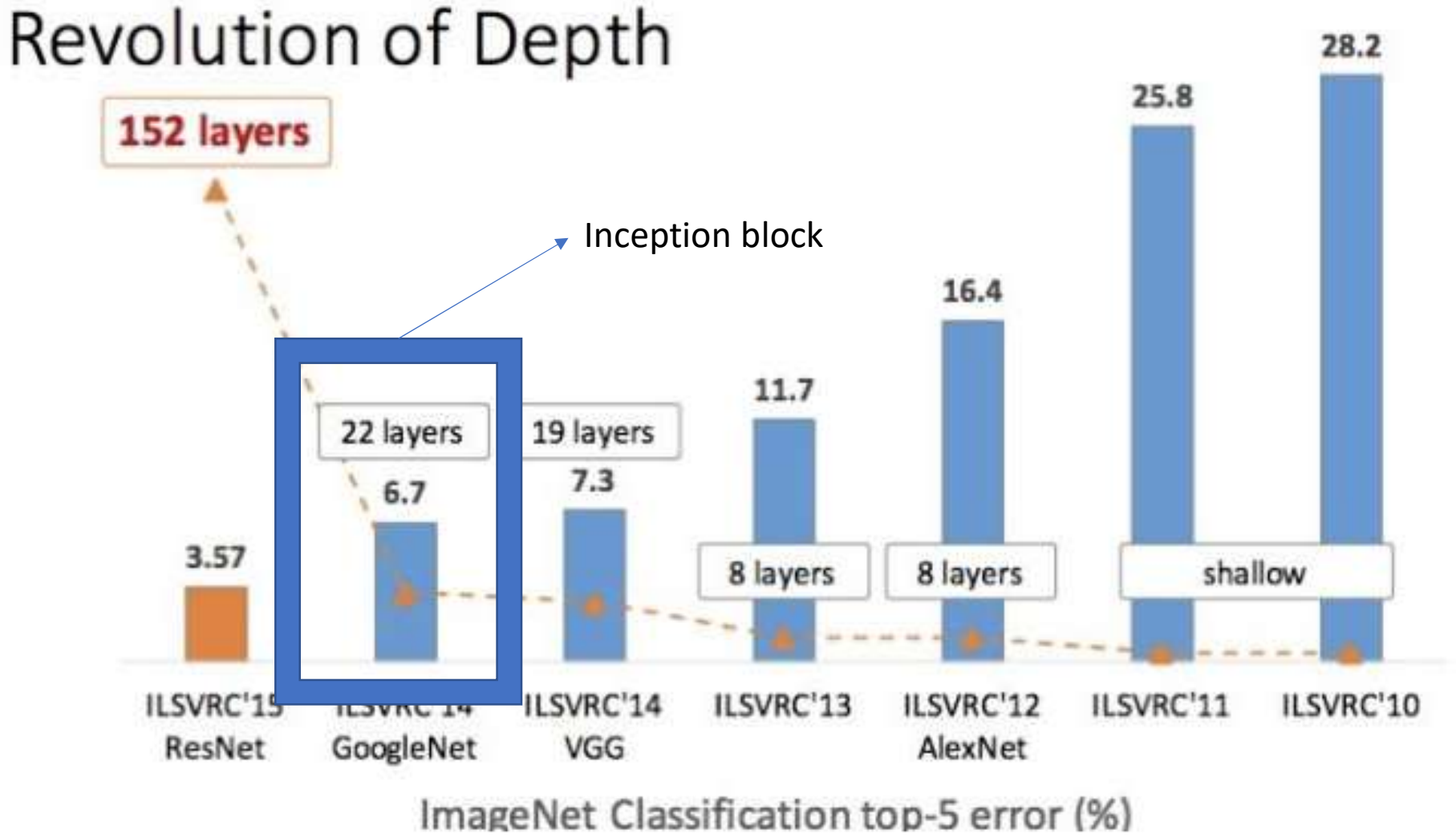
GoogLeNet



이미지넷이 주최하는 ILSVRC 대회

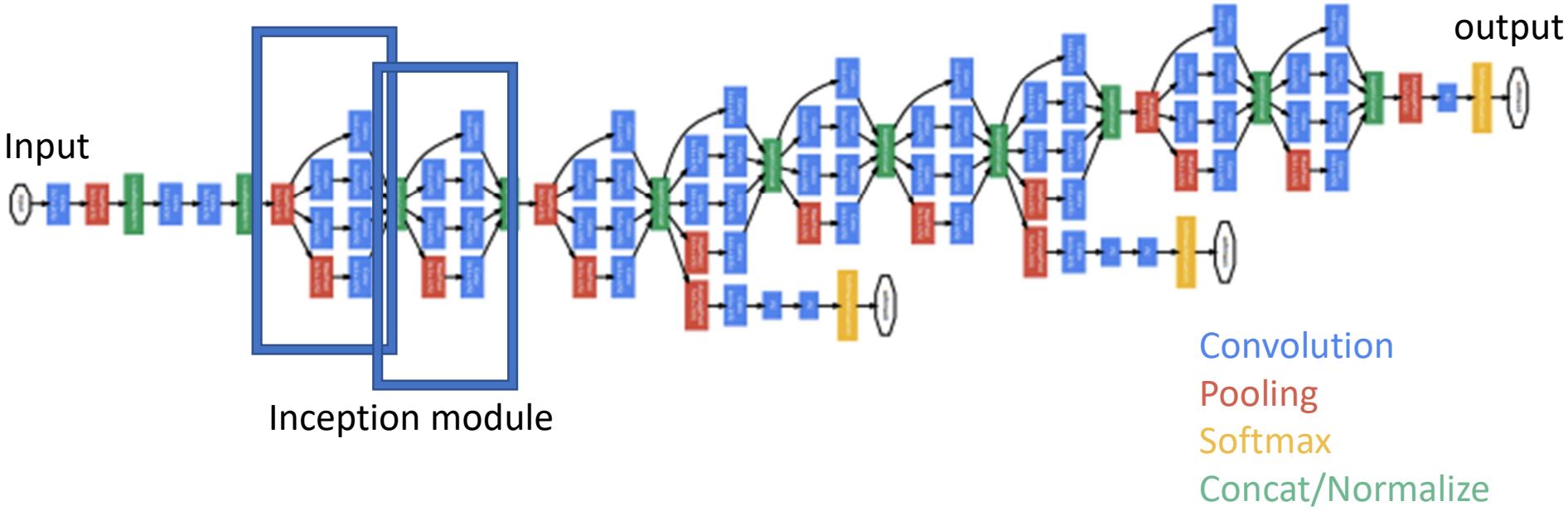
등장 배경

- Neural network의 문제점
  1. 깊어진다. -> 기하급수적으로 parameter가 증가한다. -> overfitting 하기 쉽다.
  2. 메모리 사용량이 증가한다. -> 계산 복잡도가 증가한다.

Google에서 Inception 개념을 도입하여 신경망을 더 deeper 하게 바꾸면서
연산량은 늘지않는 방법으로 네트워크 설계
즉, 효과적으로 차원을 줄이며 네트워크를 더 깊게 만드는 방향으로 진행!

GoogLeNet Architecture

GoogLeNet은 22개의 conv, 5개의 pooling 27개의 deep neural network이다.



GoogLeNet Architecture

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Table 1: GoogLeNet incarnation of the Inception architecture

GoogLeNetv

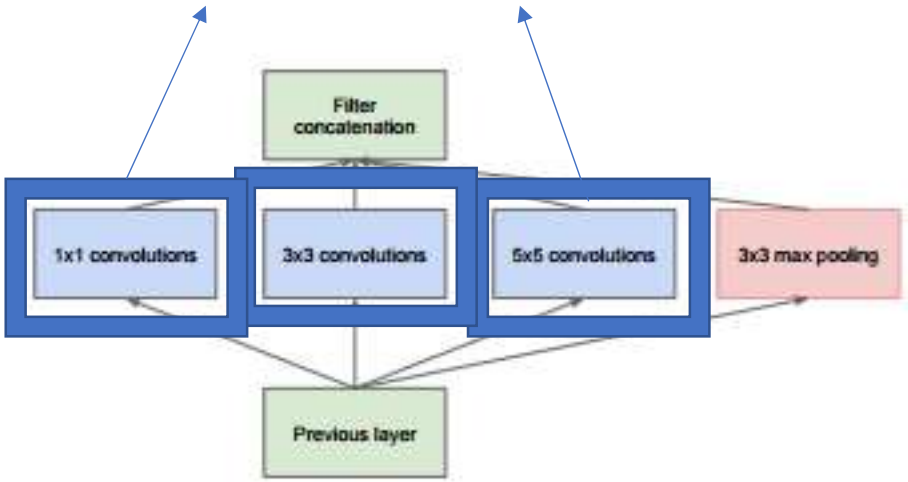
1. Pre-layer

이 부분은 일반 CNN과 비슷하다.
초기 학습에 필요한 구간인데 Inception module은 저층에서 학습 효율이 떨어지기 때문에 성능을 끌어올리기 위해 basic한 CNN layer를 두어 학습을 진행한다.

GoogLeNet

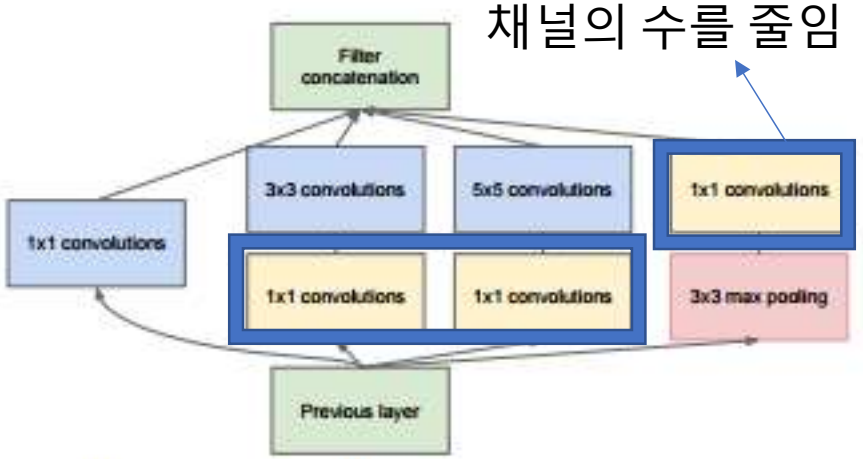
2. Inception Layer(Inception Module)

Feature을 효율적으로 추출하기 위해



(a) Inception module, naïve version

채널의 수를 줄임



(b) Inception module with dimension reductions

단점. 연산이 너무 많아진다.

re 2: Inception module

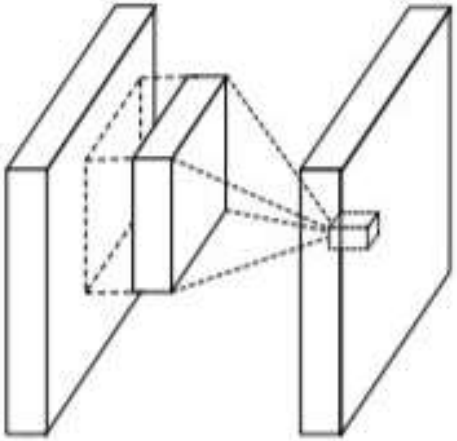
GoogLeNet

3. Global Average Pooling

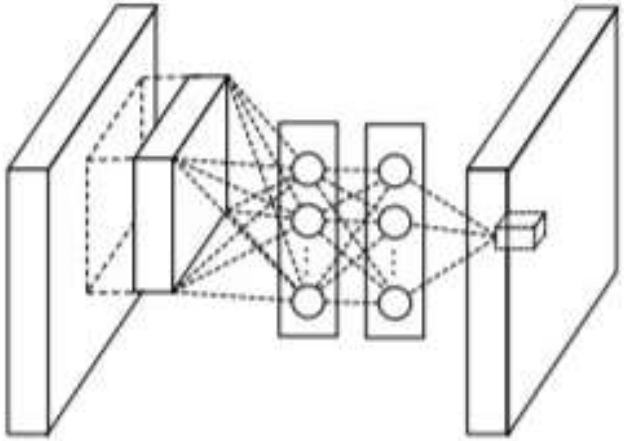
NIN(Neural In Neural)을 통해 좋은 feature을 선별했다고 판단했기 때문에
Fully Connected Layer 대신에 average pooling을 적용하였다.

Network in Network 논문

- CNN에서 데이터의 분포가 선형 관계로 표현될 수 없는 비선형관계일때
  - 관계를 표현할 수 있도록 단순한 곱/합 연산이 아닌 MLP를 추가한다.



(a) Linear convolution layer

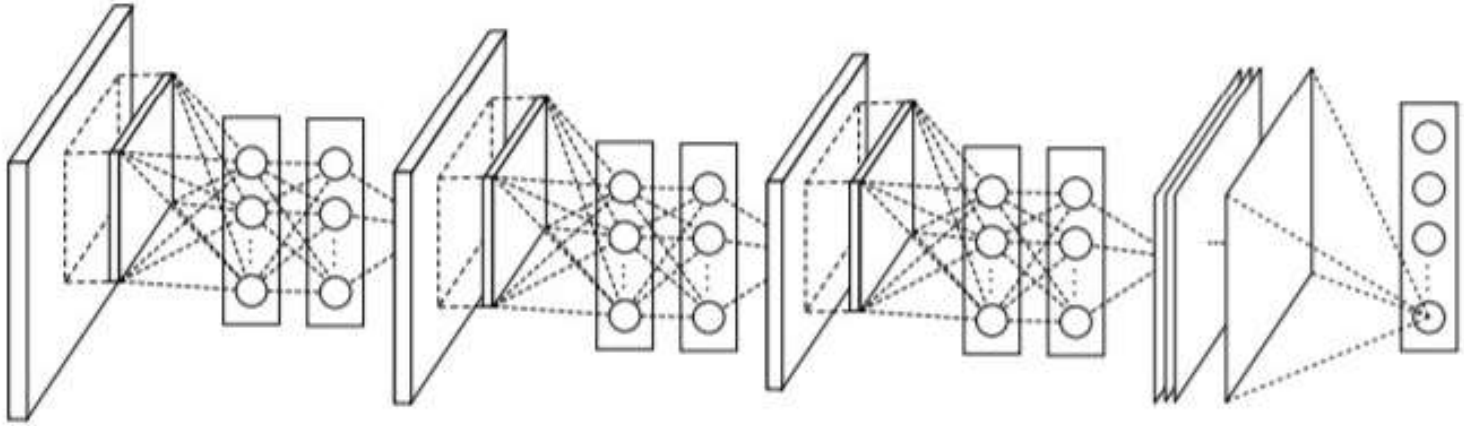


(b) Mlpconv layer

MLP 또한 Convolution filter처럼
역전파를 통해 학습된다.

Network in Network 논문

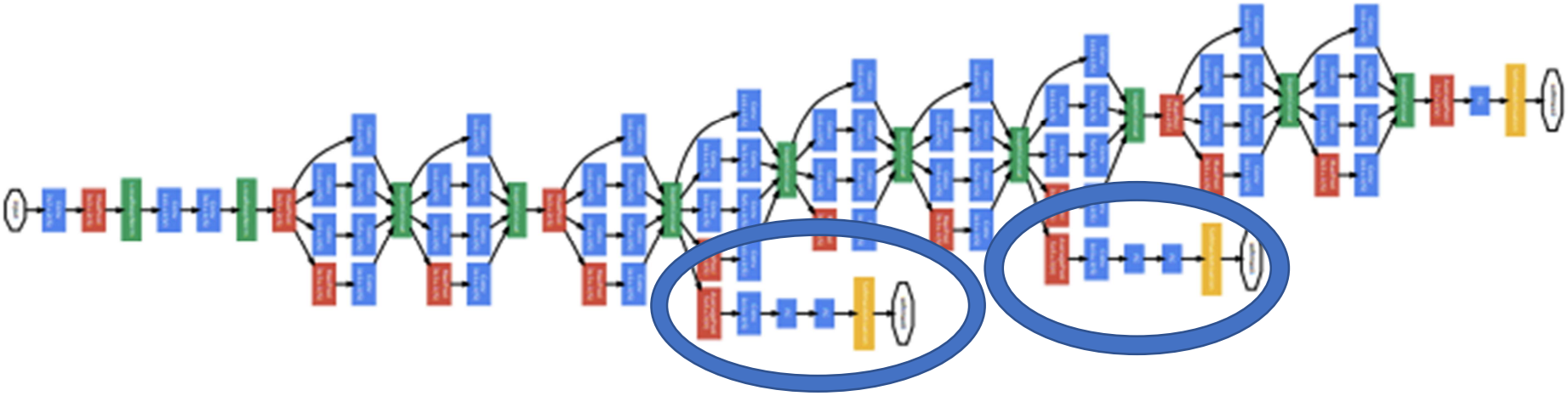
- 다음 사진은 실제 Network의 구조를 나타낸다.



GoogLeNet

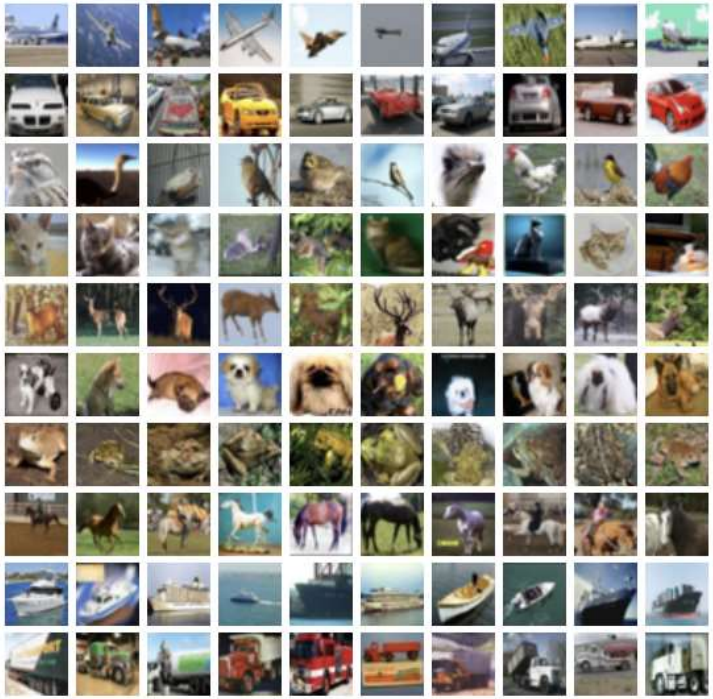
4. Auxiliary classifier

Vanishing Gradient 문제 때문에 덧붙였다.
또한 loss를 중간 중간에서 구하기 때문에 gradient가 적절하게 역전파된다.



CIFAR-10 데이터

비행기
자동차
새
고양이
사슴
개
개구리
말
배
트럭



32X32 pixel의 60000개 컬러 이미지

```
In [56]: print(x_train.shape)
         print(x_test.shape)

(50000, 32, 32, 3)
(10000, 32, 32, 3)

In [57]: print(y_train.shape)
         print(y_test.shape)

(50000, 10)
(10000, 10)
```

CIFAR-10 데이터 - CNN

In [19]:
```
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
from matplotlib import pyplot as plt
import numpy as np
from keras.utils import np_utils
from keras.layers.normalization import BatchNormalization
```

In [11]:
```
batch_size = 32
num_classes = 10
epochs = 20


(x_train, y_train), (x_test, y_test) = cifar10.load_data()
# One hot Encoding
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
```

In [44]:
```
plt.subplot(141)
plt.imshow(x_train[500], interpolation="bicubic")
plt.grid(False)
plt.subplot(142)
plt.imshow(x_train[4], interpolation="blackman")
plt.grid(False)
plt.subplot(143)
plt.imshow(x_train[8])#, interpolation="bicubic")
plt.grid(False)
plt.subplot(144)
plt.imshow(x_train[12], interpolation="bicubic")
plt.grid(False)
plt.show()
```

CIFAR-10 데이터 - CNN

```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same', input_shape=x_train.shape[1:]))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

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

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
batch_normalization (BatchNo (None, 32, 32, 32)        128
_________________________________________________________________
activation (Activation)      (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 30, 30, 32)        9248
_________________________________________________________________
batch_normalization_1 (Batch (None, 30, 30, 32)        128
_________________________________________________________________
activation_1 (Activation)    (None, 30, 30, 32)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 15, 15, 64)        18496
_________________________________________________________________
batch_normalization_2 (Batch (None, 15, 15, 64)        256
_________________________________________________________________
activation_2 (Activation)    (None, 15, 15, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 13, 13, 64)        36928
_________________________________________________________________
batch_normalization_3 (Batch (None, 13, 13, 64)        256
_________________________________________________________________
activation_3 (Activation)    (None, 13, 13, 64)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 6, 6, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 2304)              0
_________________________________________________________________
dense (Dense)                (None, 512)               1180160
_________________________________________________________________
batch_normalization_4 (Batch (None, 512)               2048
_________________________________________________________________
activation_4 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130
_________________________________________________________________
activation_5 (Activation)    (None, 10)                0
=================================================================
Total params: 1,253,674
Trainable params: 1,252,266
Non-trainable params: 1,408
```

CIFAR-10 데이터 - CNN

```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

hist = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=epochs,
                 batch_size=batch_size)
```

```
Epoch 1/20
1563/1563 [==============================] - 187s 120ms/step - loss: 1.2543 - accuracy: 0.5534 - va
l_loss: 1.0579 - val_accuracy: 0.6220
Epoch 2/20
1563/1563 [==============================] - 179s 115ms/step - loss: 0.9534 - accuracy: 0.6654 - va
l_loss: 0.7940 - val_accuracy: 0.7205
Epoch 3/20
1563/1563 [==============================] - 177s 113ms/step - loss: 0.8458 - accuracy: 0.7029 - va
l_loss: 0.8622 - val_accuracy: 0.6983
Epoch 4/20
1563/1563 [==============================] - 187s 120ms/step - loss: 0.7719 - accuracy: 0.7282 - va
l_loss: 0.7720 - val_accuracy: 0.7331
.....
Epoch 19/20
1563/1563 [==============================] - 174s 112ms/step - loss: 0.3668 - accuracy: 0.8705 - va
l_loss: 0.5239 - val_accuracy: 0.8258
Epoch 20/20
1563/1563 [==============================] - 174s 111ms/step - loss: 0.3619 - accuracy: 0.8726 - va
l_loss: 0.5701 - val_accuracy: 0.8195
```

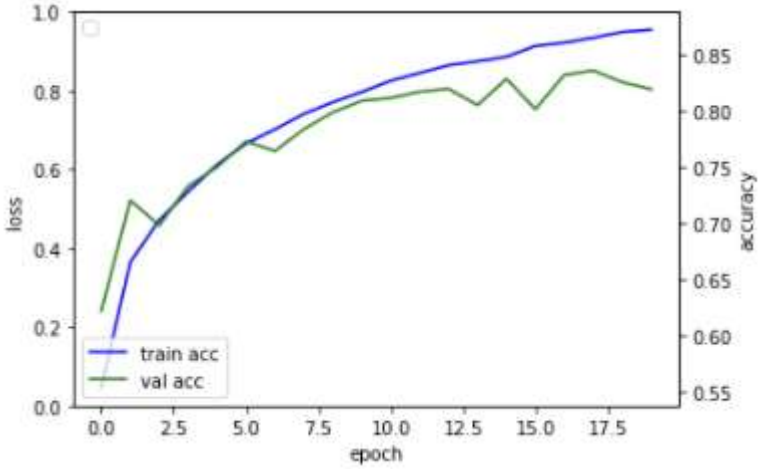
CIFAR-10 데이터 - CNN

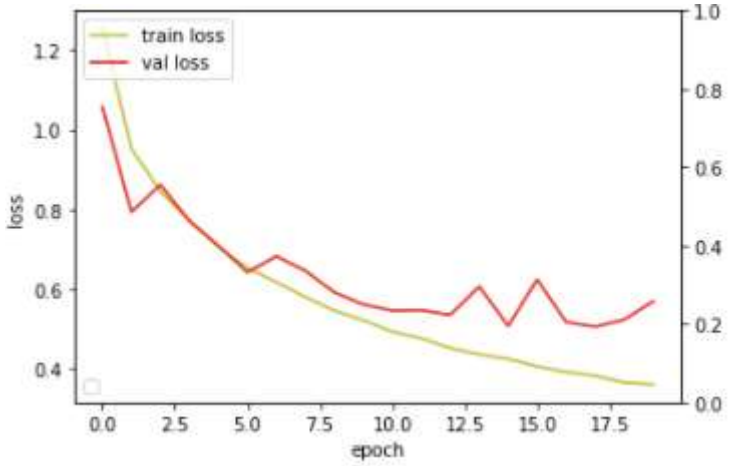
```
scores = model.evaluate(x_test, y_test, verbose=0)
print("CNN Error: %.2f%%" % (100-scores[1]*100))
```

```
CNN Error: 18.05%
```

```
scores[0]
```

```
0.5701218247413635
```





Reference

[1] GoogLeNet 논문 요약 및 정리 : https://m.blog.naver.com/PostView.nhn?blogId=soso030&logNo=221630029202&proxyReferer=https:%2F%2Fwww.google.com%2F
[2] CNN의 발자취..(주요 모델들 정리)-(1) GoogLeNet(Inception) : https://m.blog.naver.com/PostView.nhn?blogId=kmkim1222&logNo=221524681282&proxyReferer=undefined
[3] Inception(GoogLeNet) 리뷰 : https://kangbk0120.github.io/articles/2018-01/inception-googlenet-review
[4] [Paper Review] GoogLeNet : https://www.youtube.com/watch?v=W9MlakX3vko
[5]GoogLeNet 구현 코드: https://github.com/dingchenwei/googLeNet