딥러닝 이미지 분류

tensorflow에서 제공하는 샘플 이미지를 다운로드받는다.

(독립,종속), \_ = keras.datasets.mnist.load\_data()

Print(독립.shape,종속.shape)

#(60000,28,28)(60000,)

Mnist는 흑백이미지 인데, 독리변수는 총 6만장, 28\*28크기이다

종속변수. 정답을 의미 하는 6만개 숫자 1차원 배열

(독립,종속), \_ = keras.datasets.cifar10.load\_data()

Print(독립.shape,종속.shape)

#(50000,32,32,3)(50000,1)

Cifar는 컬러이미지 5만장에 32\*32크기, rgb 3비트이다.

정답을 의미하는 5만개숫자, 행이 5만이고 열이 1개인 50000\*1 2차원배열

\_ 부분은 또다른 반환값이 있지만 예제에선 다루지 않으므로 생략한다.

코드 최초 실행시 다운로드를 받고 이후에 다시 실행하면 다운로드 없이 수행된다.

이미지를 한장 출력해본다.

Matplotlib 라이브러리의 모듈 pyplot을 사용하기위해

import matplotlib.pyplot as plt

print(mnist\_y[0:10])

plt.imshow(mnist\_x[0],cmap='gray')

print(cifar\_y[0:10])

plt.imshow(cifar\_x[0])

mnist는 흑백이미지 이므로 cmap =’gray’를 명시해주어야 한다

차원을 확인한다.

import numpy as np

d1 = np.array([1,2,3,4,5])

print(d1.shape)

d2 = np.array([d1,d1,d1,d1])

print(d2.shape)

d3 = np.array([d2,d2,d2])

print(d3.shape)

d4 = np.array([d3,d3])

print(d4.shape)

결과

(5,)

(4, 5)

(3, 4, 5)

(2, 3, 4, 5)

즉 원소가 5개인 1차원 배열 d1,d1을 원소로 4개갖는 2차원 배열d2

D2를 원소로하고 3개를 갖는 3차원 배열 d3, d3를 원소로하는 2개갖는 4차원 배열 d4

이전에 mnist\_y.shape (60000,) 은 숫자 6만개인 1차원 배열

[1,2,3,…60000]

Cifar\_y.shape (50000,1)은 숫자 1개를 원소로 갖는 1차원 배열을 원소로하는 5만개 2차원 배열이다.

[[1],[2],[3]……[50000]]

이미지 학습은 기계학습에서 지도학습 그 중에서도 분류이다.

이미지 학습의 가장 기본적인 딥러닝 모델 cnn

이미지를 (60000,28,28)을 배열 형태(60000,784)로 변환 reshape – 데이터 준비단계

혹은

모델 구성단계에서 flatten레이어를 사용하여 바꾸는 방법이 있다.

(60000,784)의 독립변수가 (60000,10)의 종속변수를 판단할 때 각 784가 각 종속변수들에 대해 가중치로 판단될 때 중간에 히든 레이어(60000,84)를 부여하면 이는 컴퓨터에게 이미지들이 0~9까지의 숫자를 판단하기 위한 특징 84개를 찾아달라고 하는 것이며

인공신경망이 이를 알아서 학습하여 찾아낸다.

이미지에서 어떤 특징이 어느 위치에서 발견되는지에 대한 정보는 매우 유용함

특정한 패턴의 특징이 어디서 나타나는지를 확인하는 도구

“convolution” 합성곱이라고 부른다.

합성곱의 결과는 2차원 형태의 숫자집합으로 만들어짐 – 이미지로 표현 가능

대상으로부터 필터를 통해 특징을 잡아낸 합성곱의 결과를 특징맵이라고 부른다.(feature map)

Convolution 레이어가 2층 추가된다.

H = tf.keras.layers.Conv2D(3,kernel\_size=5, activation=’swish’)(X)

H = tf.keras.layers.Conv2D(6,kernel\_size=5, activation=’swish’)(H)

Convolution 레이어에서 결정해야 할 것은 필터셋을 몇 개 사용할 것인지.

픨터셋의 사이즈를 얼마로 할 것인지 이다.

첫 번째는 5크기의 3개의 필터셋 = 3개의 특징맵 = 3채널의 특징맵

두 번재는 5 크기의6개의 필터셋 = 6개의 특징맵 = 6채널의 특징맵

이후 H.tf.keras.layers.Flatten()(H) 로 6개의 특징맵을 픽셀단위로 한줄로 펼친 후학습한다

convolution연산은 3차원 형태의 관측치를 입력으로 받는다.

X = tf.keras.layers.Input(shape=[28,28,1])

3차원인 이유는 tensorflow 제작자가 그렇게 정했기 때문

아마 컬러이미지가 3차원이기 때문.

따라서 흑백이미지도 독립 = 독립.reshape(60000,28,28,1)로 3차원이미지로 바꿔준다

필터

필터셋은 3차원 형태로 된 가중치의 모음

convoultion구조에서 컴퓨터가 학습하는 건 필터

필터셋 하나는 앞선 레이어의 결과인 특징맵 전체를 본다.

필터셋 개수 만큼 특징맵을 만든다.

H = tf.keras.layers.Conv2D(3,kernel\_size=5, activation=’swish’)(X)

H = tf.keras.layers.Conv2D(6,kernel\_size=5, activation=’swish’)(H)

3차원 필터셋 이라는 것은 개별 필터셋들이 3차원 형태 인 것이다.

즉 첫번째 필터셋이 (3,5,5) 가 아닌

F1 = (5,5,?)

F2 = (5,5,?)

F3 = (5,5,?)

의 각 필터셋이 3차원이 되는 것이다.

?부분은 두번째 특징인 앞선 레이어의 특징맵전체를 본다와 연결된다.

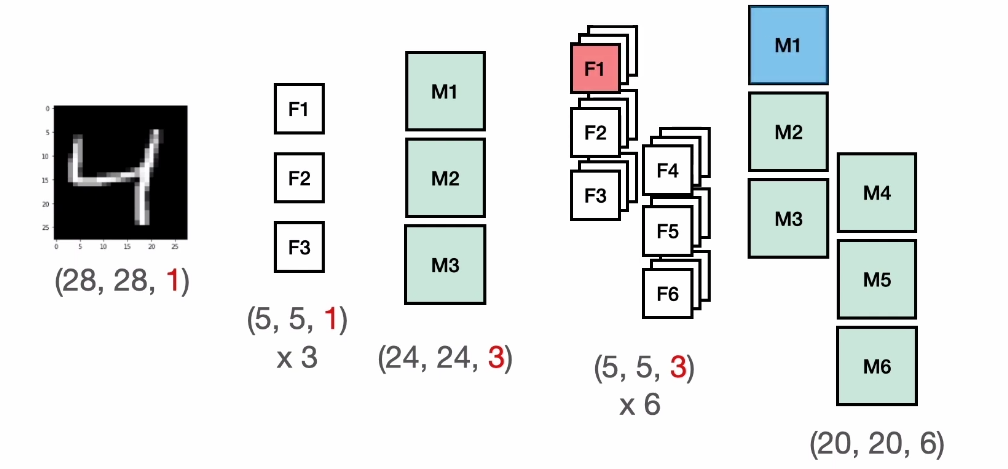
흑백의 경우 1, 컬러의 경우 3 앞선레이어의 채널수가 4 라면 4가 온다.

또한 필터셋 전체의 모양은 4차원이 된다(3(개),5,5,?(특징맵))

필터셋 이라는 말은 이해를 돕기 위한 말이며 실제로는 그냥 필터라고부른다.

필터를 6개 추가하는 행위는 컴퓨터에게 이 이미지들이 0~9까지 중 어느 숫자인지 판단하기 위해 가장 좋은 특징맵 6개를 찾아줘. 라고 하는 것과 같다.

히든레이어에서의 방식과 같다고 보면 된다.



1채널의 이미지에 대해 3개의 필터를 만들고 해당 필터들이 앞 레이어의 전체 특징맵(1)을 참고하여 각각 채널을 만든다

두번째 필터는 6개를 만들고 각 필터가 앞레이어의 전체 특징맵을 참고하여(3) 각각 채널을 만든다.

convolution실습을 하기 전 colaboratory의 런타임 유형변경을 gpu로 설정한다

---

X = tf.keras.layers.Input(shape=[28,28,1])#입력

H = tf.keras.layers.Conv2D(3,kernel\_size=5,activation='swish')(X)#필터

H = tf.keras.layers.Conv2D(6,kernel\_size=5,activation='swish')(H)#필터 총 6개의 채널 생성

H = tf.keras.layers.Flatten()(H)#이미지 데이터를 배열로 변환

H = tf.keras.layers.Dense(84,activation='swish')(H)#히든레이어 적용

Y = tf.keras.layers.Dense(10,activation='softmax')(H)#출력

Flatten이 컴퓨터가 처리할 수 있도록 이미지를 배열로 만드는 것이고,

이후에 배열에 대해서 특징을 찾는 것이 히든레이어라면, flatten이전에 이미지에서 특징을 추출하는 것이 필터라고 볼 수 있다

Maxpool2d

맵의 개수가 증가하면 flatten이후 입력으로 들어갈 colum수가 증가하고

컴퓨터가 찾아야 하는 가중치의 증가를 의미한다

가중치를 작게 유지하기 위해 입력으로 사용될 colum수를 조정하기 위해 사용한다.

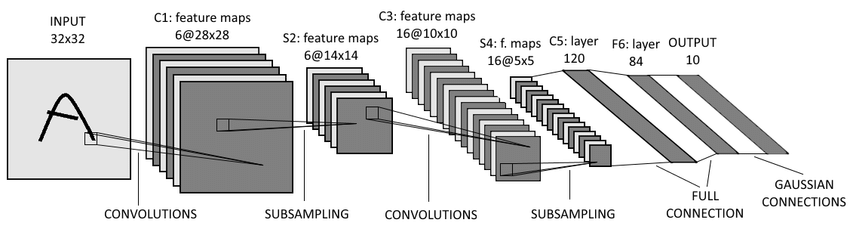
6,6 이미지가 있다고 할 때 영역내에서 가장 큰 값만 남겨 3,3이미지로 만드는 것이다.

평균값을 남기는 것은 averagepooling이 된다.

Maxpooling을 쓰는 이유는 그 대상이 특징 맵이기 때문이다.

값이 크다는 것은 그 특징이 많이 나타난 부분이기 때문이다.

Convoulution에 각 필터 적용이후마다 maxpooling을 하는 것 뿐이지만 가중치가 확연히 차이나는 것을 볼 수 있다



Lenet5 구조를 mnist에 적용해서 구현 해본다.

Lenet5에서는 32비트 이미지를 사용하므로 우리 예제의 28비트 그림이 첫 필터를 거칠 때 28비트를 유지하면 이후 과정이 같아진다. 따라서 첫 필터에 padding속성을 추가한다.

padding속성을 = ‘same’ 으로 하면 필터를 거친 이후에 사이즈에 상관없이 크기를 유지한다.

#cifar10

(독립, 종속), \_ = tf.keras.datasets.cifar10.load\_data()

print(독립.shape,종속.shape)

#독립변수가 이미 4차원이기 때문에 reshape불필요

#mnist는 종속변수가 1차원이라 상관 없었지만, 이번에는 2차원이기 때문에 1차원으로 reshape 해주어야 함

종속 = pd.get\_dummies(종속.reshape(50000))

print(독립.shape,종속.shape)