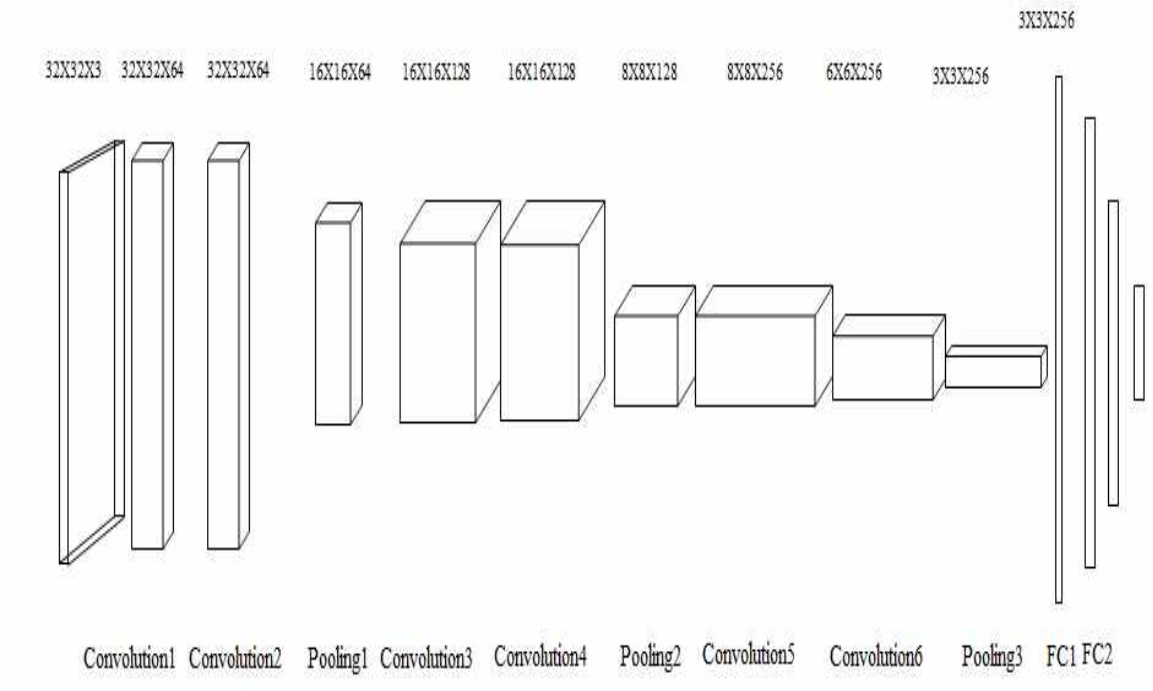


Fig. 3.5 설계한 CNN의 구조

이미지인 32x32x3(RGB칼라이미지이기 때문에 이미지가 3장으로 본다.)에 Padding을 하여 3x3 필터를 64개 Convolution1 Layer에 통과시킨다. 식 (2.4)에 대입해보면 이미지 사이즈가 줄지 않았다는 것을 볼 수가 있다. 같은 필터의 크기와 개수를 가지는 Convolution2 Layer를 Padding을 한 상태에 통과 시켜 32x32x64개가 된다. 이를 Pooling1 Layer를 통과 시켜 이미지 크기가 16x16으로 줄고 개수는 64개로 같다. 다음 Convolution3 Layer도 이전의 Convolution Layer들과 같이 이미지 사이즈가 줄지 않게 하여 128개의 필터를 통과하는 것으로 구성된다. Convolution4 layer의 구조도 Convolution3 Layer와 같다. Pooling2 Layer를 통과하여 또 다시 이미지 사이즈가 반으로 줄어 8x8로 된다. Convolution5 Layer도 Padding을 씌운 상태에서 3x3 필터를 통과시키기 때문에 이미지 사이즈가

줄지 않고 필터의 개수만 256개로 늘어난다. Convolution6 Layer에서는 Padding을 씌우지 않고 3x3 필터를 통과시키기 때문에 이미지 크기가 6x6으로 줄어들며 개수는 256개로 같다. 이를 Pooling3 Layer를 통과 시켜 최종으로 Convolution Layer들을 통과 한 어떤 미지의 특징이 되며 3x3x256의 크기를 가진다. 이것을 Fully Connected Layer에 일렬로 만들어 넣으면 1X2304의 노드를 가진다. 마지막 Output단으로 가기 전까지 Hidden Layer가 두 개이며 각각의 노드의 개수는 1024, 512개 이다. 이 모든 노드를 통과해 마지막 Output단에서 클래스 개수에 따라 Softmax로 이미지를 분류한다.[25]

Input	(3,32,32)
Conv1	(64,32,32)
Conv2	(64,32,32)
Pool1	(64,16,16)
Conv3	(128,16,16)
Conv4	(128,16,16)
Pool2	(128,8,8)
Conv5	(256,8,8)
Conv6	(256,6,6)
Pool3	(256,3,3)
FC1	2304
FC2	1024
FC3	512
Output	10

Table.3.2 설계한 CNN의 Output shape

Table. 3.2는 설계한 CNN의 이미지의 모형과 필터를 통과한 뒤의 개수 그리고 Fully Connected단의 개수를 보여준다. Table. 3.2에서와 같이 본 논문에서는 기존과 다르게 Layer의 개수를 줄이고 필터의 개수를 늘려서 결과적으로 마지막 Fully Connected 단의 파라미터의 개수를 늘렸다.

다음은 실험에서 사용된 코드의 일부이다.

```
model = Sequential()

model.add(Conv2D(64, (3, 3), input_shape=(3, 32, 32),
activation='relu', padding='same'))
model.add(Dropout(0.2))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(Dropout(0.2))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(Dropout(0.2))
model.add(Conv2D(256, (3, 3), activation='relu', padding='valid'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dropout(0.2))
model.add(Dense(1024, activation='relu',
kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu',
kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
```

Table.3.3 Keras로 설계한 CNN 모델 코드

Table.3.3은 Keras로 설계한 코드이며 중간의 CNN 모델은 위의 Table.3.3처럼 사용자가 원하는 모양으로 순서대로 작성하면 된다. 또한, Conv2D와 MaxPooling2D의 (5,5)와(2,2)처럼 사용자가 학습시키고자하는 크기의 필터와 개수를 조정 할 수가 있다.

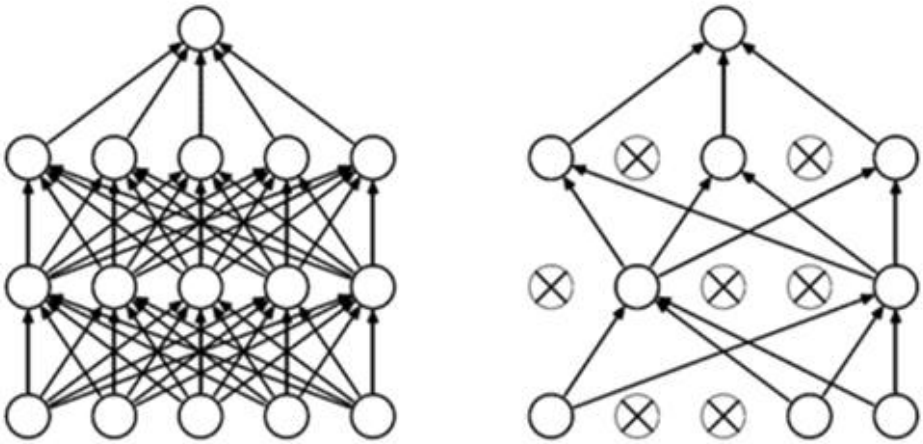


Fig. 3.6 Dropout의 예시

코드 내용 중 Dropout은 학습 시킬 때 신경회로망 전체가 아닌 일부만 통과시켜 학습 시키는 것을 Dropout이라고 한다.[26] Dropout은 신경회로망이 커질수록 Overfitting문제가 생기는데 이를 피하기 위해 고안해낸 방법이다. Dropout을 사용하면 학습을 시킬 때 줄어든 신경망 안에서도 어느 정도의 Overfitting이 생기는데 이 과정을 무작위로 반복하면, 평균의 효과를 얻는다. 또한, 신경망 안에서 특정 뉴런의 가중치, 웨이트 값이 커지면 다른 뉴런들의 학습속도가 느려지거나 학습이 제대로 안 될 수가 있는데 Dropout을 하면 특정 뉴런의 영향을 받지 않는다. 이러한 효과들로 Dropout을 사용하면 학습을 더 효과적으로 할 수가 있다.

3.4 설계한 CNN의 이미지 분류

다음 절에서는 위에서 설계한 CNN을 실험대상인 CIFAR-10,CIFAR-100, Caltech 101의 정확도와 손실을 실험 방법에 따라 다른 점을 확인하였다. 먼저 실험을 위해 Anaconda Prompt라는 윈도우에서 사용하는 가상 리눅스 환경의 창에서 Python을 실행시키고 설계한 코드를 학습을 시켰다. 다음은 Anaconda Prompt를 사용하여 코드를 실행했을 때 GPU가 작동되어 NVIDIA에서 제공하는 CUDA가 작동되는 것을 보여주는 창이다.

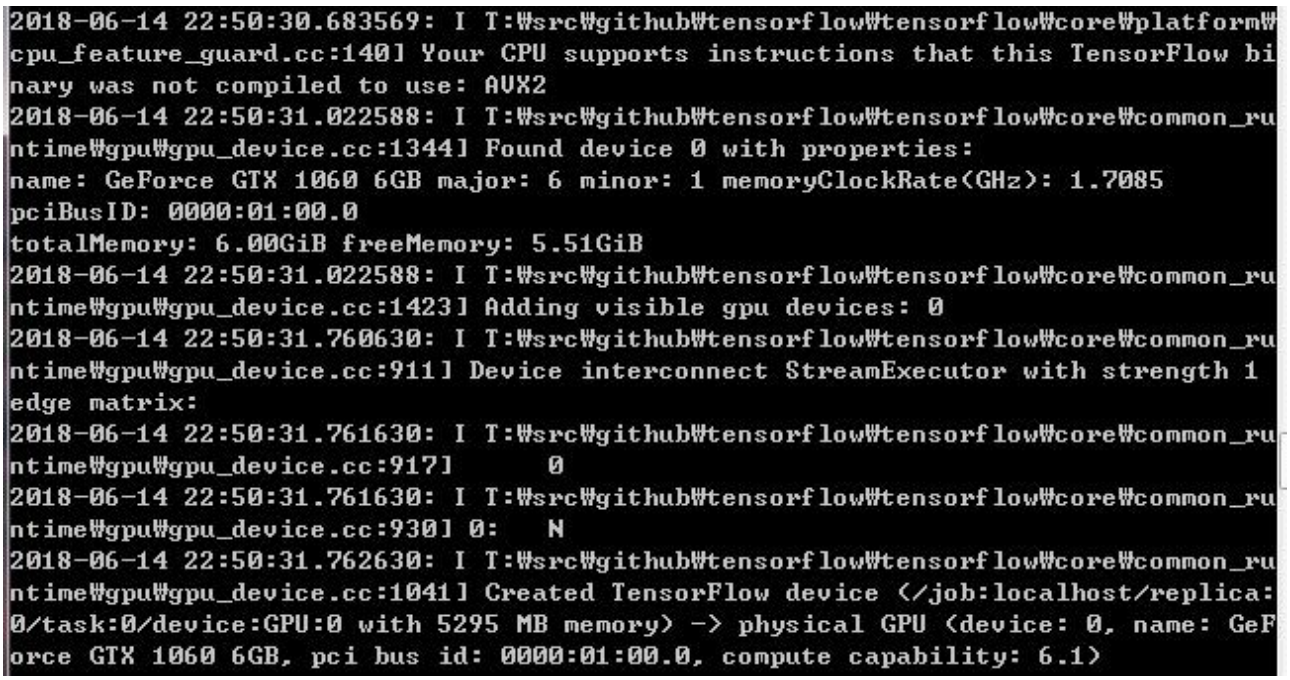


Fig. 3.7 Anaconda Prompt의 GPU가 실행되는 창

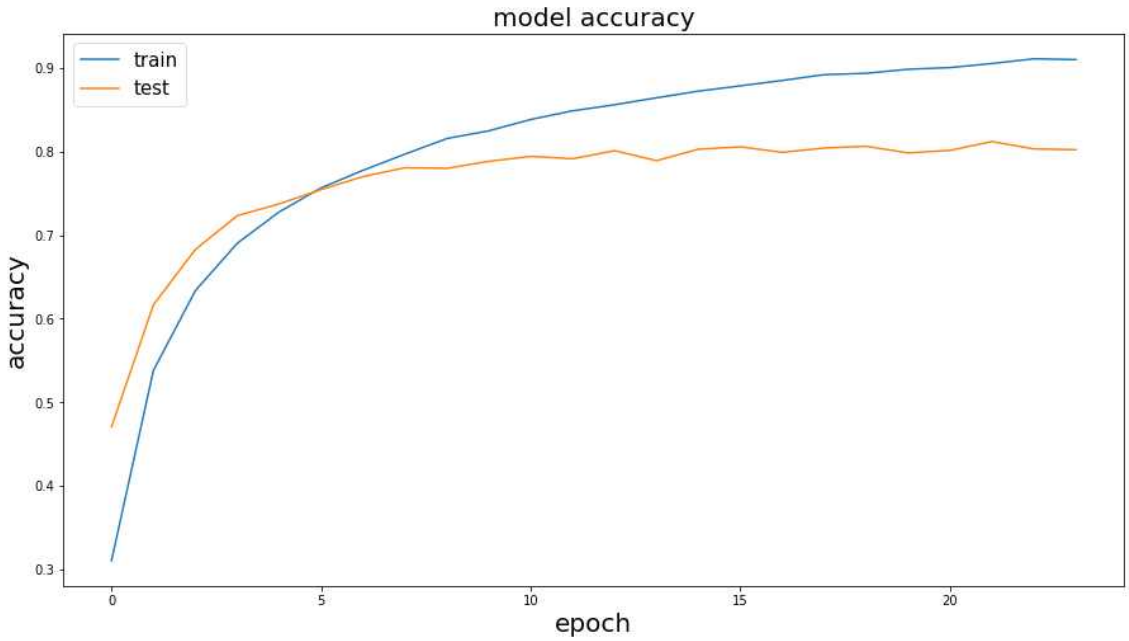


Fig. 3.8 CIFAR-10의 정확도

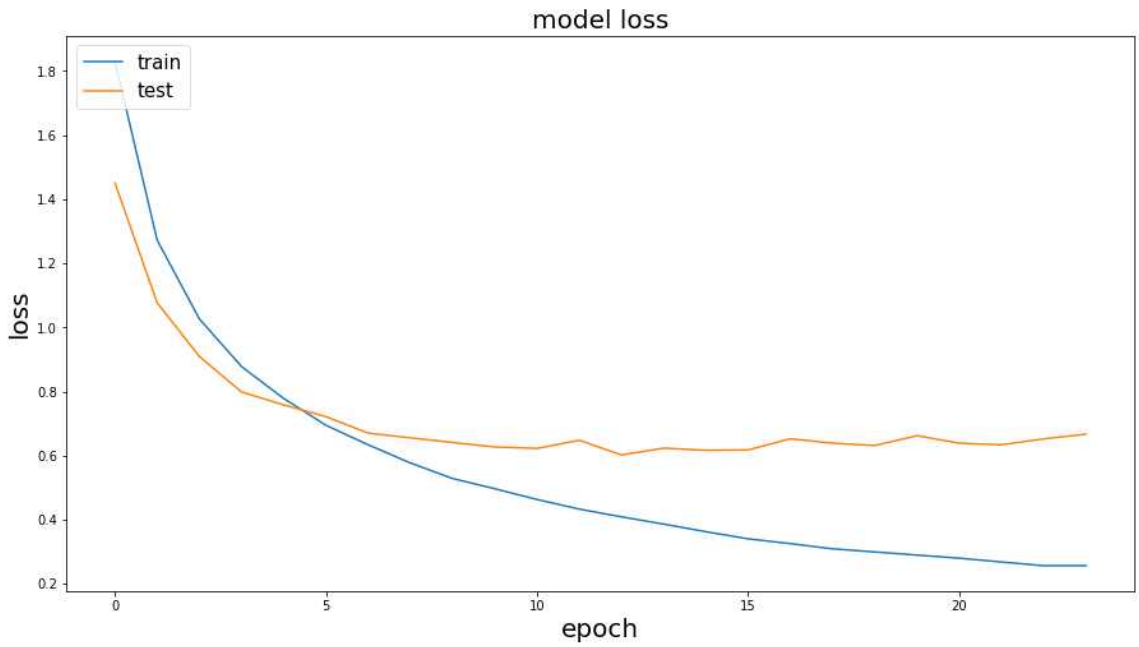


Fig. 3.9 CIFAR-10의 손실

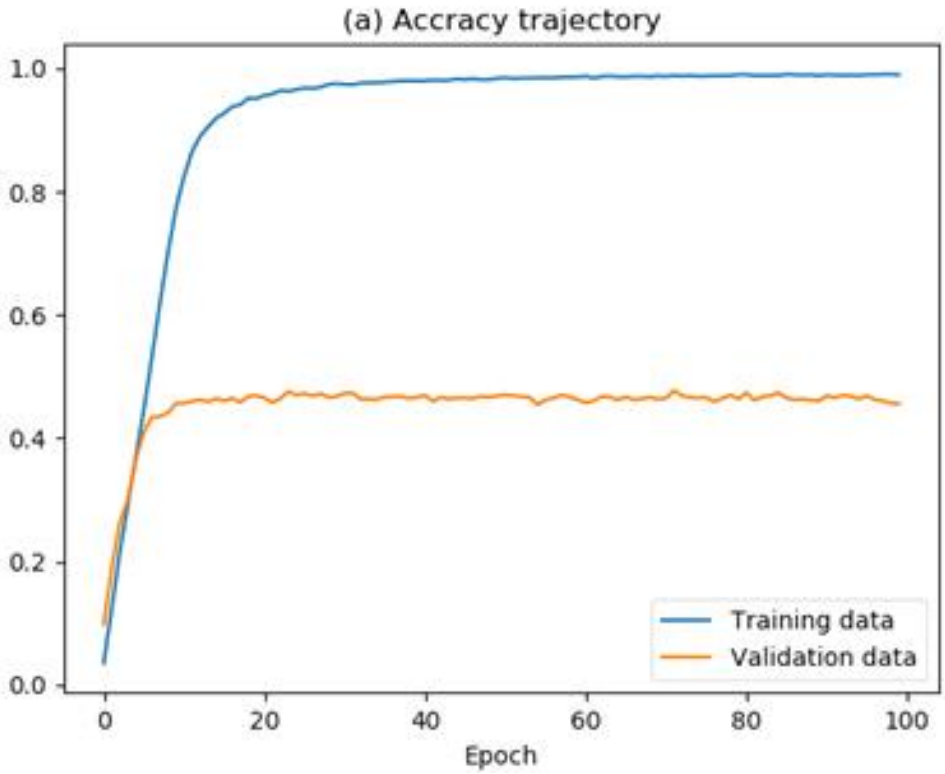


Fig. 3.10 CIFAR-100의 정확도

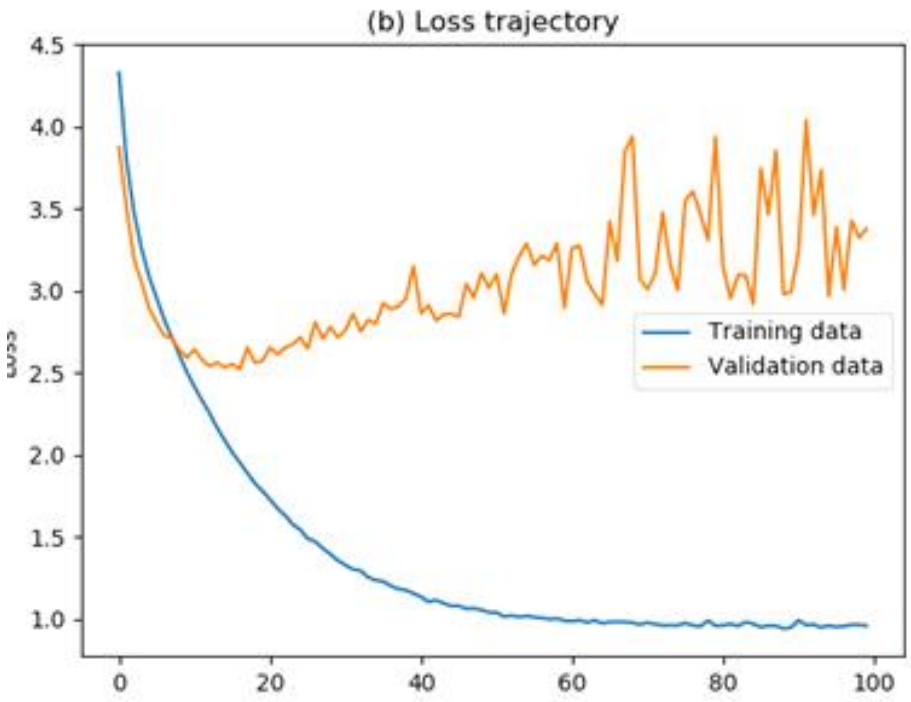


Fig. 3.11 CIFAR-100의 정확도

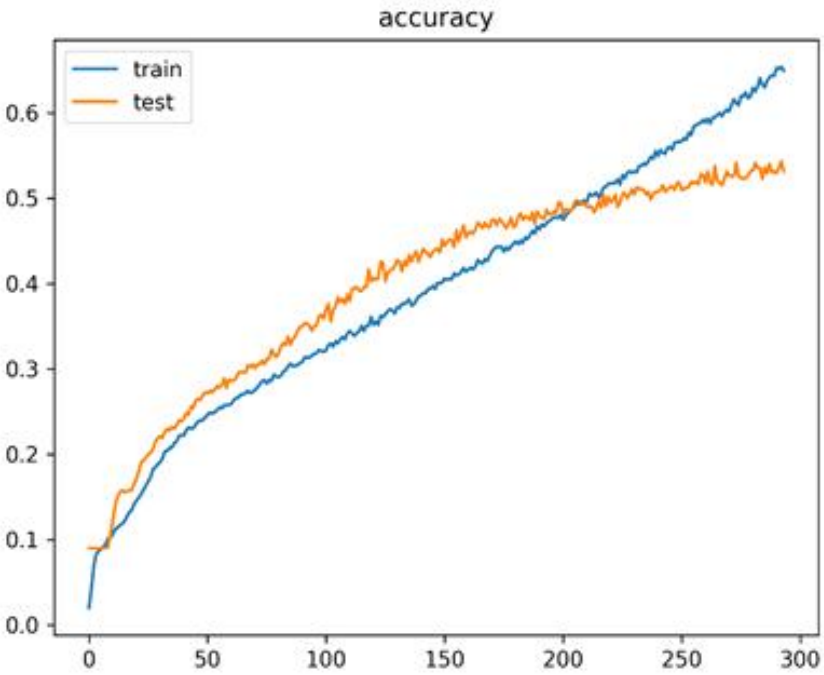


Fig. 3.12 Caltech 101의 정확도

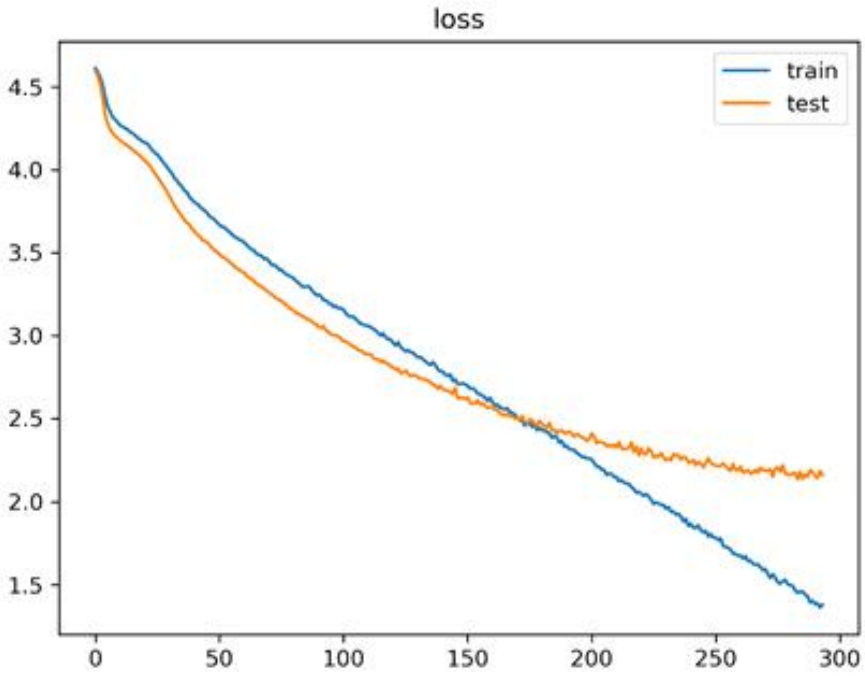


Fig. 3.13 Caltech 101의 손실

위의 그래프 들은 각각의 실험 대상의 정확도와 손실 그래프이다. 검증 데이터를 대상으로 CIFAR-10은 83.2%, CIFAR-100는 53.4% Caltech 101은 53.2%의 정확도를 보였다. 이 실험들은 Epoch를 100번 밖에 하지 않았기 때문에 정확도가 낮다고 판별하였다. CIFAR-10처럼 클래스가 10개로 낮은 것은 정확도가 비교적 높은 편이지만, CIFAR-100이나 Caltech 101처럼 클래스가 많거나 이미지 사이즈가 크면 계산이 커지거나 Grdient Explode 및 Vanish 현상이 이러 날 수가 있어 Batch Normalization 알고리즘을 추가적으로 사용하였다.

Batch Normalization를 간단히 설명하면 기존의 Regularization과정을 해준다고 볼 수 있다.[27] 이는 활성화함수를 통과한 값을 정규화(Normalization)한다고 볼 수 있다. 신경회로망에서는 학습을 시킬 때 전체 데이터를 학습시키기보단 Mini-Batch라하여 Batch사이즈를 정한다. 각 Layer의 Batch의 분포를 정규화 해주는데, 일종의 노이즈를 추가하는 방법으로 bias를 붙이는 것과 유사하다. Batch 마다 정규화하여 전체 데이터의 평균과 반산 값이 달라 질수가 있다. 학습을 할 때마다 활성화함수를 통과한 값이 정규화하기 때문에 가중치를 초기화하는 문제에서 자유로워진다. 이러한 과정으로 Batch Normalization을 통해 학습속도가 개선되고 가중치의 초기화를 선택하는데 항상 문제가 있는 신경회로망에서의 문제를 해결해준다. 또, Overfitting으로 부터의 위험을 줄이고 Gradient Vanishing 현상을 없애준다. Table.3.4는 이러한 장점을 바탕으로 Batch Normalization을 사용한 코드이다. 코드에서 보이는 weight_decay를 통해 필터의 정규화를 시켜준다.

```
model = Sequential()
weight_decay = 0.0005

model.add(Conv2D(64, (3, 3), input_shape=(3, 32, 32), padding='same',
activation='relu', kernel_regularizer=l2(weight_decay)))
model.add(BatchNormalization())
model.add(Dropout(0.3))
model.add(Conv2D(64, (3, 3), activation='relu',
padding='same',kernel_regularizer=l2(weight_decay)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), padding='same',
activation='relu',kernel_regularizer=l2(weight_decay)))
model.add(BatchNormalization())
model.add(Dropout(0.4))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(256, (3, 3), padding='same',
activation='relu',kernel_regularizer=l2(weight_decay)))
model.add(BatchNormalization())
model.add(Dropout(0.4))
model.add(Conv2D(256, (3, 3), padding='valid',
activation='relu',kernel_regularizer=l2(weight_decay)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(1024, activation='relu',kernel_regularizer=l2(weight_decay)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu',kernel_regularizer=l2(weight_decay)))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

Table.3.4 Batch Normalization을 사용한 CNN 코드

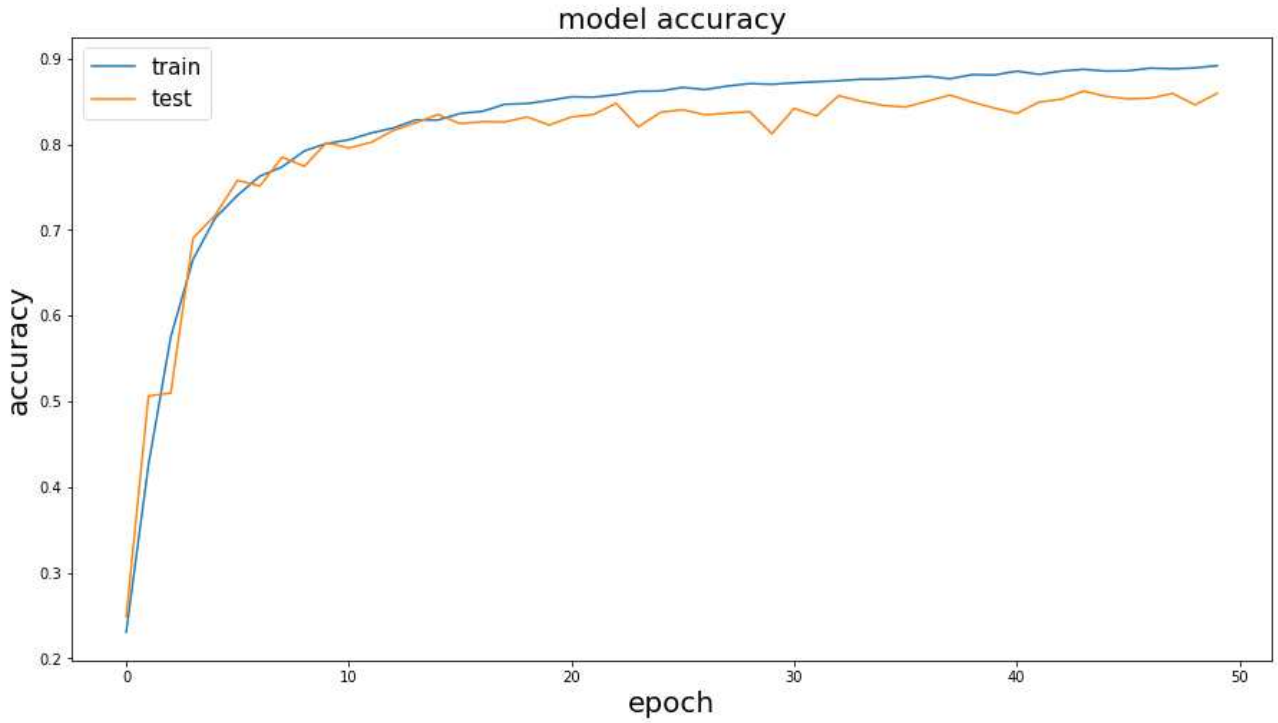


Fig. 3.14 Batch Normalization을 사용한 CIFAR-10의 정확도

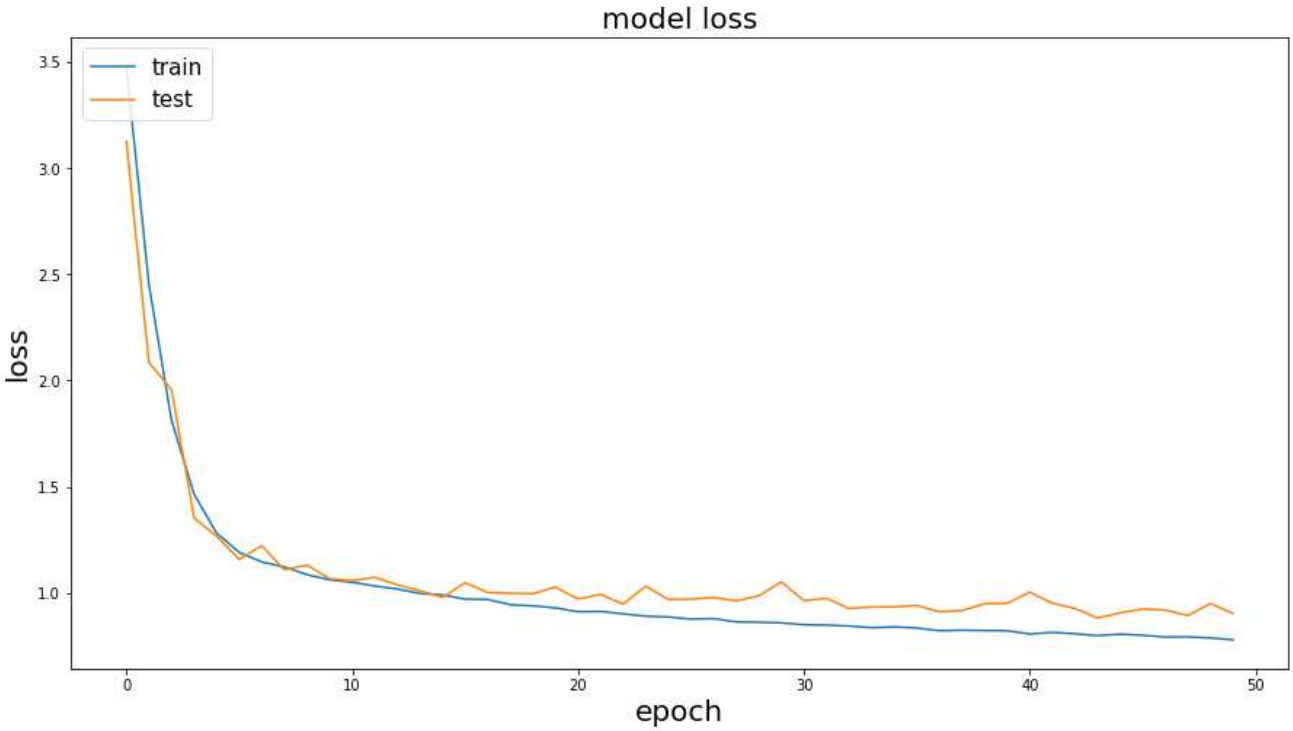


Fig. 3.15 Batch Normalization을 사용한 CIFAR-10의 정확도

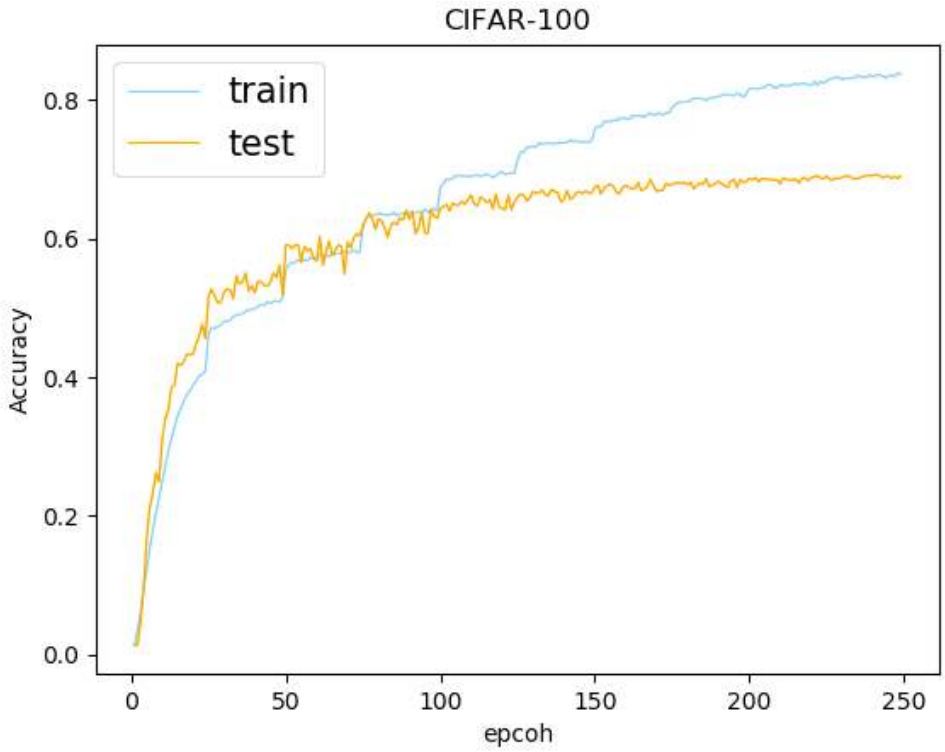


Fig. 3.16 Batch Normalization을 사용한 CIFAR-100의 정확도

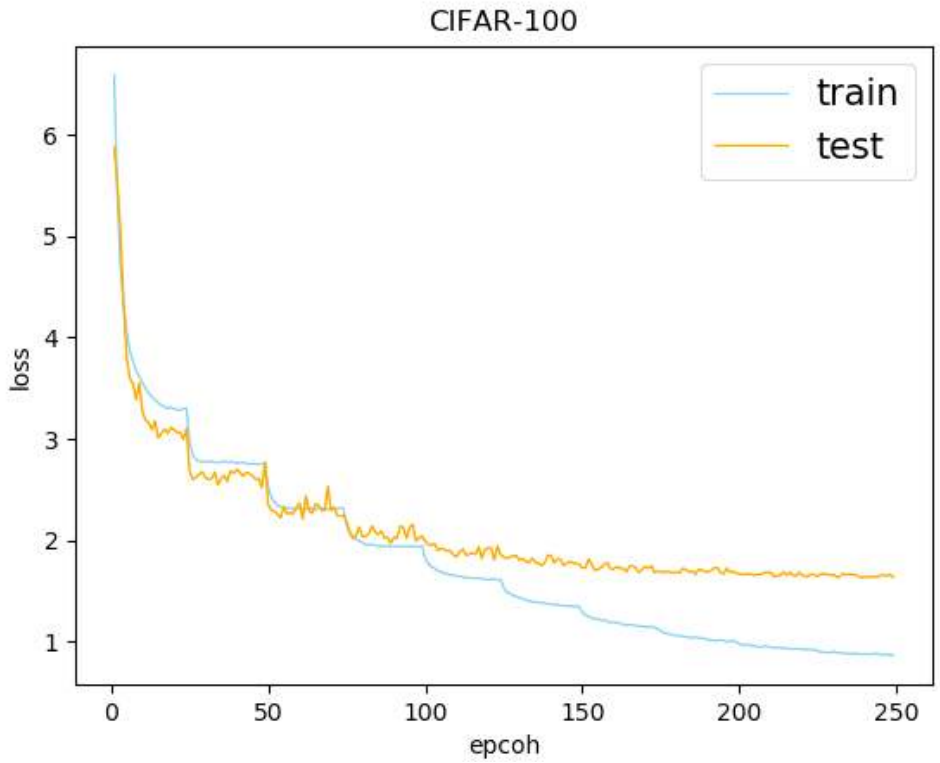


Fig. 3.17 Batch Normalization을 사용한 CIFAR-100의 정확도

위의 그래프처럼 CIFAR-10과 CIFAR-100의 정확도가 layer 숫자는 같아도 많이 오른 것을 볼 수가 있다. CIFAR-10은 86.2% CIFAR-100는 69.92%의 정확도를 보이며 CIFAR-10은 Epoch를 많이 하면 더 많이 오를 것으로 보인다.

다음은 Tensorboard를 사용하여 설계한 CNN의 구조를 확인하고 각각의 노드의 그래프를 확인하기 위한 코드의 일부분이다.[28]

```
# conv1
with tf.variable_scope('conv1') as scope:
kernel = _variable_with_weight_decay('weights',
shape=[5, 5, 3, 64],
stddev=5e-2,
wd=None)
conv = tf.nn.conv2d(images, kernel, [1, 1, 1, 1], padding='SAME')
biases = _variable_on_cpu('biases', [64], tf.constant_initializer(0.0))
pre_activation = tf.nn.bias_add(conv, biases)
conv1 = tf.nn.relu(pre_activation, name=scope.name)
_activation_summary(conv1)

# pool1
pool1 = tf.nn.max_pool(conv1, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1],
padding='SAME', name='pool1')
# norm1
norm1 = tf.nn.lrn(pool1, 4, bias=1.0, alpha=0.001 / 9.0, beta=0.75,
name='norm1')

# conv2
with tf.variable_scope('conv2') as scope:
kernel = _variable_with_weight_decay('weights',
shape=[5, 5, 64, 64],
stddev=5e-2,
wd=None)
conv = tf.nn.conv2d(norm1, kernel, [1, 1, 1, 1], padding='SAME')
biases = _variable_on_cpu('biases', [64], tf.constant_initializer(0.1))
pre_activation = tf.nn.bias_add(conv, biases)
conv2 = tf.nn.relu(pre_activation, name=scope.name)
_activation_summary(conv2)
```

Table.3.5 Tensorflow를 통해 Tensorboard를 사용하기위한 CNN코드-1

```
# conv3
with tf.variable_scope('conv3') as scope:
kernel = _variable_with_weight_decay('weights',
shape=[3, 3, 64, 128],
stddev=5e-2,
wd=None)
conv = tf.nn.conv2d(norm1, kernel, [1, 1, 1, 1], padding='SAME')
biases = _variable_on_cpu('biases', [128], tf.constant_initializer(0.0))
pre_activation = tf.nn.bias_add(conv, biases)
conv3 = tf.nn.relu(pre_activation, name=scope.name)
_activation_summary(conv3)
# conv4
with tf.variable_scope('conv4') as scope:
kernel = _variable_with_weight_decay('weights',
shape=[3, 3, 128, 128],
stddev=5e-2,
wd=None)
conv = tf.nn.conv2d(conv3, kernel, [1, 1, 1, 1], padding='SAME')
biases = _variable_on_cpu('biases', [128], tf.constant_initializer(0.0))
pre_activation = tf.nn.bias_add(conv, biases)
conv4 = tf.nn.relu(pre_activation, name=scope.name)
_activation_summary(conv4)
# pool2
pool2 = tf.nn.max_pool(conv4, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],
padding='SAME', name='pool1')
# norm2
norm2 = tf.nn.lrn(pool2, 4, bias=1.0, alpha=0.001 / 9.0, beta=0.75,
name='norm1')
# conv5
with tf.variable_scope('conv5') as scope:
kernel = _variable_with_weight_decay('weights',
shape=[5, 5, 128, 256],
stddev=5e-2,
wd=None)
conv = tf.nn.conv2d(norm2, kernel, [1, 1, 1, 1], padding='SAME')
biases = _variable_on_cpu('biases', [256], tf.constant_initializer(0.1))
pre_activation = tf.nn.bias_add(conv, biases)
conv5 = tf.nn.relu(pre_activation, name=scope.name)
_activation_summary(conv5)
# conv6
with tf.variable_scope('conv6') as scope:
kernel = _variable_with_weight_decay('weights',
shape=[5, 5, 256, 256],
stddev=5e-2,
wd=None)
conv = tf.nn.conv2d(conv5, kernel, [1, 1, 1, 1],padding='VALID' )
biases = _variable_on_cpu('biases', [256], tf.constant_initializer(0.1))
pre_activation = tf.nn.bias_add(conv, biases)
conv6 = tf.nn.relu(pre_activation, name=scope.name)
_activation_summary(conv6)

# pool3
pool3 = tf.nn.max_pool(conv6, ksize=[1, 2, 2, 1],
strides=[1, 2, 2, 1], padding='SAME', name='pool2')
```

Table.3.6 Tensorflow를 통해 Tensorboard를 사용하기위한 CNN코드-2

위의 코드는 Tensorflow기반의 코드이며, Keras보단 복잡하다 하지만 구체적으로 사용자가 원하는 만큼 신경회로망을 설계할 수가 있다. Table.3.5, Table.3.6에서 볼 수 있듯이 각 노드별로 scope를 설정했다. Fig.3.18, Fig.3.19는 설계한 신경회로망을 그래프화 한 것이다.

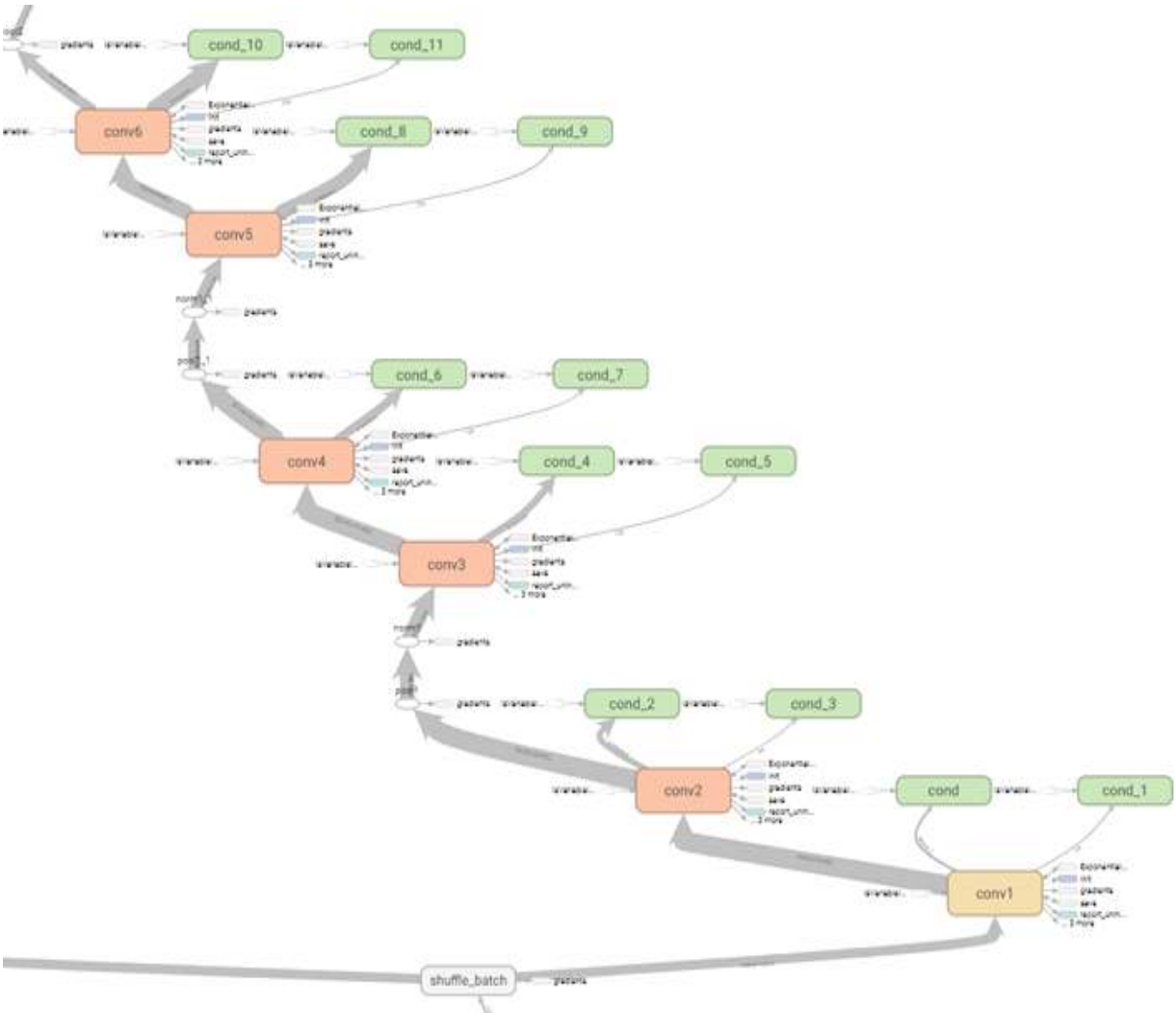


Fig. 3.18 Tensorboard로 설계한 CNN의 그래프-1

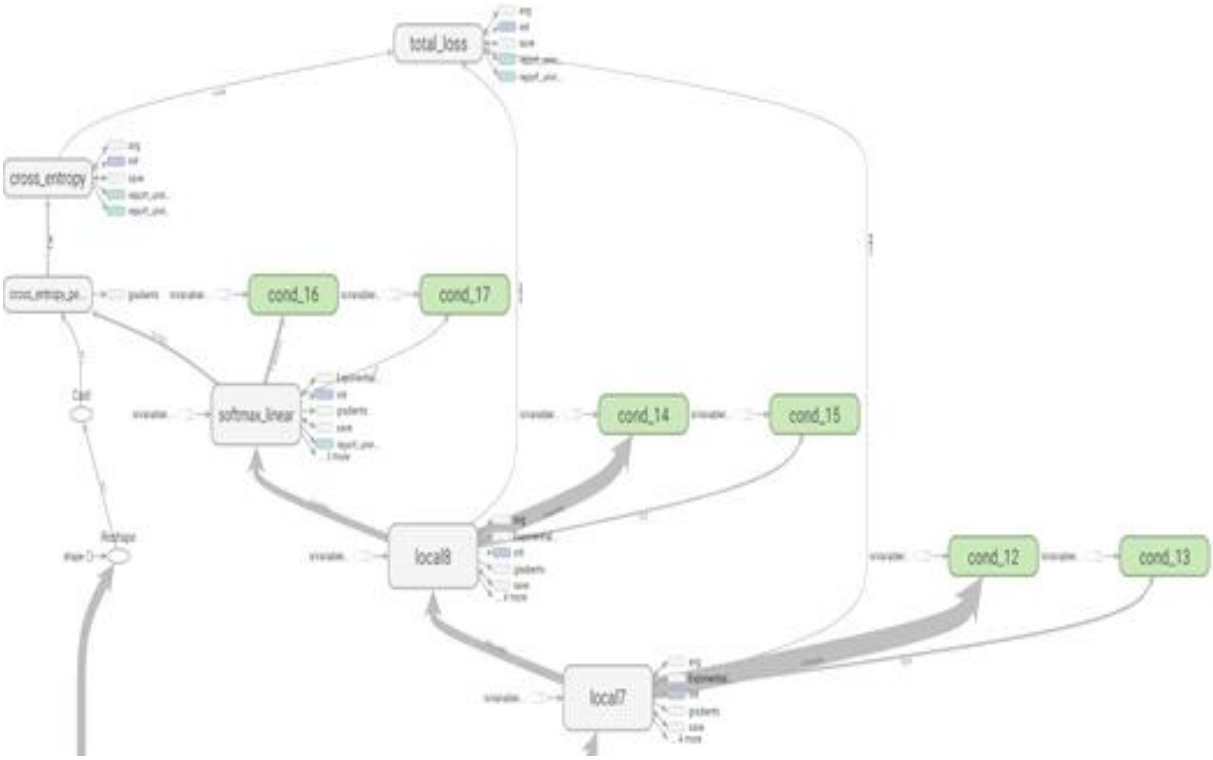


Fig. 3.19 Tensorboard로 설계한 CNN의 그래프-1

Tensorboard를 통하여 본인이 설계한 신경회로망의 구조를 마우스를 통하여 모형을 확인 할 수 가 있다. Tensorboard의 Histogram 메뉴를 통해 각 노드의 Histogram도 확인 할 수 있다.

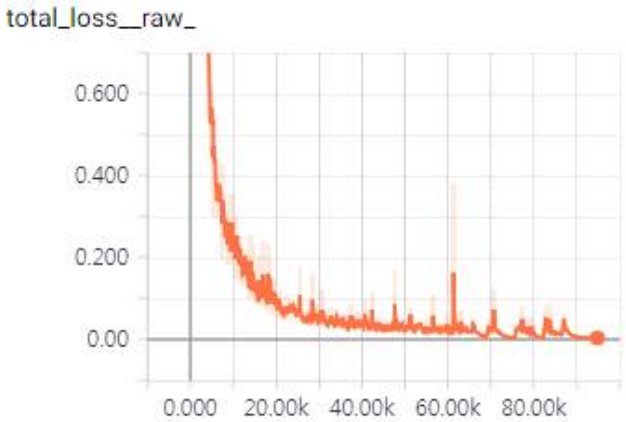
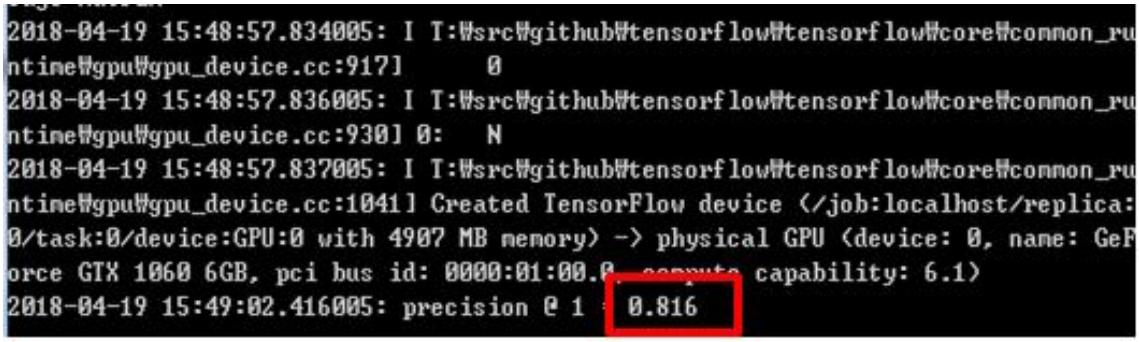


Fig. 3.20 Tensorboard를 통해 확인한 loss 그래프

마지막으로 CIFAR-10을 대상으로 Epoch의 횟수에 따라 정확도가 다름을 확인하였다.



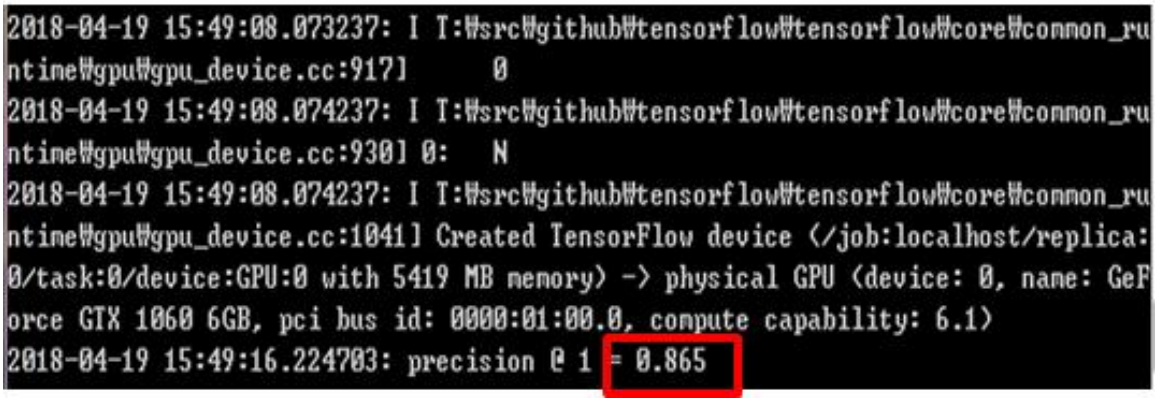


Fig. 3.21 학습 횟수를 다르게 했을 때 Anaconda 창으로 확인한 정확도

위의 실험은 CIFAR-10의 batch 사이즈를 작게 하여 위의 그림은 1만 번 밑에는 10만 번 학습했을 때의 결과이다.

3.5 높은 정확도의 논문과 비교

본 절에서는 본 논문의 실험과 각 대상의 최고 정확도의 논문의 비교를 통하여 어떻게 본 논문의 CNN을 수정 보완하여 속도와 정확도를 올리고 실제로 이미지 검색에 활용하기 위해 다른 점을 확인해보았다.

Dataset	Train	Test	최고 정확도/논문
MNIST	99.98%	99.75%	99.79% / Regularization of Neural Networks using DropConnect- ICML 2013
Cifar10	90.25%	86.5%	96.53% / Fractional Max-Pooling ARXIV 2015
Cifar100	80.3%	69.92%	75.72% /Fast and Accurate Deep Network Learning by Exponential Linear Units ARXIV2015
Caltech101	66.5%	53.2%	91.44% / Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition ARXIV 2014

Table.3.7 본 논문의 실험과 최고 정확도를 보인 논문과 비교

위의 자료들은 2016년 기준의 자료들이다.[29] 그렇기 때문에 현재 더 높은 정확도를 선보인 연구가 있을 수 있다. 먼저 MNIST 최고 정확도를 선보인 논문은 Dropout 논문이다. Dropout은 위3.2절 에서도 설명하였듯이 CNN뿐만 아니라 신경회로망이 가지는 단점을 보완하였다. MNIST 이미지가 특징이 다른 칼라 이미지보다 적고 그레이 이미지이기 때문에 가능한 것 같다. CIFAR-10과 Caltech 101의 논문은 Pooling을 Fractional 및 Spatial 하게 처리하는 차이를 보이고는 있지만 결국은 Pooling으로 인해 생기는 단점을 극복하여 층을 더 깊게 만들려고 하는 의도를 보이고 있다.[30] 마지막으로 CIFAR-100의 논문은 Activation Function을 ELU를 사용하여 정확도를 높였다.[18] 위의 논문들을 살펴보면 CNN이 가지는 단점 및 신경회로망 본연의 단점을 보완해 나가는 방향으로 더 깊은 층의 CNN혹은 신경회로망을 통해 정확도를 높였음을 확인 할 수 있었다.

제 4 장 : 결　　론

본 논문에서 설계한 CNN 구조와 이미지 분류를 위한 방법은 현재 이미 많은 이미지 검출 및 이미지 인식에도 사용되고 있으며, 이를 더욱 발전시키기 위해 다른 알고리즘들과 하이브리드화 되어 논문이 제안되거나, 신경회로망을 위한 새로운 알고리즘이 추가되어 사용되고 있다. 그 예로 R-CNN과 DCGAN이 있다.[31][32] R-CNN은 이미지를 CNN을 통하여 Tuning을 하고, Detecting하여 특징을 추출하며 이를 SVM을 이용하여 점수를 매겨 이미지를 구분하는 방식이다. 기존의 이미지 추출에 사용되었던 머신 러닝과 알고리즘을 합하여 사용하는 것을 볼 수 있다. 또 DCGAN은 GAN이라는 새로운 인공지능 알고리즘에 CNN을 붙여 사용하여 이미지 인공지능에 새로운 방향을 제시하고 있다.

본 논문에서는 6개의 Convolution Layer와 3개의 Pooling Layer를 사용하였고, 처음 Layer부터 64개의 필터를 사용하여 Pooling Layer를 지날 때마다 필터가 2배씩 증가하여 마지막 층에서는 256개의 필터를 사용하여 깊은 층의 CNN을 사용할 때와 같은 파라미터를 계산하는 구조를 보이고 있다. 이에 Dropout과 Batch Normalization을 사용하여 많은 양의 필터로 인해 계산에 Overfitting과 Gradient Vanshing 현상이 일어나지 않게 하여 더 좋은 정확도를 확인 하였다. 이를 통해 깊지 않은 층을 갖는 CNN에서도 지금까지 나온 이미지 분류 논문의 퍼포먼스를 따라 갈 수 있음을 보였다. 또한, Convolution Layer가 10개 정도로 조금 더 쌓았을 때, 신경회로망에서 중요한 Activation Function 및 Pooling들의 방식을 최근에 새로 제안되었던 논문들의 ELU 함수와 Fractional Max-Pooling 알고리즘을 실험에 추가 한다면 더 좋은 결과를 나타낼 것이라고 보고 있다. CNN은 지금까지 이미지 대상의 신경회로망 알고리즘 중에 가장 좋은 결과를 내고 있고, 속도 면에서도 현재 개발 되고 있는 CPU, GPU, RAM의 용량이 커짐에 따라 저절로 따라가고 있다. 그렇기 때문에 현재 발전되고 있는 컴퓨터 비전처리에서 빼놓을 수가 없다. 하지만 아직까지 층이 100개가 넘어가거나 그 이상일 경우 개인 연구자가 설계하기에는 사용하는 컴퓨터가 사양이 낮고, 학습 속도 면에서도 느리다. 본 논문에서는 기존의 정확도가

높은 Layer가 깊은 논문과 달리 비교적 Layer가 깊지 않은 CNN을 통하여 정확도가 높게 나오는 것을 보였다.

참고문헌

[1] Hinton, Geoffrey E. "Deep belief networks." Scholarpedia 4.5 (2009): 5947.

[2] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." nature 521.7553 (2015): 436.

[3] LeCun, Yann, and Yoshua Bengio. "Convolutional networks for images, speech, and time series." The handbook of brain theory and neural networks 3361.10 (1995): 1995.

[4] https://www.tensorflow.org/

[5] https://keras.io/

[6] Haykin, Simon S., et al. Neural networks and learning machines. Vol. 3. Upper Saddle River, NJ, USA:: Pearson, 2009.

[7] Hecht-Nielsen, Robert. "Theory of the Backpropagation Neural Network." Neural Networks for Perception.Elsevier, 1992. 65-93. Print.

[8] Mao, Xiaoyang, Yoshiyasu Nagasaka, and Atsumi Imamiya. "Automatic Generation of Pencil Drawing from 2D Images using Line Integral Convolution". CAD/Graphics. 2001. 240-248. Print.

[9] Bracewell, R. "Pentagram Notation for Cross Correlation." The Fourier Transform and Its Applications. New York: McGraw-Hill, pp. 46 and 243, 1965.

[10] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet Classification with Deep Convolutional Neural Networks". Advances in neural information processing systems. 2012. 1097-1105. Print.

[11] Ciresan, Dan; Meier, Ueli; Schmidhuber, Jürgen (June 2012). "Multi-column deep neural networks for image classification". 2012 IEEE Conference on Computer Vision and Pattern Recognition. New York, NY: Institute of Electrical and Electronics Engineers (IEEE): 3642 -3649. arXiv:1202.2745

[12] Graham, Benjamin. "Fractional Max-Pooling." arXiv preprint arXiv:1412.6071 (2014)Print.

[13] Haykin, Simon S. (1999). Neural Networks: A Comprehensive Foundation

[14] Han, Jun, and Claudio Moraga. "The influence of the sigmoid function parameters on the speed of backpropagation learning." International Workshop on Artificial Neural Networks. Springer, Berlin, Heidelberg, 1995.

[15] Hochreiter, Sepp. "The vanishing gradient problem during learning recurrent neural nets and problem solutions." International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 6.02 (1998): 107-116.

[16] Nair, Vinod; Hinton, Geoffrey E. (2010), "Rectified Linear Units Improve Restricted Boltzmann Machines", 27th International Conference on International Conference on Machine Learning, ICML'10, USA: Omnipress, pp. 807-814, ISBN 9781605589077

[17] Maas, Andrew L.; Hannun, Awni Y.; Ng, Andrew Y. (June 2013). "Rectifier nonlinearities improve neural network acoustic models"

[18] Clevert, Djork-Arné; Unterthiner, Thomas; Hochreiter, Sepp (2015-11-23). "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)". arXiv:1511.07289

[19] Scherer, Dominik, Andreas Müller, and Sven Behnke. "Evaluation of pooling operations in convolutional architectures for object recognition." International conference on artificial neural networks. Springer, Berlin, Heidelberg, 2010.

[20] https://anaconda.org/

[21] Yann, LeCun, Cortes Corinna, and J. B. Christopher. "The MNIST database of handwritten digits." URL http://yhann. lecun. com/exdb/mnist (1998).

[22] Krizhevsky, Alex, Vinod Nair, and Geoffrey Hinton. "The CIFAR-10 dataset." online: http://www. cs. toronto. edu/kriz/cifar. html (2014).

[23] Krizhevsky, Alex, Vinod Nair, and Geoffrey Hinton. "The CIFAR-100 dataset." online: http://www. cs. toronto. edu/kriz/cifar. html (2014).

[24] Griffin, Gregory, Alex Holub, and Pietro Perona. "Caltech-101 object category dataset." (2007).

[25]Function, Softmax. "Available online: https://en. wikipedia. org/wiki." Error_function (accessed on 22 November 2012).

[26] Srivastava, Nitish, et al. "Dropout: A simple way to prevent neural networks from overfitting." The Journal of Machine Learning Research 15.1 (2014): 1929-1958.

[27] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv 2015." arXiv preprint arXiv:1502.03167 (2015).

[28] Girija, Sanjay Surendranath. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems." (2016).

[29] Benenson, Rodrigo. "Classification datasets results." URL http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results. html (2016).

[30] He, Kaiming, et al. "Spatial pyramid pooling in deep convolutional networks for visual recognition." european conference on computer vision. Springer, Cham, 2014.

[31] Girshick, Ross. "Fast r-cnn." arXiv preprint arXiv:1504.08083 (2015).

[32] Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." arXiv preprint arXiv:1511.06434 (2015).

국 문 초 록

이미지 컨텐츠 검색을 위한 CNN의 구조

이현수
전자전기공학부 제어 및 시스템
중앙대학교 대학원

본 논문에서는 이미지 인식 및 분류에 널리 이용되는 CNN (convolutional neural network)을 이용하여 심층 학습 알고리즘으로 이미지 컨텐츠를 분류한다. CNN은 이미 높은 정확도와 뛰어난 분류 성능을 가지고 있다. 그러나 기존의 신경 회로망의 overfitting 및 gradient vanishing 문제와 같은 단점은 여전히 높은 컴퓨팅 파워와 더 깊은 층의 필요성과 함께 남아 있습니다. 이 과부하로 인해 학습 속도는 이미지의 크기에 따라 느려진다. 따라서 우리는 CNN의 상세한 분석과 실험을 통해 CNN의 장점과 단점을 확인했다. 이 논문에서 CNN은 최근의 높은 정확도 논문의 성과에 근접한 결과를 제공하기 위해 심층적이지 않도록 설계하였다. 여섯 개의 깊은 convolution layer와 세 개의 pooling layer가 쌓여있고, fully connected layer에는 두 개의 hidden layer가 있다. 이렇게 설계된 CNN은 CIFAR-10, CIFAR-100 및 Caltech 101과 같은 벤치마크 데이터베이스로 테스트하였다. 또한 기존의 convolution layer, pooling layer 및 수많은 필터와 함께 dropout과 batch normalization 알고리즘이 추가로 사용하였다. 결과는 매우 깊은 구조 없이 CNN이 높은 정확도를 제공 할 수 있음을 확인시켜주었다. 마지막으로 우리는 R-CNN과 DCGAN 알고리즘을 통해 CNN의 미래 방향을 논의했다.

핵심어 (또는 주요어) : Convolutional Neural Network, Overfitting, Gradient Vanishing, Dropout, Batch Normalization, 이미지 분류

Abstract

A Structure of Convolutional Neural Networks for Image Contents Search

Hyun Su Lee
Electrical & Electronic Engineering
Intelligent Robot and Vision
Graduate School of Chung-Ang University

In this paper, a CNN (convolutional neural network), which is popular in image recognition and classification, is used to classify image contents with deep learning algorithm. CNNs have already high accuracy and outstanding performance in classification. However, disadvantages such as over-fitting and gradient vanishing problems of existing neural networks, still remain, along with the necessity of higher computing power and deeper layers. Due to this overload, the speed of learning slows down depending on how large the size of the image is. Therefore, we have confirmed the advantages and disadvantages of CNN through detailed analyses and experiments of CNN. In this thesis, a CNN is designed not to be very deep, to give results that are close to the performances of recent high-accuracy papers. Six deep convolutional layers and three pooling layers are stacked, and fully connected layers have two hidden layers. Now, the designed CNN has been tested with benchmark databases such as CIFAR-10, CIFAR-100 and Caltech 101. In addition, dropout and batch normalization algorithms were additionally used with the existing convolution layer, pooling layer and numerous filters. The result confirmed that, even without a very deep structure, this CNN can give high accuracy. Finally, we discussed the future direction of CNN through R-CNN and DCGAN algorithm.

핵심어 (또는 주요어) : Convolutional Neural Network, Overfitting, Gradient Vanishing, Dropout, Batch Normalization, Image Classification