CNN Model

* Convoulutional Neural Networks의 약자
* 이미지나 영상 데이터를 처리할 때 사용
* 전처리작업이 들어가는 Neural Network 모델
* DNN 모델의 문제점에서 출발
* 이미지를 날 것으로 그대로 받음으로써 공간적/지역적 정보를 유지한 채 특성들의 계층을 빌드업
* 전체 보다는 부분을 살리고 이미지의 한 픽셀과 주변 픽셀들의 연관성을 살림

Ex) 새의 이미지를 판단할 때 부리를 이용하는데 사진마다 새의 부리 위치는 다르지만 이미지의 부분을 캐치하여 효율적으로 판단 가능

* 이상행동에 관련해서는 부분 이미지(사람의 행동 이미지)만 가지고 하면 될지도?
* 컬러이미지는 RGB 세가지의 채널로 3차원 이미지로 order-3텐서라고 칭함
* 익숙한 2차원의 이미지는 order-2텐서(Matrix)라고도 칭함
* 작동 원리

: 2차원의 이미지 예시

1. 픽셀 단위로 구성
2. Matrix(행렬)로 표현할 수 있음
3. CNN에 넣어줄 입력값은 Matrix로 표현된 이미지
4. CNN에는 필터가 존재
5. 이미지에 필터를 곱하여 필터 Matrix 크기의 결과값이 나옴

🡺Stride(필터를 우측으로 얼마나 움직이나)을 조절

🡺Stride값이 클수록 건너뛰는 칸이 커지면서 결과값의 이미지가 작아짐

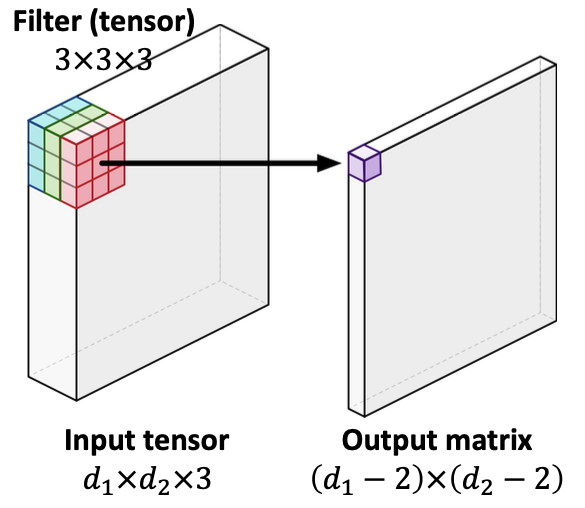
1. 하지만 이미지에 필터를 씌우면 결과가 줄어들면서 손실 발생

🡺Padding을 통해 손실 해결 (0으로 구성된 테두리를 이미지에 감싸줌)

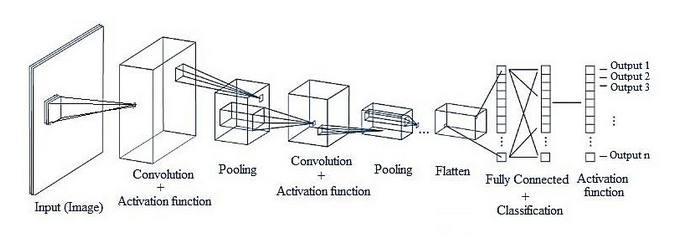
🡺입력값과 결과값의 크기가 같아지면서 손실이 줄어들었음

: 3차원의 이미지

1. R, G, B의 값 3\*3\*3의 3차원이 된 경우 Filter을 이용하여 matrix로 바꿔줌



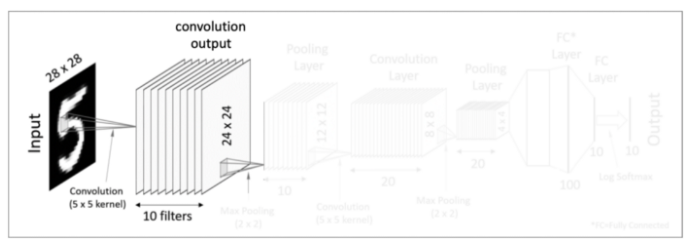
CNN 모델의 네트워크 구조



* 완전연결계층(이전 계층의 모든 뉴런과 결합되어 있는 Affine계층으로 구현)과는 다름

모델의 계층별 설명

* 첫번째 convolution layer



1. 입력값 28\*28크기의 이미지
2. 여러 개의 필터(커널) 사용해 결과값 얻기
3. 한 개의 28\*28 이미지 입력값에 10개의 5\*5 필터를 사용해 10개의 24\*24 matrics 합성곱 결과값을 만들어냄
4. 해당 결과값에 활성화함수(ex. ReLU) 적용
5. 첫번째 합성곱 레이어 완성
6. 합성곱 레이어 구성: 합성곱 처리와 활성화 함수

\*활성화 함수는 선형함수인 convolution(합성곱)에 비선형성을 추가하기 위해 사용

* 첫번째 Pooling Layer

1. Convolution 과정을 통해 많은 수의 결과값(이미지)들이 생성됨
2. 한 개의 이미지에서 10개의 결과값이 도출되어 값이 너무 많아짐
3. 이를 위해 Pooling이라는 방법을 사용(예시에선 2\*2 Pooling 사용)

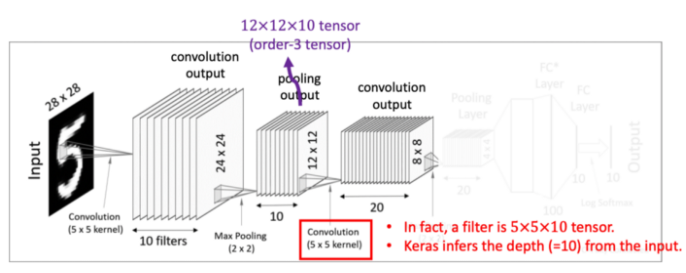
Pooling은 각 결과값의 dimentionality를 축소해주는 것을 목적으로 둠

Correlation이 낮은 부분을 삭제해 각 결과값을 크기를 줄이는 과정

Pooling에는 Max pooling(1)과 Average pooling(2)이 존재

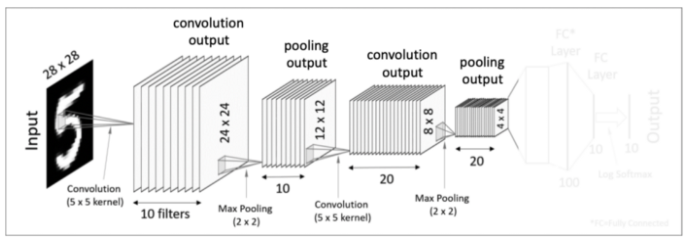
1. Pooling의 크기에 따라서 해당 크기의 Matrix 내의 가장 큰 값을 가져옴
2. Pooling의 크기에 따라서 해당 크기의 Matrix 내의 평균값을 가져옴
3. Pooling을 사용해 결과값이 10개의 12\*12 Matrix로 바뀜

* 두번째 Convolution Layer



1. 이전의 pooling layer에서 얻어낸 12\*12\*10 텐서(12\*12의 Matrix 10개 order-3 tensor)를 대상으로 5\*5\*10 크기의 텐서 필터 20개 사용
2. 8\*8 크기를 가진 결과값 20개 생성

* 두번째 Pooling Layer



1. Pooling 과정 처리 (2\*2 Matrix)
2. 20개의 4\*4 생성

* Flatten

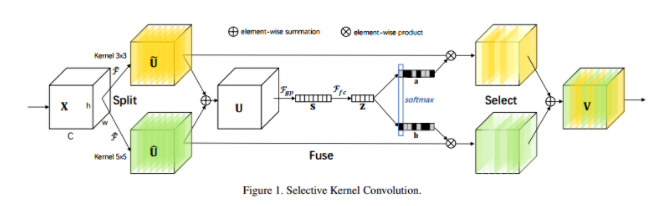
1. 4\*4\*20의 텐서를 일자 형태의 데이터로 쭉 펼쳐주는 과정을 Flatten 또는 Vectorization이라 함
2. 각 세로 줄을 일렬로 쭉 세워서 320-dimension을 가진 Vector의 형태로 만들어냄
3. 4\*4 크기의 이미지는 이미지 자체가 아닌 입력된 이미지에서 얻어온 특이점 데이터
4. 즉 1차원 데이터로 변형시켜도 무관한 상태가 됨

* Fully-Connected Layers(Dense Layers)

1. 하나 혹은 하나 이상의 Fully-Connected Layer을 적용시키고 마지막에 Softmax 활성화함수를 적용하면 최종 결과물 출력

SK-CNN 모델

* Kernel\_size(Padding을 할 때 사용하는 것)를 고정시키지 않고 입력 이미지의 정보에 따라 크기가 변화
* 입력 이미지 내의 객체 크기가 커지면 kernel\_size도 마찬가지고 커짐
* Split, fuse, select 세가지 연산으로 이루어져 있음
* 여러 kernel\_size를 지닌 합성곱 연산을 수행하고 입력값에 따라 각 합성곱 연산의 중요도 가치를 계산해 다양한 kernel\_size를 사용하는 효과 제공



1. Split
   1. 입력 피쳐 맵에 m개의 convolution 연산 수행
   2. 위에서는 m=2이며, 3\*3, 5\*5 kernel size를 지닌 합성곱으로 분할
   3. 각 합성곱 연산은 resnext의 group convolution과 xception의 epthwise separable convolution을 적용하며, 후에 batch norm과 relu 연산 사용
   4. Kernel size는 dilated conv나 더 큰 크기의 kernel size를 사용할 수 있음
2. Fuse
   1. 생성된 m개의 피쳐맵을 하나로 통합(element-wise summation 연산 사용) : U
   2. 생성된 피쳐맵에 global average pooling 적용 🡺 채널 단위의 값을 저장하고 있는 C\*1\*1의 벡터 생성 : s
   3. Fc layer로 채널 축소, batch norm과 relu 연산도 함께 적용 : z
3. select
   1. z 벡터에 m개의 1\*1 합성곱 또는 fc 연산을 적용해 m개의 벡터 생성
      1. 기존의 피쳐맵 채널 c로 확장하기 위해
   2. 각 백터에 softmax 연산을 하여 0~1의 값으로 변경
      1. 부모가 두 벡터 값의 합이므로 A에서는 높은 값을, B에서는 낮은 값을 갖는 원소, a + b = 1
   3. Split이 생성한 m개의 feature map에 곱하고 m개의 feature map을 element-wise summation하여 하나의 피쳐맵 생성

테이블이(가) 표시된 사진

자동 생성된 설명