고양이 종 예측 CNN 탐구 보고서

대건고등학교 20928 정준영

목차

I. 탐구 동기 2
II. 인공 신경망과 합성곱 신경망 3
1. 인공 신경망(Artificial Neural Network) 3
2. 합성곱 신경망(Convolutional Neural Network) 4
III. CNN 공부/실습 7
1. 이산 합성곱 수행(p.623~31) 7
IV. 고양이 종 분류 CNN 제작 14
1. 데이터 수집 14
2. 데이터 학습 15
3. 결과 15
V. 결론 17

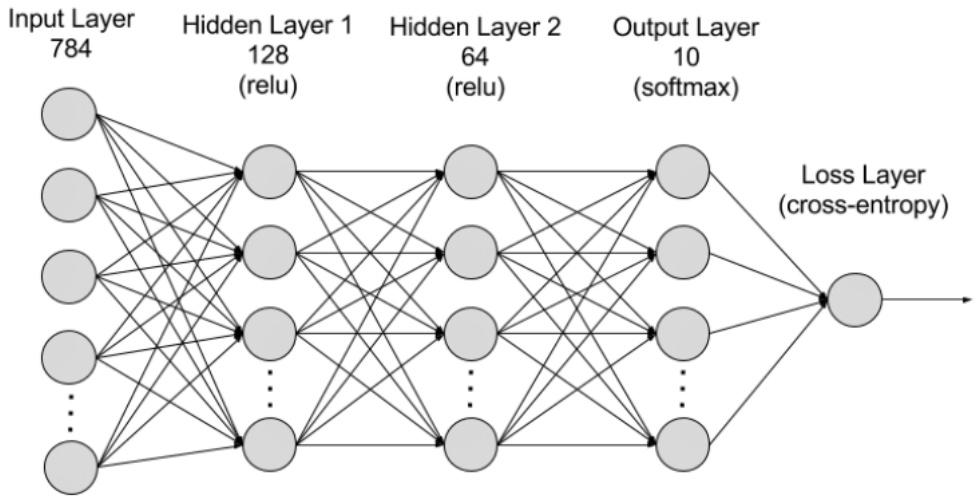
I. 탐구 동기

머신러닝과 딥러닝 등 인공 신경망에 대해 공부를 해보고 싶어 여러 논문과 책을 찾는 중 1학년 자율시간 때 보게 된 '안정근, 우탁. (2022).『합성곱 신경망 알고리즘이 도입된 인공지능 인지 시스템을 통한 게임 AI의 행동 메커니즘 향상 및 FPS 게임의 난이도 조절 방법론 제안』. 디지털콘텐츠학회논문지, 23(1), 21-30, 10.9728/dcs.2022.23.1.21'과 '세바스찬 라시카, 바히드 미자리리. 박해선 옮김. 길벗 출판사.『머신 러닝 교과서 with 파이썬, 사이킷런, 텐서플로』.'를 읽어보게 되었고 합성곱 신경망 즉 CNN에 대해 더욱 자세히 탐구하고 이를 직접 사용해보고 싶어졌다. 이에 인공 신경망과 합성곱 신경망에 대해 조사를 해보고 이를 이용해 직접 고양이의 종을 예측해보는 활동을 진행하게 되었다.

II. 인공 신경망과 합성곱 신경망

1. 인공 신경망(Artificial Neural Network)

인공신경망이란 사람 또는 동물의 두뇌 신경망에 착안하여 구현된 컴퓨팅 시스템이다. 기계 학습의 형태 중 하나로 신경 세포인 뉴런이 여러 개 연결되어있는 망 형태이다.[1] 인공 신경망은 교사 신호(정답)의 입력에 의해 문제에 최적화 되어가는 지도 학습과 교사 신호를 필요로하지 않는 비지도 학습으로 나뉘어 명확한 정답이 존재하는 경우 지도 학습이, 데이터 클러스팅(주어진 데이터들을 분석하여 데이터 집단(클러스터)를 정의하고 데이터 집단의 대표점을 찾는 데이터 마이닝의 한 방법[2]) 같은 경우는 비지도 학습이 사용된다.[3] 이번 탐구의 경우 고양이들의 종은 정해져 있으므로 지도 학습이 사용된다.



인공 신경망은 여러 개의 계층으로 나누어져 있다. 기본적으로 3 개의 층으로 구성된다.

---

[1] 인공 신경망, IT용어사전, 네이버 지식백과

[2] 클리스터 분석, 위키피디아

[3] 인공 신경망, 위키피디아

입력 계층

데이터들이 입력 계층에서 인공 신경망으로 들어간다. 입력 노드는 데이터를 처리하여 분석 또는 분류한 후 다음 계층으로 전달하게 된다. 고양이 신경망을 만들기 위해서는 입력 계층에 고양이 데이터들이 입력될 것이다.

숨겨진 계층

숨겨진 계층은 입력 계층이나 다른 숨겨진 계층에서 입력을 받는다. 각 숨겨진 계층은 이전 계층의 출력을 분석하고 추가 처리하여 다음 계층으로 전달한다. 숨겨진 계층의 수는 많을 수 있다.

출력 계층

출력 계층은 인공 신경망이 처리한 모든 데이터의 최종 결과를 출력한다. 단일 노드 또는 다중 노드를 가질 수 있다. 예를 들어 이진(네 / 아니요) 분류 문제가 있는 경우 출력 계층에는 하나의 출력 노드가 있고 결과는 1 또는 0 일것이다. 그러나 다중 클래스 분류 문제가 있는 경우 출력 계층은 둘 이상의 출력 노드로 구성될 수 있다.[4] 이번 탐구에서 고양이 종은 여러 개이므로 다중 클래스 분류 문제이며 출력 노드는 둘 이상 일 것이다.

2. 합성곱 신경망(Convolutional Neural Network)

합성곱 신경망(CNN)은 합성곱을 이용한 인공신경망이다. CNN은 이미지와 같은 형태의 데이터들을 처리하는 데에 적합하며 인간의 시각 피질 구조를 모방하여 데이터들로부터 주요 특징을 추출하고 데이터의 차원에 따라 1차원, 2차원, 3차원 합성곱 신경망 등으로 표현한다. 합성곱 신경망과 다른 신경망의 차이점은 합성곱 신경망은 데이터의 공간적 특성을 보존할 수 있다는 것이다. 단순 다중 퍼셉트론 인공 신경망의 경우에는 10x10 크기의 2차원 이미지 데이터를 넣으면 100크기의 1차원 데이터로 변환되지만 CNN은 2차원 데이터 그대로 보존할 수 있다.[5]

---

[4] 신경망이란 무엇인가요?, AWS

[5] 합성곱 신경망, 두산백과

합성곱 신경망에 대해 알려면 우선 합성곱에 대해 먼저 알아야 한다. 합성곱(Convolution)은 하나의 함수를 반전 이동한 것과 또 다른 함수를 곱한 다음 구간에 대해 적분하여 새로운 함수를 구하는 수학 연산자이다. 두 함수 f와 g의 합성곱을 기호로 $\boldsymbol{f * g}$로 표현한다. g또는 f를 반전시킨 합성곱은,

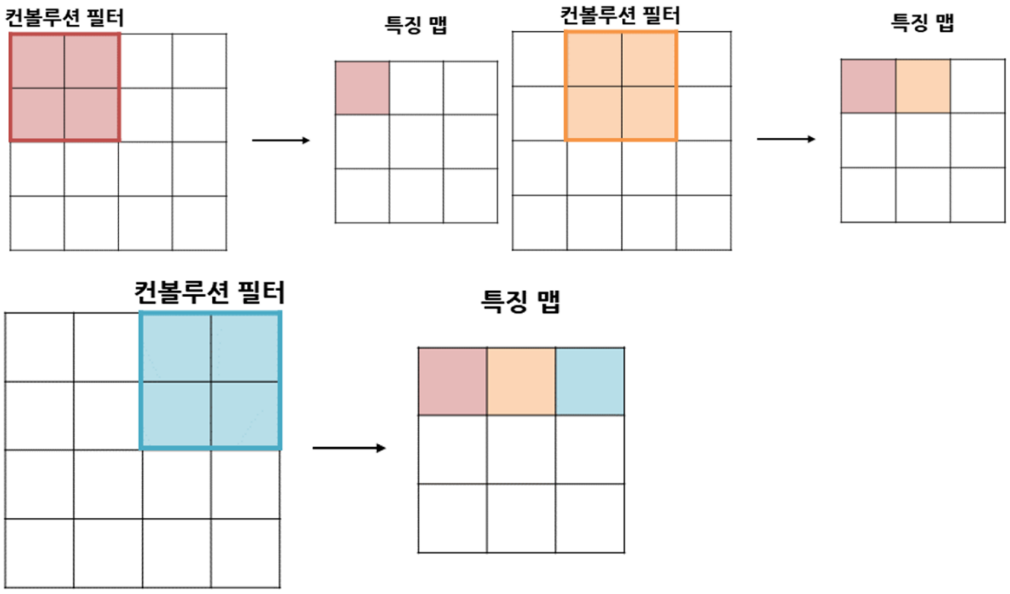
$$(\boldsymbol{f} * \boldsymbol{g})(\boldsymbol{t}) = \int_{-\infty}^{\infty} \boldsymbol{f}(\boldsymbol{\tau})\boldsymbol{g}(\boldsymbol{t}-\boldsymbol{\tau})\boldsymbol{d\tau}$$

$$= (\boldsymbol{f} * \boldsymbol{g})(\boldsymbol{t}) = \int_{-\infty}^{\infty} \boldsymbol{f}(\boldsymbol{t}-\boldsymbol{\tau})\boldsymbol{g}(\boldsymbol{\tau})\boldsymbol{d\tau}$$

로 나타낸다. 이산 함수 f, g에 대하여 두 함수의 합성곱은,

$$(\boldsymbol{f} * \boldsymbol{g})(\boldsymbol{m}) = \sum_{\boldsymbol{n}} \boldsymbol{f}(\boldsymbol{n})\boldsymbol{g}(\boldsymbol{m}-\boldsymbol{n})$$

로 나타낸다.

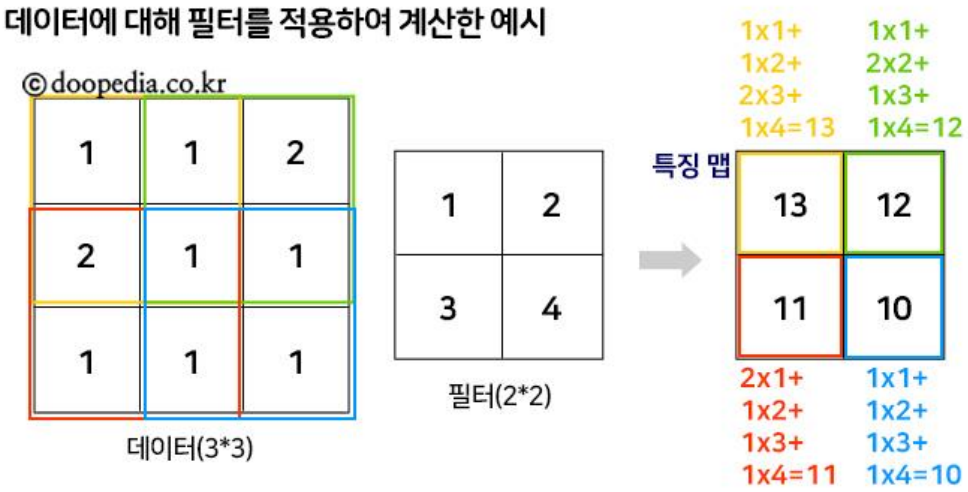


기계학습에서 합성곱 연산은 합성곱 신경망의 컨볼루션 계층에서 활용된다. 합성곱 신경망은 비정형 데이터를 입력받아 분류하는 모델이지만 이 데이터를 모델이 바로 처리할 수가 없다. 따라서 이를 1차원 선형 벡터로 전환하기 위해 입력받은 데이터에 컨볼루션 필터를 순차 적용하여 곱한 값을 합하는 합성곱 연산을 수행하여 특징 맵을 생성하게 된다.[6]

---

[6] 합성곱, AI 용어사전

예시로 3x3 크기의 데이터에 2x2 크기의 컨볼루션 필터를 적용시켜 보면 아래와 같이 된다.



기존 데이터를 필터의 크기에 맞게 2x2로 나누고 같은 자리에 있는 수끼리 곱하여 이들을 모두 더한 합을 특징맵에 해당하는 칸에 넣는다. 예로 위 사진에서 노란 박스안에 들어있는 1, *1*, **2**, 1은 각각 필터의 수인 1, *2*, **3**, 4와 곱해져서 1×1 + *1×2* + **2×3** + 1×4 = 13이 된다.

III. CNN 공부/실습

CNN을 직접 사용해보기 위해 앞서 말한 도서인『머신 러닝 교과서 with 파이썬, 사이킷런, 텐서플로』를 읽어보며 코드를 실습해보며 공부하였다.

1. 이산 합성곱 수행(p.623~31)

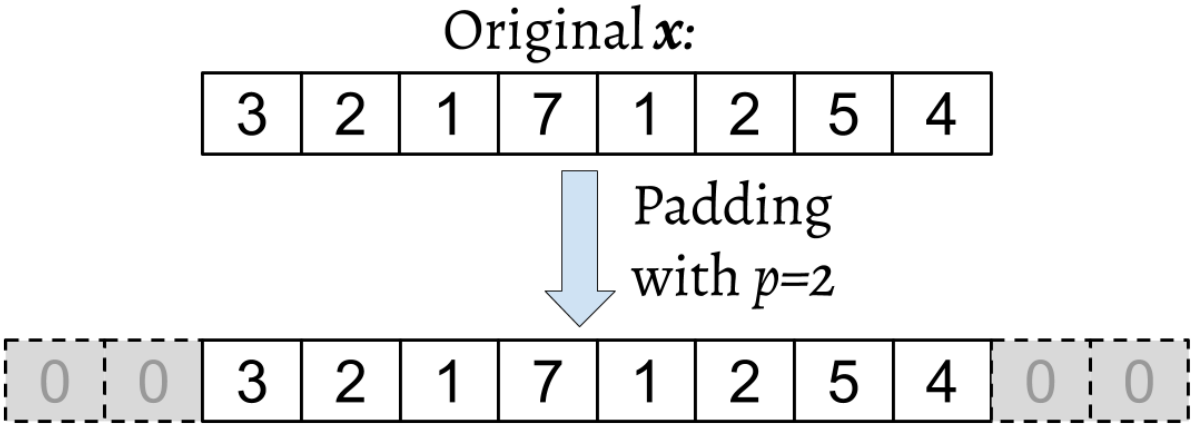
이산합성곱은 두 개의 벡터 x와 w에 대해 앞에서 조사한 바와 같이

$y = x * w$로 나타낸다. 이때 x는 입력(또는 신호), w는 필터 또는 커널이라고 부른다. 이산합성곱의 수학적 정의는

$$y = x * w \rightarrow y[i] = \sum_{k=-\infty}^{+\infty} x[i-k]w[k]$$

이다. 대괄호 [ ]는 벡터 원소의 인덱스를 나타내는 데 사용된다.

여기서 특이한 점이 두가지가 존재한다. 첫번째로 인덱스 k 의 범위가 $-\infty$에서 $+\infty$ 이다. 머신 러닝에서는 항상 유한한 특성 벡터만을 다루는데 합성곱의 범위가 무한이다. 예로 x 를 [0, 1, 2, ..., 8, 9]로 총 10 개의 특성을 가지고 있다 하면 $[-\infty$:-1]과 [10: $+\infty]$ 인덱스는 x 범위 밖이 된다. 이러한 문제를 해결하기 위해 x 와 w 그리고 y 를 0 으로 채워 무한한 크기로 만든다. 이를 제로 패딩(zero padding)이라 한다. 실제로는 무한한 인덱스가 유용하지 않으므로 유한한 개수로 채우게 되며 각 방향으로 추가된 이 개수를 p 로 나타낸다. 1 차원 벡터 x 의 p=2 인 패딩을 예로 들면

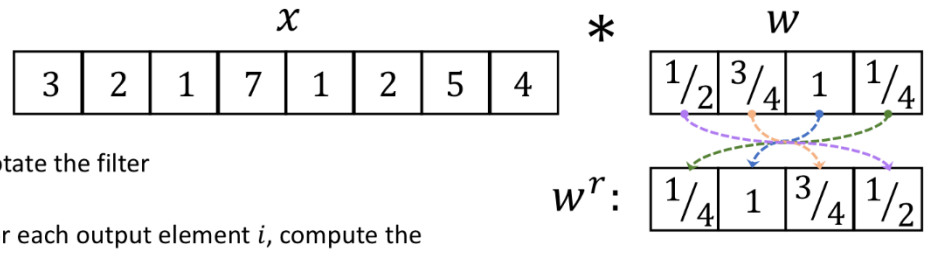


와 같이 된다.

원본 입력 x 와 필터 w 가 각각 n, m 개의 원소를 가지고 m<=n, 패딩 크기는 p 라 하면 패딩된 벡터 x^p 의 크기는 n+2p 이다. 이산 합성곱을 계산하기 위한 실제 공식은

$$y = x * w \rightarrow y[i] = \sum_{k=0}^{k=m-1} x^p[i+m-k]w[k]$$

로 바뀌게 된다. 이로서 인덱스가 무한인 문제가 해결되었다. 두번째 문제는 i+m-k 로 x 를 인덱싱 하는 것 이다. x 와 w 가 다른 방향으로 인덱싱 하게 된다. 이러한 것은 패딩 된 후의 x 또는 w 중 하나를 뒤집어서 계산하는 것과 동일하다. 필터 w 를 뒤집어서 w^r 을 얻었다 할 때 점곱 x[i:i+m]w^r(x[i:i+m]는 크기가 m 인 x 의 패치)을 계산하면 y[i]원소가 하나 얻어지게 된다. 이 연산을 모든 출력 원소를 얻기 위해 슬라이딩 윈도우(특정 크기의 구간을 나누어 순차적으로 수행하는 방식[7])방식으로 반복된다.예를 들어 x=(3, 2, 1, 7, 1, 2, 5, 4) 이고 w=(1/2, 3/4, 1, 1/4), p=0 일 때 처음 3 개의 출력 원소를 계산하면,



Step 1: Rotate the filter

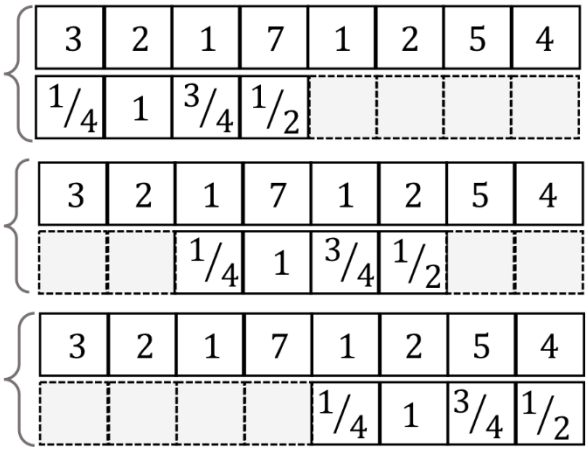
Step 2: For each output element i, compute the dot-product $x[i:i+4].w^r$

(move the filter two cells)

$$y[0] = 3\times{}^1/_4 + 2\times1 + 1\times{}^3/_4 + 7\times{}^1/_2$$
$$\rightarrow y[0] = 7$$

$$y[1] = 1\times{}^1/_4 + 7\times1 + 1\times{}^3/_4 + 2\times{}^1/_2$$
$$\rightarrow y[1] = 9$$

$$y[2] = 1\times{}^1/_4 + 2\times1 + 5\times{}^3/_4 + 4\times{}^1/_2$$
$$\rightarrow y[2] = 8$$



[7] 슬라이딩 윈도우 프로토콜, 데이터 통신과 컴퓨터 네트워크

가 된다. 이때 회전된 필터 w^r 은 2 칸씩 이동하며 이 이동하는 양을 스트라이드(stride)라고 한다.

합성곱 출력 크기는 입력 벡터 위를 필터 w 가 이동한 전체 횟수로 결정된다. 입력 벡터의 크기 n, 필터 크기 m, 패딩 p, 스트라이드 s 인 $x * w$출력 크기 o 는

$$o = \left\lfloor \frac{n + 2p - m}{s} \right\rfloor + 1$$

이다. 이때 $\lfloor \square \rfloor$는 버림 연산을 나타낸다.

1 차원 합성곱 계산 방법을 익히기 위해 넘파이와 비교하여 실습코드를 직접 실행해보았다.

```python

def conv1d(x, w, p=0, s=1):
    w_rot = np.array(w[::-1])
    x_padded = np.array(x)
    if p > 0:
        zero_pad = np.zeros(shape=p)
        x_padded = np.concatenate(
            [zero_pad, x_padded, zero_pad])
    res = []
    for i in range(0, int((len(x_padded) - len(w_rot)) / s) + 1, s):
        res.append(np.sum(
            x_padded[i:i+w_rot.shape[0]] * w_rot))
    return np.array(res)


## 테스트:
x = [1, 3, 2, 4, 5, 6, 1, 3]
w = [1, 0, 3, 1, 2]

print('Conv1d 구현:',
    conv1d(x, w, p=2, s=1))

print('넘파이 결과:',
    np.convolve(x, w, mode='same'))
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Conv1d 구현: [ 5. 14. 16. 26. 24. 34. 19. 22.]
넘파이 결과: [ 5 14 16 26 24 34 19 22]
```

실행 결과는

Conv1d 구현: [5. 14. 16. 26. 24. 34. 19. 22.]

넘파이 결과: [5 14 16 26 24 34 19 22]

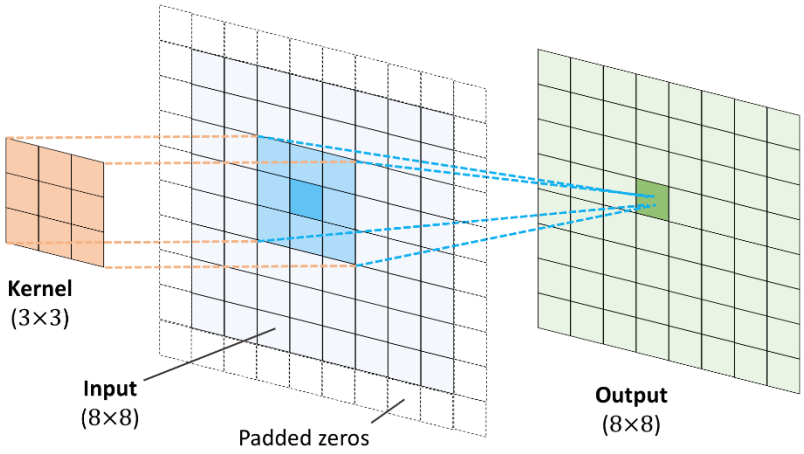
이다.[8]

이번엔 2D 이산 합성곱을 수행해보자. $m_1<=n_1$ 이고 $m_2<=n_2$ 인 행렬 X_{n1*n2} 와 필터 행렬 W_{m1*m2} 의 2D 합성곱 결과는 행렬 Y= $X * W$ 이며 수학적으로 정의하면

$$Y = X * W \rightarrow Y[i,j] = \sum_{k1=-\infty}^{+\infty} \sum_{k2=-\infty}^{+\infty} x[i-k1, j-k2]w[k1,k2]$$

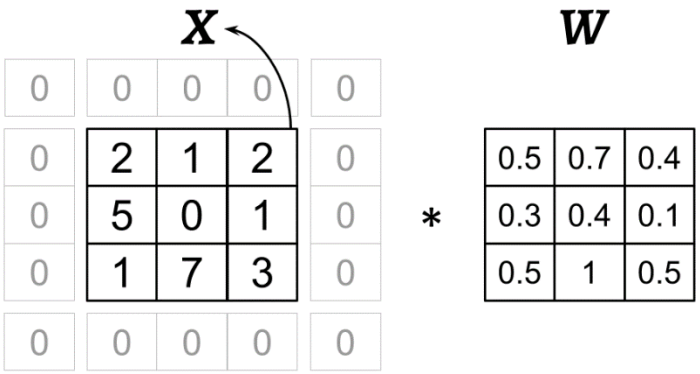
이다. (k1 = k_1, k2 = k_2 이다. 이하 밑첨자는 일반적인 보통문자로 작성한다.)

차원 하나를 줄이면 남은 공식은 이전의 1 차원 합성곱과 동일하다. 따라서 앞에서 한 제로 패딩, 행렬 회전, 스트라이드 등도 모두 적용가능하다. 다음 예는 크기가 8x8 인 입력 행렬과 3x3 의 필터를 사용한 2 차원 합성곱이다. P=1 이며 풀력의 크기는 8x8 이다.



다음 예는 p=(1, 1), s=(2, 2) 일 때 입력 행렬 X3*3 과 커널 행렬 W3*3 사이의 2 차원 합성곱 계산이다. 여기서는 입력행렬 네 면에 0 이 한줄씩 추가되므로 패딩된 행렬 X^p_{5*5} 를 만든다.

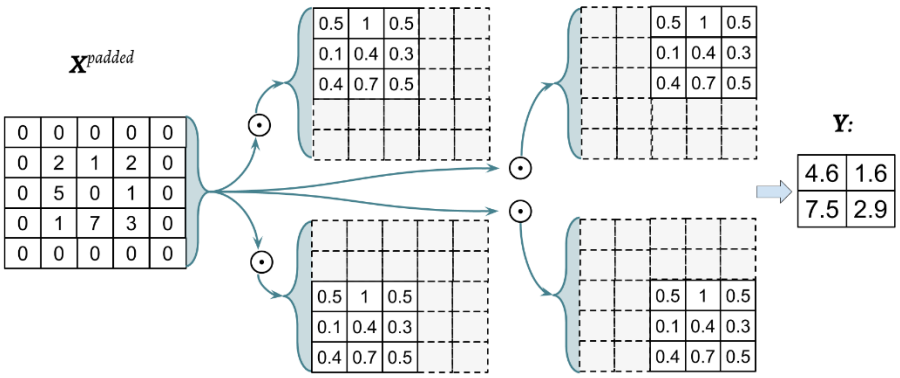
[8] https://github.com/jjy0809/catSpeciesCNNpy/blob/main/conv1d.py



W 필터를 뒤집으면

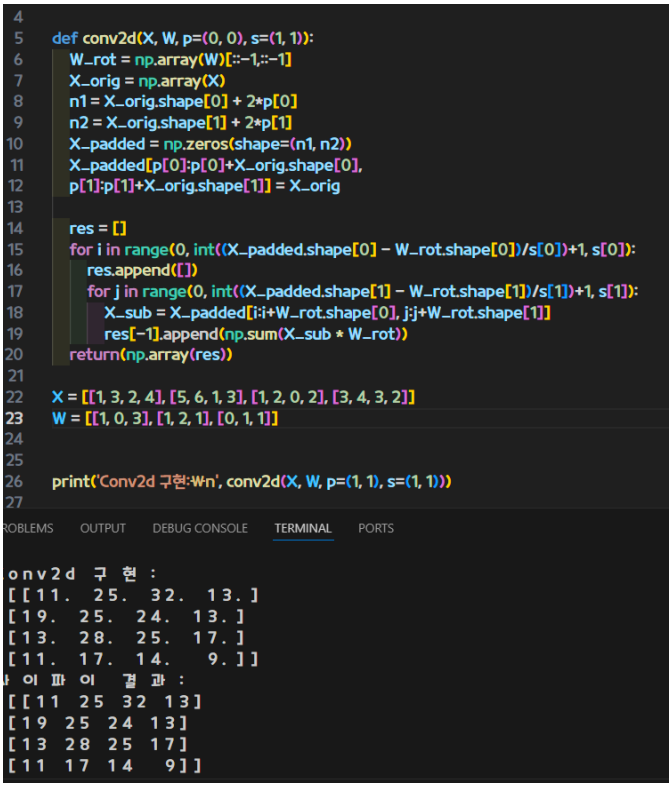
$$W^r = \begin{bmatrix} 0.5 & 1 & 0.5 \\ 0.1 & 0.4 & 0.3 \\ 0.4 & 0.7 & 0.5 \end{bmatrix}$$

가 된다. 아래는 슬라이딩 윈도우 방식을 이용해 W^r을 이동시켜 가며 원소별 곱의 합을 계산하는 과정이다.



결괏값 Y는 2x2 행렬이 된다.

1 차원때와 마찬가지로 2 차원도 코드를 통해 실습해보고 사이파이와 비교해보았다.



```
def conv2d(X, W, p=(0, 0), s=(1, 1)):
    W_rot = np.array(W)[::-1,::-1]
    X_orig = np.array(X)
    n1 = X_orig.shape[0] + 2*p[0]
    n2 = X_orig.shape[1] + 2*p[1]
    X_padded = np.zeros(shape=(n1, n2))
    X_padded[p[0]:p[0]+X_orig.shape[0],
    p[1]:p[1]+X_orig.shape[1]] = X_orig

    res = []
    for i in range(0, int((X_padded.shape[0] - W_rot.shape[0])/s[0])+1, s[0]):
        res.append([])
        for j in range(0, int((X_padded.shape[1] - W_rot.shape[1])/s[1])+1, s[1]):
            X_sub = X_padded[i:i+W_rot.shape[0], j:j+W_rot.shape[1]]
            res[-1].append(np.sum(X_sub * W_rot))
    return(np.array(res))

X = [[1, 3, 2, 4], [5, 6, 1, 3], [1, 2, 0, 2], [3, 4, 3, 2]]
W = [[1, 0, 3], [1, 2, 1], [0, 1, 1]]


print('Conv2d 구현:₩n', conv2d(X, W, p=(1, 1), s=(1, 1)))
```

```
ROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

onv2d 구 현 :
[[11.  25.  32.  13.]
[19.  25.  24.  13.]
[13.  28.  25.  17.]
[11.  17.  14.   9.]]
ㅏ 이 파 이  결 과 :
[[11 25 32 13]
[19 25 24 13]
[13 28 25 17]
[11 17 14  9]]
```

출력 결과는

Conv2d 구현:

[[11. 25. 32. 13.]

[19. 25. 24. 13.]

[13. 28. 25. 17.]

[11. 17. 14. 9.]]

싸이파이 결과:

[[11 25 32 13]

[19 25 24 13]

[13 28 25 17]

[11 17 14 9]]

이다.[9]

이하 서브 샘플링, 이미지 파일 읽기, 채널, 특성 맵, 드롭 아웃, 과소(대)적합, 손실 함수, 케라스(keras), Adam 옵티마이저 등은 시간상 생략하며 아래 활동에서 중간중간 설명될 예정이다.

[9] https://github.com/jjy0809/catSpeciesCNNpy/blob/main/conv2d.py

IV. 고양이 종 분류 CNN 제작

이제 앞에서 배운 내용들을 토대로 고양이 이미지를 보고 종을 분류해주는 합성곱 신경망을 만들어보겠다.파이썬과 텐서플로를 이용하였으며 코드를 짤 때 앞에서 사용한 책과 ChatGPT를 참고하였다.

1. 데이터 수집

CNN을 만들기에 앞서 우선 고양이 사진을 수집해야한다.

고양이의 종은 많으므로 그 중 한국에서 인기있는 10종을 골랐다.

- *코리안 쇼트헤어 (Korean Shorthair)*
- *페르시안 (Persian)*
- *샴 (Siamese)*
- *메인쿤 (Maine Coon)*
- *러시안 블루 (Russian Blue)*
- *벵갈 (Bengal)*
- *스코티시 폴드 (Scottish Fold)*
- *브리티시 쇼트헤어 (British Shorthair)*
- *아메리칸 쇼트헤어 (American Shorthair)*
- *랙돌 (Ragdoll)*

이 10 개의 종에 대해 각 종마다 고양이 이미지를 수집할 것이다. 데이터 수집을 직접 하기는 어려우므로 파이썬을 이용하여 자동으로 수집하였다.

구글[10] / 야휴, 빙, 얀덱스[11] / 네이버[12]에서 사진들을 가져왔다.

총 8610 장의 이미지를 수집하였다. 수집된 이미지는 학습 데이터(train)과 검증 데이터(validation) 그리고 테스트 데이터(test)로 구분 하였으며 이들은 모두 구글드라이브에 업로드 해두었다.

2. 데이터 학습

본격적으로 CNN 을 만들기 위해 코드를 짰다[13][14]. 이후 여러 번의 학습을 진행하였으며 정확도를 높이고 손실은 낮추기 위해 에포크(epoch)와 배치(batch)크기 등을 조정하고 드롭 아웃, 조기 종료 콜백, 학습률 감소 콜백 등의 기능을 추가해가며 학습했다.

3. 결과

학습 데이터에 대해 정확도는 약 50%대, 손실은 약 1.2 정도였으며 검증 데이터에 대해 정확도는 약 40%대, 손실은 1.6 정도였다. 정확도가 낮은 이유로는 학습 데이터의 부족이라 생각된다. 또한 학습 데이터의 정확도와 손실이 검증에 비해 비교적 우수한걸로 보아 과대적합 현상도 발생한 것으로 보인다. 테스트에서는 각 종별로 테스트를 진행하였으며 결과를 ChatGPT 를 이용하여 분석하였다. 아래는 각 종별 및 전체 정확도이다.

- *American Shorthair Accuracy: 45.24%*
- *Bengal Accuracy: 70%*
- *British Shorthair Accuracy: 47.73%*

---

[10] https://github.com/jjy0809/catSpeciesCNNpy/blob/main/googleImage.py

[11] https://github.com/jjy0809/catSpeciesCNNpy/blob/main/yahooYandexBingImages.py

[12] https://github.com/jjy0809/catSpeciesCNNpy/blob/main/naverImage.py

[13] https://github.com/jjy0809/catSpeciesCNNpy/blob/main/cat_cnn.py (모델 학습 코드)

[14] https://github.com/jjy0809/catSpeciesCNNpy/blob/main/cat_species.py (예측 코드)

- *Korean Shorthair Accuracy: 3.45%*
- *Maine Coon Accuracy: 41.30%*
- *Persian Accuracy: 60.98%*
- *Ragdoll Accuracy: 28.21%*
- *Russian Blue Accuracy: 54.35%*
- *Scottish Fold Accuracy: 8.11%*
- *Siamese Accuracy: 75%*
- *Overall Accuracy: 44.75%*

전체적으로 정확도가 45%대에 있으며 특히 코리안쇼트헤어와 스코티시폴드에 대해 정확도가 현저히 낮음을 확인 할 수 있다. 그 까닭은 코리안쇼트헤어는 데이터가 타 종에 비해 50%정도로 적었으며 스코티시폴드는 다른종들과의 특별한 구분점이 없는것으로 보인다.

전체 코드 및 결과 그래프, 최종 학습 모델은 깃허브(코드, 모델)와 구글드라이브(결과, 정확도/손실 그래프)에 업로드 해두었다.

V. 결론

합성곱 신경망 CNN에 대해 탐구하고 직접 고양이 종을 예측하는 CNN을 만들어 보았다. CNN을 이해하는 과정부터 코드를 짜고 데이터를 수집하고 모델을 학습하고 정확도를 개선시키고 결과를 분석하는 모든 과정 중간 중간 수많은 어려움과 오류 등으로 인해 힘들었지만 CNN이란 흥미로운 개념을 직접 탐구하고 사용해본다는 것에 재미를 느끼며 거의 1주 가까이 열심히 공부하고 코드를 짜고 수많은 오류들을 고치며 이번 탐구를 진행하였다. 시간이 부족하여 탐구한 내용을 다 적지 못한 것이 아쉽다. 또한 학습 데이터 부족으로 정확도가 낮은 것이 아쉽지만 그래도 목표 정확도인 50%에 근접하였다. 추후 더 많은 데이터를 수집하고 사용할 능력이 될 때 더 많은 고양이 종에 대한 CNN을 직접 만들어보고 싶다.