

# CHAPTER 4

## Table of contents

### 1. Introduction to CNN

### 2. Convolution Neural Network

- 1) Images are numbers
- 2) Feature Extraction with Convolution
- 3) Convolution neural network

### 3. Example model

- 1) Example 1 : convolution neural network model
- 2) Example 2 : convolution neural network for 2D
- 3) Example 3 : convolution neural network for 1D

### 4. Study case

- 1) MNIST data
- 2) Cifar10 data
- 3) Pre-trained model

## 1. Introduction to CNN

시력은 비언어적 의사소통 수단이며 물체를 감지하여 등 인간에게 중요한 감각 중 하나이다. 그렇다면, 인간의 시각을 어떻게 기계로 형성화할 수 있을까? 이 질문으로부터 이미지 인식 기술이 발전되기 시작하였다.

이미지 인식 및 분류에서 특히 뛰어난 성능을 보이는 합성곱 신경망(Convolutional Neural Network)은 입력 데이터의 모든 특징을 한번에 학습하지 않고 부분적인 특징을 여러 구간에서 학습하고자 하는 알고리즘으로, CNN을 구성하는 층은 크게 다음과 같이 세 종류의 층으로 구분할 수 있다

- (1) 합성곱 계층(Convolusion Layer)
- (2) 풀링 계층(Pooling Layer)
- (3) 완전 연결 계층(Fully Connected Layer)

먼저 합성곱 계층에서는 일정한 사이즈로 입력 데이터를 지역적으로 구분한 후 다수의 필터와 곱연산을 통해 특성맵을 추출한다. 풀링 계층에서는 추출된 특성 맵을 지정한 풀링 사이즈로 구분하여 각 구역당 최대값(Max Pooling) 또는 평균값(Average Pooling) 하나만을 남기는 방식으로 특성 맵의 크기를 축소한다. 마지막 합성곱(및 풀링) 층에서 출력된 특성 맵을 일렬로 변형하고 이후 완전 연결 층에서 출력 클래스들과의 완전 연결 연산을 통한 예측 및 분류가 이루어진다.

일반적으로 분류 문제를 위한 CNN은 다음과 같이 합성곱-풀링 계층을 반복하여 입력 데이터에 대한 특징을 추출하고 난 뒤 마지막 단계에 1개 이상의 완전 연결 층을 두어 클래스 분류가 이루어지도록 하는 구조를 가지고 있다.



### 합성곱 층의 2가지 핵심 파라미터

- (1) 입력으로부터 필터를 적용할 패치의 크기(kernel size)
- (2) 특성 맵의 출력 깊이 : 합성곱 연산에 사용할 필터의 수(filters)

CNN의 학습 목적은 층마다 설정된 개수의 필터 가중치를 학습함으로써 클래스를 구별하는데 영향을 미치는 중요한 특징을 뽑아낼 수 있도록 하는 것이다. 각 필터의 크기를 크게 설정한다면 한번에 넓은 구역에서 클래스를 구별하는 특징을 찾도록 하는 것이고 작게 설정한다면 좀더 작은 구역에서의 특징을 찾아내도록 하는 것이다. 따라서 모형에서 기대하는 바에 따라 필터를 적용할 패치의 크기라는 하이퍼 파라미터를 다르게 설정할 수 있다.

입력 데이터가 이미지하고 할 때, 하나의 필터는 이미지의 전 구역을 돌아다니며 동일한 연산을 진행한다. 즉 필터를 공유 파라미터로 사용하기 때문에 기존의 완전 연결로만 이루어진 신경망에 비해 추정 파라미터의 수를 대폭 줄일 수 있다.

개별 필터를 구성하는 각 원소의 초기값은 일반적인 심층신경망과 같이 랜덤하게 결정하고 반복학습에 따라 값을 업데이트한다. 이미지 분류 모형을 구성하는 경우 'Alexnet', 'VGG' 등과 같이 대규모 이미지 분류 경연대회에서 우승한 사전학습된 모형의 합성곱 층을 불러와 초기 단계부터 부분적 특징을 효과적으로 학습하고, 여기에 예측을 위한 완전 연결 층을 쌓는 형태로 사용하기도 한다.

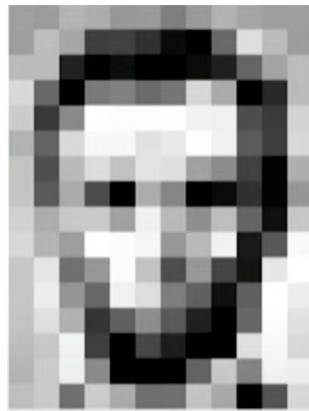
합성곱 신경망은 2D의 이미지 뿐 아니라 1D의 시퀀스 형태의 데이터에도 적용이 가능하다. 대표적으로 시계열 시퀀스나 문장, 즉 단어들의 시퀀스 형태가 있다. 이 경우 적용하는 필터 역시 1D로 적용되며 연산과정은 2D CNN과 동일하다.

jmkim21@ewhain.net

## 2. Convolution Neural Network

### 1) Images are Numbers

우선, 이미지를 이해하기 위해 어떤 처리과정이 필요한지 알아보도록 한다. 아래의 이미지는 해상도가 낮은 링컨 대통령의 흑백 사진이다.



다음으로 이미지의 최소 단위인 각 pixel을 숫자로 나타낼 것이다. 이때, 흑백 사진이므로 각 pixel의 색을 나타내는 숫자는 1개이다.

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	6	124	131	111	120	204	164	15	56	180
194	68	197	251	237	239	230	228	227	87	71	201
172	106	207	233	233	214	220	220	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	166	84	10	168	134	11	31	62	22	148
199	164	191	193	158	227	178	149	182	106	36	190
206	174	155	252	236	231	149	178	228	43	95	234
190	216	216	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	103	96	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	236	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	297	177	121	123	200	178	13	96	218

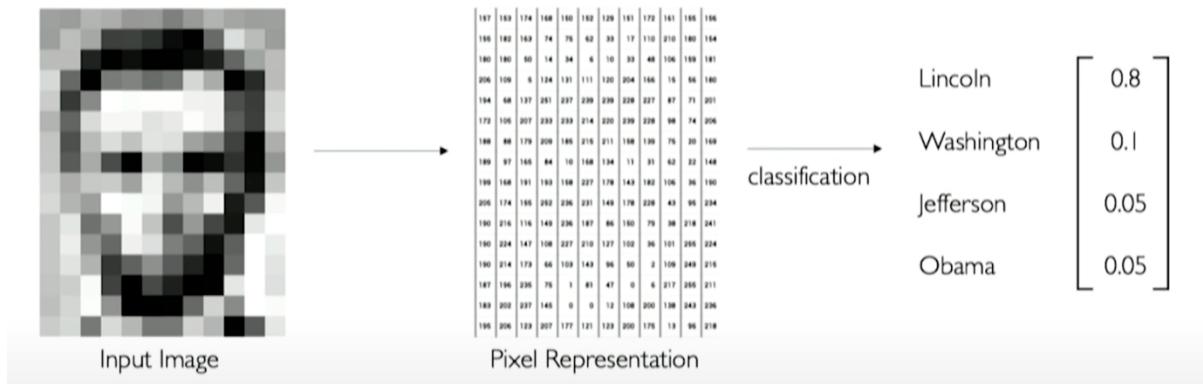
다음은 컴퓨터가 인식하는 이미지이다.

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	6	124	131	111	120	204	164	16	66	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	166	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	149	182	106	36	190
206	174	155	252	236	231	149	178	228	43	95	234
190	216	216	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	96	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	236	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	297	177	121	123	200	178	13	96	218

각 pixel의 값을 [0,255] 사이의 숫자로 변환한 후 matrix로 변환하면 이미지를 분석 model에 넣을 수 있는 데이터의 형태가 되는 것이다.

NOTE)

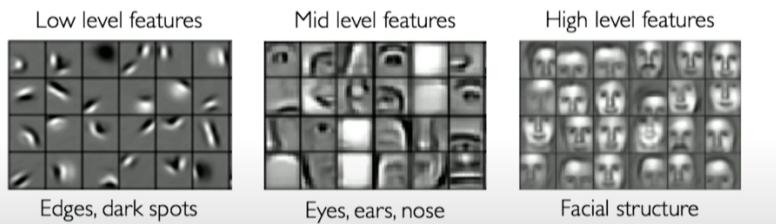
0은 black, 255은 white이다.



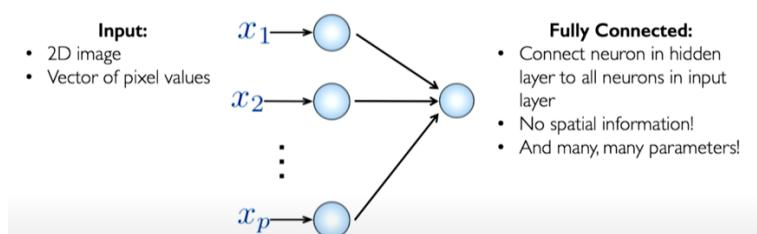
컬러 이미지의 경우에는 각 pixel의 값을 RGB(red, green, blue) 컬러를 이용하여 3가지 숫자로 나타낸다. 예를 들어, 1080X1080 pixel의 컬러 이미지는 각 pixel이 3가지의 숫자로 (red, green, blue) 표현되기 때문에 1080X1080X3 형식의 3차원 data matrix를 컴퓨터가 인식하게 된다.

## 2) Feature Extraction with Convolution

신경망은 학습과정에서 입력된 이미지의 특징을 찾아나간다. 방대한 양의 이미지에서 특징을 추출하고(feature extract) 여러 특징들의 계층을 생성해 이미지 분류 model을 만든다.



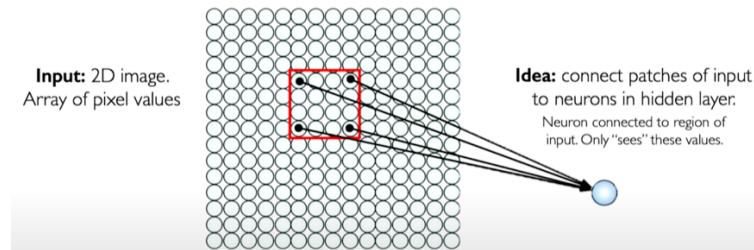
예를 들어, 사람의 얼굴을 학습하는 과정에서 신경망은 하나로 이어지는 선들과 모서리를 우선적으로 추출하고 다음으로 눈, 귀, 코등의 특징들을 추출할 것이다. 이런 특징들이 모여 하나의 얼굴을 완성시키는데 선, 모서리 그리고 눈, 귀, 코 등의 특징이 모여 최상위 특징인 얼굴을 구성하는 것이다.



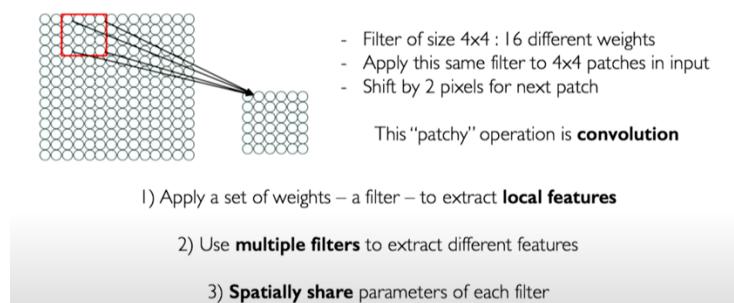
가장 먼저 개발되었던 DNN에 이미지 데이터를 넣게 될 경우 신경망의 모든 층은 fully connected layer(완전 연결층)로 구성된다. 각 layer를 통과하기 위해서는 2-dimension인 데이터를 1-dimension으로 변환해야 하는데,

데이터가 숫자들의 나열로 변환되어 되어 fully connected layer를 지나가면서 이미지 데이터가 가지는 공간적 구조를 무시하게 되는 문제가 생긴다. 동시에 모든 pixel이 모든 node와 연결이 되므로 계산해야 하는 parameter의 수가 천문학적인 양으로 늘어가게 된다.

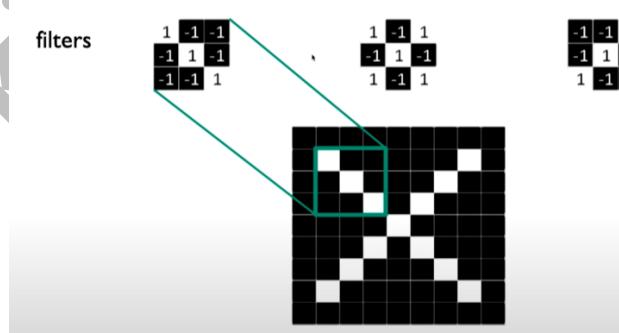
그렇다면, 어떻게 이미지 데이터의 공간적 구조를 보존하면서 data의 feature를 찾아낼 수 있을까?



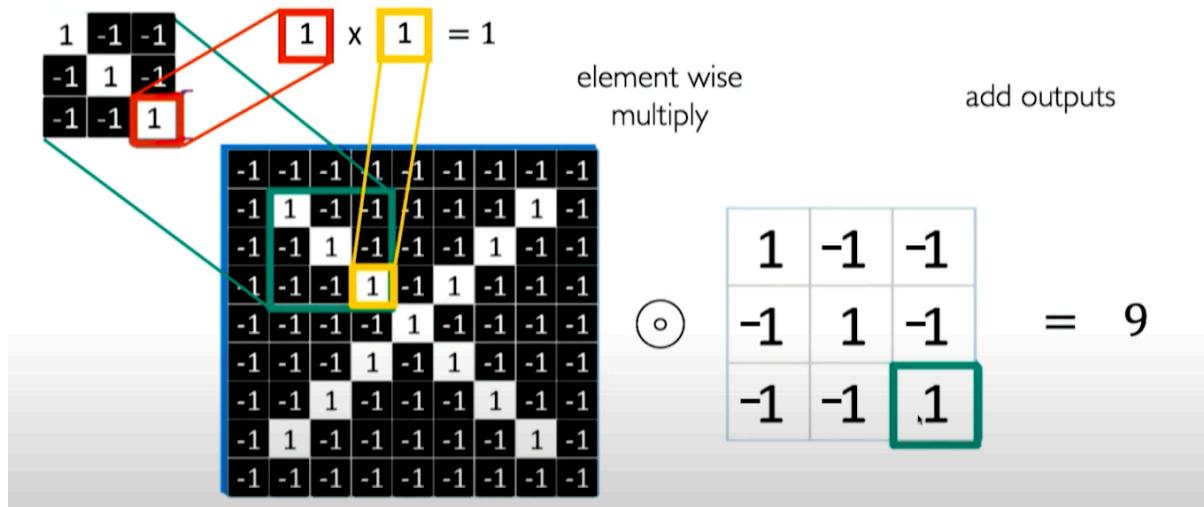
Convolution layer는 이미지 데이터를 보존함과 동시에 parameter를 찾기 위해 하나의 숫자를 node와 연결시키는 것이 아니라 데이터의 작은 구역(patch)을 node와 연결시킨다. 위 그림의  $4 \times 4$  size의 red box가 input된 이미지를 지나가면서 각 부분들이 single node와 연결이 되고 이 과정에서 feature를 찾아나간다.



feature가 추출되는 과정을 좀 더 자세하게 살펴보도록 한다. 위의 red box를 filter라고 정의하며 filter는 CNN에서 중요한 hyperparameter이다. filter는 weight의 집합체로 여러 종류의 filter를 쓰면서 이미지에서 다양한 feature를 찾을 수 있도록 해준다.



위 그림에서 3개의  $3 \times 3$  filter들은 하나의 이미지에서 각각 다른 feature들을 찾는다. 첫번째 filter는 X에서 왼쪽 방향의 대각선을 찾으며, 두번째 filter는 두 대각선이 직교하는 부분을 찾고 마지막 filter는 오른쪽 방향의 대각선을 찾는다. 3개의 filter가 지나가면서 구한 feature map의 조합을 이용하여 X라고 판단할 수 있는 것이다. 이때, 합성곱의 개념이 정의되는데 convolution(합성곱)은 filter가 feature를 찾아가는 계산과정을 의미한다.



convolution operation은 filter와 overlap되는 이미지의 작은 구역과 filter의 weight를 각각 곱한 후 계산된 숫자들을 모두 더한다. 계산 결과로 나온 숫자들은 이미지의 feature를 나타내는 값들이다.

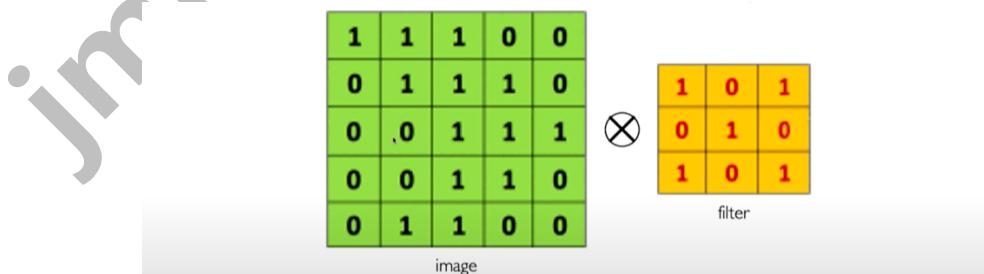


여러 filter들을 이용하여 하나의 사진에서 발견할 수 있는 feature를 다시 이미지로 형성화시키면 위와 같은 결과를 얻을 수 있다.

NOTE)

Example (The convolution Operation)

아래에 5x5 image과 3x3 filter가 있다. filter가 image를 지나면서 convolution operation(합성곱)을 통해 나오는 output을 계산한다.



filter가 image를 지나가면서 계산하는 과정은 다음과 같다.

The diagram illustrates three rows of convolution operations. Each row shows an input image (3x3 grid), a filter (3x3 grid), and the resulting feature map (3x3 grid). The first row uses a filter of  $\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$ . The second row uses a filter of  $\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$ . The third row uses a filter of  $\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$ .

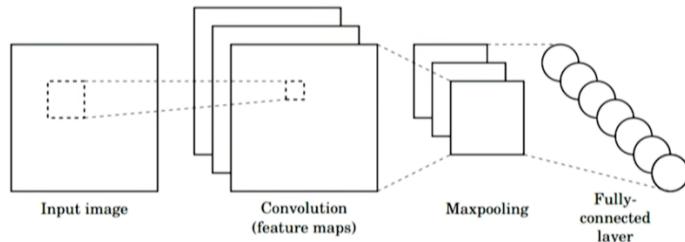
NOTE)

이미지 출처 : [http://introtodeeplearning.com/slides/6S191\\_MIT\\_DeepLearning\\_L3.pdf](http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L3.pdf)

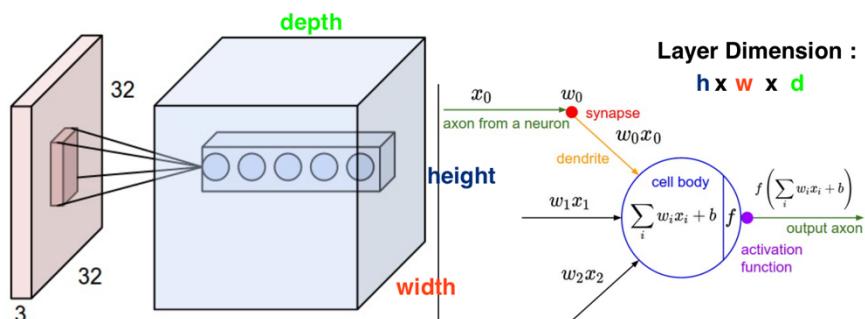
### 3) Convolution Neural Network

본격적으로 CNN model을 구성하는 layer에 대해 알아보도록 한다.

#### a) Convolution layer



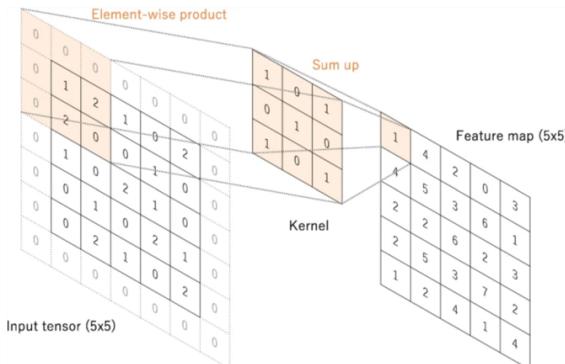
우선, CNN의 hidden layer는 convolution layer로 구성되는데 하나의 layer 안에서의 계산은 DNN의 hidden layer와 동일하다. 각 filter들의 weight를 이용하여 convolution operation을 진행하고 bias를 더한 다음 nonlinear activation function(ReLU 등)을 거치며 결과값이 출력된다.



하나의 이미지를 학습하는데 있어 filter를 여러개 이용하는 경우 위의 layer에 대해서 여러가지 옵션을 추가

해야 한다. 몇개의 filter를 이용할 것인지(filters), filter의 사이즈는 어떻게 설정할 것인지(kernel\_size, filter\_size), filter가 어느 정도의 보폭으로 이미지를 쓸고 지나갈 것인지(stride)를 설정해야 한다.

```
tf.keras.layers.Conv2D(filters = d, kernel_size = (h,w), strides = s)
tf.keras.layers.Conv2D(filters = d, filter_size = n, strides = s)
# filter 가 정사각형인 경우 filter_size 이용
```

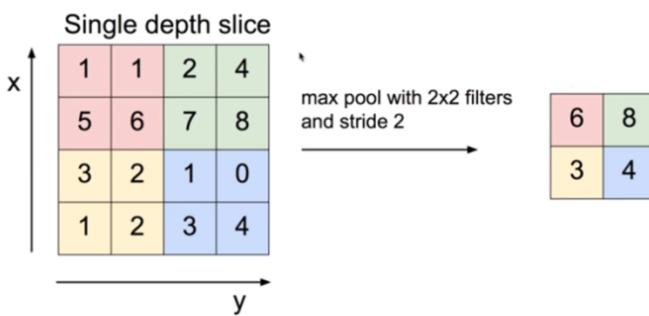


filter가 image를 지나가면서 stride에 따라 output size가 작아진다(dimension reduction). padding은 위의 그림처럼 이미지의 가장자리에 0을 추가한다. 이 단계를 통해 이미지가 down scale되는 것을 막을 수 있고 input size와 output size가 동일하게 유지된다. padding을 하지 않을 경우 convolution layer를 거칠수록 feature map의 크기는 줄어든다.

stride는 filter가 한번에 이동하는 보폭을 의미한다. convolution layer에서 stride를 1로 설정할 경우 한칸씩 이동하며, pooling layer에서는 stride를 filter size와 동일하게 설정하여 서로 겹치지 않도록(No overlapping)하는 것이 기본 설정이며 일반적으로 사용된다.

### b) Pooling layer

다음은 pooling layer이다. pooling layer에는 최댓값을 추출하는 max pooling, 평균값을 추출하는 mean pooling 등 다양한 종류가 있다. 먼저, max pooling과 이 과정의 output에 대해 알아보도록 한다.

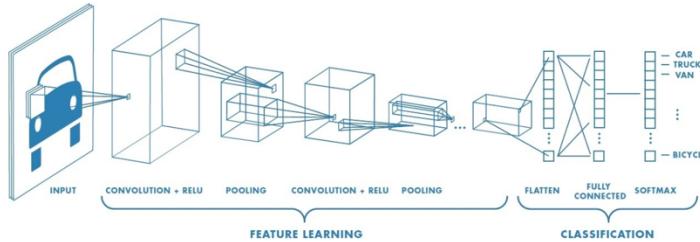


위 그림에서 pooling filter의 크기는 2x2이며 stride는 2로 설정함으로써 겹쳐지는 pixel이 발생하지 않는다. 또한, max pooling이기 때문에 filter에 들어오는 4개의 값 중 가장 큰 값을 가지는 pixel만을 추출한다. 따라서, 4x4 image가 2x2 image로 scale이 축소된다.

```
tf.keras.layers.MaxPool2D(pool_size = (2,2), strides = 2)
```

### c) Fully-connected layer

convolution layer와 pooling layer를 번갈아가며 통과하게 되면 이미지의 scale이 점점 줄어들게 된다. 줄어든 image를 1차원 숫자의 나열로 만들게 되면 fully connected layer를 통과할 수 있고 각 클래스에 속할 확률을 계산한다.



CNN model은 크게 convolution layer, pooling layer, fully connected layer로 구성된다. CNN model의 학습과정은 크게 두가지로 나뉜다. 첫번째로 convolution layer와 pooling layer를 번갈아 지나가면서 이미지의 feature를 학습함과 동시에 이미지의 크기가 축소된다. 두번째로 fully connected layer에서 찾아낸 feature들을 이용하여 어떤 클래스인지 확률을 구한다.

### 3. Example models

#### 1) Example 1 : convolution neural network model

앞의 과정을 convolution neural network 로 만들면 아래와 같은 model을 만들 수 있다.

```
import tensorflow as tf

def generate_model():
    model = tf.keras.Sequential([
        # first convolutional layer
        tf.keras.layers.Conv2D(32, filter_size = 3, activation = 'relu'),
        tf.keras.layers.MaxPool2D(pool_size = 2, strides = 2),

        # second convolutional layer
        tf.keras.layers.Conv2D(64, filter_size = 3, activation = 'relu'),
        tf.keras.layers.MaxPool2D(pool_size = 2, strides = 2),

        # fully connected classifier
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(1024, activation = 'relu'),
        tf.keras.layers.Dense(10, activation = 'softmax') # 10 classes
    ])

    return model
```

## 2) Example 2 : convolution neural network for 2D

CNN의 각 층에서의 연산을 그림으로 표현하기 위해 2D 형태의 input을 가정한다.

하나의 입력 이미지의 넓이와 높이가 각각 28 pixel로 이루어져 있다고 하자. 컬러 이미지의 경우 각 픽셀에는 RGB, 3가지 차원의 특징이 존재하므로 입력의 깊이 깊이, 즉 입력 특징(feature)의 수는 3이다. 흑백 이미지의 경우 pixel 별로 1가지 음영값 만을 가지므로 깊이는 1이 된다. 이미지가 분류될 클래스는 10개가 존재한다고 할 때, 다음과 같이 두 개의 합성곱 및 풀링 계층을 거치는 CNN 모형의 구조를 예시로 고려하였다.

### a) Hyper-parameter

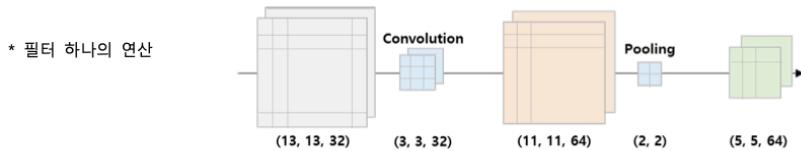
합성곱 층	필터의 개수	필터 사이즈
1	32	(3, 3)
2	64	(3, 3)

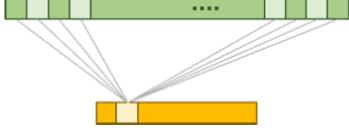
### b) 구조 요약

Layer	Output Shape	Parameters
Input	(None, 28, 28, 3)	
<b>Conv2D(filters=32, kernel_size=(3,3), activation='relu', input_shape=(28,28,3))</b>	(None, 26, 26, 32)	896
<b>MaxPooling2D(pool_size=(2,2))</b>	(None, 13, 13, 32)	0
<b>Conv2D(filters=64, kernel_size=(3,3), activation='relu')</b>	(None, 11, 11, 64)	18496
<b>MaxPooling2D(pool_size=(2,2))</b>	(None, 5, 5, 64)	0
<b>Flatten</b>	(None, 1600)	0
<b>Dense(units=10, activation='softmax')</b>	(None, 10)	16010

- 합성곱(2D) 층에서 입력의 높이 및 너비가 n이고 필터의 높이 및 너비가 l이라고 할 때, 패딩을 하지 않고 (n, n) 이미지 (또는 특성 맵)에 (l, l) 필터를 적용한 결과 출력되는 특성맵의 크기는  $(n-l+1, n-l+1)$ 이 된다.
- 합성곱 층의 파라미터의 개수는  $(\text{입력 채널의 수}) \times (\text{필터의 넓이}) \times (\text{출력 채널의 수})$ 의 가중치(weight)와 출력 채널의 수와 동일한 개수의 편향(bias)를 더한 만큼 존재한다.
- 완전 연결 층의 파라미터는  $(\text{입력 층 노드 수}) \times (\text{출력 층 노드 수})$ 의 가중치와 출력 노드 수만큼의 편향으로 이루어진다.

Convolution Layer 1		Shape
	(1) 입력 이미지는 한번에 필터와 동일한 크기의 윈도우로 구분된다. 예시의 경우 필터 사이즈가 (3, 3)이므로 패딩을 하지 않고 스트라이드가 1인 경우 총 26*26개의 구역이 생긴다. (앞서 구조에서는 3차원으로 입력 채널을 가정하였으나 이해를 돋기 위해 그림에서는 입력 채널 1개에 대한 그림으로 표현하였다.)  (2) 하나의 필터는 이미지를 슬라이딩하며 각 구역당 점곱을 통해 하나의 값을 계산하고 각 값에 활성화 함수를 적용하여 26*26 크기의 특성 맵 하나를 형성한다.  (3) 총 32개의 필터에 대해 각각 26*26 크기의 특성 맵이 생성되므로 출력되는 특징 맵은 (26, 26, 32) 차원의 배열이 된다.	(26, 26, 1)  (26, 26, 32)
	Pooling Layer 1	
	(4) 폴링 사이즈는 2로 최대 폴링할 경우 출력된 각 특징 맵의 2칸 당 큰 값만을 추출하여 특징 맵의 사이즈를 축소한다. (특징 맵의 개수는 동일하게 유지된다.) 이 층에서는 파라미터를 사용한 연산이 일어나지 않는다.	(13, 13, 32)
* 필터 하나의 연산 	Convolution  Pooling	(28, 28)      (3, 3)      (26, 26)      (2, 2)      (13, 13)
Convolution Layer 2		Shape
	(5) 이전 층에서 출력된 특징 맵이 두 번째 합성곱 층의 입력 특징 맵이 된다. 첫 번째 합성곱 층과 같은 과정으로, 필터 사이즈와 동일한 크기로 특징 맵의 구역이 구분되어 채널의 깊이는 유지된다.  (6) 각 필터는 입력되는 특징 맵과 동일한 깊이를 가진다. 개별 필터가 입력 특징 맵의 구역을 이동하며 점곱과 활성화 함수를 거쳐 3차원의 윈도우 당 하나의 값을 계산함으로써 11*11 크기의 특징 맵 하나가 형성된다.  (7) 두 번째 합성곱 층에서 설정한 필터의 개수는 64개이므로 11*11 크기의 특징 맵이 64개 형성되어 출력되는 특징 맵은 (11, 11, 64)의 차원을 가진다.	(13, 13, 32)  (11, 11, 64)
	Pooling Layer 1	
	(8) 마찬가지로 특징 맵의 각 두 칸 당 큰 값만을 추출함으로써 각 특징 맵의 크기를 축소하여 개수는 유지된다.	(5, 5, 64)



Flatten & Output Layer	Shape
 (9) 클래스 분류를 위한 마지막 출력 계층과 완전 연결하기 위해 이전 층에서 출력된 (5, 5, 64)의 특징 맵을 1차원의 배열로 변형 한다.	(1600,)
 (10) 1600개의 모든 특징과 10개의 클래스를 완전 연결한 연산을 통해 각 클래스로 분류될 Score를 계산하고 활성함수로 Softmax 를 취하여 값이 제일 큰 클래스로 분류한다.	(10,)

### 3) Example 3 : convolution neural network for 1D

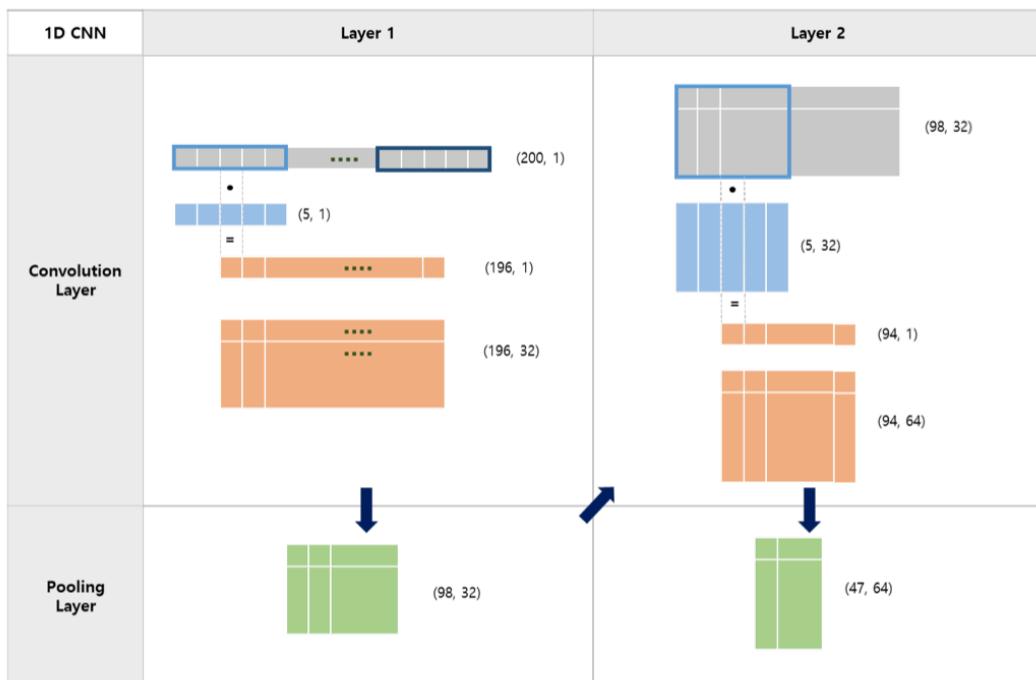
1D CNN 역시 2D와 동일한 연산 과정을 거치며 차이점은 입출력되는 데이터 및 특성맵과 필터의 차원에 있다. 1D CNN에서 필터는 데이터와 동일하게 길이 차원만 존재하며, 입력 시퀀스를 한 반향으로 슬라이딩하며 합성곱 연산을 진행한다. 하나의 데이터가 총 길이가 200인 시퀀스 하나로 이루어진 형태라고 하자. 데이터 하나에 대한 특징이 여러 개라면 여러개의 시퀀스가 동시에 채널로서 존재할 것이다. 다음에서 이를 2가지 클래스로 분류하는 모형의 구조를 예시로 사용하여 합성곱 층과 폴링층에서 일어나는 연산을 도식화하였다. 마지막 출력된 특징맵의 일렬 배열과 완전연결 층의 연산은 2D와 동일하다. 합성곱 층에서 학습할 파라미터는 (입력 채널의 수)x(시퀀스 길이)x(출력 채널의 수) 만큼의 가중치와 출력 채널의 수 만큼의 편향으로 구성된다.

#### a) Hyper-parameter

합성곱 층	필터의 개수	필터 사이즈
1	32	5
2	64	5

#### b) 구조 요약

Layer	Output Shape	Parameters
Input	(None, 200, 1)	
<b>Conv1D(filters=32, kernel_size=5, activation='relu', input_shape=(200, 1))</b>	(None, 196, 32)	192
<b>MaxPooling1D(pool_size=2)</b>	(None, 98, 32)	0
<b>Conv1D(filters=64, kernel_size=5, activation='relu')</b>	(None, 94, 64)	10304
<b>MaxPooling1D(pool_size=2)</b>	(None, 47, 64)	0
<b>Flatten</b>	(None, 3008)	0
<b>Dense(units=2, activation='softmax')</b>	(None, 2)	6018



## 4. Study Case

### 1) MNIST data

MNIST dataset은 총 70,000개의 28x28 저해상도 흑백 이미지를 포함하고 있다. 따라서 하나의 이미지는 (28, 28, 1) 배열의 형태로 정의할 수 있다. 전체 데이터 셋에서 이미지가 속한 클래스는 총 10개로 0부터 9까지의 숫자에 대한 손글씨 이미지이다. 70,000개 중 60,000개는 train과 validation set에, 10,000개는 test set에 포함된다.

```
# import library
import sys
import tensorflow as tf
from tensorflow.keras import layers

import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
from tensorflow.keras.utils import to_categorical, plot_model

import numpy as np
import matplotlib.pyplot as plt

# hyperparameter
batch_size = 128
num_classes = 10
epochs = 12
```

keras에 내장된 mnist dataset을 불러들인 후 train data와 test data로 분리한다.

```
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

print(f'x_train : {x_train.shape}') # (데이터 개수, 28 X 28 pixel)
print(f'y_train : {y_train.shape}')
print(f'x_test : {x_test.shape}')
print(f'y_test : {y_test.shape}')
```

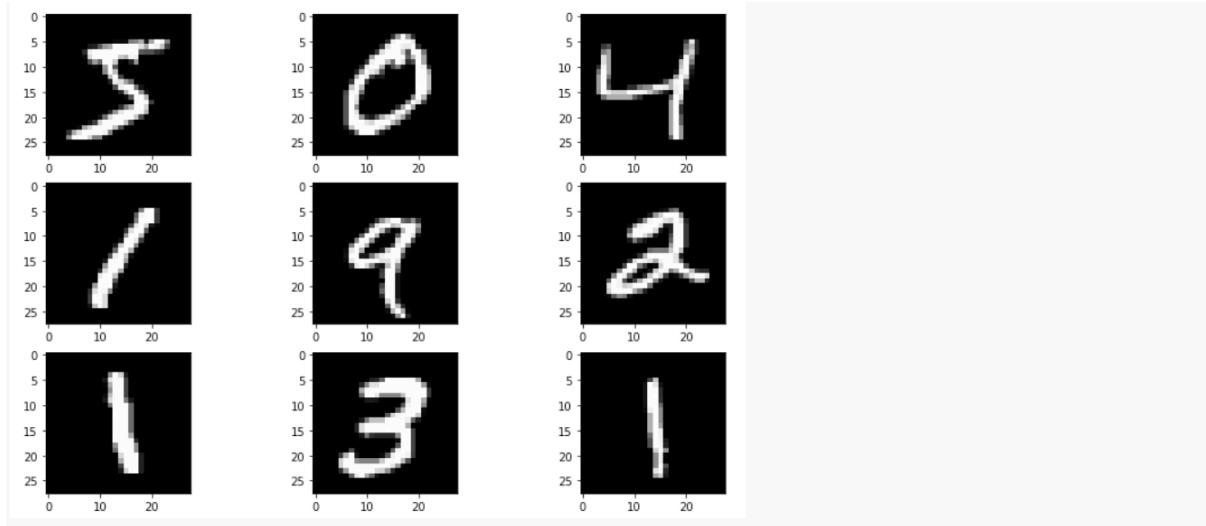
9개의 이미지를 출력해보도록 한다.

```
plt.figure(figsize=(12,8))

count = 0

for i in range(9):
    count += 1
    plt.subplot(3,3,count)
    plt.imshow(x_train[i].reshape(28,28), cmap = plt.get_cmap('gray'), interpolation='nearest')

plt.show()
```



다음으로 데이터를 학습하기 위해 전처리 과정이 필요하다. 이미지를 model에 넣기 위해 각 pixel의 값을 숫자로 변형시킨다.

```
img_rows = 28
img_cols = 28

input_shape = (img_rows, img_cols, 1)

# reshaping mnist dataset
x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1).astype('float32')
x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1).astype('float32')
```

DNN과 마찬가지로 data normalization 과정이 필요하다.

```
# Scale images to the [0, 1] range
x_train /= 255.
x_test /= 255.
```

y에 해당하는 label 값은 0부터 9까지 10개의 숫자이므로 model에 넣기 위해 one-hot encoding 과정이 필요하다.

```
# convert class vectors to binary class matrices
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)
```

cnn model을 만든 후 데이터를 학습시킨다.

```
# convolutional neural network
model = keras.Sequential([
    layers.Conv2D(32, kernel_size = (5, 5), strides = 1, activation = 'relu', input_shape =
input_shape),
    layers.MaxPooling2D(pool_size = (2, 2), strides = 2),
    layers.Conv2D(64, kernel_size = (2, 2), activation = 'relu'),
    layers.MaxPooling2D(pool_size = (2, 2), strides = 2),
    layers.Flatten(),
    layers.Dropout(0.5),
    layers.Dense(num_classes, activation = 'softmax')
])
```

```

model.summary()

Model: "sequential"
=====
Layer (type)          Output Shape       Param #
=====
conv2d (Conv2D)        (None, 24, 24, 32)    832
max_pooling2d (MaxPooling2D) (None, 12, 12, 32)    0
)
conv2d_1 (Conv2D)      (None, 11, 11, 64)     8256
max_pooling2d_1 (MaxPooling2D) (None, 5, 5, 64)    0
flatten (Flatten)      (None, 1600)         0
dropout (Dropout)      (None, 1600)         0
dense (Dense)          (None, 10)           16010
=====
Total params: 25,098
Trainable params: 25,098
Non-trainable params: 0

```

분류 문제이므로 model.compile 층에서 option은 classification의 option과 동일하게 설정한다.

```

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
hist = model.fit(x_train, y_train, batch_size=batch_size, epochs=10, verbose=1,
                  validation_data=(x_test, y_test)) # epochs 조정

score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

Test loss : 0.022359047085046768
Test accuracy : 0.9930999875068665

```

Test data에서 accuracy는 0.99로 높은 정확도를 보여준다. 다음은 학습의 결과를 그림으로 확인하기 위한 코드이다.

```

import random
import matplotlib.pyplot as plt

predicted_result = model.predict(x_test)
predicted_labels = np.argmax(predicted_result, axis=1) # 확률의 최대값의 index 를 추출하며 여기서는
index 가 label 과 동일하다.

test_labels = np.argmax(y_test, axis=1)

count = 0

```

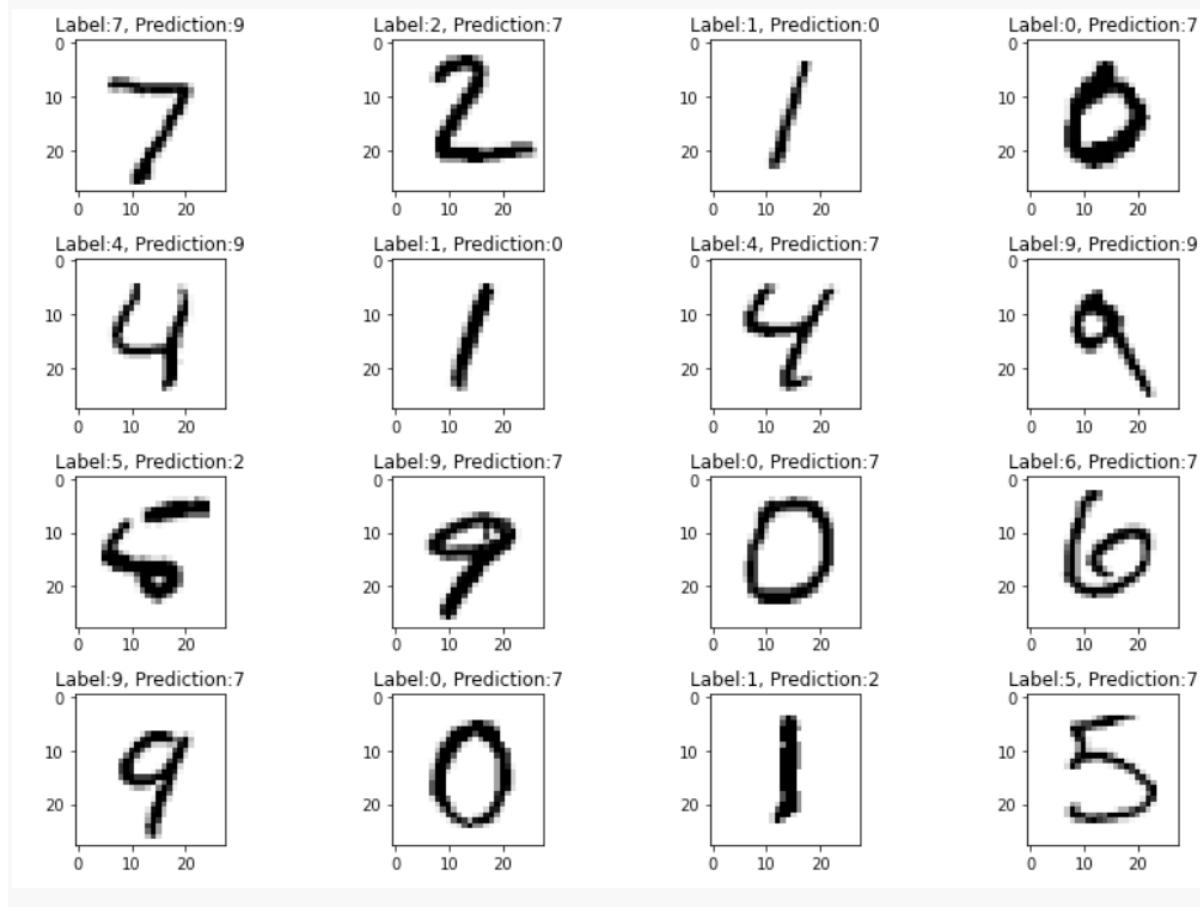
```

plt.figure(figsize=(12,8))

for n in range(16):
    count += 1
    plt.subplot(4, 4, count)
    plt.imshow(x_test[n].reshape(28, 28), cmap='Greys', interpolation='nearest')
    tmp = "Label:" + str(test_labels[n]) + ", Prediction:" + str(predicted_labels[n])
    plt.title(tmp)

plt.tight_layout()
plt.show()

```



classification 문제이므로 confusion matrix를 구한다.

```

from sklearn.metrics import confusion_matrix
y_pred=model.predict(x_test)
ypred_class=np.argmax(y_pred,axis=1)
ytest_class=np.argmax(y_test,axis=1)
print(ypred_class.size)
print(ytest_class.size)

10000
10000

conf_mat=confusion_matrix(ytest_class,ypred_class)
print(conf_mat)

```

```
[[978  0  0  0  0  0  0  1  1  0]
 [ 0 1133  1  0  0  0  1  0  0  0]
 [ 1  0 1028  0  0  0  0  2  1  0]
 [ 0  0  1 1006  0  1  0  1  1  0]
 [ 0  0  0  0  976  0  1  0  1  4]
 [ 1  0  0  4  0  885  1  0  1  0]
 [ 3  2  1  0  1  1  946  0  4  0]
 [ 0  5  7  1  0  1  0 1011  1  2]
 [ 3  0  1  1  0  0  0  0  968  1]
 [ 0  0  1  0  3  1  0  2  2 1000]]
```

Dataframe으로 confusion matrix를 만든다.

```
label = range(10)

import pandas as pd
table = pd.DataFrame(conf_mat, columns = label, index = label)
table
```

	0	1	2	3	4	5	6	7	8	9
0	978	0	0	0	0	0	0	1	1	0
1	0	1133	1	0	0	0	1	0	0	0
2	1	0	1028	0	0	0	0	2	1	0
3	0	0	1	1006	0	1	0	1	1	0
4	0	0	0	0	976	0	1	0	1	4
5	1	0	0	4	0	885	1	0	1	0
6	3	2	1	0	1	1	946	0	4	0
7	0	5	7	1	0	1	0	1011	1	2
8	3	0	1	1	0	0	0	0	968	1
9	0	0	1	0	3	1	0	2	2	1000

어떤 경우에 잘못된 분류가 나왔는지 관찰하기 위해 오분류 된 예시를 본다.

```
misclassified=np.where(ypred_class!=ytest_class)
aa=np.asarray(misclassified)
print(misclassified)
print(len(misclassified[0]))
```

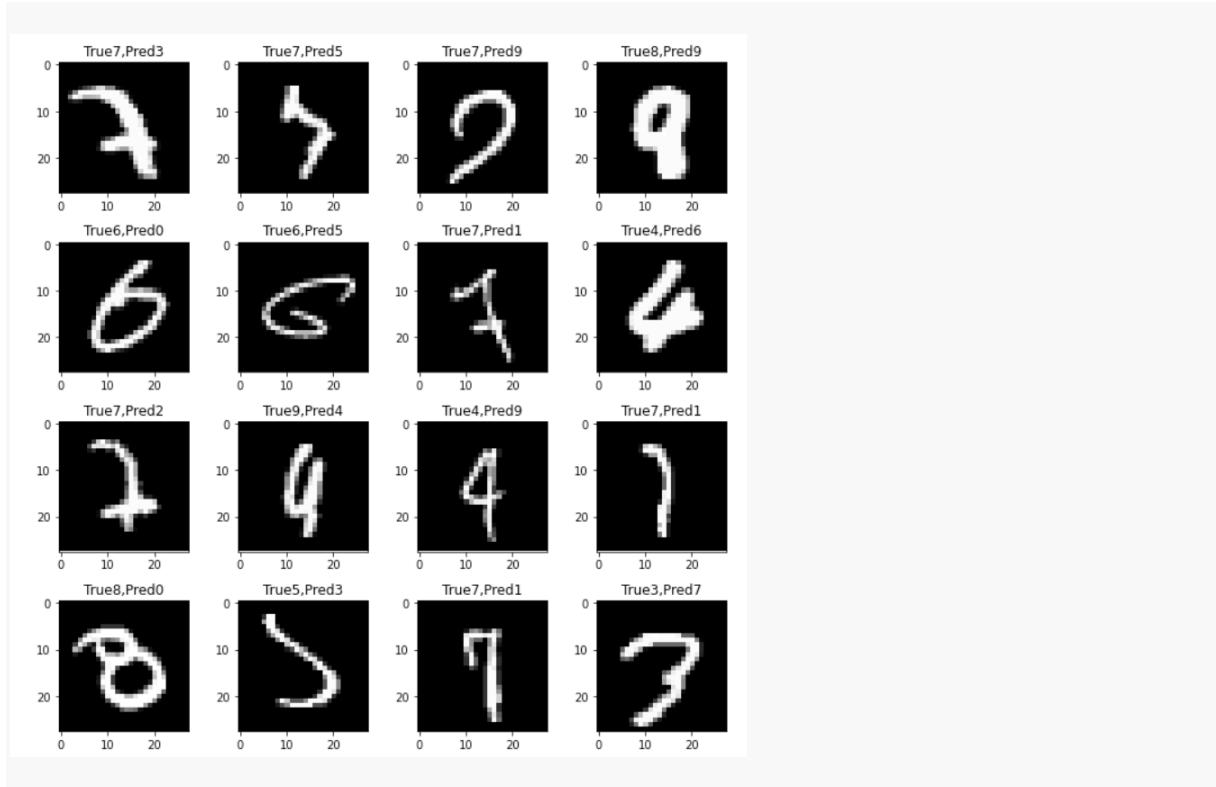
69

69개의 이미지가 잘못 분류되었다. 몇가지 예시를 살펴보도록 하자.

```
plt.rcParams['figure.figsize'] = (9,9) # Make the figures a bit bigger

for i in range(16):
    plt.subplot(4,4,i+1)
    plt.imshow(x_test[aa[0,i]].reshape(28, 28), cmap='gray', interpolation='none')
    plt.title("True{}, Pred{}".format(ytest_class[aa[0,i]], ypred_class[aa[0,i]]))

plt.tight_layout()
```



첫번째 줄의 네번째 이미지의 경우 8이라는 손글씨를 9로 잘못 분류하였다.

## 2) CIFAR10 data

CIFAR10 dataset은 총 60,000개의 32x32 저해상도 컬러 이미지를 포함하고 있다. 따라서 하나의 이미지는 R, G, B 3차원의 특징을 가지는 (32, 32, 3) 배열의 형태로 정의할 수 있다. 전체 데이터 셋에서 이미지가 속한 클래스는 총 10개로 각 클래스 당 6,000개의 이미지가 존재하며 이중 각 클래스당 5,000개는 훈련 및 검증용 셋에, 1,000개는 테스트 셋에 포함된다. 각각의 클래스는 서로 오버랩되지 않는다.

### a) Class

Index	0	1	2	3	4	5	6	7	8	9
Label	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck

앞에서 다룬 CNN의 층별 속성 및 구조를 변화시켜가며 다양한 모형을 고려하여 CIFAR10 data set에 적용해 보기로 한다.

### b) 구조 요약

Layer	Number of Output Channels	Filter Size
Input	3	
Convolution 1	16	(3, 3)
Convolution 2	32	(3, 3)
Convolution 3	64	(3, 3)
Dense	128	-
Output	10	-

적합할 CNN 모형의 전반적인 구조는 총 세 블록의 합성곱(convolution)-풀링(pooling) 층을 필수적으로 거쳐 최종 출력(output) 층과 fully connected 되도록 설정하였다.

먼저 각 블록당 하나의 합성곱 층을 거쳐 출력된 특징 맵을 일렬로 배열하고 마지막 출력 층과 완전 연결 함으로써 클래스 분류가 이루어지도록 하는 비교적 간단한 구조부터, 각 블록의 연속된 합성곱 층의 수를 증가시키고 일렬로 배열한 특징 맵과 최종 출력 층 사이에 완전 연결 층(fully-connected hidden layer)을 추가하는 등 점점 모형의 구조를 깊게 설계하여 결과의 차이를 확인하였다. 모형이 깊어짐에 따라 발생할 수 있는 과대적합의 위험을 해소하기 위해 일부 모형에서는 배치 정규화 또는 드롭아웃 방법을 적용하였다.

### c) Hyper-parameter

epochs	batch size	activation	kernel initializer	dropout rate
100	32	'relu'	'he_uniform'	0.1

### d) Call back

모든 모형은 최대 100번의 반복 동안 하나의 epoch 안에서 여러 번의 미니 배치를 반복해 가중치를 업데이트

트 한다. 즉 배치 사이즈를 32로 설정할 경우 학습에 사용하는 훈련 이미지는 총 5만개 이므로 epoch 하나 당  $50000/32 = 1562$ 번의 미니배치가 진행된다. 검증용 셋에 대한 loss와 accuracy의 평가는 epoch 하나가 종료될 때마다 진행되며 keras의 callback option을 통해 학습의 초기 종료 및 최적 모형 저장 등을 수행할 수 있다.

```
# EarlyStopping
callback = tf.keras.callbacks.EarlyStopping(monitor = 'loss', patience = 3)
```

monitor로 설정한 측정치가 지정한 patience의 epoch 동안 향상되지 않을 경우 학습을 초기 종료한다. val\_loss나 val\_accuracy를 monitor하고 있다면 loss대신 val\_loss나 val\_accuracy를 사용할 수 있다.

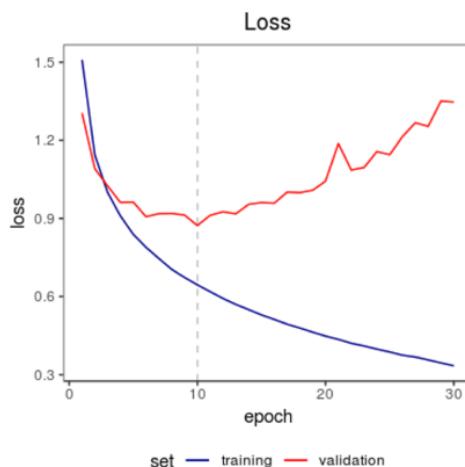
```
# ModelCheckpoint
callback = tf.keras.callbacks.ModelCheckpoint(
    filepath = './model/checkpoint.{epoch:02d}.hdf5', monitor = 'val_loss',
    save_best_only = FALSE,
    save_freq = 'epoch')
```

지정한 경로에 모형의 체크포인트를 저장하여 save\_best\_only = TRUE로 설정할 경우 총 반복횟수(epochs) 중 설정한 측정치(monitor) 기준으로 최적의 모형을 저장하도록 한다. monitor로 설정할 측정치는 validation loss 또는 accuracy가 될 수 있다.

예를 들어 다음과 같이 callback 옵션을 설정하였다고 하자. 지정하지 않은 다른 옵션들은 default 값으로 가정한다.

```
callback = tf.keras.callbacks.EarlyStopping(monitor = 'val_loss', patience = 20)
callback = tf.keras.callbacks.ModelCheckpoint(filepath = './model/checkpoint.{epoch:02d}.hdf5',
                                              monitor = 'val_loss', save_best_only = TRUE)
```

위 옵션을 지정한 모형은 학습이 진행됨에 따라 각 epoch의 마지막에 검증용 셋에 대해 평가되는 loss를 monitoring한다. 각 epoch 종료 시마다 현재의 loss가 이전 epoch의 loss보다 감소했다면 checkpoint에 지정한 경로에 현재의 모형을 저장한다. (감소하지 않았다면 저장하지 않고 다음 epoch를 진행한다) 아래의 그림은 전체 epochs를 100으로 설정하고 학습을 시작한 모형의 히스토리 예시다. 10번째 epoch에서 loss가 가장 작으며 이후 20번의 epoch 동안 loss가 10번째 loss보다 감소하지 않았다. 따라서 30번째 epoch를 마지막으로 전체 학습이 중단되었음을 알 수 있다. 최종적으로 10번째 epoch에 학습된 모형 및 가중치가 checkpoint의 경로에 저장된다.



본 예시에서는 'validation accuracy'를 기준으로 20번의 epoch 동안 모형이 향상되지 않을 경우 학습을 종료하고 최적 모형을 저장하도록 하였다.

### Basic model

Block 1		Block 2		Block 3		Flatten	Output
Convolution	Pooling	Convolution	Pooling	Convolution	Pooling		

총 3번의 합성곱-최대 풀링을 거쳐 일렬 배열 후 최종 출력층과 완전 연결된다.

```
# import Library
from tensorflow.keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Activation
from tensorflow.keras import optimizers
from tensorflow.keras.layers import BatchNormalization, Dropout

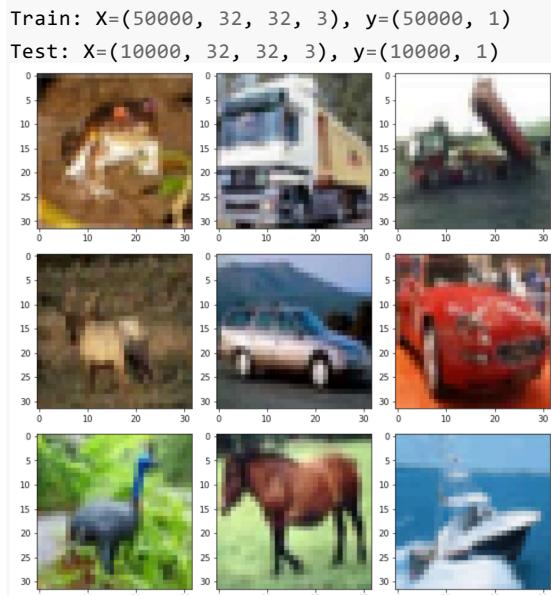
# example of Loading the cifar10 dataset
import matplotlib.pyplot as plt
from keras.datasets import cifar10

# Load dataset
(trainX, trainy), (testX, testy) = cifar10.load_data()

# summarize Loaded dataset
print('Train: X=%s, y=%s' % (trainX.shape, trainy.shape))
print('Test: X=%s, y=%s' % (testX.shape, testy.shape))

# plot first few images
for i in range(9):
    plt.subplot(3,3,i+1)
    plt.imshow(trainX[i])

# show the figure
plt.tight_layout()
```



```

trainY = to_categorical(trainy)
testY = to_categorical(testy)

# convert from integers to floats
train_norm = trainX.astype('float32')
test_norm = testX.astype('float32')

# normalize to range 0-1
train_norm = train_norm / 255.0
test_norm = test_norm / 255.0

```

## 1. Define model

```

# basic model

def basic_model():
    model = Sequential()
    model.add(Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_uniform',
                    padding='same', input_shape=(32, 32, 3)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform',
                    padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform',
                    padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())

    model.add(Dense(10, activation='softmax'))

    # compile model
    opt = optimizers.Adam(learning_rate=0.001)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])

    return model

# model summary
model = basic_model()
model.summary()

```

```
Model: "sequential_2"

Layer (type)          Output Shape         Param #
=================================================================
conv2d_4 (Conv2D)      (None, 32, 32, 16)    448
max_pooling2d_4 (MaxPooling 2D) (None, 16, 16, 16)    0
conv2d_5 (Conv2D)      (None, 16, 16, 32)    4640
max_pooling2d_5 (MaxPooling 2D) (None, 8, 8, 32)    0
conv2d_6 (Conv2D)      (None, 8, 8, 64)     18496
max_pooling2d_6 (MaxPooling 2D) (None, 4, 4, 64)    0
flatten_2 (Flatten)   (None, 1024)        0
dense_3 (Dense)       (None, 10)           10250
=====
Total params: 33,834
Trainable params: 33,834
Non-trainable params: 0
```

```
# setting callback option
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

es = EarlyStopping(monitor = 'val_accuracy', patience = 20)
mc = ModelCheckpoint("./model/checkpoint.{epoch:02d}.hdf5", monitor = "val_accuracy")
```

## 2. training

```
epochs = 100
batch_size = 32
history = model.fit(train_norm, trainY, epochs = epochs, batch_size = batch_size,
                     validation_data = (test_norm, testY), verbose =1, callbacks = [es, mc])
```

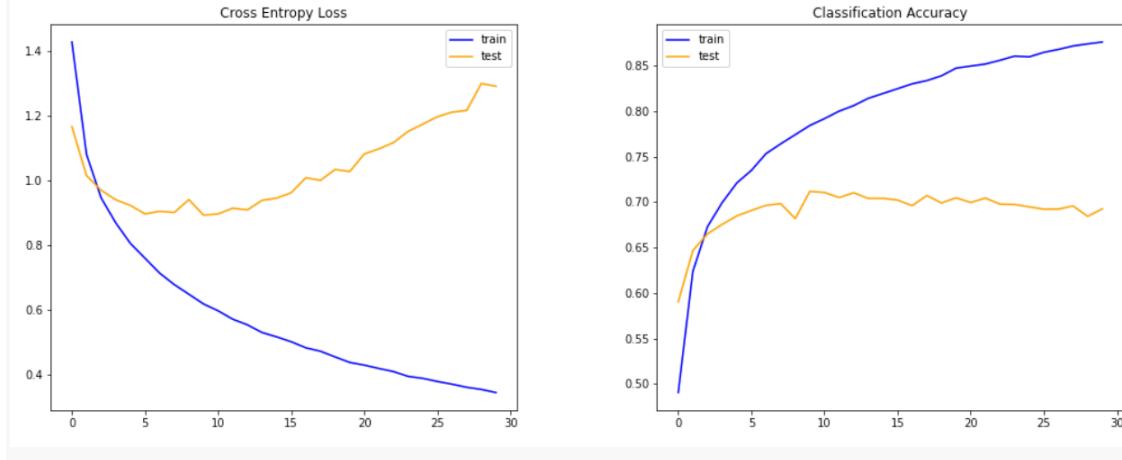
## 3. History

```
# plot diagnostic Learning curves
def summarize_diagnostics(history):
    plt.subplots(figsize = (15,6))
    plt.subplots_adjust(left=0.1, bottom=0.1, right=0.99, top=0.9, wspace=0.3, hspace=0.2)

    # plot loss
    plt.subplot(121)
    plt.title('Cross Entropy Loss')
    plt.plot(history.history['loss'], color='blue', label='train')
    plt.plot(history.history['val_loss'], color='orange', label='test')
    plt.legend()
```

```
# plot accuracy
plt.subplot(122)
plt.title('Classification Accuracy')
plt.plot(history.history['accuracy'], color='blue', label='train')
plt.plot(history.history['val_accuracy'], color='orange', label='test')
plt.legend()

summarize_diagnostics(history)
```



#### 4. Evaluation

```
# Evaluation
score = model.evaluate(test_norm, testY, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

Test loss: 1.303773283958435
Test accuracy: 0.6927000284194946
```

정확도가 0.69이다.

#### 5. Confusion matrix

```
import numpy as np
from sklearn.metrics import confusion_matrix

y_pred=model.predict(test_norm)
ypred_class=np.argmax(y_pred,axis=1)
ytest_class=np.argmax(testY,axis=1)

conf_mat=confusion_matrix(ytest_class,ypred_class)

label = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

import pandas as pd
table = pd.DataFrame(conf_mat, columns = label, index = label)
table
```

	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	764	23	39	29	15	7	17	10	64	32
automobile	27	846	6	9	3	8	11	3	21	66
bird	76	10	561	77	44	75	98	25	20	14
cat	27	16	69	508	44	181	103	28	11	13
deer	26	10	100	86	569	48	102	46	11	2
dog	15	1	59	190	34	593	52	40	8	8
frog	5	8	30	49	21	24	843	4	11	5
horse	22	7	40	60	59	80	25	690	2	15
ship	56	34	12	21	4	4	20	6	827	16
truck	27	130	11	17	6	14	16	19	34	726

```

misclassified=np.where(ypred_class!=ytest_class)
aa=np.asarray(misclassified)
print(misclassified)
print(len(misclassified[0]))


(array([ 2,  7,  8, ..., 9994, 9995, 9998]),)
3073

```

오분류된 이미지는 총 3073개이다. 잘못 예측된 몇가지 이미지를 보도록 한다.

```

import random
import matplotlib.pyplot as plt
import numpy as np

predicted_result = model.predict(test_norm)
predicted_labels = np.argmax(predicted_result, axis=1) # 확률의 최대값의 index 를
# 추출하며 여기서는 index 가 label 과 동일하다.

test_labels = np.argmax(testY, axis=1)

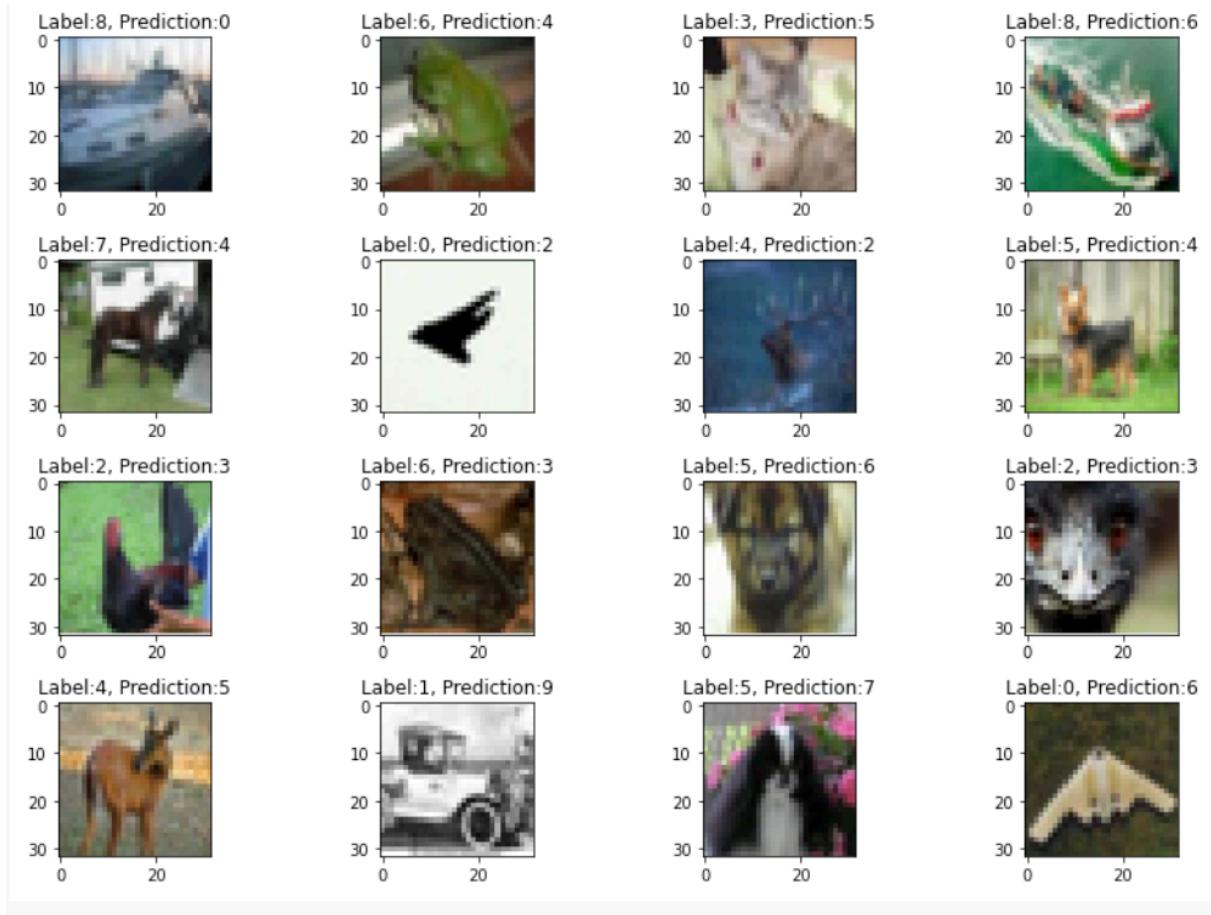
count = 0

plt.figure(figsize=(12,8))

for n in range(16):
    i = misclassified[0][n]
    count += 1
    plt.subplot(4, 4, count)
    plt.imshow(test_norm[i], cmap='Greys', interpolation='nearest')
    tmp = "Label:" + str(test_labels[i]) + ", Prediction:" + str(predicted_labels[i])
    plt.title(tmp)

plt.tight_layout()
plt.show()

```



## Deep model

Block 1				Block 2				Block 3				Flatten	Dense	Dense	Output
Conv	Conv	Conv	Pool	Conv	Conv	Conv	Pool	Conv	Conv	Conv	Pool				

블록 하나당 연속된 세 개의 합성곱 층 및 최대 풀링 층을 포함하여 총 세 블록을 거쳐 일렬 배열 후 마지막 출력 층 이전에 2개의 완전연결 은닉층을 거쳐 분류되도록 한다. 블락마다 합성곱 층의 개수를 변화시키거나, 중간에 배치 정규화 스텝의 유무를 변화시키기도하고, 완전 연결 은닉층의 개수를 변화시키며 최적의 성능을 보이는 model을 찾아보도록 한다. 위의 예시와 동일하게 epoch = 100, batch\_size = 32로 설정하고, callback option을 설정한다.

- 각 합성곱 층에서 활성화 함수에 입력되기 전 배치 정규화 스텝을 거치도록 설정하였다.
- 각 완전 연결 은닉 층 이후에는 드롭아웃(rate = 0.1) 층을 추가하여 학습마다 노드 중 10%를 제거하도록 하였다.

데이터를 불러온 뒤 전처리 과정을 거친 후 몇개의 training data를 출력하였다.

```
# import Library
from tensorflow.keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Activation
from tensorflow.keras import optimizers
from tensorflow.keras.layers import BatchNormalization, Dropout

# example of Loading the cifar10 dataset
import matplotlib.pyplot as plt
from keras.datasets import cifar10

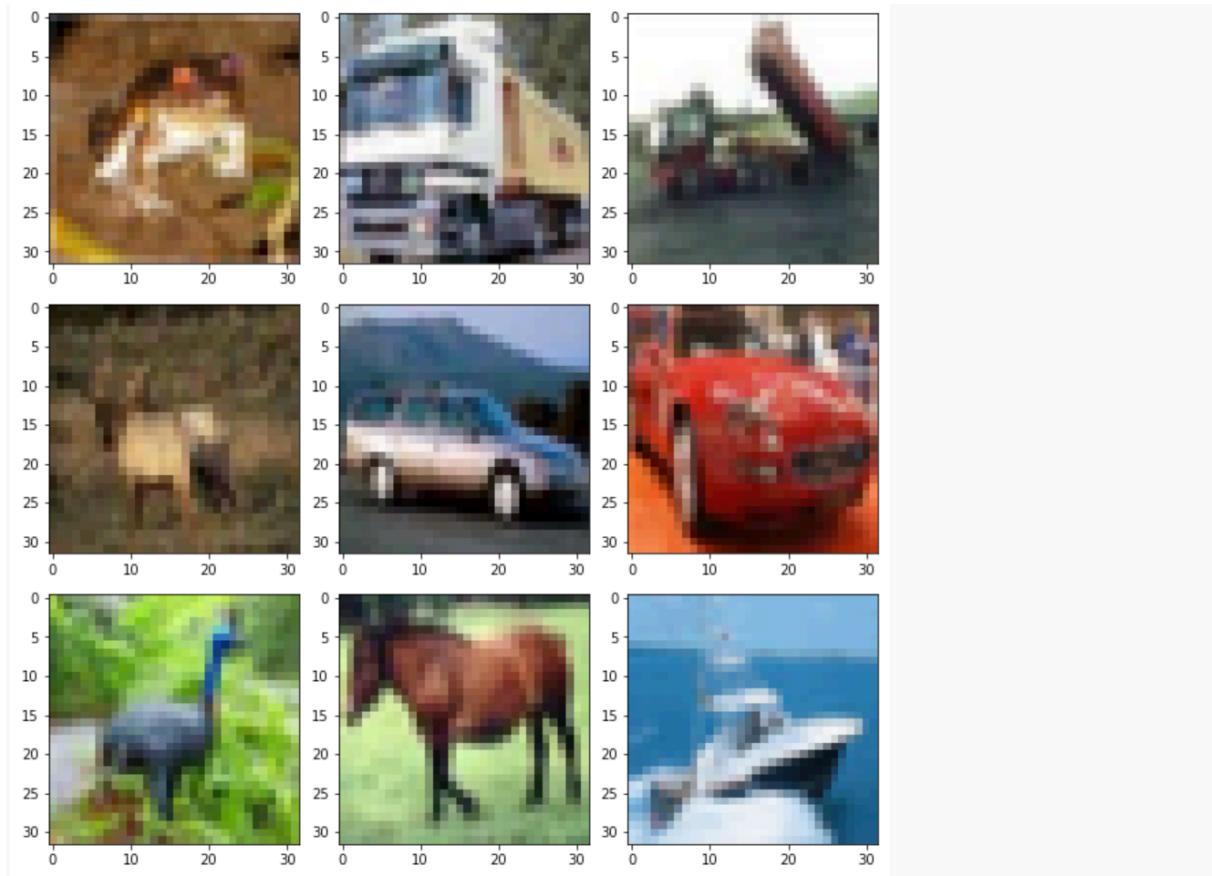
# Load dataset
(trainX, trainy), (testX, testy) = cifar10.load_data()

# summarize loaded dataset
print('Train: X=%s, y=%s' % (trainX.shape, trainy.shape))
print('Test: X=%s, y=%s' % (testX.shape, testy.shape))

# plot first few images
for i in range(9):
    plt.subplot(3,3,i+1)
    plt.imshow(trainX[i])

# show the figure
plt.tight_layout()

Train: X=(50000, 32, 32, 3), y=(50000, 1)
Test: X=(10000, 32, 32, 3), y=(10000, 1)
```



```

trainY = to_categorical(trainy)
testY = to_categorical(testy)

# convert from integers to floats
train_norm = trainX.astype('float32')
test_norm = testX.astype('float32')

# normalize to range 0-1
train_norm = train_norm / 255.0
test_norm = test_norm / 255.0

```

Callback option을 사용하여 model의 accuracy가 10번동안 좋아지지 않으면 학습을 멈추도록 설정하고, training 과정을 시각화하기 위한 함수를 정의한다.

```

# setting callback option
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

es = EarlyStopping(monitor = "val_accuracy", patience = 10)
mc = ModelCheckpoint("./model/checkpoint.{epoch:02d}.hdf5", monitor = "val_accuracy")

# plot diagnostic Learning curves
def summarize_diagnostics(history):
    plt.subplots(figsize = (15,6))
    plt.subplots_adjust(left=0.1, bottom=0.1, right=0.99, top=0.9, wspace=0.3, hspace=0.2)

    # plot loss
    plt.subplot(121)
    plt.title('Cross Entropy Loss')
    plt.plot(history.history['loss'], color='blue', label='train')

```

```
plt.plot(history.history['val_loss'], color='orange', label='test')
plt.legend()

# plot accuracy
plt.subplot(122)
plt.title('Classification Accuracy')
plt.plot(history.history['accuracy'], color='blue', label='train')
plt.plot(history.history['val_accuracy'], color='orange', label='test')
plt.legend()
```

## Model 1

### 1. Define model

```
# model 1
# NC = 1, BN = NO, FC = 0

def model_1():
    model = Sequential()
    model.add(Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_uniform',
                    padding='same', input_shape=(32, 32, 3)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform',
                    padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform',
                    padding='same'))
    model.add(MaxPooling2D((2, 2)))

    model.add(Flatten())

    model.add(Dense(10, activation='softmax'))
    model.add(Dropout(0.1))

    # compile model
    opt = optimizers.Adam(learning_rate=0.001)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])

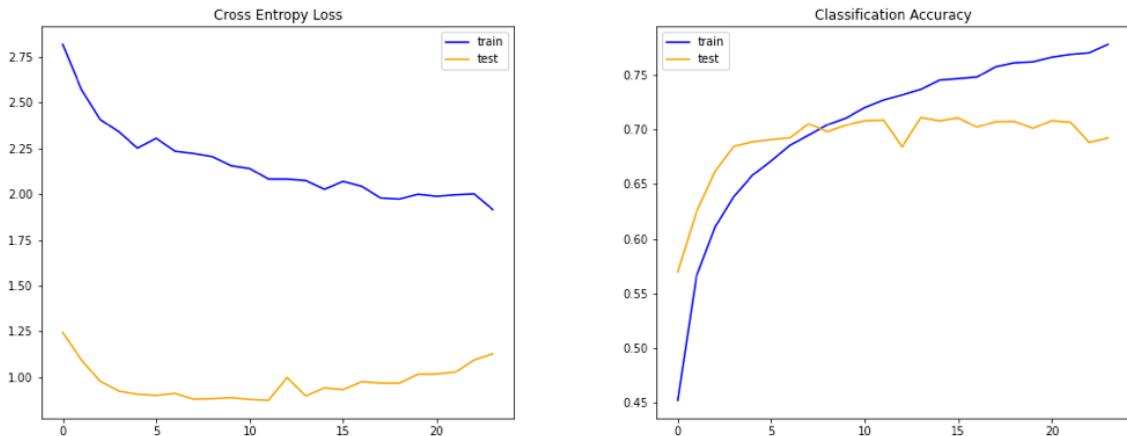
    return model
```

```
# hyperparameter
epochs = 100
batch_size = 32

# training
history = model.fit(train_norm, trainY, epochs = epochs, batch_size = batch_size,
                     validation_data = (test_norm, testY), verbose =1, callbacks = [es, mc])
```

### 2. History

```
# History
summarize_diagnostics(history)
```



### 3. Evaluation

```
# Evaluation
score = model.evaluate(test_norm, testY, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

Test loss: 1.1286035776138306
Test accuracy: 0.6923999786376953
```

모델들의 accuracy를 히스토그램으로 시각화하여 비교하기 위해 model의 accuracy를 저장하는 list를 만든다.

```
accuracy_list = []
accuracy_list.append(round(score[1],4))
```

### 4. Confusion matrix

```
# confusion matrix

import numpy as np
from sklearn.metrics import confusion_matrix

y_pred=model.predict(test_norm)

ypred_class=np.argmax(y_pred, axis=1)
ytest_class=np.argmax(testY, axis=1)

conf_mat=confusion_matrix(ytest_class,ypred_class)

label = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

import pandas as pd
table = pd.DataFrame(conf_mat, columns = label, index = label)
table
```

	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	850	13	28	15	12	10	2	7	44	19
automobile	50	816	3	11	5	12	2	8	31	62
bird	93	6	600	38	87	84	34	43	12	3
cat	60	11	70	438	85	234	30	41	21	10
deer	45	4	69	43	642	57	29	97	11	3
dog	33	4	44	121	42	669	13	63	11	0
frog	16	8	82	57	66	67	683	11	4	6
horse	37	2	38	30	51	70	1	757	3	11
ship	113	28	15	5	8	12	3	5	796	15
truck	81	118	10	7	6	20	9	27	49	673

## Model 2

### 1. Define model

```
# model 2
# NC = 2, BN = NO, FC = 0

def model_2():
    model = Sequential()
    model.add(Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_uniform',
                    padding='same', input_shape=(32, 32, 3)))
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform',
                    padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_uniform',
                    padding='same'))
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform',
                    padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_uniform',
                    padding='same'))
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform',
                    padding='same'))
    model.add(MaxPooling2D((2, 2)))

    model.add(Flatten())

    model.add(Dense(10, activation='softmax'))
    model.add(Dropout(0.1))

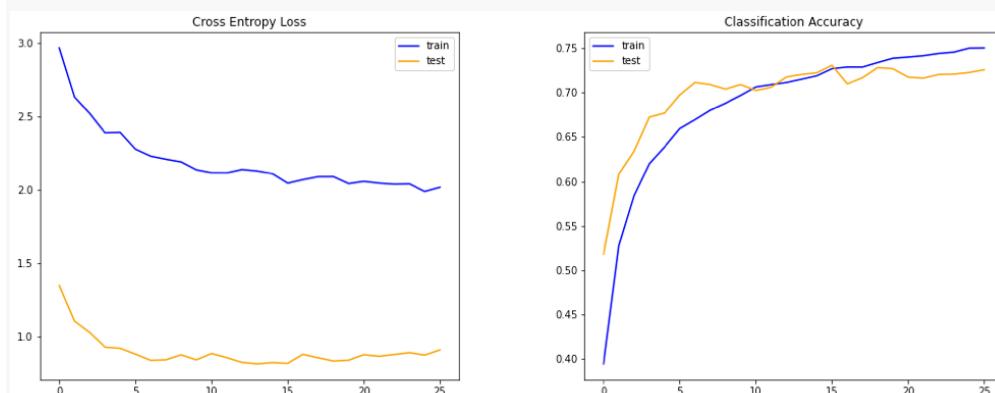
    # compile model
    opt = optimizers.Adam(learning_rate=0.001)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])

    return model
```

```
# training
history = model.fit(train_norm, trainY, epochs = epochs, batch_size = batch_size,
                      validation_data = (test_norm, testY), verbose =1, callbacks = [es, mc])
```

### 2. History

```
# History
summarize_diagnostics(history)
```



### 3. Evaluation

```
# Evaluation
score = model.evaluate(test_norm, testY, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

Test loss: 0.9082899689674377
Test accuracy: 0.7257999777793884

accuracy_list.append(round(score[1],4))
```

### 4. Confusion matrix

```
# confusion matrix

import numpy as np
from sklearn.metrics import confusion_matrix

y_pred=model.predict(test_norm)
ypred_class=np.argmax(y_pred,axis=1)
ytest_class=np.argmax(testY,axis=1)

conf_mat=confusion_matrix(ytest_class,ypred_class)

label = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

import pandas as pd
table = pd.DataFrame(conf_mat, columns = label, index = label)
table
```

	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	781	17	38	12	13	7	7	11	56	58
automobile	11	864	3	4	1	3	5	3	9	97
bird	92	6	557	34	102	75	54	47	10	23
cat	29	17	56	403	83	213	77	73	18	31
deer	21	6	35	35	685	41	34	127	8	8
dog	13	6	37	98	37	674	19	89	10	17
frog	9	7	37	49	56	25	778	12	12	15
horse	21	3	24	12	41	47	3	832	4	13
ship	81	35	12	4	5	5	8	6	801	43
truck	22	55	4	1	3	5	5	11	11	883

### Model 3

#### 1. Define model

```
# model 3
# NC = 2, BN = YES, FC = 0

def model_3():
    model = Sequential()

    # first convolutional Layer
    model.add(Conv2D(16, (3, 3), kernel_initializer='he_uniform', padding='same',
                    input_shape=(32, 32, 3)))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Conv2D(32, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2)))

    # second convolutional layer
    model.add(Conv2D(16, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Conv2D(32, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2)))

    # third convolutional layer
    model.add(Conv2D(16, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Conv2D(32, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2)))

    # fully connected classifier
    model.add(Flatten())
    model.add(Dense(10, activation='softmax'))
    model.add(Dropout(0.1))

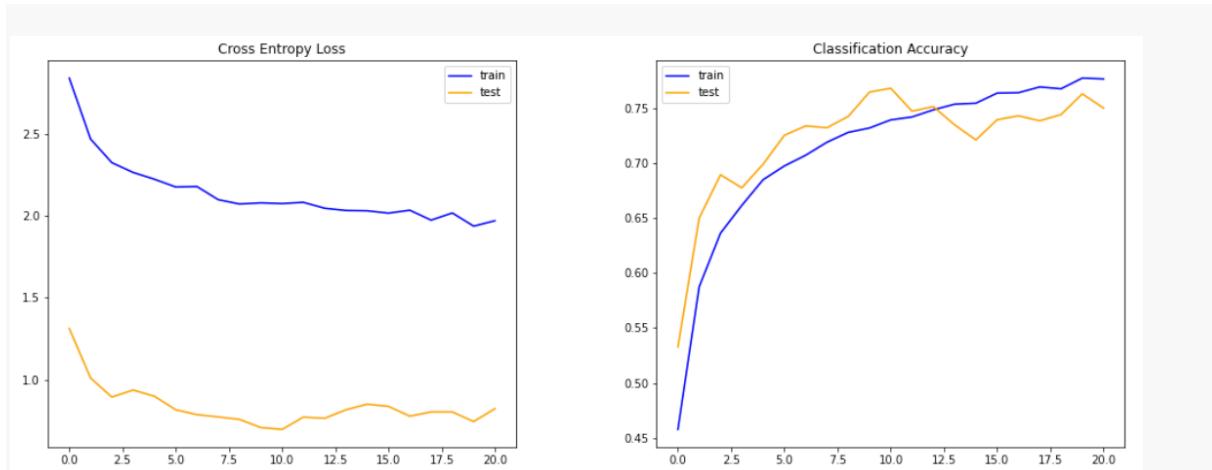
    # compile model
    opt = optimizers.Adam(learning_rate=0.001)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])

    return model

# training
history = model.fit(train_norm, trainY, epochs = epochs, batch_size = batch_size,
                     validation_data = (test_norm, testY), verbose =1, callbacks = [es, mc])
```

#### 2. History

```
# History
summarize_diagnostics(history)
```



### 3. Evaluation

```
# Evaluation
score = model.evaluate(test_norm, testY, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

Test loss: 0.839942455291748
Test accuracy: 0.744700014591217

accuracy_list.append(round(score[1],4))
```

### 4. Confusion matrix

Confusion matrix를 구하는 코드는 위의 예제와 동일하므로 생략한다.

	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	766	16	84	19	23	3	9	5	45	30
automobile	16	853	14	2	3	5	12	1	21	73
bird	38	4	746	53	50	53	35	11	5	5
cat	19	2	106	580	50	176	38	14	7	8
deer	16	1	134	74	662	47	42	18	5	1
dog	11	1	85	120	35	709	11	22	2	4
frog	7	3	71	93	21	33	764	1	5	2
horse	14	4	88	44	71	91	3	673	1	11
ship	67	16	24	12	5	5	3	2	850	16
truck	35	62	7	11	5	3	5	4	24	844

## Model 4

### 1. Define model

```
# model 4
# NC = 2, BN = YES, FC = 1

def model_4():
    model = Sequential()

    # first convolutional Layer
    model.add(Conv2D(16, (3, 3), kernel_initializer='he_uniform', padding='same',
                    input_shape=(32, 32, 3)))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Conv2D(32, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2)))

    # second convolutional Layer
    model.add(Conv2D(16, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Conv2D(32, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2)))

    # third convolutional Layer
    model.add(Conv2D(16, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Conv2D(32, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2)))

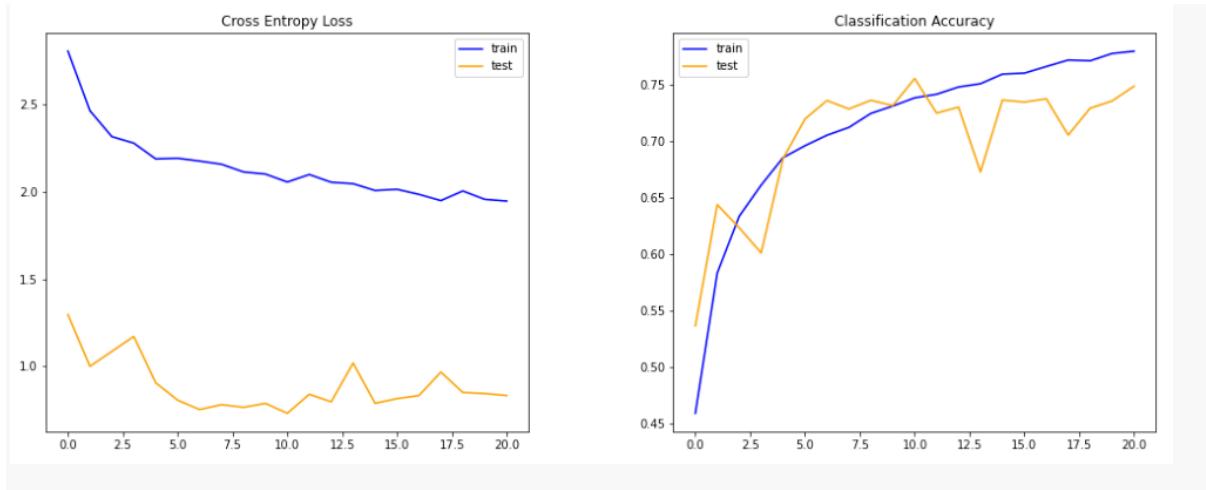
    # fully connected classifier
    model.add(Flatten())
    model.add(Dense(50, activation='relu'))
    model.add(Dense(10, activation='softmax'))
    model.add(Dropout(0.1))

    # compile model
    opt = optimizers.Adam(learning_rate=0.001)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])

    return model
```

### 2. History

```
# History
summarize_diagnostics(history)
```



### 3. Evaluation

```
# Evaluation
score = model.evaluate(test_norm, testY, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

Test loss: 0.8327572345733643
Test accuracy: 0.748199999332428

accuracy_list.append(round(score[1],4))
```

### 4. Confusion matrix

	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	771	33	38	36	8	5	13	10	39	47
automobile	2	914	1	4	1	1	8	0	13	56
bird	63	12	593	65	33	94	88	32	2	18
cat	9	12	33	553	15	257	74	20	7	20
deer	15	5	61	94	562	80	103	69	5	6
dog	9	4	24	118	10	780	22	23	1	9
frog	7	4	24	52	5	28	867	5	3	5
horse	12	5	20	48	19	105	14	757	2	18
ship	69	45	8	23	3	4	11	2	797	38
truck	18	64	3	9	1	1	4	3	9	888

## Model 5

### 1. Define model

```
# model 5
# NC = 2, BN = YES, FC = 2

def model_5():
    model = Sequential()

    # first convolutional Layer
    model.add(Conv2D(16, (3, 3), kernel_initializer='he_uniform', padding='same',
                    input_shape=(32, 32, 3)))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Conv2D(32, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2)))

    # second convolutional Layer
    model.add(Conv2D(16, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Conv2D(32, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2)))

    # third convolutional Layer
    model.add(Conv2D(16, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Conv2D(32, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2)))

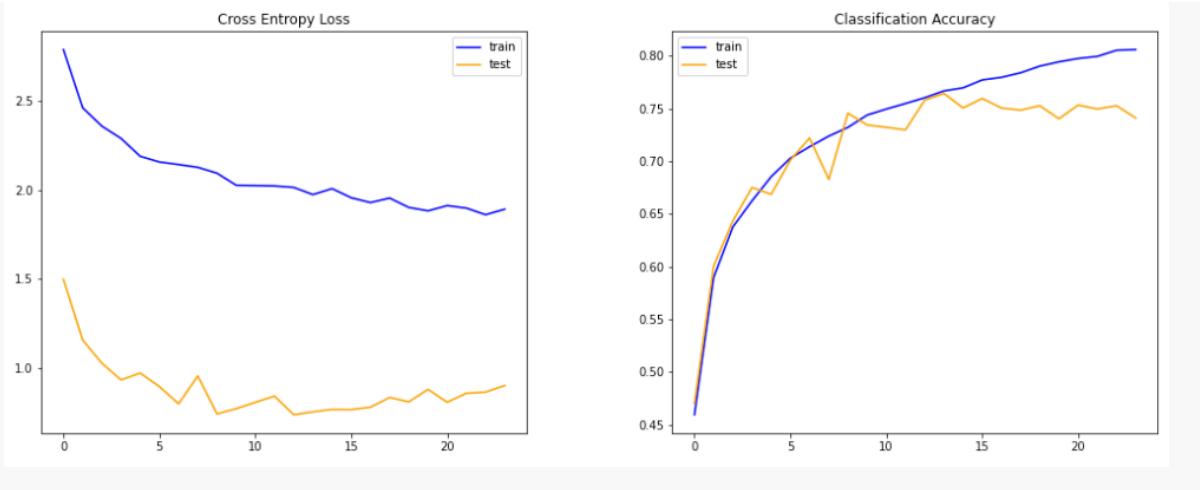
    # fully connected classifier
    model.add(Flatten())
    model.add(Dense(100, activation='relu'))
    model.add(Dense(50, activation='relu'))
    model.add(Dense(10, activation='softmax'))
    model.add(Dropout(0.1))

    # compile model
    opt = optimizers.Adam(learning_rate=0.001)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])

    return model
```

### 2. History

```
# History
summarize_diagnostics(history)
```



### 3. Evaluation

```
# Evaluation
score = model.evaluate(test_norm, testY, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

Test loss: 0.9000278115272522
Test accuracy: 0.7408999800682068

accuracy_list.append(round(score[1],4))
```

### 4. Confusion matrix

	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	674	38	98	61	16	15	15	7	31	45
automobile	5	913	1	7	2	5	11	3	2	51
bird	41	1	634	77	51	95	73	21	2	5
cat	8	6	41	652	30	177	63	22	0	1
deer	6	2	54	104	654	80	45	53	1	1
dog	5	4	21	166	23	742	17	21	0	1
frog	5	2	31	69	14	28	843	3	2	3
horse	5	10	16	50	32	104	10	766	0	7
ship	44	78	22	55	4	12	23	5	730	27
truck	16	119	4	13	3	15	12	6	11	801

## Model 6

### 1. Define model

```
# model 6
# NC = 3, BN = YES, FC = 0

def model_6():
    model = Sequential()

    # first convolutional Layer
    model.add(Conv2D(16, (3, 3), kernel_initializer='he_uniform', padding='same',
                    input_shape=(32, 32, 3)))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Conv2D(32, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Conv2D(64, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2)))

    # second convolutional Layer
    model.add(Conv2D(16, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Conv2D(32, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Conv2D(64, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2)))

    # third convolutional Layer
    model.add(Conv2D(16, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Conv2D(32, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Conv2D(64, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2)))

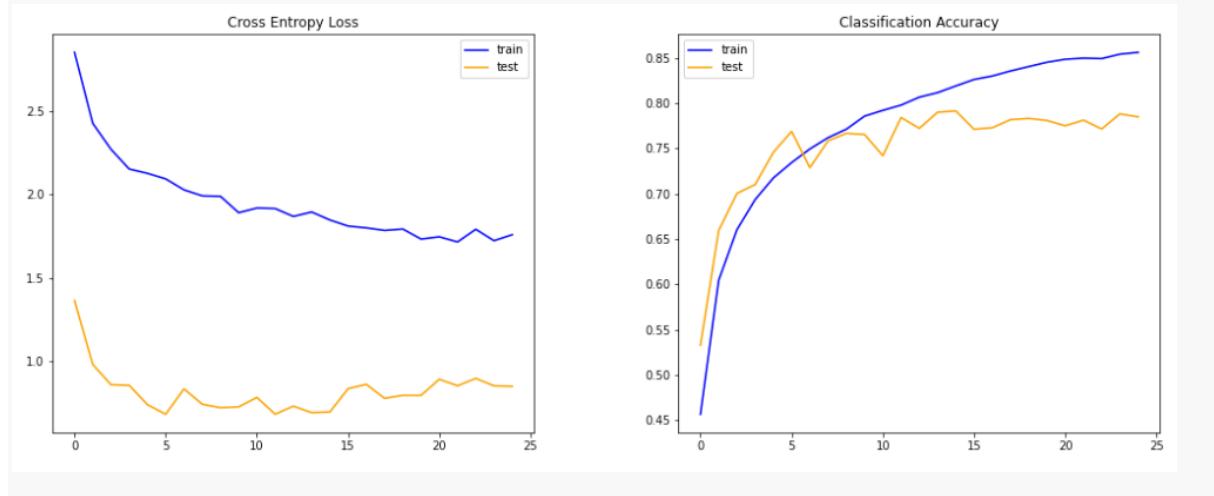
    # fully connected classifier
    model.add(Flatten())
    model.add(Dense(10, activation='softmax'))
    model.add(Dropout(0.1))

    # compile model
    opt = optimizers.Adam(learning_rate=0.001)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])

    return model
```

## 2. History

```
# History
summarize_diagnostics(history)
```



## 3. Evaluation

```
# Evaluation
score = model.evaluate(test_norm, testY, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

Test loss: 0.8507578372955322
Test accuracy: 0.7849000096321106
```

```
accuracy_list.append(round(score[1],4))
```

## 4. Confusion matrix

	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	806	21	47	13	20	4	6	8	53	22
automobile	8	937	0	2	1	0	4	1	10	37
bird	57	3	731	56	50	28	45	12	13	5
cat	24	9	77	674	49	85	34	20	18	10
deer	11	5	70	58	774	22	29	22	8	1
dog	16	5	63	212	41	600	25	30	3	5
frog	9	17	38	64	26	6	827	2	8	3
horse	16	4	51	44	60	28	8	774	3	12
ship	47	36	16	4	3	1	4	2	873	14
truck	19	84	7	5	4	1	2	8	17	853

## Model 7

### 1. Define model

```
# model 7
# NC = 3, BN = YES, FC = 1

def model_7():
    model = Sequential()

    # first convolutional layer
    model.add(Conv2D(16, (3, 3), kernel_initializer='he_uniform', padding='same',
                    input_shape=(32, 32, 3)))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Conv2D(32, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Conv2D(64, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2)))

    # second convolutional layer
    model.add(Conv2D(16, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Conv2D(32, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Conv2D(64, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2)))

    # third convolutional layer
    model.add(Conv2D(16, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Conv2D(32, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Conv2D(64, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2)))

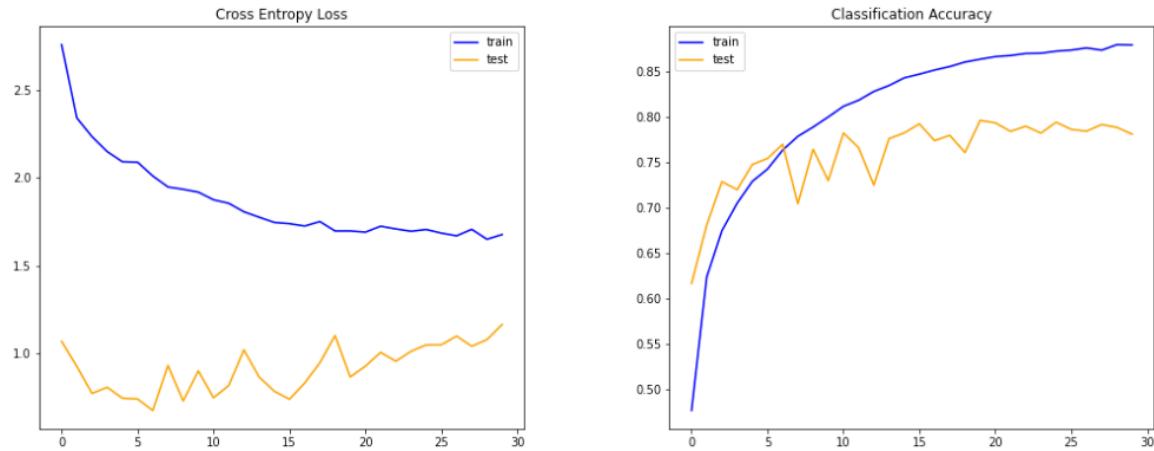
    # fully connected classifier
    model.add(Flatten())
    model.add(Dense(1024, activation='relu'))
    model.add(Dense(10, activation='softmax'))
    model.add(Dropout(0.1))

    # compile model
    opt = optimizers.Adam(learning_rate=0.001)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])

    return model
```

## 2. History

```
# History
summarize_diagnostics(history)
```



## 3. Evaluation

```
# Evaluation
score = model.evaluate(test_norm, testY, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

Test loss: 1.1642721891403198
Test accuracy: 0.7811999917030334
```

```
accuracy_list.append(round(score[1],4))
```

## 4. Confusion matrix

	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck	
airplane	806		18	78	18	17	2	16	11	22	12
automobile	10		904	8	2	1	0	19	0	20	36
bird	41		4	723	39	54	30	81	22	4	2
cat	17		3	81	591	57	63	139	35	10	4
deer	11		1	44	45	798	7	56	36	2	0
dog	6		3	67	151	46	599	73	46	4	5
frog	4		2	25	23	14	3	923	1	4	1
horse	7		5	25	31	72	20	13	823	1	3
ship	82		24	8	14	5	0	17	3	832	15
truck	24		94	8	7	7	2	17	12	16	813

## Model 8

### 1. Define model

```
# model 8
# NC = 3, BN = YES, FC = 2

def model_8():
    model = Sequential()

    # first convolutional layer
    model.add(Conv2D(16, (3, 3), kernel_initializer='he_uniform', padding='same',
                    input_shape=(32, 32, 3)))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Conv2D(32, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Conv2D(64, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2)))

    # second convolutional layer
    model.add(Conv2D(16, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Conv2D(32, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Conv2D(64, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2)))

    # third convolutional layer
    model.add(Conv2D(16, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Conv2D(32, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Conv2D(64, (3, 3), kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2)))

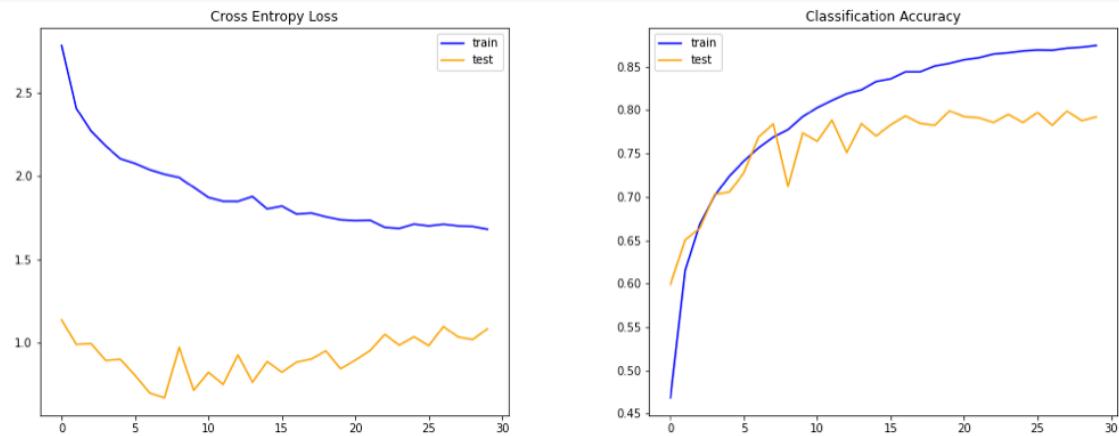
    # fully connected classifier
    model.add(Flatten())
    model.add(Dense(1024, activation='relu'))
    model.add(Dense(512, activation='relu'))
    model.add(Dense(10, activation='softmax'))
    model.add(Dropout(0.1))

    # compile model
    opt = optimizers.Adam(learning_rate=0.001)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
```

```
return model
```

## 2. History

```
# History
summarize_diagnostics(history)
```



## 3. Evaluation

```
# Evaluation
score = model.evaluate(test_norm, testY, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

Test loss: 1.082869291305542
Test accuracy: 0.7919999957084656
```

```
accuracy_list.append(round(score[1],4))
```

## 4. Confusion matrix

	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	827	16	52	12	7	1	5	6	53	21
automobile	20	919	4	1	2	0	5	0	16	33
bird	52	3	763	31	46	24	59	9	10	3
cat	25	9	79	601	45	84	109	20	17	11
deer	13	1	68	50	775	13	46	22	7	5
dog	11	2	72	179	44	599	54	33	3	3
frog	11	2	46	24	16	3	887	1	9	1
horse	17	6	58	26	60	26	12	782	5	8
ship	36	15	10	4	5	2	5	0	906	17
truck	19	66	3	7	1	1	6	6	30	861

8개의 모델들 간의 성능을 비교하기 위해 accuracy를 histogram으로 그린다.

```
from pandas import Series, DataFrame
import numpy as np

df = {'model' : ['model1', 'model2', 'model3', 'model4', 'model5', 'model6', 'model7', 'model8'],
      'accuracy' : accuracy_list}

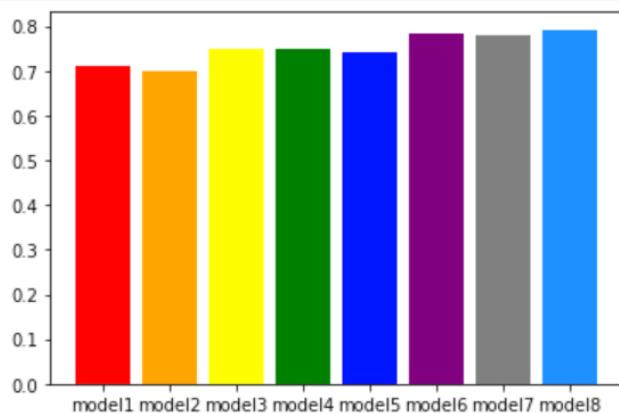
acc_df = DataFrame(df)
acc_df
```

	model	accuracy
0	model1	0.7100
1	model2	0.7001
2	model3	0.7486
3	model4	0.7482
4	model5	0.7409
5	model6	0.7849
6	model7	0.7812
7	model8	0.7920

```
import matplotlib.pyplot as plt

colors = ['red', 'orange', 'yellow', 'green', 'blue', 'purple', 'Gray', 'dodgerblue']
plt.bar(acc_df['model'], acc_df['accuracy'], color = colors)
plt.show()

print("The best model :", acc_df.loc[np.argmax(accuracy_list)][0])
print("Accuracy of the best model :", acc_df.loc[np.argmax(accuracy_list)][1] )
```



```
The best model : model8
Accuracy of the best model : 0.792
```

Model	CL	BN	FC	accuracy
Model 1	1	NO	0	0.7100
Model 2	2	NO	0	0.7001
Model 3	2	YES	0	0.7486
Model 4	2	YES	1	0.7482
Model 5	2	YES	2	0.7409
Model 6	3	YES	0	0.7849
Model 7	3	YES	1	0.7812
Model 8	3	YES	2	0.7920

- **CL**: 블록 당 연속된 합성곱 층의 수
- **BN**: 합성곱 층의 배치 정규화 여부
- **FC**: 마지막 출력 층 이전의 완전 연결 은닉층 수

Test data set에 대하여 accuracy가 가장 높은 모형은 블록당 세개의 합성곱 층과 배치 정규화를 거쳐 은닉 층 없이 최종 출력층으로 완전연결되는 model6과 2개의 은닉층을 통과 후 완전 연결되는 model8이다.

블록 당 합성곱 층의 수가 증가할수록 test data에서 accuracy가 향상되었다. 또한, 배치 정규화 단계를 추가 함으로써 정확도가 높아가는 것을 확인하였다. 그러나, 마지막 블록에 완전 연결 층을 추가하면 정확도 향상에 도움이 된다고 하기에는 어려워 보인다.

cifar10 데이터 셋을 이용해 합성곱 신경망의 다양한 구조를 도입하여 정확도를 향상시켜 보았다. 데이터 및 다양한 하이퍼 파라미터, 옵티마이저 등의 설정에 따라 최적의 성능을 보이는 모형의 구조는 정해져 있지 않으므로, 모형의 옵션을 점진적으로 조정해보고 Cross-Validation을 통해 최적의 모형을 선택하도록 할 수 있다. 예를 들면, `kernel_initializer = 'he_normal'`을 설정할 수도 있고, `optimizer`를 SGD, RMSprop를 사용해 볼 수도 있다.

마지막으로 딥러닝 모델의 경우 랜덤 시드를 설정하지 않기 때문에 모델을 학습시킬 때 마다 결과값이 매번 달라진다. 따라서, model 8이 가장 좋은 성능을 보이는 경우가 있고 model 6이 가장 좋은 성능을 보일 때도 있는데, 위의 결과를 통해서도 알 수 있듯 블록당 연결된 합성곱 층의 개수(CL)가 증가하는 것은 모델의 performance 향상에 큰 도움이 된다.

### 3) Pre-trained model

DNN에서 모델의 성능을 높이기 위한 하나의 hyper-parameter로 초기값 설정에 대한 내용을 배웠다. 'he'나 'Xavier'처럼 초기값을 생성하는 것이 아닌 비슷한 문제를 학습한 모델의 가중치를 사용하는 방법이 있다. 이런 방법을 pre-trained model이라고 한다.

유명한 이미지 분류대회인 ILSVRC 이미지넷 대회에서 최근 몇년간 좋은 성능을 보이는 CNN 모델이 개발되었다. 대회에서 우승한 모델들을 pre-trained model로 가지고 와 학습을 하는 경우 model의 성능이 얼만큼 향상되는지 본다.

CNN의 2가지 pre-trained model의 구조와 model을 불러와 코드를 작성하는 방법에 대해 알아보고자 한다. VGG-16(2014), Inception(2014)에 대해 소개한다.

#### a) Import image data

Example 3에서 이용할 데이터는 kaggle에서 제공하는 강아지와 고양이 이미지 데이터로 25,000개의 train data와 10,000개의 test data로 구성되어 있다. 케글에서 제공하는 데이터를 모두 이용할 경우 모델을 돌리는데 많은 시간이 소요되므로 데이터의 개수를 축소하여 이용하도록 한다. 아래의 코드는 인터넷에서 이미지 zip 파일을 다운받는 코드로 wget을 미리 설치해야 이용가능하다.

NOTE)

Window

<https://sound10000w.tistory.com/229>

Mac

<https://somjang.tistory.com/entry/MAC-맥북-터미널에서-wget-설치하여-사용하는-방법-feat-HomeBrew>

```
!wget --no-check-certificate \
  https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip \
  -O /tmp/cats_and_dogs_filtered.zip
```

```
# Import Library
import os
import zipfile
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers
from tensorflow.keras import Model
import matplotlib.pyplot as plt
```

#### b) Preparing the dataset

데이터를 우선 train data/validation data로 나눈 후, 강아지 사진과 고양이 사진을 분리한다.

```
local_zip = '/tmp/cats_and_dogs_filtered.zip'
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('/tmp')
zip_ref.close()
```

```

base_dir = '/tmp/cats_and_dogs_filtered'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')

# Directory with our training cat pictures
train_cats_dir = os.path.join(train_dir, 'cats')

# Directory with our training dog pictures
train_dogs_dir = os.path.join(train_dir, 'dogs')

# Directory with our validation cat pictures
validation_cats_dir = os.path.join(validation_dir, 'cats')

# Directory with our validation dog pictures
validation_dogs_dir = os.path.join(validation_dir, 'dogs')

# Set up matplotlib fig, and size it to fit 4x4 pics

import matplotlib.image as mpimg
nrows = 4
ncols = 4

fig = plt.gcf()
fig.set_size_inches(ncols*4, nrows*4)
pic_index = 100
train_cat_fnames = os.listdir( train_cats_dir )
train_dog_fnames = os.listdir( train_dogs_dir )

next_cat_pix = [os.path.join(train_cats_dir, fname)
                for fname in train_cat_fnames[ pic_index-8:pic_index]
                ]

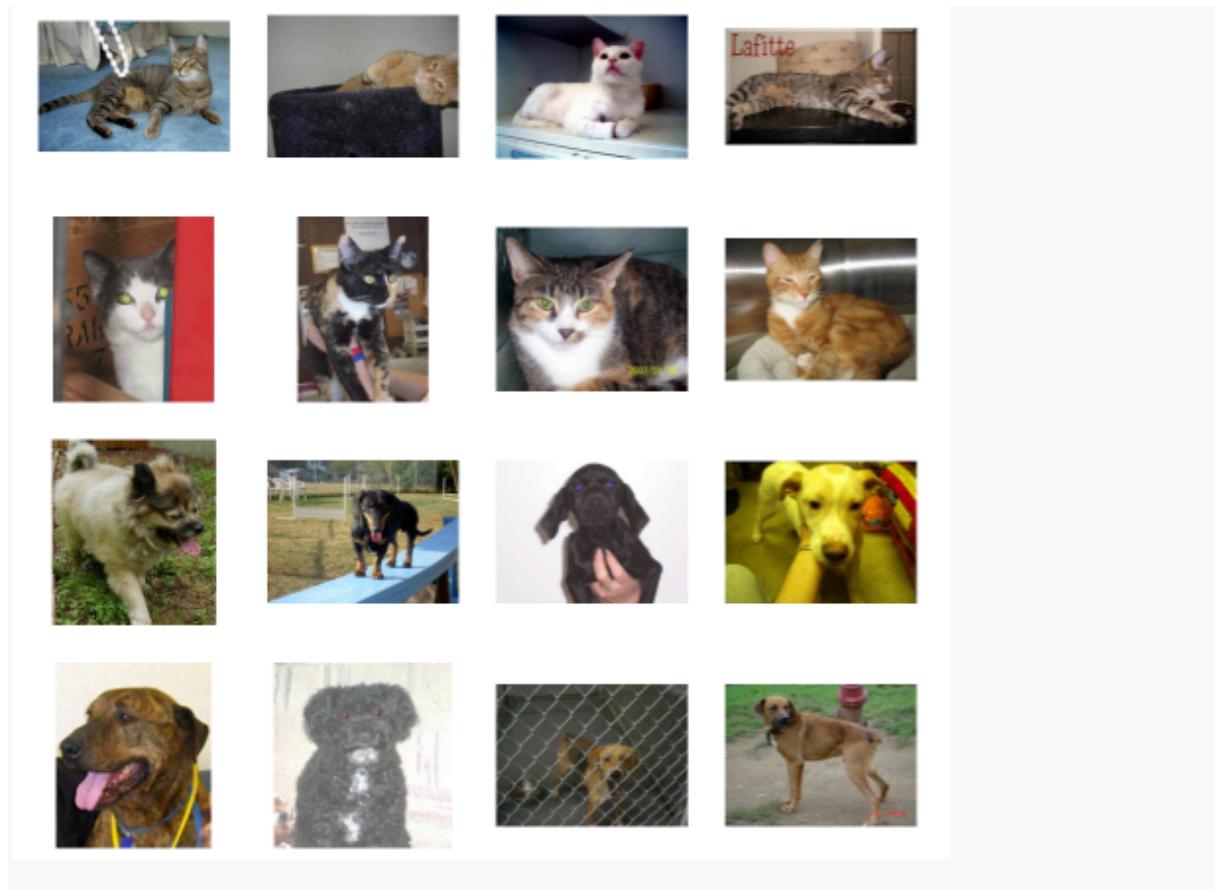
next_dog_pix = [os.path.join(train_dogs_dir, fname)
                for fname in train_dog_fnames[ pic_index-8:pic_index]
                ]

for i, img_path in enumerate(next_cat_pix+next_dog_pix):
    # Set up subplot; subplot indices start at 1
    sp = plt.subplot(nrows, ncols, i + 1)
    sp.axis('Off') # Don't show axes (or gridLines)

    img = mpimg.imread(img_path)
    plt.imshow(img)

plt.show()

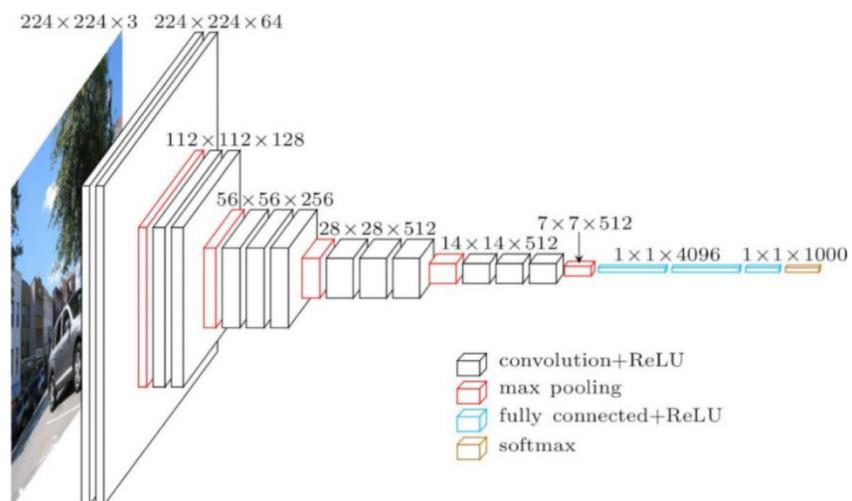
```



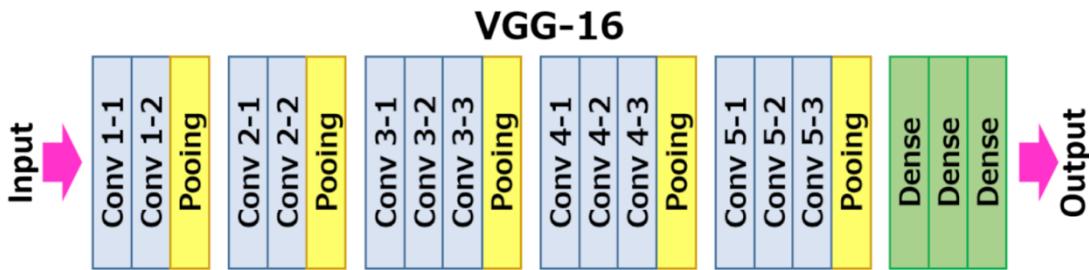
### c) Pre-trained models for image classification

#### c-1) VGG-16

ILSVRC 2014 대회 2등은 옥스퍼드 대학교 VGG 연구소의 Karen Simonyan과 Andrew Zisserman이 개발한 VGGNet이다. VGGNet은 매우 단순하고 고전적인 구조이다. 2개 또는 3개의 합성곱 층 뒤에 풀링 층이 나오고 다시 2개 또는 3개의 합성곱 층과 풀링 층이 등장하는 식이다. VGG-16은 16개의 합성곱 층이 존재하는 것으로 의미하며 19개의 합성곱 층으로 이루어진 VGG-19도 있다. 마지막 완전 연결 층은 2개의 은닉층과 출력층으로 이루어진다. 다른 모델과 다르게 VGGNet은 3x3 필터만 사용한다.



아래 그림은 VGG-16의 model의 구조이다.



그림에서 알 수 있듯 convolutional layers = 13, dense layers = 3으로 총 16개의 층으로 이루어져 있다. 각 층에서 3x3 size의 filter를 사용하며 activation function은 relu로 설정하여 parameter의 수를 줄일 수 있다.

VGG-16 model을 Pre-trained model로 이용하여 모델을 만들어 보고자 한다. 모델을 정의하기 이전에 데이터를 생성하는 방법에 소개한다.

Data Augmentation(이미지 증식)이란 train data를 랜덤하게 여러 간격으로 이동하거나 수평으로 뒤집고 조명을 바꾸는 식으로 train sample을 인공적으로 생성하여 train data의 수를 늘리는 방법이다. 이미지 증식 기법은 overfitting을 줄이므로 규제기법의 하나이며 이 기법을 이용해 생성된 샘플은 가능한 진짜에 가까워야 한다. 다시 말해, 생성된 훈련 세트에서 이미지를 뽑았을 때 이미지 증식 기법으로 만들어진 것인지 원본 이미지인지 구분할 수 없어야 한다는 것이다. 예를 들어, 각기 다른 양으로 train dataset에 있는 모든 이미지의 크기를 조금 변경하거나 이동, 회전을 한다. 이렇게 만든 이미지를 train dataset에 추가하면 model이 학습하는 과정에서 그림에 있는 물체의 위치, 방향, 크기 변화에 덜 민감해진다. 또 다른 방식으로 조명 조건에 민감하지 않은 모델을 만들기 위해 비슷하게 여러 가지 명암을 가진 이미지를 생성할 수 있다.



train data에 대해서는 scaling 과정과 Image Augmentation을 모두 적용시키고 validation data에는 scaling 과정만 적용시킨다.

```
# Add our data-augmentation parameters to ImageDataGenerator
train_datagen = ImageDataGenerator(rescale = 1./255.,      # normalization
                                    rotation_range = 40, width_shift_range = 0.2,
                                    height_shift_range = 0.2, shear_range = 0.2, zoom_range = 0.2, horizontal_flip = True)

# Note that the validation data should not be augmented!
test_datagen = ImageDataGenerator( rescale = 1.0/255. )

# Flow training images in batches of 20 using train_datagen generator
train_generator = train_datagen.flow_from_directory(train_dir, batch_size = 20, class_mode =
'binary', target_size = (224, 224))
```

```
# Flow validation images in batches of 20 using test_datagen generator
validation_generator = test_datagen.flow_from_directory(validation_dir, batch_size = 20,
class_mode = 'binary', target_size = (224, 224))
```

다음으로 VGG-16 기본 모델을 import 한다. 이때, model의 마지막 층만 변경 가능하다. pre-trained VGG-16의 경우 1000 classes의 classification 문제에 대한 model이었으므로 본 예제에서 다루는 데이터에 맞춰 마지막 층을 binary classification problem으로 변경한다.

```
from tensorflow.keras.applications.vgg16 import VGG16

base_model = VGG16(input_shape = (224, 224, 3), # Shape of our images
                    include_top = False, # Leave out the last fully connected layer
                    weights = 'imagenet')

for layer in base_model.layers:
    layer.trainable = False
```

binary classification problem에 맞게 마지막 층을 설정 후 모델을 학습한다.

```
# Flatten the output layer to 1 dimension
x = layers.Flatten()(base_model.output)

# Add a fully connected layer with 512 hidden units and ReLU activation
x = layers.Dense(512, activation='relu')(x)

# Add a dropout rate of 0.5
x = layers.Dropout(0.5)(x)

# Add a final sigmoid layer with 1 node for classification output
x = layers.Dense(1, activation='sigmoid')(x)

model = tf.keras.models.Model(base_model.input, x)

model.compile(optimizer = tf.keras.optimizers.RMSprop(learning_rate=0.0001), loss =
'binary_crossentropy', metrics = ['acc'])

vgghist = model.fit(train_generator, validation_data = validation_generator,
                     steps_per_epoch = 100, epochs = 10)
```

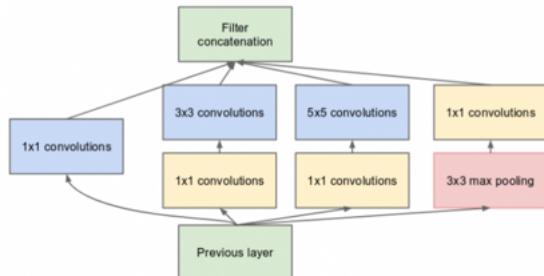
```
Epoch 1/10
100/100 [=====] - 31s 290ms/step - loss: 0.7211 - acc: 0.6595 - val_loss: 0.3012 - val_acc: 0.8790
Epoch 2/10
100/100 [=====] - 31s 307ms/step - loss: 0.4779 - acc: 0.7795 - val_loss: 0.2665 - val_acc: 0.8870
Epoch 3/10
100/100 [=====] - 28s 280ms/step - loss: 0.4323 - acc: 0.8065 - val_loss: 0.2544 - val_acc: 0.8940
Epoch 4/10
100/100 [=====] - 29s 288ms/step - loss: 0.3730 - acc: 0.8305 - val_loss: 0.4388 - val_acc: 0.7970
Epoch 5/10
100/100 [=====] - 31s 308ms/step - loss: 0.3751 - acc: 0.8290 - val_loss: 0.2783 - val_acc: 0.8740
Epoch 6/10
```

```
100/100 [=====] - 27s 265ms/step - loss: 0.3646 - acc: 0.8445 - val_loss:  
0.2082 - val_acc: 0.9120  
Epoch 7/10  
100/100 [=====] - 31s 311ms/step - loss: 0.3504 - acc: 0.8475 - val_loss:  
0.2229 - val_acc: 0.9090  
Epoch 8/10  
100/100 [=====] - 27s 273ms/step - loss: 0.3319 - acc: 0.8545 - val_loss:  
0.2098 - val_acc: 0.9150  
Epoch 9/10  
100/100 [=====] - 28s 281ms/step - loss: 0.3326 - acc: 0.8585 - val_loss:  
0.2703 - val_acc: 0.8790  
Epoch 10/10  
100/100 [=====] - 27s 271ms/step - loss: 0.3250 - acc: 0.8660 - val_loss:  
0.1962 - val_acc: 0.9240
```

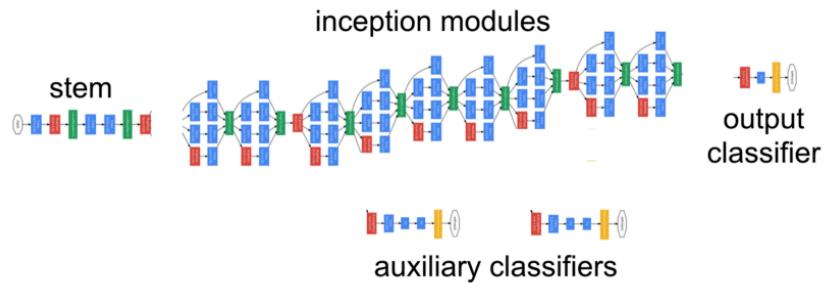
validation accuracy가 90%를 넘어가므로 좋은 성능을 보이고 있다.

### c-2) Inception

2014년 대회에서 2등을 했던 VGG-16를 이긴 모델은 Google에서 개발한 GoogLeNet 혹은 Inception이라 불리는 model이다. 처음 개발되었던 model은 inceptionv1 model로 VGG나 AlexNet에 비해 10배나 적은 parameter를 가졌다. 또한, Inception model에서 제안된 새로운 개념은 Inception Module이라는 것인데 다른 사이즈의 filter를 합성곱 층에서 동시에 다루는 것이다. 이 과정을 통해 parameter의 수를 급격하게 감소시킬 수 있었다.

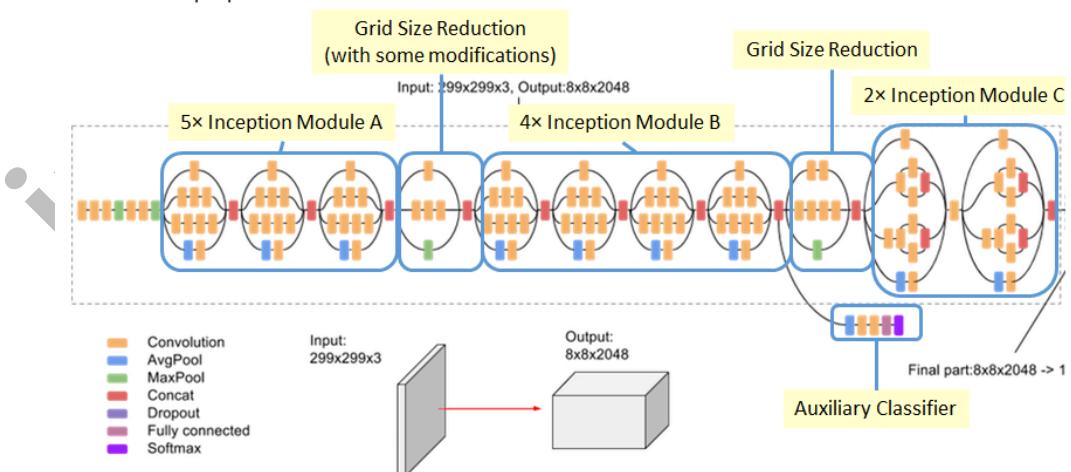


Inceptionv1의 layer는 22개이다. 아래의 그림은 Inceptionv1의 구조이다.



Inceptionv1 model을 기반으로 개발된 Inceptionv2는 좀더 복잡하며 accuracy를 향상시켰다. 배치 정규화가 추가되었으며 optimizer로 RMSProp를 쓴다는 것에 변화가 있었다.

- Introduction of Batch Normalisation
- More factorization
- RMSProp Optimiser



모델 사이의 성능을 정확하게 비교하기 위해 위의 VGG-16 을 이용한 모델의 data augmentation 과 scaling 과정을 동일하게 적용한다.

```
# Add our data-augmentation parameters to ImageDataGenerator
train_datagen = ImageDataGenerator(rescale = 1./255., rotation_range = 40,
                                   width_shift_range = 0.2, height_shift_range = 0.2,
                                   shear_range = 0.2, zoom_range = 0.2, horizontal_flip = True)

test_datagen = ImageDataGenerator( rescale = 1.0/255. )

train_generator = train_datagen.flow_from_directory(train_dir, batch_size = 20,
                                                    class_mode = 'binary',
                                                    target_size = (150, 150))
validation_generator = test_datagen.flow_from_directory(validation_dir, batch_size = 20,
                                                       class_mode = 'binary',
                                                       target_size = (150, 150))

Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
```

Inception model을 불러온 후 학습을 수행한다.

```
from tensorflow.keras.applications.inception_v3 import InceptionV3
base_model = InceptionV3(input_shape = (150, 150, 3), include_top = False, weights = 'imagenet')

for layer in base_model.layers:
    layer.trainable = False

from tensorflow.keras.optimizers import RMSprop

x = layers.Flatten()(base_model.output)
x = layers.Dense(1024, activation='relu')(x)
x = layers.Dropout(0.2)(x)

# Add a final sigmoid layer with 1 node for classification output
x = layers.Dense(1, activation='sigmoid')(x)

model = tf.keras.models.Model(base_model.input, x)

model.compile(optimizer = RMSprop(learning_rate=0.0001), loss = 'binary_crossentropy',
              metrics = ['acc'])

inc_history = model.fit_generator(train_generator, validation_data = validation_generator,
                                  steps_per_epoch = 100, epochs = 10)

Epoch 1/10
100/100 [=====] - 25s 203ms/step - loss: 1.3554 - acc: 0.8410 - val_loss: 0.4878 - val_acc: 0.8690
Epoch 2/10
100/100 [=====] - 18s 184ms/step - loss: 0.4877 - acc: 0.8865 - val_loss: 0.1347 - val_acc: 0.9560
Epoch 3/10
100/100 [=====] - 17s 166ms/step - loss: 0.3770 - acc: 0.9015 - val_loss: 0.1468 - val_acc: 0.9570
Epoch 4/10
100/100 [=====] - 17s 174ms/step - loss: 0.3609 - acc: 0.9035 - val_loss: 0.2703 - val_acc: 0.9330
Epoch 5/10
100/100 [=====] - 19s 186ms/step - loss: 0.3332 - acc: 0.9105 - val_loss: 0.2397 - val_acc: 0.9360
```

```
Epoch 6/10
100/100 [=====] - 18s 175ms/step - loss: 0.3180 - acc: 0.9115 - val_loss:
0.2168 - val_acc: 0.9430
Epoch 7/10
100/100 [=====] - 18s 178ms/step - loss: 0.2996 - acc: 0.9155 - val_loss:
0.3358 - val_acc: 0.9400
Epoch 8/10
100/100 [=====] - 19s 187ms/step - loss: 0.2750 - acc: 0.9195 - val_loss:
0.4196 - val_acc: 0.9180
Epoch 9/10
100/100 [=====] - 17s 173ms/step - loss: 0.3157 - acc: 0.9200 - val_loss:
0.1181 - val_acc: 0.9580
Epoch 10/10
100/100 [=====] - 17s 167ms/step - loss: 0.2634 - acc: 0.9235 - val_loss:
0.1544 - val_acc: 0.9610
```

VGG-16 model에 비해 validation accuracy가 더 높다.