08

CNN 을 이용한 CIFAR-10 분류

CNN

CNN 이란 Convolution Neural Network 의 약자로 일반 Deep Neural Network (DNN)에서 이미지나 영상과 같은 데이터를 처리할 때 발생하는 문제점을 보완한 방법입니다. DNN 은 기본적으로 1 차원의 데이터를 사용합니다. 쉽게 생각하면 iris 데이터를 생각해보면 됩니다.

Index	꽃받침길이	꽃받침너비	꽃잎길이	꽃잎너비	꽃의종류(Target)
1	5.1	3.5	1.4	0.2	setosa

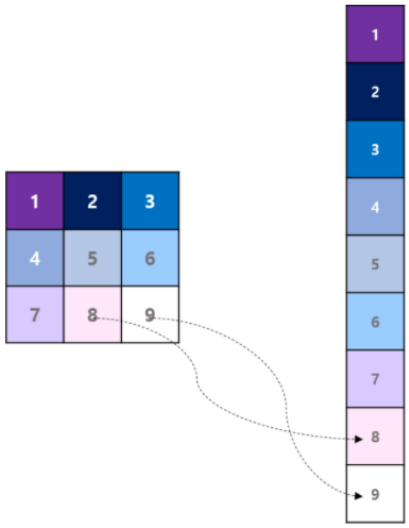
iris 데이터는 이러한 형태의 데이터로 존재하는데 여기서 꽃받침과 꽃잎의 정보는 input 데이터(=X)라 하고, 꽃의 종류는 output 데이터(=Y)로 설정하여 학습하게 됩니다. 그리고 학습된 모델에게 꽃받침과 꽃잎에 대한 정보를 넣음으로써 꽃의 종류를 맞추게 합니다.

하지만 이미지는 어떤가요? 이미지는 위의 iris 데이터처럼 하나의 row 로 표현되지 않습니다. 이렇게 표현하게 되면 이미지 데이터는 큰 손실을 갖게될 수 밖에 없습니다.

이를 테면 위와 같이 강아지 이미지를 한 개의 row 로 표현할 때, 원래 이미지를 예측할 수 없게되는 문제가 발생합니다. 따라서 이미지 데이터는 어떤 픽셀을 중심으로 주변 픽셀들의

연관관계를 유지할 수 있어야 합니다. CNN 에서는 이 이미지의 한 픽셀과 주변 픽셀들의 연관관계를 유지시키며 학습시키는 것을 목표로 합니다.

Convolution

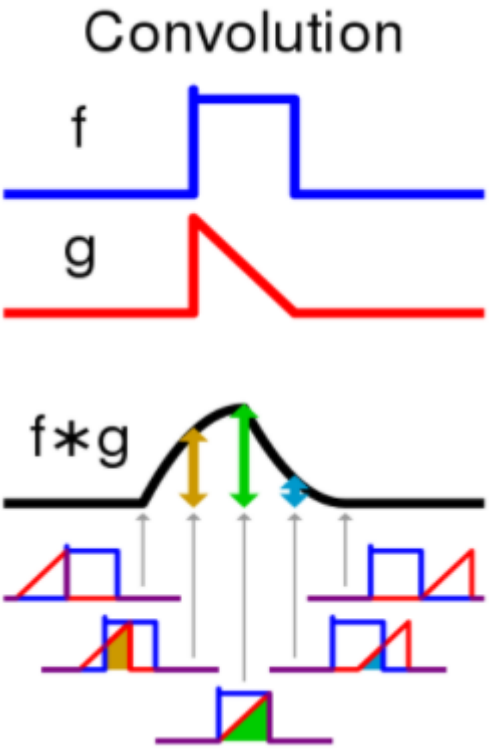


일반적인 DNN 에서 flattening 하여 이미지데이터의 공간적 구조를 무시하게 되면 정확한 분류가 어렵다고 앞선 예시를 통해 설명하였습니다. 이러한 문제는 convolutional layer 의 등장에 큰 동기가 되었습니다.

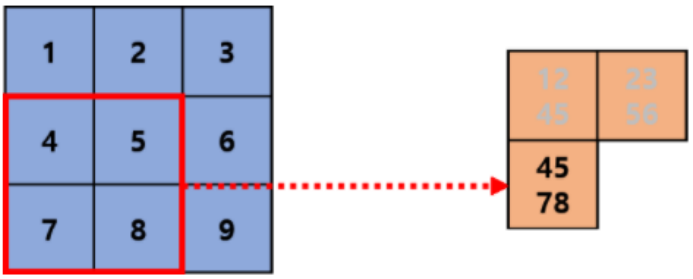
Convolution 은 두 함수(f, g)를 이용해서 한 함수(f)가 나머지 함수(g)에 의해 모양이 수정된 제 3 의 함수 (f*g)로 만들어주는 연산으로 컴퓨터비전, 자연어처리, 이미지 처리 등 다양한 분야에서 이용되는 방법입니다.

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

Convolution 의 정의를 그대로 이해하면 y 축 기준 좌우가 반전된 함수 g 를 우측으로 t 만큼 이동한 함수 g(t-)와 f()가 곱해진 함수의 적분입니다.

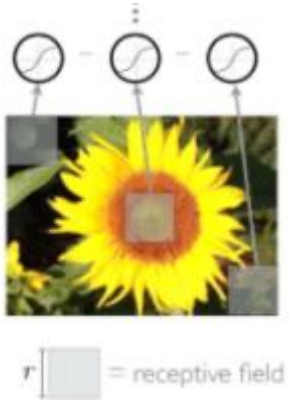


Convolution layer 는 이러한 필터링 기법을 인공신경망에 적용함으로써, 전체를 인식하는 것이 아니라, 일부분을 필터라 불리는 아래의 빨간 상자에 투영하고 그를 우측의 주황색 수용영역(recdptive field)에 연결하여 복합적으로 해석하는 것을 모방한 것입니다. 즉, 아래와 같은 빨간 상자를 sliding window 방식으로 가중치와 입력값을 곱한 값을 더하여 활성함수를 취한 값을 다음 층으로 넘겨줍니다.

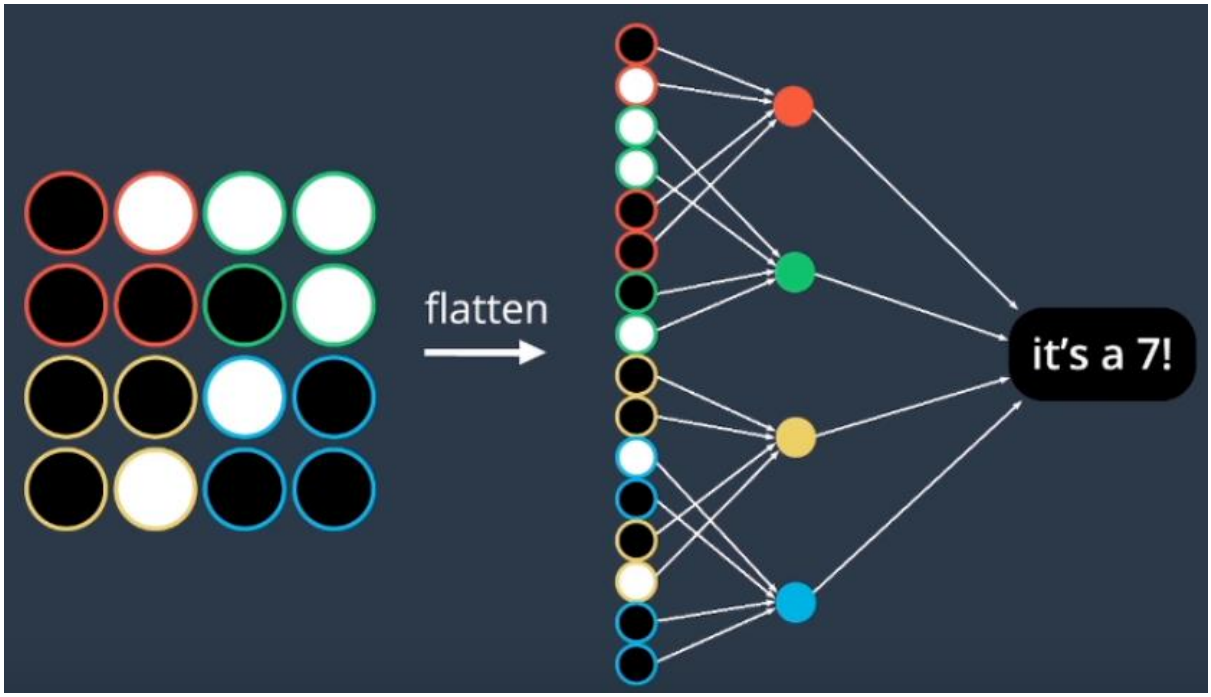


Local connectivity

이미지를 인식하는데 있어서 한 번에 모든 부분을 처리하는 것은 아닙니다. 아래 꽃 사진을 인식하는 과정을 예로 들자면, 꽃의 중심부분인 해바라기 씨를 처리하는 부분, 모서리 부분을 처리하는 부분이 따로 있을 것입니다.



local connectivity 에 대한 보다 자세한 설명을 위해 아래 그림을 참고해보도록 하겠습니다.



위 그림에서 이미지를 빨, 노, 초, 파로 영역을 나누어 담당한 그 영역의 이미지에 대한 패턴을 찾아낼 수 있게 합니다. 그리고 이렇게 영역별 패턴 찾기를 담당한 노드들은 output layer 에게 본인이 담당한 영역에서 찾아낸 패턴을 알려주고, output layer 은 영역 별로 찾아낸 패턴을 조합하여 최종 결과를 찾아냅니다. 이를 local connection 이라고 합니다.

Parameter sharing

CNN 이 이미지를 분류하는 데 있어 parameter 를 공유하는 것이 얼마나 도움이 될 지는 예를 들어보면 쉽게 파악할 수 있습니다.

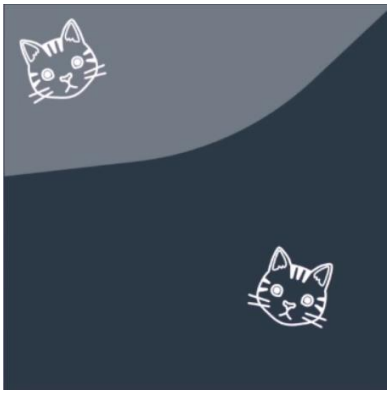
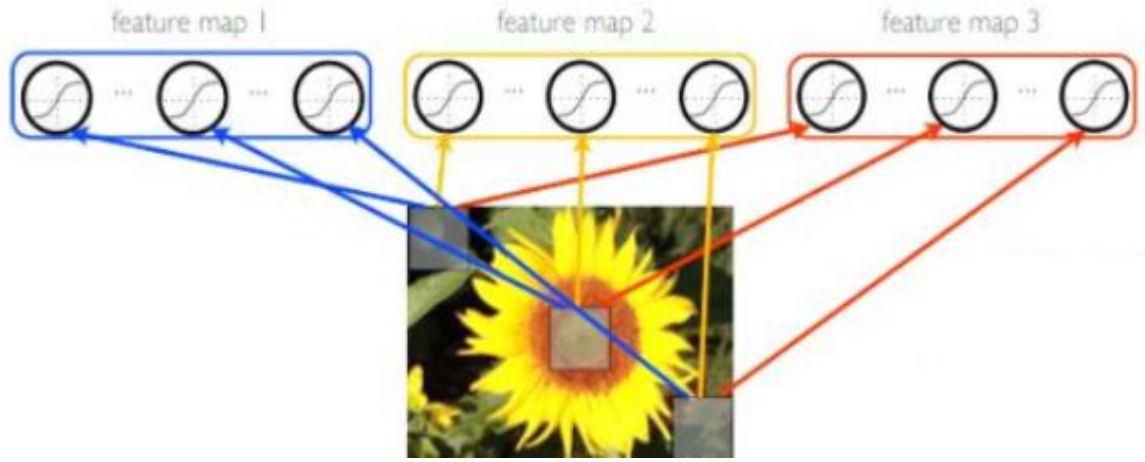


사진 내에 고양이가 독립적으로 임의의 위치에 존재한다고 가정해봅시다. 이 경우 고양이가 그림의 어느 쪽에 있든 위치만 다른 것이지 고양이라는 사실은 동일하다는 것을 네트워크에 명시적으로 알려줘야합니다.

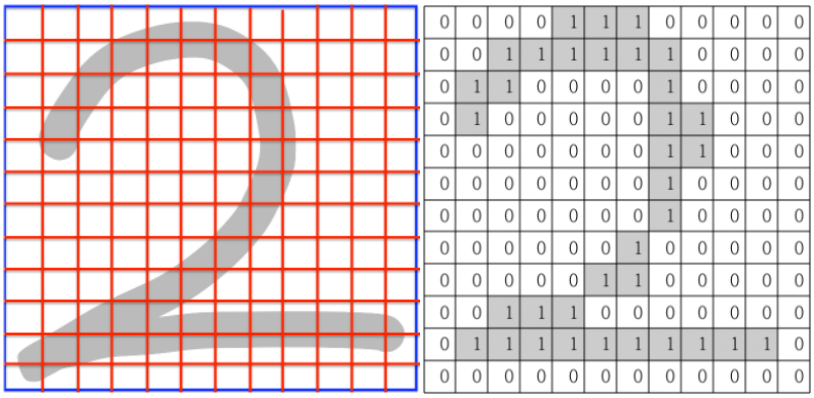
쉽게 이야기하여 같은 정보를 담은 parameter 는 서로 공유하자는 말로 이해할 수 있습니다. 이처럼 동일한 feature 정보를 처리하는 층을 구성하여 해당 층에서는 해당 feature 만 처리하게 하는 것입니다.



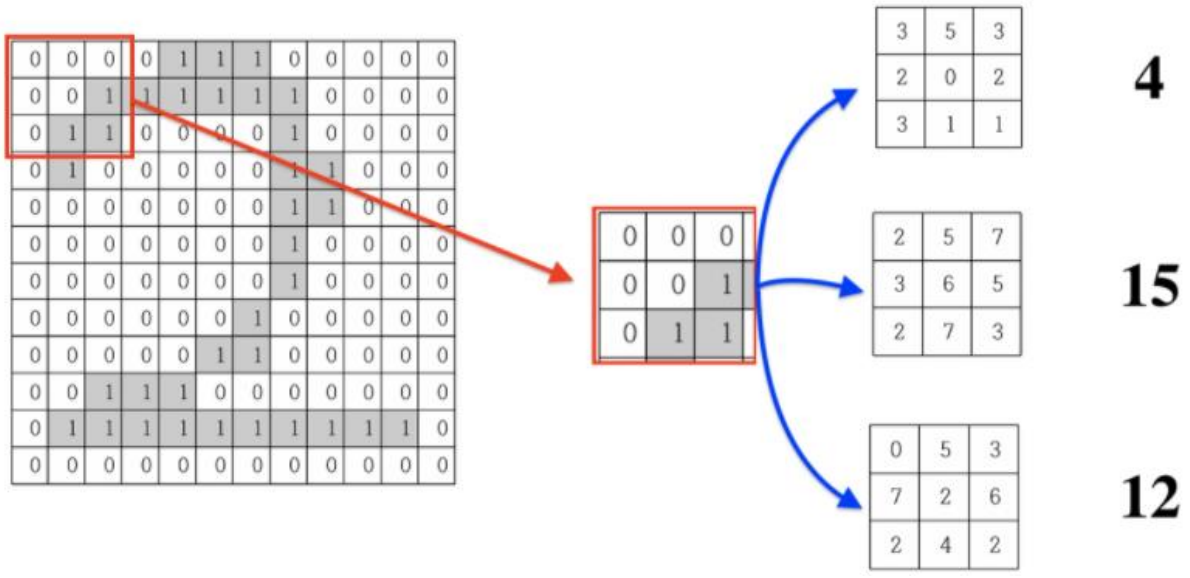
위의 그림에서 처럼 각기 다른 feature 를 처리하는 feature map 이 있고, 같은 map 은 같은 feature 를 공유하여 처리하게 됩니다. 따라서 CNN 은 local connectivity 와 parameter sharing 을 통해 처리해야할 parameter 수를 획기적으로 줄일 수 있게 됩니다.

Filter, Stride

아래는 12x12 크기의 이미지를 숫자인 곳은 1, 아닌 곳은 0 의 픽셀값을 주어 표현한 그림입니다.



CNN 은 일단 하나의 이미지로부터 픽셀 간의 연관성을 살린 여러 개의 이미지를 생성하는 것에서 시작합니다. 방법은 아래와 같습니다. 3x3 의 크기로 이미지를 뽑아내서 마찬가지로 3x3 크기의 랜덤값을 갖고있는 데이터(필터)와 곱하여 더해줍니다.

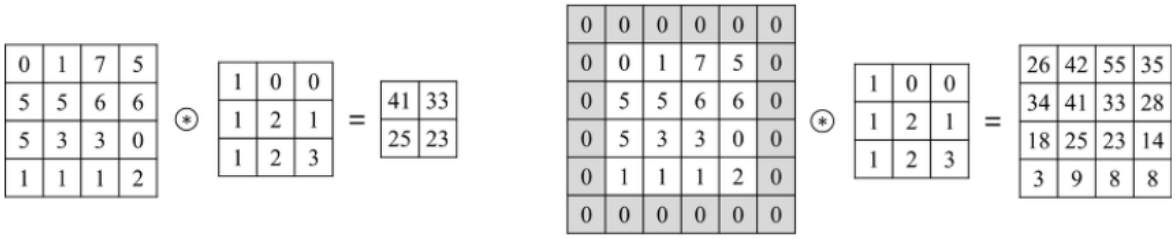


계산해보면 각 필터의 값에 따라서 4, 15, 12 와 같은 서로 다른 값들이 발생하는 것을 알 수 있습니다. 즉 9 개의 픽셀로부터 하나의 값이 생성되며, 위의 빨간색 칸을 한 칸씩(혹은 여러 칸일 수 있고 이 값을 stride 라고 부릅니다) 옮겨가며 이 값들을 하나씩 생성하게 되면 한 필터 당

10x10 크기의 이미지를 생성하게 됩니다. 이러한 필터가 윗 그림처럼 3 개 있을 경우 10x10 이미지 3 개가 생성됩니다.

Padding

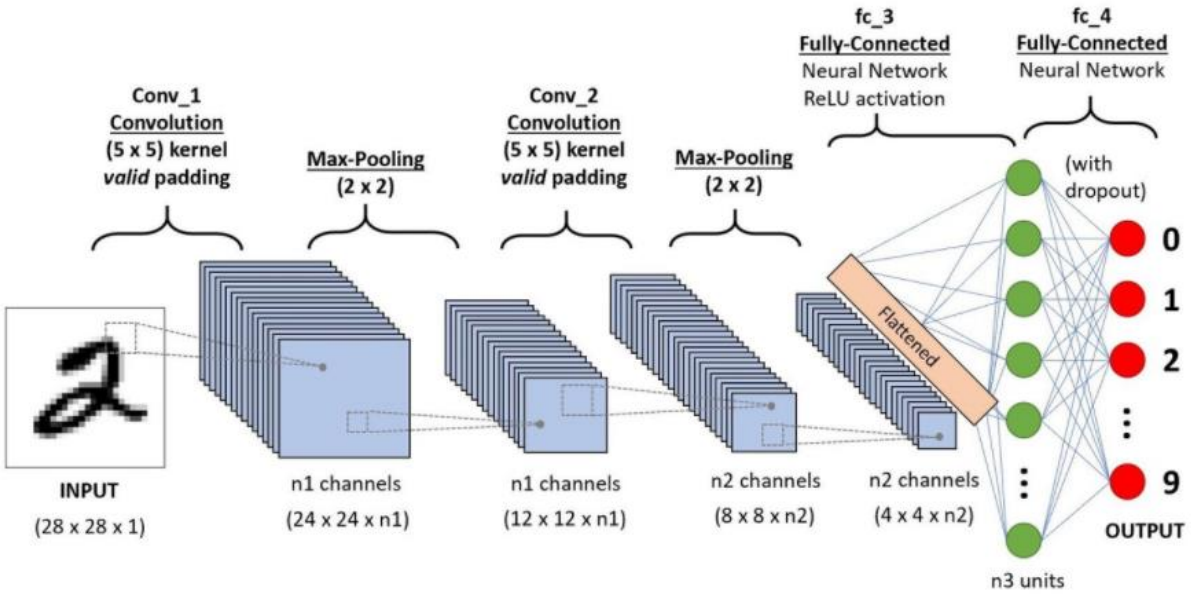
앞선 예시에서 12x12 이미지가 filter, stride 를 거쳐 10x10 으로 크기가 줄어든 현상을 발견할 수 있습니다. 그럼 12x12 이미지가 10x10 으로 줄어드는데 괜찮은건가? 의문이 들 수 있습니다. 이러한 문제를 해결하기 위해서 padding 이라는 값을 주게 됩니다. padding 이란 원래 이미지 양 옆에 일정 데이터를를 추가하여 원래의 크기와 동일한 이미지를 유지하도록 하는 방법입니다. 이미지의 가장자리에 0 의 값을 갖는 픽셀을 추가한 것을 zero-padding 이라고 하며, CNN 에서는 주로 이러한 zero-padding 이 이용됩니다.



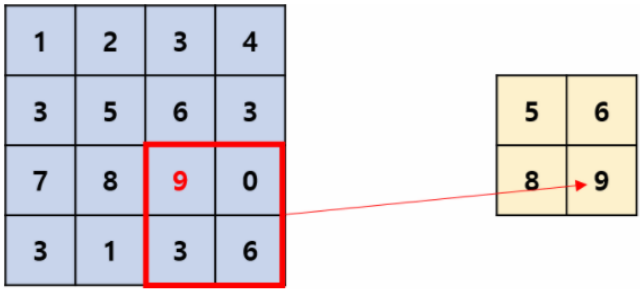
Pooling

Convolution 과정을 통해 filter 에 따라 이미지도 덩달아 많아지게 됩니다. 예를들어 filter 가 5 개라면 convolution 과정을 거칠 때마다 5 배씩 이미지가 늘어납니다. 이러한 문제점에서 도달한 것이 pooling 이라는 과정입니다. Pooling 은 이미지의 개수를 증가시키지 않고, 1 개의 이미지에서 1 개의 출력을 만들면서 동시에 기존 이미지에 filter 만 적용하여 크기를 줄이는 방법입니다.

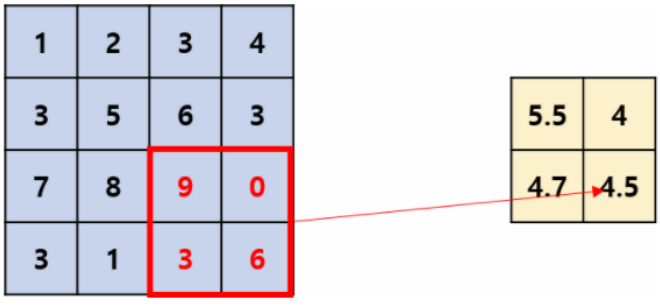
Pooling layer 는 데이터의 공간적인 특성을 유지하면서 크기를 줄여주는 층으로 연속적인 convolution layer 사이에 주기적으로 넣어줍니다. 이를 통해 크기가 줄어들어 학습할 parameter 를 줄일 수 있게됩니다.



Pooling 은 주어진 픽셀에서 값을 추출하는 방식에 따라서 max pooling 과 average pooling 으로 구분할 수 있습니다. Maxpooling 은 pooling filter 영역에서 가장 큰 값을 추출하는 pooling 기법입니다.

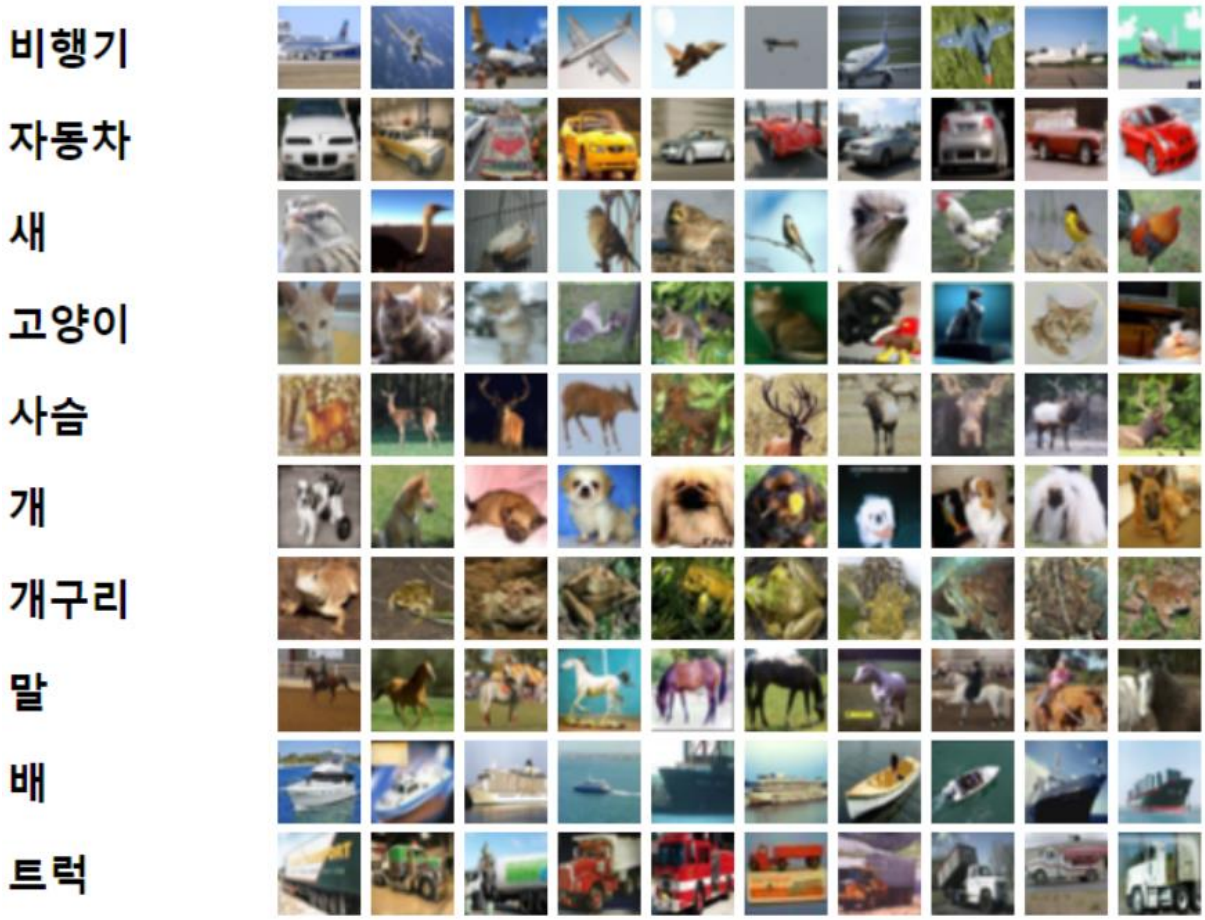


이어서 average pooling 은 영역 내의 값의 평균을 추출하는 pooling 기법입니다.



CIFAR-10

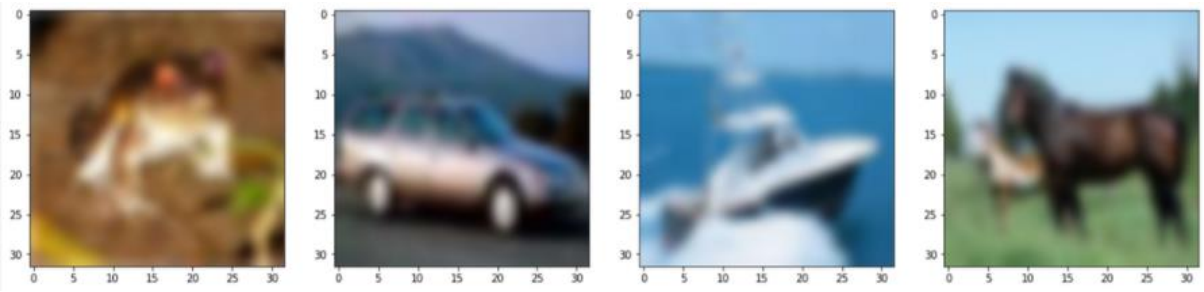
CIFAR-10 데이터셋은 32x32 픽셀의 60000 개 컬러이미지가 포함되어 있으며 각 이미지는 10 개의 클래스로 라벨링이 되어 있습니다.



총 60000 개의 이미지 중, 50000 개는 training, 10000 개는 test 용도로 사용됩니다. keras 에서 제공하는 API 를 통해 데이터를 로드하게 되면 아래와 같은 모습입니다.

```
#cifar10에서 데이터를 로드
#from keras.datasets import cifar10 이 필요
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

print ("Training data:")
print ("Number of examples: ", X_train.shape[0])
print ("Number of channels:",X_train.shape[3])
print ("Image size:", X_train.shape[1], X_train.shape[2])
print
print ("Test data:")
print ("Number of examples:", X_test.shape[0])
print ("Number of channels:", X_test.shape[3])
print ("Image size:", X_test.shape[1], X_test.shape[2])
```

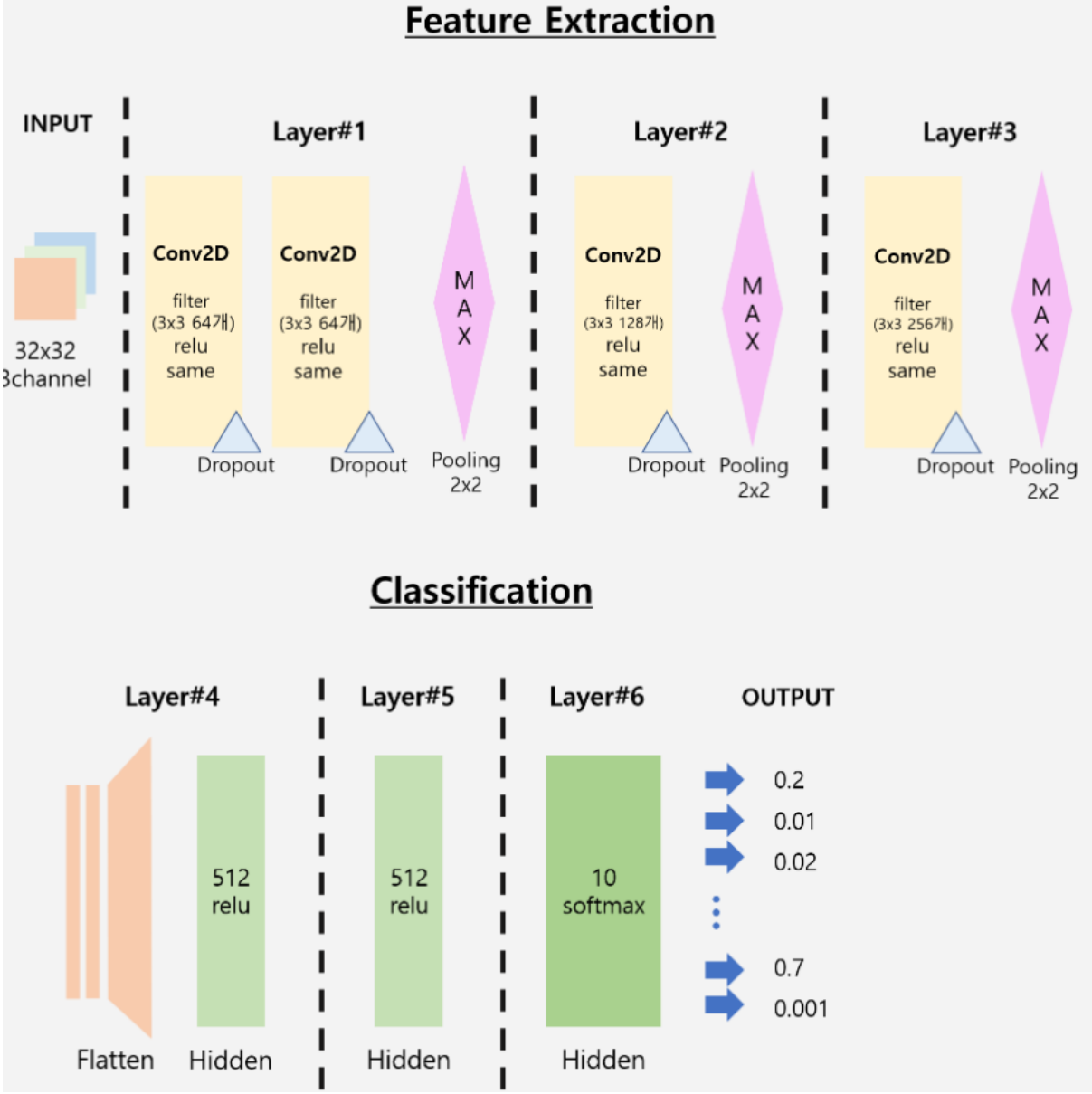
```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 2s 0us/step
Training data:
Number of examples:  50000
Number of channels: 3
Image size: 32 32
Test data:
Number of examples: 10000
Number of channels: 3
Image size: 32 32
(50000, 32, 32, 3) uint8
```



keras를 활용한 CIFAR-10 분류

https://gruuuuu.github.io/machine-learning/cifar10-cnn/#

아래는 이번 시간에 만들어 볼 CNN의 구조도입니다.



다음은 데이터 전처리입니다. 저번 machine learning 강의에서 normalization 과 standardization 에 대하여 설명하였습니다. 간단히 리뷰해보자면 normalization 은 아래 공식을 통해 전체 구간을 [0,1]로 맞추어주는 과정입니다.

$$X = \frac{x - x_{min}}{x_{max} - x_{min}}$$

standardization 은 데이터들이 gaussian distribution 을 따른다고 가정하고 평균 0, 표준편차 1 인 형태를 갖도록 변환해주는 방식입니다.

$$X = \frac{x - x_{mean}}{x_{std}}$$

이번 예제에서는 stadardization 을 사용하여 데이터 전처리를 진행해보겠습니다.

```
print ("mean before normalization:", np.mean(X_train))
print ("std before normalization:", np.std(X_train))

mean=[0,0,0]
std=[0,0,0]
newX_train = np.ones(X_train.shape)
newX_test = np.ones(X_test.shape)
#train set에 있는 데이터로만 평균과 표준편차를 구함
for i in range(3):
    mean[i] = np.mean(X_train[:,:,:,i])
    std[i] = np.std(X_train[:,:,:,i])

#train과 test셋 모두 정규화 작업
for i in range(3):
    newX_train[:,:,:,i] = X_train[:,:,:,i] - mean[i]
    newX_train[:,:,:,i] = newX_train[:,:,:,i] / std[i]
    newX_test[:,:,:,i] = X_test[:,:,:,i] - mean[i]
    newX_test[:,:,:,i] = newX_test[:,:,:,i] / std[i]

X_train = newX_train
X_test = newX_test

print ("mean after normalization:", np.mean(X_train))
print ("std after normalization:", np.std(X_train))
print(X_train.max())
```

```
mean before normalization: 120.70756512369792
std before normalization: 64.1500758911213
mean after normalization: 4.917991939616 21e-17
std after normalization: 0.9999999999999996
```

이어서 본격적으로 training 입니다.

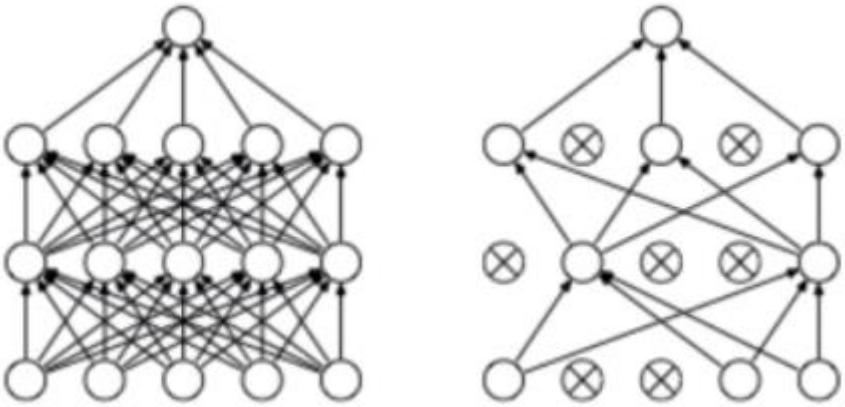
```
batchSize = 512             #-- Training Batch Size
num_classes = 10            #-- Number of classes in CIFAR-10 dataset
num_epochs = 50             #-- Number of epochs for training
learningRate= 0.001         #-- Learning rate for the network
lr_weight_decay = 0.95      #-- Learning weight decay. Reduce the learn rate by 0.95 after epoc


img_rows = 32               #-- input image dimensions
img_cols = 32

Y_train = np_utils.to_categorical(y_train, num_classes)
Y_test = np_utils.to_categorical(y_test, num_classes)
```

모델 구현은 아래와 같습니다.

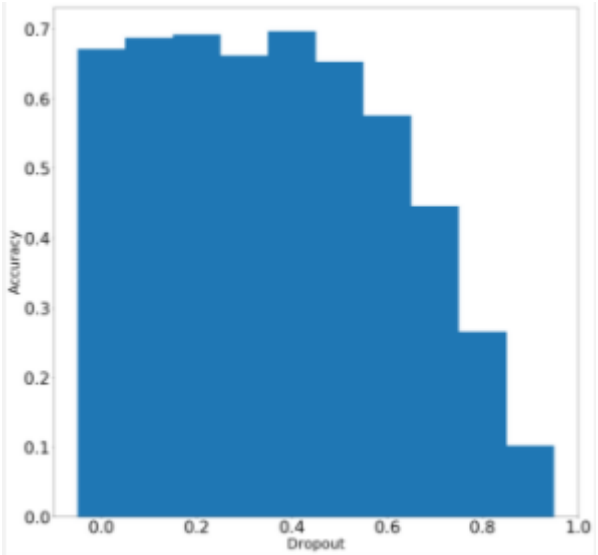
```
from keras import initializers
import copy
result = {}
y = {}
loss = []
acc = []
dropouts = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
for dropout in dropouts:
    print ("Dropout: ", (dropout))
    model = Sequential()

    #-- layer 1
    model.add(Conv2D(64, 3, 3,
                             border_mode='same',
                             activation='relu'
                             input_shape=(img_rows, img_cols,3)))
    model.add(Dropout(dropout))
    model.add(Conv2D(64, 3, 3, activation='relu',border_mode='same'))
    model.add(Dropout(dropout))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    ##--layer 2
    model.add(Conv2D(128, 3, 3, activation='relu',border_mode='same'))
    model.add(Dropout(dropout))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    ##--layer 3
    model.add(Conv2D(256, 3, 3, activation='relu',border_mode='same'))
    model.add(Dropout(dropout))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    ##-- layer 4
    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
```

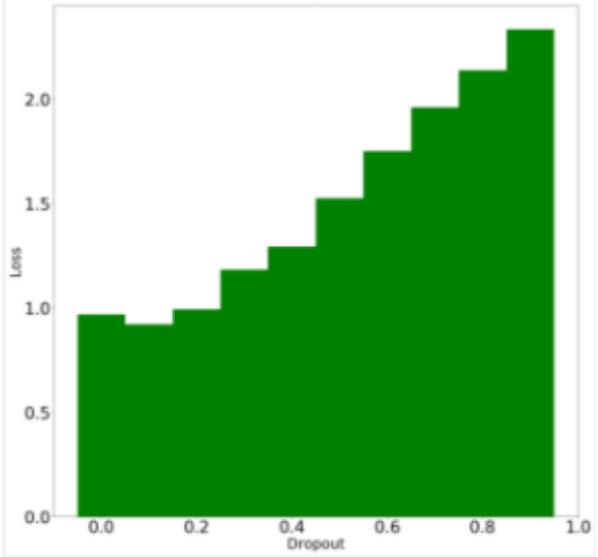
```
#-- layer 5
model.add(Dense(512, activation='relu'))

#-- layer 6
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])

model_cce = model.fit(X_train, Y_train, batch_size=batchSize, nb_epoch=num_epochs, verbose=1, shuffle=True,
score = model.evaluate(X_test, Y_test, verbose=0)
y[dropout] = model.predict(X_test)
print('Test score:', score[0])
print('Test accuracy:', score[1])
result[dropout] = copy.deepcopy(model_cce.history)
loss.append(score[0])
acc.append(score[1])
```

dropout이란 overfitting을 방지하기 위해 fully-connected layer에서 몇 개의 노드 연결을 끊고, 나머지 남은 노드들을 통해서만 훈련을 하게하는 것입니다.



이번 예제에서 dropout 을 0.0-0.9 까지 바꿔가며 테스트를 마친 뒤 dropout 별 accuracy 와 loss 입니다.


Accuracy
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.0
0.2
0.4
0.6
0.8
1.0
Dropout


Loss
0.0
0.5
1.0
1.5
2.0
0.0
0.2
0.4
0.6
0.8
1.0
Dropout