Lenet-5 Review

Gradient-Based Learning Applied to Document Recognition_1998_Yann_LeCun

Lenet-5는 손으로 적은 우편번호를 보다 전문적인 방법으로 효율적으로 확인하기 위해 고안된 CNN 구조이다.

전통적인 패턴인식 모델은 수작업된(Hand-crafted) 특징 추출기를 통해 Input에서 정보를 수집했으나, input의 사이즈가 커지면서 parameter의 개수가 증가하고, 메모리 사용량도 커지기 시작함

이러한 문제들을 효율적으로 해결하고자 제안한 CNN구조가 Lenet-5이다.

사용된 Architecture의 Idea는 다음과 같다

1. Local Receptive Fields
2. Shared Weights
3. Sub-sampling

Local Receptive Fields(수용 영역)

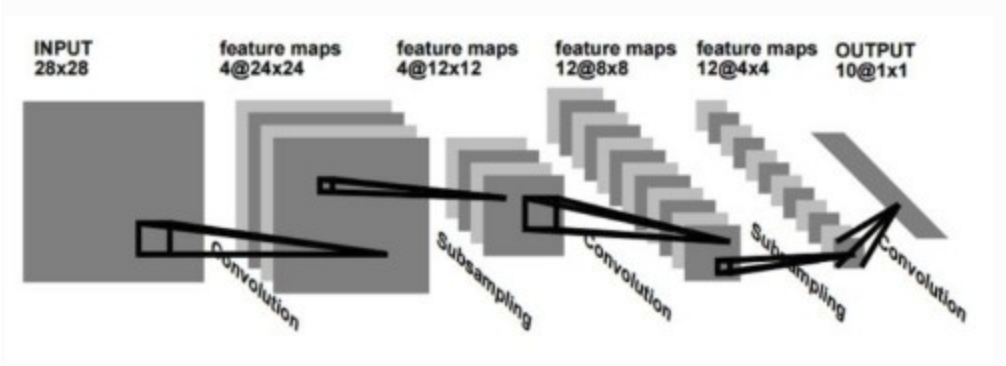
Visual cortex(시각 피질) 의 메커니즘에서 아이디어를 착안,

고양이에게 그림을 보여줬을때, 특정 그림에서만 뉴런이 반응하는것을 모티브로 하여 입력된 이미지의 Local의 특징을 추출하도록 하였다.

이때, Kernel 또는 Filter를 적용한다고 하며, 특정 사이즈로 설정하여 적용 시킨다. 이러한 과정을 Convolution이라고 하며 Filter는 즉 Weight가 된다.

이러한 Convolution이 끝나면 Feature Map을 얻게 되며, Filter를 통해서 network의 다음 layer를 Fully-connected가 아닌 Local하게 구성하여 weight 파라미터의 수도 줄어들게 된다.

Lenet-1의 구조



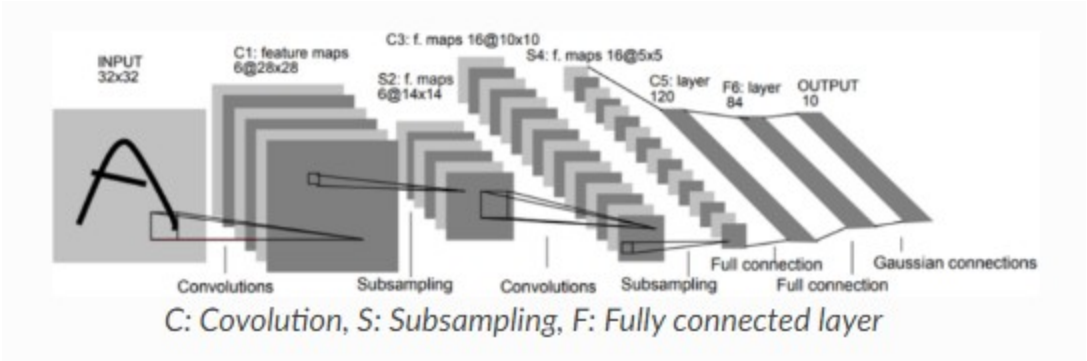
Shared Weights(동일한 가중치 적용)

Input에서 Filter를 활용한 Convolution으로 Feature map을 만드는데 있어, N개의 Filter를 사용하여 N개의 Feature map을 만들게 된다. 또한 모든 이미지에 동일한 Filter를 사용하여 가중치가 동일하게 적용된다.

Subsampling(출력값에서 일부분만 취함)

Pooling과 비슷한 개념으로, 위 논문에서는 average pooling, Convolution된 값의 평균값을 받아오면서 Feature map을 생성한다.

Lenet-5의 구조



Lenet-5는 Convolution Layer 3개, Subsampling Layer 3개, Fully Connected Layer 1개로 구성.

Fully connected layer와 Convolution layer의 차이

Fully connected layer는 Input image를 1차원으로 Flatten 시키는데, 이와 동시에 공간 정보가 손실되고, 정보가 부족해져 특징 추출 및 학습이 비효율적으로 진행될 수 있다.

Convolution layer는 input image의 형상을 최대한 유지하여 공간 정보를 계속 가지고 가며 N개의 필터와 Sub-sampling을 통해 이미지의 Local Feature들을 추출해내고, 이것들이 쌓여 Global Feature를 만든다. 또한 Filter가 전 과정에 있어 동일하기때문에 parameter수가 상대적으로 적다.

계층 별 이해

[Input]

32 x 32 사이즈의 Image를 Input으로 받는다.

[C1]

5 x 5 Filter로 Convolution 28 x 28의 Local Feature Map 6개를 생성한다.

[S2]

2 x 2 Subsampling(Average Pooling)을 통해, 14 x 14의 Feature Map 6개를 생성한다.

[C3]

5 x 5 Filter로 Convolution을 진행하여 10 x 10의 Feature Map 16개를 만든다.

[S4]

2 x 2 의 Subsampling(Average Pooling)을 통해 5 x 5 Feature Map 16개를 생성한다.

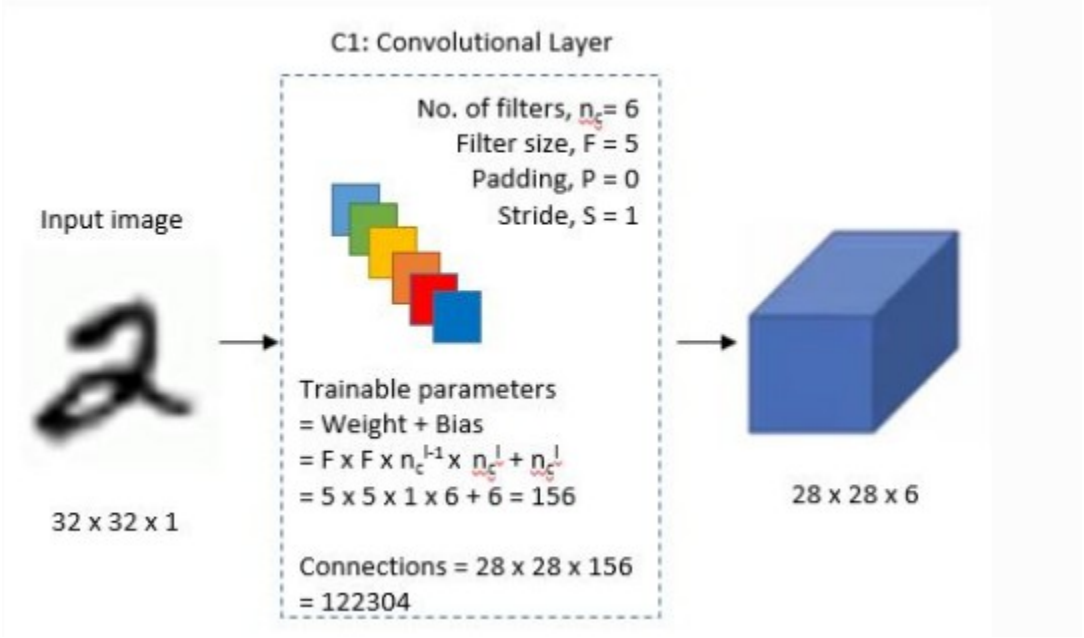
[C5]

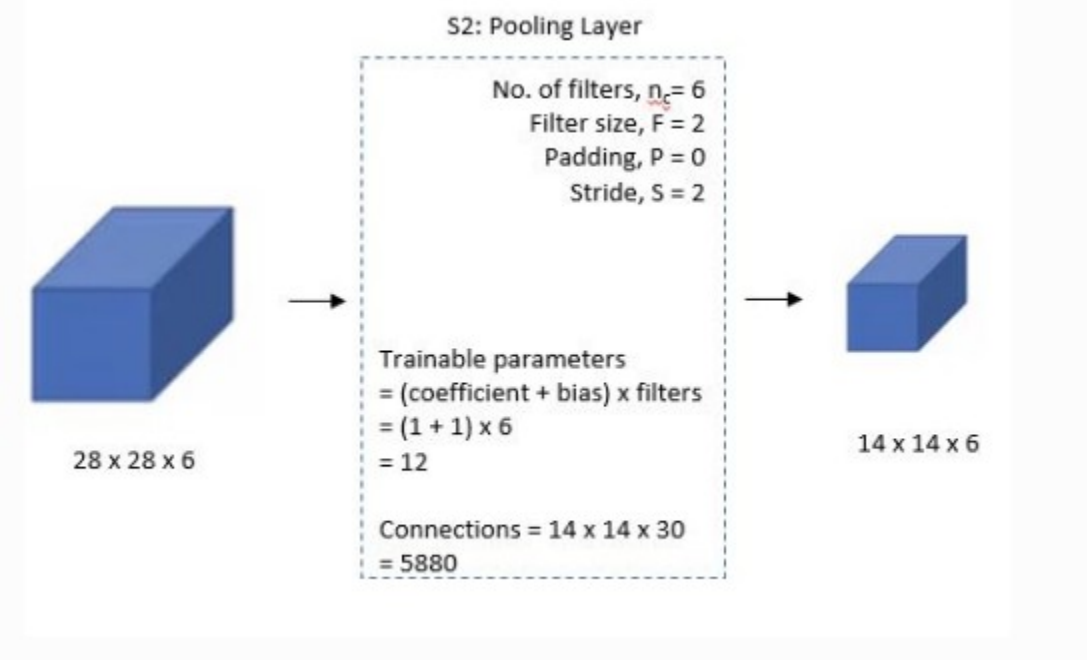
5 x 5 Feature Map을 5 x 5 Filter로 다시 Convolution하여 1 x 1 Feature Map 120개를 생성한다.

[F6]

120개의 Feature Map을 크기가 84인 Fully Connected Layer로 연결하고, 크기가 10은 Layer와 연결하여 10개의 Class를 구분할 수 있게 만든다.

Parameter 개수


C1: Convolutional Layer
No. of filters, n_c= 6
Filter size, F = 5
Padding, P = 0
Stride, S = 1
Input image
Trainable parameters
= Weight + Bias
= F x F x n_c^{l-1} x n_c^{l} + n_c^{l}
= 5 x 5 x 1 x 6 + 6 = 156
Connections = 28 x 28 x 156
= 122304
32 x 32 x 1
28 x 28 x 6


S2: Pooling Layer
No. of filters, n_c= 6
Filter size, F = 2
Padding, P = 0
Stride, S = 2
Trainable parameters
= (coefficient + bias) x filters
= (1 + 1) x 6
= 12
Connections = 14 x 14 x 30
= 5880
28 x 28 x 6
14 x 14 x 6

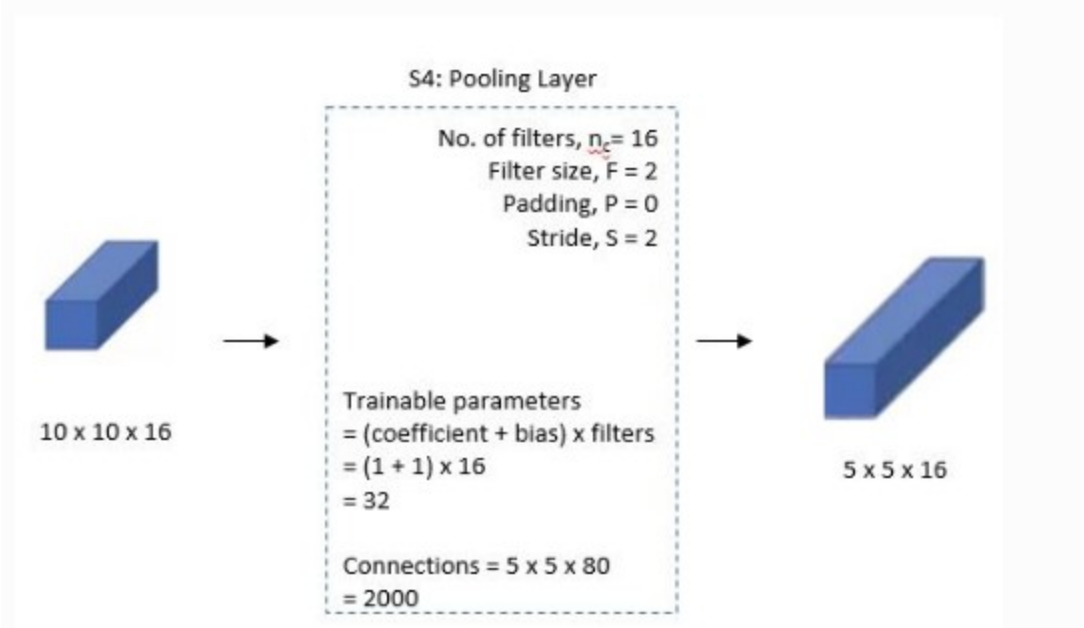
Third layer

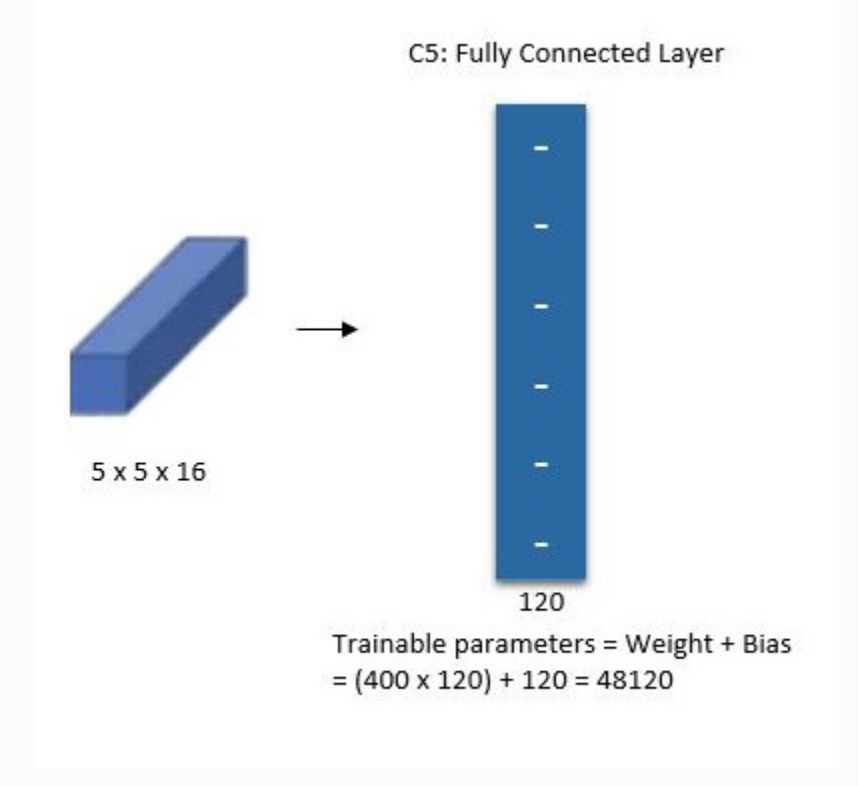
C3는 C1과 유사한데, 6개 Feature map을 모두 16개의 Feature map으로 연결하는 것이 아니라 선택적으로 10개만 연결했다 (아래 그림).

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

TABLE I

EACH COLUMN INDICATES WHICH FEATURE MAP IN S2 ARE COMBINED BY THE UNITS IN A PARTICULAR FEATURE MAP OF C3.





Code로 구현 및 실습(Mnist, Cifar-10)

```
import tensorflow as tf
import pandas as pd
# 데이터를 준비합니다.
(x,y), _ = tf.keras.datasets.mnist.load_data()
# 독립변수 x를 Reshape하기
x = x.reshape(60000,28,28,1)
# 종속변수 y를 one-hot encoding
y = pd.get_dummies(y)
print(x.shape, y.shape)
# 모델 완성하기
X = tf.keras.layers.Input(shape=[28,28,1])
# 6장의 Filter를 사용하고 6개의 Feature map을 만들고, 32x32에서 28x28이 됐으므로 커널사이즈는 5
C = tf.keras.layers.Conv2D(6, kernel_size = 5, padding = 'same',activation='swish')(X)
# Subsampling(MaxPooling으로 진행)
C = tf.keras.layers.MaxPool2D()(C)
# Feature map이 16장이 됐으므로 filter는 16개, 28x28에서 24x24가 됐으므로 커널사이즈는 5
C = tf.keras.layers.Conv2D(16,kernel_size = 5, activation = 'swish')(C)
# Subsampling(MaxPooling으로 진행)
C = tf.keras.layers.MaxPool2D()(C)
# C5 : 120 layer로 펼쳐주기전에 Flatten을 사용하여 펼쳐주기
C = tf.keras.layers.Flatten()(C)
# C5로 120개의 Feature 잡기
C = tf.keras.layers.Dense(120, activation='swish')(C)
# F6 으로 Fully connected layer, 84개 Feature 잡기
```

```
C = tf.keras.layers.Dense(84, activation='swish')(C)
# Output값으로 10개의 Feature(Y) 잡기
Y = tf.keras.layers.Dense(10, activation='softmax')(C)

model = tf.keras.models.Model(X,Y)
model.compile(loss='categorical_crossentropy', metrics='accuracy')
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 28, 28, 1)]       0
_________________________________________________________________
conv2d (Conv2D)              (None, 28, 28, 6)         156
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 16)          0
_________________________________________________________________
flatten (Flatten)            (None, 400)               0
_________________________________________________________________
dense (Dense)                (None, 120)               48120
_________________________________________________________________
dense_1 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_2 (Dense)              (None, 10)                850
=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
_________________________________________________________________
```

```
model.fit(x,y,epochs=10)
Epoch 1/10
1875/1875 [==============================] - 12s 6ms/step - loss: 0.1163 - accuracy: 0.9678
Epoch 2/10
1875/1875 [==============================] - 12s 6ms/step - loss: 0.0777 - accuracy: 0.9793
Epoch 3/10
1875/1875 [==============================] - 12s 6ms/step - loss: 0.0689 - accuracy: 0.9825
Epoch 4/10
1875/1875 [==============================] - 12s 6ms/step - loss: 0.0588 - accuracy: 0.9852
Epoch 5/10
1875/1875 [==============================] - 12s 6ms/step - loss: 0.0597 - accuracy: 0.9857
Epoch 6/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.0587 - accuracy: 0.9875
Epoch 7/10
1875/1875 [==============================] - 12s 7ms/step - loss: 0.0596 - accuracy: 0.9874
```

```
Epoch 8/10
1875/1875 [==============================] - 12s 6ms/step - loss: 0.0582 - accuracy: 0.9879
Epoch 9/10
1875/1875 [==============================] - 12s 6ms/step - loss: 0.0594 - accuracy: 0.9881
Epoch 10/10
1875/1875 [==============================] - 12s 6ms/step - loss: 0.0634 - accuracy: 0.9872
```

```
#Mnist말고 Cifar-10으로 해보기!!!!!!!!!!!!!!!!

(x,y), _ = tf.keras.datasets.cifar10.load_data()
print(x.shape, y.shape)
# y를 one-hot encoding 해주기
# mnist는 표형태가 아니라 1차원이라서 one-hot이 됐었지만 reshape를 먼저해야함
y = pd.get_dummies(y.reshape(50000))

# 모델 완성하기, input shape를 32,32,3로 바꿔주기
X = tf.keras.layers.Input(shape=[32,32,3])
# 6장의 Filter를 사용하고 6개의 Feature map을 만들고, 32x32에서 28x28이 됐으므로 커널사이즈는 5
C = tf.keras.layers.Conv2D(6, kernel_size = 5,activation='swish')(X)
# Subsampling(MaxPooling으로 진행)
C = tf.keras.layers.MaxPool2D()(C)
# Feature map이 16장이 됐으므로 filter는 16개, 28x28에서 24x24가 됐으므로 커널사이즈는 5
C = tf.keras.layers.Conv2D(16,kernel_size = 5, activation = 'swish')(C)
# Subsampling(MaxPooling으로 진행)
C = tf.keras.layers.MaxPool2D()(C)
# C5 : 120 layer로 펼쳐주기전에 Flatten을 사용하여 펼쳐주기
C = tf.keras.layers.Flatten()(C)
# C5로 120개의 Feature 잡기
C = tf.keras.layers.Dense(120, activation='swish')(C)
# F6 으로 Fully connected layer, 84개 Feature 잡기
C = tf.keras.layers.Dense(84, activation='swish')(C)
# Output값으로 10개의 Feature(Y) 잡기
Y = tf.keras.layers.Dense(10, activation='softmax')(C)

model = tf.keras.models.Model(X,Y)
model.compile(loss='categorical_crossentropy', metrics='accuracy')
```

```
model.fit(x,y,epochs=10)
Epoch 1/10
1563/1563 [==============================] - 13s 8ms/step - loss: 2.5520 - accuracy: 0.2577
Epoch 2/10
1563/1563 [==============================] - 13s 8ms/step - loss: 1.5743 - accuracy: 0.4317
Epoch 3/10
1563/1563 [==============================] - 13s 8ms/step - loss: 1.4650 - accuracy: 0.4830
Epoch 4/10
1563/1563 [==============================] - 13s 8ms/step - loss: 1.4075 - accuracy: 0.5075
Epoch 5/10
1563/1563 [==============================] - 13s 8ms/step - loss: 1.3536 - accuracy: 0.5270
Epoch 6/10
```

```
1563/1563 [==============================] - 13s 8ms/step - loss: 1.3274 - accuracy: 0.5334
Epoch 7/10
1563/1563 [==============================] - 12s 8ms/step - loss: 1.3064 - accuracy: 0.5443
Epoch 8/10
1563/1563 [==============================] - 12s 8ms/step - loss: 1.2952 - accuracy: 0.5508
Epoch 9/10
1563/1563 [==============================] - 13s 8ms/step - loss: 1.2756 - accuracy: 0.5580
Epoch 10/10
1563/1563 [==============================] - 13s 8ms/step - loss: 1.2783 - accuracy: 0.5617
```

```
test_loss, test_acc = model.evaluate(x,y,verbose=2)
print('\n테스트 정확도 :', test_acc)
1563/1563 - 5s - loss: 1.3538 - accuracy: 0.5428

테스트 정확도 : 0.5428000092506409
```