3 Additional Functions with Keras

- 3.1 모델 합치기 with CIFAR10
- 3.2 데이터 증강
- 3.3 Finetuning

3.1 모델 합치기 with CIFAR10

convolution과 maxpooling layer로 구성된 feature extractor 모델과

fully connected layer로 구성된 ANN classifier 모델을 따로 정의하고

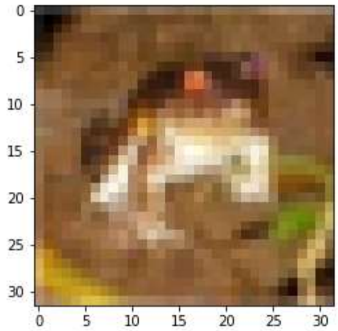
두 모델을 합쳐서 CNN 모델을 만듬

In [1]:
```
import os
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

(1) 데이터셋 : CIFAR 10

In [2]:
```
import tensorflow.keras.utils as utils
from tensorflow.keras import datasets

# Dataset Load
(X_train, Y_train), (X_test, Y_test) = datasets.cifar10.load_data()

# Dataset Confirm
print(X_train.shape, Y_train.shape)
print('label : ',Y_train[0])
plt.imshow(X_train[0])

# Dataset Preprocessing
X_train = X_train/255.0
X_test = X_test/255.0
Y_train = utils.to_categorical(Y_train)
Y_test = utils.to_categorical(Y_test)

print(X_train.shape, Y_train.shape)
```

```
(50000, 32, 32, 3) (50000, 1)
label :  [6]
(50000, 32, 32, 3) (50000, 10)
```



(2) 모델링

In [3]:
```
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Input, Dense, Activation
from tensorflow.keras.layers import Flatten, BatchNormalization, Dropout, ReLU
from tensorflow.keras.layers import Conv2D, MaxPooling2D
```

In [4]:
```
n_in = X_train.shape[1:]
n_out = Y_train.shape[-1]

def conv_maxpool_layers(n_in):
    model = Sequential()
    model.add(Conv2D(16, kernel_size=(3, 3), padding='same', activation='relu', input_shape=(n_in)))
    model.add(Conv2D(32, kernel_size=(3, 3), padding='same', strides=(2, 2), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model

def fc_layers(n_out):
    model = Sequential()
    model.add(Dense(units =128, input_shape=(2048,), activation='relu'))
    model.add(Dense(units =n_out, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model

def CNN_sum(n_in, n_out):

    # Coding Time
```

```
    #각 부분 모델
    feature_extractor=conv_maxpool_layers(n_in)
    feature_extractor.trainable=True # .trainable : 해당 layer에 대한 학습 여부 결정
    ann_classifier = fc_layers(n_out)
    ann_classifier.trainable=True

    #두 모델을 합쳐 새로운 모델 정의(Functional Style)
    x = Input(shape=n_in)
    feature = feature_extractor(x)
    y = ann_classifier(feature)
    model = Model(inputs = x, outputs = y)

    '''
    Sequential Style
    model = Sequential()
    model.add(feature_extractor)
    model.add(ann_classifier)
    '''
    return model
```

In [5]:

```
model = CNN_sum(n_in, n_out)
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 32, 32, 3)]       0

 sequential (Sequential)     (None, 2048)              5088

 sequential_1 (Sequential)   (None, 10)                263562

=================================================================
Total params: 268,650
Trainable params: 268,650
Non-trainable params: 0
_________________________________________________________________
```

(3-4) 모델의 학습과정 설정 / 모델 학습

In [6]:

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

from tensorflow.keras.callbacks import EarlyStopping
earlystopper = EarlyStopping(monitor='val_accuracy', patience=7, verbose=1, mode='auto', restore_best_weights=True)
history = model.fit(X_train, Y_train, batch_size=128, epochs=50, validation_split=0.2, callbacks = [earlystopper])
```

```
Epoch 1/50
313/313 [==============================] - 4s 4ms/step - loss: 1.5609 - accuracy: 0.4430 - val_loss: 1.3450 - val_accuracy: 0.5314
Epoch 2/50
313/313 [==============================] - 1s 4ms/step - loss: 1.2049 - accuracy: 0.5751 - val_loss: 1.1523 - val_accuracy: 0.6040
Epoch 3/50
313/313 [==============================] - 1s 4ms/step - loss: 1.0810 - accuracy: 0.6224 - val_loss: 1.0996 - val_accuracy: 0.6216
Epoch 4/50
313/313 [==============================] - 1s 4ms/step - loss: 0.9961 - accuracy: 0.6512 - val_loss: 1.0757 - val_accuracy: 0.6246
Epoch 5/50
313/313 [==============================] - 1s 4ms/step - loss: 0.9385 - accuracy: 0.6727 - val_loss: 1.0490 - val_accuracy: 0.6339
Epoch 6/50
313/313 [==============================] - 1s 4ms/step - loss: 0.8660 - accuracy: 0.6992 - val_loss: 1.0011 - val_accuracy: 0.6522
Epoch 7/50
313/313 [==============================] - 1s 4ms/step - loss: 0.8172 - accuracy: 0.7172 - val_loss: 0.9900 - val_accuracy: 0.6561
Epoch 8/50
313/313 [==============================] - 1s 3ms/step - loss: 0.7731 - accuracy: 0.7303 - val_loss: 0.9783 - val_accuracy: 0.6623
Epoch 9/50
313/313 [==============================] - 1s 4ms/step - loss: 0.7196 - accuracy: 0.7498 - val_loss: 0.9847 - val_accuracy: 0.6703
Epoch 10/50
313/313 [==============================] - 1s 4ms/step - loss: 0.6631 - accuracy: 0.7674 - val_loss: 0.9835 - val_accuracy: 0.6721
Epoch 11/50
313/313 [==============================] - 1s 4ms/step - loss: 0.6208 - accuracy: 0.7843 - val_loss: 0.9745 - val_accuracy: 0.6751
Epoch 12/50
313/313 [==============================] - 1s 4ms/step - loss: 0.5700 - accuracy: 0.8031 - val_loss: 1.0233 - val_accuracy: 0.6654
Epoch 13/50
313/313 [==============================] - 1s 4ms/step - loss: 0.5223 - accuracy: 0.8210 - val_loss: 1.0782 - val_accuracy: 0.6639
Epoch 14/50
313/313 [==============================] - 1s 4ms/step - loss: 0.4674 - accuracy: 0.8405 - val_loss: 1.0746 - val_accuracy: 0.6720
Epoch 15/50
313/313 [==============================] - 1s 4ms/step - loss: 0.4185 - accuracy: 0.8584 - val_loss: 1.1313 - val_accuracy: 0.6713
Epoch 16/50
313/313 [==============================] - 1s 4ms/step - loss: 0.3786 - accuracy: 0.8694 - val_loss: 1.1914 - val_accuracy: 0.6688
Epoch 17/50
313/313 [==============================] - 1s 4ms/step - loss: 0.3326 - accuracy: 0.8884 - val_loss: 1.2327 - val_accuracy: 0.6686
Epoch 18/50
312/313 [============================>.] - ETA: 0s - loss: 0.2815 - accuracy: 0.9084Restoring model weights from the end of the best
epoch: 11.
313/313 [==============================] - 1s 4ms/step - loss: 0.2816 - accuracy: 0.9083 - val_loss: 1.3489 - val_accuracy: 0.6560
Epoch 18: early stopping
```

(5) 모델 평가

In [7]:

```
# Coding Time
loss_and_accuracy = model.evaluate(X_test, Y_test, batch_size=128)
print('loss : %.4f, accruracy : %.4f'%(loss_and_accuracy[0],loss_and_accuracy[1]))
```

```
79/79 [==============================] - 0s 2ms/step - loss: 0.9910 - accuracy: 0.6665
loss : 0.9910, accruracy : 0.6665
```

3.2 Image data augmentation

케라스에서는 ImageDataGenerate 함수로 Data augmentation 기능을 제공

https://keras.io/preprocessing/image/#imagedatagenerator-class

(1)-2 데이터 증강 적용

In [8]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator, img_to_array
```

In [9]:
```
datagen = ImageDataGenerator(
    featurewise_center = False,
    samplewise_center = False,
    featurewise_std_normalization = False,
    samplewise_std_normalization = False,
    zca_whitening = False,
    rotation_range = 2, # 회전
    zoom_range = 0.1, # 확대 축소
    width_shift_range = 0.1, # 수평 이동
    height_shift_range = 0.1, # 수직 이동
    horizontal_flip = True, # 수평 반전|
    vertical_flip = False # 수직 반전
)

datagen.fit(X_train)
```

(3-4) 모델의 학습과정 설정 / 모델 학습

In [10]:
```
model = CNN_sum(n_in, n_out)
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
earlystopper = EarlyStopping(monitor='val_accuracy', patience=7, verbose=1, mode='auto', restore_best_weights=True)
# Coding Time
model.fit(datagen.flow(X_train[:-10000], Y_train[:-10000], batch_size = 128),
          epochs = 50, validation_data = (X_train[-10000:], Y_train[-10000:]), verbose = 1, callbacks = [earlystopper])
```

```
Epoch 1/50
313/313 [==============================] - 9s 27ms/step - loss: 1.7201 - accuracy: 0.3855 - val_loss: 1.4450 - val_accuracy: 0.4806
Epoch 2/50
313/313 [==============================] - 8s 27ms/step - loss: 1.4038 - accuracy: 0.5003 - val_loss: 1.2836 - val_accuracy: 0.5405
Epoch 3/50
313/313 [==============================] - 9s 27ms/step - loss: 1.3025 - accuracy: 0.5375 - val_loss: 1.2124 - val_accuracy: 0.5726
Epoch 4/50
313/313 [==============================] - 8s 27ms/step - loss: 1.2336 - accuracy: 0.5640 - val_loss: 1.1382 - val_accuracy: 0.6037
Epoch 5/50
313/313 [==============================] - 9s 27ms/step - loss: 1.1773 - accuracy: 0.5831 - val_loss: 1.0655 - val_accuracy: 0.6244
Epoch 6/50
313/313 [==============================] - 9s 27ms/step - loss: 1.1362 - accuracy: 0.5994 - val_loss: 1.0553 - val_accuracy: 0.6320
Epoch 7/50
313/313 [==============================] - 9s 28ms/step - loss: 1.1107 - accuracy: 0.6091 - val_loss: 1.0476 - val_accuracy: 0.6417
Epoch 8/50
313/313 [==============================] - 8s 27ms/step - loss: 1.0797 - accuracy: 0.6203 - val_loss: 1.0650 - val_accuracy: 0.6291
Epoch 9/50
313/313 [==============================] - 8s 27ms/step - loss: 1.0552 - accuracy: 0.6270 - val_loss: 1.0304 - val_accuracy: 0.6445
Epoch 10/50
313/313 [==============================] - 8s 27ms/step - loss: 1.0293 - accuracy: 0.6378 - val_loss: 0.9912 - val_accuracy: 0.6583
Epoch 11/50
313/313 [==============================] - 8s 27ms/step - loss: 1.0094 - accuracy: 0.6451 - val_loss: 1.0018 - val_accuracy: 0.6501
Epoch 12/50
313/313 [==============================] - 8s 27ms/step - loss: 0.9970 - accuracy: 0.6481 - val_loss: 0.9554 - val_accuracy: 0.6683
Epoch 13/50
313/313 [==============================] - 9s 27ms/step - loss: 0.9795 - accuracy: 0.6548 - val_loss: 0.9602 - val_accuracy: 0.6729
Epoch 14/50
313/313 [==============================] - 9s 27ms/step - loss: 0.9599 - accuracy: 0.6624 - val_loss: 0.9548 - val_accuracy: 0.6688
Epoch 15/50
313/313 [==============================] - 8s 27ms/step - loss: 0.9530 - accuracy: 0.6649 - val_loss: 0.9917 - val_accuracy: 0.6585
Epoch 16/50
313/313 [==============================] - 8s 27ms/step - loss: 0.9372 - accuracy: 0.6699 - val_loss: 0.9157 - val_accuracy: 0.6823
Epoch 17/50
313/313 [==============================] - 8s 26ms/step - loss: 0.9275 - accuracy: 0.6754 - val_loss: 0.9315 - val_accuracy: 0.6803
Epoch 18/50
313/313 [==============================] - 8s 27ms/step - loss: 0.9204 - accuracy: 0.6757 - val_loss: 0.9133 - val_accuracy: 0.6854
Epoch 19/50
313/313 [==============================] - 8s 27ms/step - loss: 0.9067 - accuracy: 0.6794 - val_loss: 0.9492 - val_accuracy: 0.6719
Epoch 20/50
313/313 [==============================] - 9s 27ms/step - loss: 0.9016 - accuracy: 0.6841 - val_loss: 0.8951 - val_accuracy: 0.6966
Epoch 21/50
313/313 [==============================] - 9s 27ms/step - loss: 0.8923 - accuracy: 0.6877 - val_loss: 0.9448 - val_accuracy: 0.6743
Epoch 22/50
313/313 [==============================] - 9s 27ms/step - loss: 0.8870 - accuracy: 0.6887 - val_loss: 0.8897 - val_accuracy: 0.6976
Epoch 23/50
313/313 [==============================] - 8s 26ms/step - loss: 0.8801 - accuracy: 0.6877 - val_loss: 0.9220 - val_accuracy: 0.6860
Epoch 24/50
313/313 [==============================] - 8s 26ms/step - loss: 0.8621 - accuracy: 0.6957 - val_loss: 0.8910 - val_accuracy: 0.6998
Epoch 25/50
313/313 [==============================] - 8s 27ms/step - loss: 0.8580 - accuracy: 0.6992 - val_loss: 0.8934 - val_accuracy: 0.6963
Epoch 26/50
313/313 [==============================] - 8s 27ms/step - loss: 0.8546 - accuracy: 0.7019 - val_loss: 0.8339 - val_accuracy: 0.7161
Epoch 27/50
313/313 [==============================] - 8s 26ms/step - loss: 0.8457 - accuracy: 0.7009 - val_loss: 0.8769 - val_accuracy: 0.7042
Epoch 28/50
313/313 [==============================] - 8s 27ms/step - loss: 0.8395 - accuracy: 0.7062 - val_loss: 0.9057 - val_accuracy: 0.6936
Epoch 29/50
313/313 [==============================] - 8s 27ms/step - loss: 0.8376 - accuracy: 0.7079 - val_loss: 0.8768 - val_accuracy: 0.7009
Epoch 30/50
313/313 [==============================] - 8s 27ms/step - loss: 0.8263 - accuracy: 0.7082 - val_loss: 0.8935 - val_accuracy: 0.6995
Epoch 31/50
313/313 [==============================] - 8s 27ms/step - loss: 0.8217 - accuracy: 0.7114 - val_loss: 0.8584 - val_accuracy: 0.7074
Epoch 32/50
313/313 [==============================] - 8s 27ms/step - loss: 0.8194 - accuracy: 0.7085 - val_loss: 0.9068 - val_accuracy: 0.6975
Epoch 33/50
312/313 [============================>.] - ETA: 0s - loss: 0.8116 - accuracy: 0.7123Restoring model weights from the end of the best
epoch: 26.
313/313 [==============================] - 9s 27ms/step - loss: 0.8114 - accuracy: 0.7123 - val_loss: 0.9245 - val_accuracy: 0.6822
Epoch 33: early stopping
```

Out[10]:
```
<keras.callbacks.History at 0x272ce92d850>
```

In [11]:
```python
loss_and_accuracy = model.evaluate(X_test, Y_test, batch_size=128)
print('loss : %.4f, accruracy : %.4f'%(loss_and_accuracy[0],loss_and_accuracy[1]))
```

```
79/79 [==============================] - 0s 3ms/step - loss: 0.8390 - accuracy: 0.7086
loss : 0.8390, accruracy : 0.7086
```

3.3 Transfer learning

Transfer learning을 통해 현재 쓰이고 있는 네트워크를 가져와 학습하는 방법을 배워본다(Classifier만 / Entire)

(2) 모델링1 : Classifier learning

사용가능 네트워크 : https://keras.io/api/applications/

In [12]:
```python
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
```

In [13]:
```python
# Coding Time
base_model = VGG16(weights='imagenet', input_shape=(32,32,3), include_top=False)
base_model.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_3 (InputLayer)        [(None, 32, 32, 3)]       0         
                                                                 
 block1_conv1 (Conv2D)       (None, 32, 32, 64)        1792      
                                                                 
 block1_conv2 (Conv2D)       (None, 32, 32, 64)        36928     
                                                                 
 block1_pool (MaxPooling2D)  (None, 16, 16, 64)        0         
                                                                 
 block2_conv1 (Conv2D)       (None, 16, 16, 128)       73856     
                                                                 
 block2_conv2 (Conv2D)       (None, 16, 16, 128)       147584    
                                                                 
 block2_pool (MaxPooling2D)  (None, 8, 8, 128)         0         
                                                                 
 block3_conv1 (Conv2D)       (None, 8, 8, 256)         295168    
                                                                 
 block3_conv2 (Conv2D)       (None, 8, 8, 256)         590080    
                                                                 
 block3_conv3 (Conv2D)       (None, 8, 8, 256)         590080    
                                                                 
 block3_pool (MaxPooling2D)  (None, 4, 4, 256)         0         
                                                                 
 block4_conv1 (Conv2D)       (None, 4, 4, 512)         1180160   
                                                                 
 block4_conv2 (Conv2D)       (None, 4, 4, 512)         2359808   
                                                                 
 block4_conv3 (Conv2D)       (None, 4, 4, 512)         2359808   
                                                                 
 block4_pool (MaxPooling2D)  (None, 2, 2, 512)         0         
                                                                 
 block5_conv1 (Conv2D)       (None, 2, 2, 512)         2359808   
                                                                 
 block5_conv2 (Conv2D)       (None, 2, 2, 512)         2359808   
                                                                 
 block5_conv3 (Conv2D)       (None, 2, 2, 512)         2359808   
                                                                 
 block5_pool (MaxPooling2D)  (None, 1, 1, 512)         0         
                                                                 
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

In [14]:

```
# Get Feature Extroactor from VGG16
x = base_model.output

# Add Classifier
x = Flatten()(x)
x = Dropout(0.5)(x)
x = Dense(256, activation='relu')(x)
x = BatchNormalization()(x)
predictions = Dense(Y_train.shape[1], activation='softmax')(x) #Y_train.shape[1] :10

model = Model(inputs=base_model.input, outputs=predictions)
```

In [15]:

```
# first: train only the top layers (which were randomly initialized)
for layer in base_model.layers:
    layer.trainable = False
```

In [16]:

```
model.summary()
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 32, 32, 3)]       0

 block1_conv1 (Conv2D)       (None, 32, 32, 64)        1792

 block1_conv2 (Conv2D)       (None, 32, 32, 64)        36928

 block1_pool (MaxPooling2D)  (None, 16, 16, 64)        0

 block2_conv1 (Conv2D)       (None, 16, 16, 128)       73856

 block2_conv2 (Conv2D)       (None, 16, 16, 128)       147584

 block2_pool (MaxPooling2D)  (None, 8, 8, 128)         0

 block3_conv1 (Conv2D)       (None, 8, 8, 256)         295168

 block3_conv2 (Conv2D)       (None, 8, 8, 256)         590080

 block3_conv3 (Conv2D)       (None, 8, 8, 256)         590080

 block3_pool (MaxPooling2D)  (None, 4, 4, 256)         0

 block4_conv1 (Conv2D)       (None, 4, 4, 512)         1180160

 block4_conv2 (Conv2D)       (None, 4, 4, 512)         2359808

 block4_conv3 (Conv2D)       (None, 4, 4, 512)         2359808

 block4_pool (MaxPooling2D)  (None, 2, 2, 512)         0

 block5_conv1 (Conv2D)       (None, 2, 2, 512)         2359808

 block5_conv2 (Conv2D)       (None, 2, 2, 512)         2359808

 block5_conv3 (Conv2D)       (None, 2, 2, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 1, 1, 512)         0

 flatten_2 (Flatten)         (None, 512)               0

 dropout (Dropout)           (None, 512)               0

 dense_4 (Dense)             (None, 256)               131328

 batch_normalization (BatchN  (None, 256)              1024
 ormalization)

 dense_5 (Dense)             (None, 10)                2570

=================================================================
Total params: 14,849,610
Trainable params: 134,410
Non-trainable params: 14,715,200
_________________________________________________________________
```

(3) 모델의 학습과정 설정

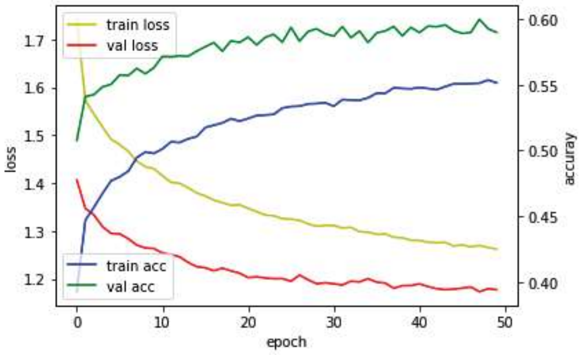
In [17]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

(4) 모델 학습시키기

In [18]:
```
earlystopper = EarlyStopping(monitor='val_accuracy', patience=7, verbose=1, mode='auto', restore_best_weights=True)
history = model.fit(X_train, Y_train, batch_size=128, epochs=50, validation_split=0.2, callbacks = [earlystopper])
```

```
Epoch 1/50
313/313 [==============================] - 4s 11ms/step - loss: 1.7419 - accuracy: 0.3924 - val_loss: 1.4068 - val_accuracy: 0.5077
Epoch 2/50
313/313 [==============================] - 3s 9ms/step - loss: 1.5729 - accuracy: 0.4469 - val_loss: 1.3468 - val_accuracy: 0.5410
Epoch 3/50
313/313 [==============================] - 3s 10ms/step - loss: 1.5437 - accuracy: 0.4568 - val_loss: 1.3333 - val_accuracy: 0.5425
Epoch 4/50
313/313 [==============================] - 3s 10ms/step - loss: 1.5172 - accuracy: 0.4672 - val_loss: 1.3090 - val_accuracy: 0.5484
Epoch 5/50
313/313 [==============================] - 3s 10ms/step - loss: 1.4913 - accuracy: 0.4769 - val_loss: 1.2946 - val_accuracy: 0.5503
Epoch 6/50
313/313 [==============================] - 3s 9ms/step - loss: 1.4804 - accuracy: 0.4798 - val_loss: 1.2938 - val_accuracy: 0.5575
Epoch 7/50
313/313 [==============================] - 3s 9ms/step - loss: 1.4661 - accuracy: 0.4842 - val_loss: 1.2837 - val_accuracy: 0.5570
Epoch 8/50
313/313 [==============================] - 3s 9ms/step - loss: 1.4457 - accuracy: 0.4945 - val_loss: 1.2705 - val_accuracy: 0.5624
Epoch 9/50
313/313 [==============================] - 3s 9ms/step - loss: 1.4341 - accuracy: 0.4989 - val_loss: 1.2645 - val_accuracy: 0.5584
Epoch 10/50
313/313 [==============================] - 3s 10ms/step - loss: 1.4299 - accuracy: 0.4978 - val_loss: 1.2633 - val_accuracy: 0.5630
Epoch 11/50
313/313 [==============================] - 3s 10ms/step - loss: 1.4153 - accuracy: 0.5012 - val_loss: 1.2537 - val_accuracy: 0.5716
Epoch 12/50
313/313 [==============================] - 3s 10ms/step - loss: 1.4016 - accuracy: 0.5066 - val_loss: 1.2508 - val_accuracy: 0.5714
Epoch 13/50
313/313 [==============================] - 3s 10ms/step - loss: 1.3992 - accuracy: 0.5060 - val_loss: 1.2459 - val_accuracy: 0.5720
Epoch 14/50
313/313 [==============================] - 3s 10ms/step - loss: 1.3901 - accuracy: 0.5088 - val_loss: 1.2342 - val_accuracy: 0.5717
Epoch 15/50
313/313 [==============================] - 3s 10ms/step - loss: 1.3792 - accuracy: 0.5106 - val_loss: 1.2252 - val_accuracy: 0.5756
Epoch 16/50
313/313 [==============================] - 3s 9ms/step - loss: 1.3729 - accuracy: 0.5174 - val_loss: 1.2232 - val_accuracy: 0.5790
Epoch 17/50
313/313 [==============================] - 3s 9ms/step - loss: 1.3644 - accuracy: 0.5192 - val_loss: 1.2169 - val_accuracy: 0.5821
Epoch 18/50
313/313 [==============================] - 3s 9ms/step - loss: 1.3590 - accuracy: 0.5210 - val_loss: 1.2223 - val_accuracy: 0.5756
Epoch 19/50
313/313 [==============================] - 3s 9ms/step - loss: 1.3534 - accuracy: 0.5242 - val_loss: 1.2166 - val_accuracy: 0.5835
Epoch 20/50
313/313 [==============================] - 3s 9ms/step - loss: 1.3547 - accuracy: 0.5223 - val_loss: 1.2117 - val_accuracy: 0.5822
Epoch 21/50
313/313 [==============================] - 3s 10ms/step - loss: 1.3471 - accuracy: 0.5243 - val_loss: 1.2027 - val_accuracy: 0.5863
Epoch 22/50
313/313 [==============================] - 3s 10ms/step - loss: 1.3405 - accuracy: 0.5266 - val_loss: 1.2040 - val_accuracy: 0.5803
Epoch 23/50
313/313 [==============================] - 3s 9ms/step - loss: 1.3332 - accuracy: 0.5270 - val_loss: 1.2017 - val_accuracy: 0.5862
Epoch 24/50
313/313 [==============================] - 3s 10ms/step - loss: 1.3316 - accuracy: 0.5275 - val_loss: 1.2002 - val_accuracy: 0.5884
Epoch 25/50
313/313 [==============================] - 3s 10ms/step - loss: 1.3256 - accuracy: 0.5322 - val_loss: 1.2004 - val_accuracy: 0.5824
Epoch 26/50
313/313 [==============================] - 3s 10ms/step - loss: 1.3247 - accuracy: 0.5335 - val_loss: 1.1949 - val_accuracy: 0.5937
Epoch 27/50
313/313 [==============================] - 3s 9ms/step - loss: 1.3222 - accuracy: 0.5337 - val_loss: 1.2077 - val_accuracy: 0.5832
Epoch 28/50
313/313 [==============================] - 3s 9ms/step - loss: 1.3145 - accuracy: 0.5354 - val_loss: 1.1980 - val_accuracy: 0.5906
Epoch 29/50
313/313 [==============================] - 3s 9ms/step - loss: 1.3096 - accuracy: 0.5357 - val_loss: 1.1896 - val_accuracy: 0.5926
Epoch 30/50
313/313 [==============================] - 3s 9ms/step - loss: 1.3114 - accuracy: 0.5364 - val_loss: 1.1917 - val_accuracy: 0.5888
Epoch 31/50
313/313 [==============================] - 3s 9ms/step - loss: 1.3111 - accuracy: 0.5337 - val_loss: 1.1897 - val_accuracy: 0.5870
Epoch 32/50
313/313 [==============================] - 3s 10ms/step - loss: 1.3061 - accuracy: 0.5388 - val_loss: 1.1870 - val_accuracy: 0.5943
Epoch 33/50
313/313 [==============================] - 3s 9ms/step - loss: 1.3075 - accuracy: 0.5384 - val_loss: 1.1951 - val_accuracy: 0.5859
Epoch 34/50
313/313 [==============================] - 3s 9ms/step - loss: 1.2983 - accuracy: 0.5382 - val_loss: 1.1935 - val_accuracy: 0.5909
Epoch 35/50
313/313 [==============================] - 3s 9ms/step - loss: 1.2974 - accuracy: 0.5400 - val_loss: 1.2004 - val_accuracy: 0.5822
Epoch 36/50
313/313 [==============================] - 3s 10ms/step - loss: 1.2928 - accuracy: 0.5436 - val_loss: 1.1931 - val_accuracy: 0.5895
Epoch 37/50
313/313 [==============================] - 3s 9ms/step - loss: 1.2939 - accuracy: 0.5435 - val_loss: 1.1912 - val_accuracy: 0.5910
Epoch 38/50
313/313 [==============================] - 3s 9ms/step - loss: 1.2866 - accuracy: 0.5477 - val_loss: 1.1803 - val_accuracy: 0.5944
Epoch 39/50
313/313 [==============================] - 3s 9ms/step - loss: 1.2856 - accuracy: 0.5472 - val_loss: 1.1857 - val_accuracy: 0.5873
Epoch 40/50
313/313 [==============================] - 3s 9ms/step - loss: 1.2805 - accuracy: 0.5469 - val_loss: 1.1860 - val_accuracy: 0.5937
Epoch 41/50
313/313 [==============================] - 3s 10ms/step - loss: 1.2802 - accuracy: 0.5480 - val_loss: 1.1894 - val_accuracy: 0.5898
Epoch 42/50
313/313 [==============================] - 3s 10ms/step - loss: 1.2763 - accuracy: 0.5472 - val_loss: 1.1839 - val_accuracy: 0.5947
Epoch 43/50
313/313 [==============================] - 3s 9ms/step - loss: 1.2752 - accuracy: 0.5462 - val_loss: 1.1794 - val_accuracy: 0.5942
Epoch 44/50
313/313 [==============================] - 3s 10ms/step - loss: 1.2762 - accuracy: 0.5486 - val_loss: 1.1775 - val_accuracy: 0.5956
Epoch 45/50
313/313 [==============================] - 3s 9ms/step - loss: 1.2684 - accuracy: 0.5507 - val_loss: 1.1784 - val_accuracy: 0.5912
Epoch 46/50
313/313 [==============================] - 3s 9ms/step - loss: 1.2709 - accuracy: 0.5506 - val_loss: 1.1805 - val_accuracy: 0.5891
Epoch 47/50
313/313 [==============================] - 3s 9ms/step - loss: 1.2668 - accuracy: 0.5509 - val_loss: 1.1830 - val_accuracy: 0.5898
Epoch 48/50
313/313 [==============================] - 3s 9ms/step - loss: 1.2695 - accuracy: 0.5512 - val_loss: 1.1730 - val_accuracy: 0.5999
Epoch 49/50
313/313 [==============================] - 3s 9ms/step - loss: 1.2653 - accuracy: 0.5536 - val_loss: 1.1793 - val_accuracy: 0.5928
Epoch 50/50
313/313 [==============================] - 3s 9ms/step - loss: 1.2622 - accuracy: 0.5515 - val_loss: 1.1772 - val_accuracy: 0.5899
```

In [19]:
```
fig, loss_ax = plt.subplots()
acc_ax = loss_ax.twinx()

loss_ax.plot(history.history['loss'], 'y', label='train loss')
loss_ax.plot(history.history['val_loss'], 'r', label='val loss')

acc_ax.plot(history.history['accuracy'], 'b', label='train acc')
acc_ax.plot(history.history['val_accuracy'], 'g', label='val acc')

loss_ax.set_xlabel('epoch')
loss_ax.set_ylabel('loss')
acc_ax.set_ylabel('accuray')

loss_ax.legend(loc='upper left')
acc_ax.legend(loc='lower left')

plt.show()
```



(5) 모델 평가하기

In [20]:
```
loss_and_accuracy = model.evaluate(X_test, Y_test, batch_size=128)
print('loss : %.4f, accruracy : %.4f'%(loss_and_accuracy[0],loss_and_accuracy[1]))
```

```
79/79 [==============================] - 1s 6ms/step - loss: 1.1906 - accuracy: 0.5835
loss : 1.1906, accruracy : 0.5835
```

(2) 모델링1 : Entire

사용가능 네트워크 : https://keras.io/api/applications/

In [21]:
```
# Get Feature Extroactor from VGG16
x = base_model.output

# Add Classifier
x = Flatten()(x)
x = Dropout(0.5)(x)
x = Dense(256, activation='relu')(x)
x = BatchNormalization()(x)
predictions = Dense(Y_train.shape[1], activation='softmax')(x) #Y_train.shape[1] :10

model = Model(inputs=base_model.input, outputs=predictions)
```

In [22]:
```
# first: train the all layers (which were randomly initialized)
for layer in base_model.layers:
    layer.trainable = True
```

In [23]:
```
model.summary()
```

```
Model: "model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 32, 32, 3)]       0

 block1_conv1 (Conv2D)       (None, 32, 32, 64)        1792

 block1_conv2 (Conv2D)       (None, 32, 32, 64)        36928

 block1_pool (MaxPooling2D)  (None, 16, 16, 64)        0

 block2_conv1 (Conv2D)       (None, 16, 16, 128)       73856

 block2_conv2 (Conv2D)       (None, 16, 16, 128)       147584

 block2_pool (MaxPooling2D)  (None, 8, 8, 128)         0

 block3_conv1 (Conv2D)       (None, 8, 8, 256)         295168

 block3_conv2 (Conv2D)       (None, 8, 8, 256)         590080

 block3_conv3 (Conv2D)       (None, 8, 8, 256)         590080

 block3_pool (MaxPooling2D)  (None, 4, 4, 256)         0

 block4_conv1 (Conv2D)       (None, 4, 4, 512)         1180160

 block4_conv2 (Conv2D)       (None, 4, 4, 512)         2359808

 block4_conv3 (Conv2D)       (None, 4, 4, 512)         2359808

 block4_pool (MaxPooling2D)  (None, 2, 2, 512)         0

 block5_conv1 (Conv2D)       (None, 2, 2, 512)         2359808

 block5_conv2 (Conv2D)       (None, 2, 2, 512)         2359808

 block5_conv3 (Conv2D)       (None, 2, 2, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 1, 1, 512)         0

 flatten_3 (Flatten)         (None, 512)               0

 dropout_1 (Dropout)         (None, 512)               0

 dense_6 (Dense)             (None, 256)               131328

 batch_normalization_1 (Batc  (None, 256)              1024
 hNormalization)

 dense_7 (Dense)             (None, 10)                2570

=================================================================
Total params: 14,849,610
Trainable params: 14,849,098
Non-trainable params: 512
_________________________________________________________________
```

(3) 모델의 학습과정 설정

In [24]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

(4) 모델 학습시키기

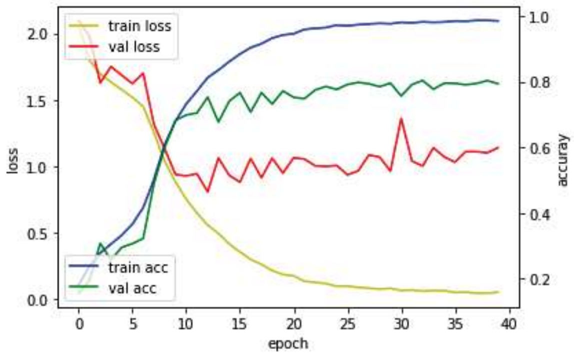
In [25]:
```
earlystopper = EarlyStopping(monitor='val_accuracy', patience=7, verbose=1, mode='auto', restore_best_weights=True)
history = model.fit(X_train, Y_train, batch_size=128, epochs=50, validation_split=0.2, callbacks = [earlystopper])
```

```
Epoch 1/50
313/313 [==============================] - 7s 21ms/step - loss: 2.0626 - accuracy: 0.1814 - val_loss: 2.0975 - val_accuracy: 0.1557
Epoch 2/50
313/313 [==============================] - 6s 20ms/step - loss: 1.7981 - accuracy: 0.2383 - val_loss: 1.9782 - val_accuracy: 0.1944
Epoch 3/50
313/313 [==============================] - 6s 20ms/step - loss: 1.6957 - accuracy: 0.2782 - val_loss: 1.6277 - val_accuracy: 0.3076
Epoch 4/50
313/313 [==============================] - 6s 20ms/step - loss: 1.6340 - accuracy: 0.3053 - val_loss: 1.7503 - val_accuracy: 0.2578
Epoch 5/50
313/313 [==============================] - 6s 20ms/step - loss: 1.5768 - accuracy: 0.3324 - val_loss: 1.6855 - val_accuracy: 0.2953
Epoch 6/50
313/313 [==============================] - 6s 20ms/step - loss: 1.5190 - accuracy: 0.3664 - val_loss: 1.6224 - val_accuracy: 0.3070
Epoch 7/50
313/313 [==============================] - 6s 20ms/step - loss: 1.4504 - accuracy: 0.4164 - val_loss: 1.7010 - val_accuracy: 0.3224
Epoch 8/50
313/313 [==============================] - 6s 20ms/step - loss: 1.2539 - accuracy: 0.5014 - val_loss: 1.3159 - val_accuracy: 0.4895
Epoch 9/50
313/313 [==============================] - 6s 20ms/step - loss: 1.0370 - accuracy: 0.6079 - val_loss: 1.1320 - val_accuracy: 0.5995
Epoch 10/50
313/313 [==============================] - 6s 19ms/step - loss: 0.8838 - accuracy: 0.6798 - val_loss: 0.9394 - val_accuracy: 0.6818
Epoch 11/50
313/313 [==============================] - 6s 19ms/step - loss: 0.7534 - accuracy: 0.7317 - val_loss: 0.9289 - val_accuracy: 0.6984
Epoch 12/50
313/313 [==============================] - 6s 19ms/step - loss: 0.6502 - accuracy: 0.7712 - val_loss: 0.9449 - val_accuracy: 0.7043
Epoch 13/50
313/313 [==============================] - 6s 19ms/step - loss: 0.5593 - accuracy: 0.8123 - val_loss: 0.8090 - val_accuracy: 0.7530
Epoch 14/50
313/313 [==============================] - 6s 19ms/step - loss: 0.4955 - accuracy: 0.8351 - val_loss: 1.0644 - val_accuracy: 0.6769
Epoch 15/50
313/313 [==============================] - 6s 19ms/step - loss: 0.4173 - accuracy: 0.8607 - val_loss: 0.9330 - val_accuracy: 0.7404
Epoch 16/50
313/313 [==============================] - 6s 19ms/step - loss: 0.3574 - accuracy: 0.8839 - val_loss: 0.8818 - val_accuracy: 0.7661
Epoch 17/50
313/313 [==============================] - 6s 20ms/step - loss: 0.3009 - accuracy: 0.9034 - val_loss: 1.0592 - val_accuracy: 0.7073
Epoch 18/50
313/313 [==============================] - 6s 21ms/step - loss: 0.2642 - accuracy: 0.9153 - val_loss: 0.9151 - val_accuracy: 0.7666
Epoch 19/50
313/313 [==============================] - 6s 21ms/step - loss: 0.2172 - accuracy: 0.9324 - val_loss: 1.0622 - val_accuracy: 0.7316
Epoch 20/50
313/313 [==============================] - 6s 21ms/step - loss: 0.1876 - accuracy: 0.9420 - val_loss: 0.9498 - val_accuracy: 0.7715
Epoch 21/50
313/313 [==============================] - 6s 20ms/step - loss: 0.1769 - accuracy: 0.9457 - val_loss: 1.0655 - val_accuracy: 0.7520
Epoch 22/50
313/313 [==============================] - 6s 20ms/step - loss: 0.1371 - accuracy: 0.9588 - val_loss: 1.0558 - val_accuracy: 0.7481
Epoch 23/50
313/313 [==============================] - 6s 19ms/step - loss: 0.1285 - accuracy: 0.9621 - val_loss: 1.0045 - val_accuracy: 0.7745
Epoch 24/50
313/313 [==============================] - 6s 20ms/step - loss: 0.1193 - accuracy: 0.9644 - val_loss: 0.9999 - val_accuracy: 0.7850
Epoch 25/50
313/313 [==============================] - 6s 20ms/step - loss: 0.0994 - accuracy: 0.9720 - val_loss: 1.0076 - val_accuracy: 0.7762
Epoch 26/50
313/313 [==============================] - 6s 19ms/step - loss: 0.1007 - accuracy: 0.9700 - val_loss: 0.9363 - val_accuracy: 0.7912
Epoch 27/50
313/313 [==============================] - 6s 20ms/step - loss: 0.0907 - accuracy: 0.9733 - val_loss: 0.9672 - val_accuracy: 0.7977
Epoch 28/50
313/313 [==============================] - 6s 20ms/step - loss: 0.0845 - accuracy: 0.9756 - val_loss: 1.0862 - val_accuracy: 0.7932
Epoch 29/50
313/313 [==============================] - 6s 20ms/step - loss: 0.0779 - accuracy: 0.9775 - val_loss: 1.0705 - val_accuracy: 0.7847
Epoch 30/50
313/313 [==============================] - 6s 20ms/step - loss: 0.0839 - accuracy: 0.9761 - val_loss: 0.9644 - val_accuracy: 0.7955
Epoch 31/50
313/313 [==============================] - 6s 19ms/step - loss: 0.0673 - accuracy: 0.9805 - val_loss: 1.3610 - val_accuracy: 0.7562
Epoch 32/50
313/313 [==============================] - 6s 20ms/step - loss: 0.0711 - accuracy: 0.9790 - val_loss: 1.0408 - val_accuracy: 0.7910
Epoch 33/50
313/313 [==============================] - 6s 20ms/step - loss: 0.0621 - accuracy: 0.9819 - val_loss: 1.0026 - val_accuracy: 0.8034
Epoch 34/50
313/313 [==============================] - 6s 19ms/step - loss: 0.0661 - accuracy: 0.9804 - val_loss: 1.1410 - val_accuracy: 0.7769
Epoch 35/50
313/313 [==============================] - 6s 19ms/step - loss: 0.0658 - accuracy: 0.9815 - val_loss: 1.0713 - val_accuracy: 0.7951
Epoch 36/50
313/313 [==============================] - 6s 19ms/step - loss: 0.0523 - accuracy: 0.9845 - val_loss: 1.0322 - val_accuracy: 0.7947
Epoch 37/50
313/313 [==============================] - 6s 20ms/step - loss: 0.0555 - accuracy: 0.9840 - val_loss: 1.1113 - val_accuracy: 0.7905
Epoch 38/50
313/313 [==============================] - 6s 19ms/step - loss: 0.0467 - accuracy: 0.9863 - val_loss: 1.1115 - val_accuracy: 0.7943
Epoch 39/50
313/313 [==============================] - 6s 20ms/step - loss: 0.0475 - accuracy: 0.9862 - val_loss: 1.1016 - val_accuracy: 0.8030
Epoch 40/50
313/313 [==============================] - ETA: 0s - loss: 0.0544 - accuracy: 0.9848Restoring model weights from the end of the best epoch: 33.
313/313 [==============================] - 6s 20ms/step - loss: 0.0544 - accuracy: 0.9848 - val_loss: 1.1413 - val_accuracy: 0.7938
Epoch 40: early stopping
```

In [26]:

```
fig, loss_ax = plt.subplots()
acc_ax = loss_ax.twinx()

loss_ax.plot(history.history['loss'], 'y', label='train loss')
loss_ax.plot(history.history['val_loss'], 'r', label='val loss')

acc_ax.plot(history.history['accuracy'], 'b', label='train acc')
acc_ax.plot(history.history['val_accuracy'], 'g', label='val acc')

loss_ax.set_xlabel('epoch')
loss_ax.set_ylabel('loss')
acc_ax.set_ylabel('accuray')

loss_ax.legend(loc='upper left')
acc_ax.legend(loc='lower left')
```

```
plt.show()
```



(5) 모델 평가하기

In [27]:
```
loss_and_accuracy = model.evaluate(X_test, Y_test, batch_size=128)
print('loss : %.4f, accruracy : %.4f'%(loss_and_accuracy[0],loss_and_accuracy[1]))
```

```
79/79 [==============================] - 1s 7ms/step - loss: 1.0334 - accuracy: 0.8012
loss : 1.0334, accruracy : 0.8012
```

3.4 Keras MNIST - 모델의 성능을 직접 높혀보자

- CNN의 구조를 바꾸어 나만의 모델을 만들어보자
- 목표 정확도: 99.5% 만들기
- 바꿀 수 있는 하이퍼 파라미터: Learning Rate, Batch size, Epochs, Optimizer, Activation Function, 모델 레이어 구조, BN, DO, DA, Fine Tuning 등

(1) 데이터셋

In [28]:
```
(X_train, Y_train),(X_test, Y_test) = datasets.mnist.load_data()
print(X_train.shape, Y_train.shape)
```

```
(60000, 28, 28) (60000,)
```

In [29]:
```
from tensorflow.keras import backend
backend.image_data_format()
```

Out[29]:
```
'channels_last'
```

In [30]:
```
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], X_train.shape[2], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], X_test.shape[2], 1)

Y_train = utils.to_categorical(Y_train)
Y_test = utils.to_categorical(Y_test)

print(X_train.shape, Y_train.shape)

n_in = X_train.shape[1:]
n_out = Y_train.shape[-1]
```

```
(60000, 28, 28, 1) (60000, 10)
```

(2) Keras 모델링

In [ ]:

(3) 모델의 학습과정 설정

In [ ]:

(4) 모델 학습시키기

In [ ]:

(5) 모델 평가하기

In [ ]: