딥러닝 (Deep Learning)

양승우

딥러닝은 머신러닝 알고리즘의 한 종류입니다.

데이터 처리 유닛의 층을 여러 개 쌓아 구조적이지 않은 데이터로부터 고수준 표현을 학습합니다.

그림 2-2 딥러닝의 개념도

먼저 책에 나와 있는 예제를 따라 해보고

어떻게 하면 딥러닝 모델의 성능을 향상 할 수 있는지 고민해 보겠습니다.

첫 번째 심층 신경망

라이브러리 임포트

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt

from keras.layers import Input, Flatten, Dense, Conv2D
from keras.models import Model
from keras.optimizers import Adam
from keras.utils import to_categorical

from keras.datasets import cifar10
```

```
Using TensorFlow backend.
C:₩Users₩User₩anaconda3₩envs₩testGAN₩lib₩site-packages₩tensorflow₩python₩framework₩dtypes.py:526: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:₩Users₩User₩anaconda3₩envs₩testGAN₩lib₩site-packages₩tensorflow₩python₩framework₩dtypes.py:527: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:₩Users₩User₩anaconda3₩envs₩testGAN₩lib₩site-packages₩tensorflow₩python₩framework₩dtypes.py:528: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:₩Users₩User₩anaconda3₩envs₩testGAN₩lib₩site-packages₩tensorflow₩python₩framework₩dtypes.py:529: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:₩Users₩User₩anaconda3₩envs₩testGAN₩lib₩site-packages₩tensorflow₩python₩framework₩dtypes.py:530: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:₩Users₩User₩anaconda3₩envs₩testGAN₩lib₩site-packages₩tensorflow₩python₩framework₩dtypes.py:535: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

데이터 적재

In [2]:
```
NUM_CLASSES = 10
```

In [3]:
```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

In [4]:
```
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

y_train = to_categorical(y_train, NUM_CLASSES)
y_test = to_categorical(y_test, NUM_CLASSES)
```

In [5]:
```
# 색 채널 R : 빨간색(0) , G : 초록색 (1) , B : 파란색 (2)
```

In [6]:
```
x_train[54, 12, 13, 1] # 인덱스 54의 이미지에서 (12, 13)에 해당하는 픽셀의 초록색 채널(1)의 값
```

Out[6]:
```
0.36862746
```

In [7]:
```
x_train[2050, 12, 16, 0] # 인덱스 2050의 이미지에서 (12, 16)에 해당하는 픽셀의 빨간색 채널(0)의 값
```

Out[7]:
```
0.1254902
```

여기서는 Sequential 모델보다 함수형 API를 사용하여 유연성을 높입니다.

In [8]:
```
input_layer = Input((32,32,3))

x = Flatten()(input_layer)

x = Dense(200, activation = 'relu')(x)
x = Dense(150, activation = 'relu')(x)

output_layer = Dense(NUM_CLASSES, activation = 'softmax')(x)

model = Model(input_layer, output_layer)
```

```
WARNING:tensorflow:From C:₩Users₩User₩anaconda3₩envs₩testGAN₩lib₩site-packages₩tensorflow₩python₩framework₩op_def_library.py:263: colocate_
with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
```

In [9]:
```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 32, 32, 3)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3072)              0
_________________________________________________________________
dense_1 (Dense)              (None, 200)               614600
_________________________________________________________________
dense_2 (Dense)              (None, 150)               30150
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1510
=================================================================
Total params: 646,260
Trainable params: 646,260
Non-trainable params: 0
_________________________________________________________________
```

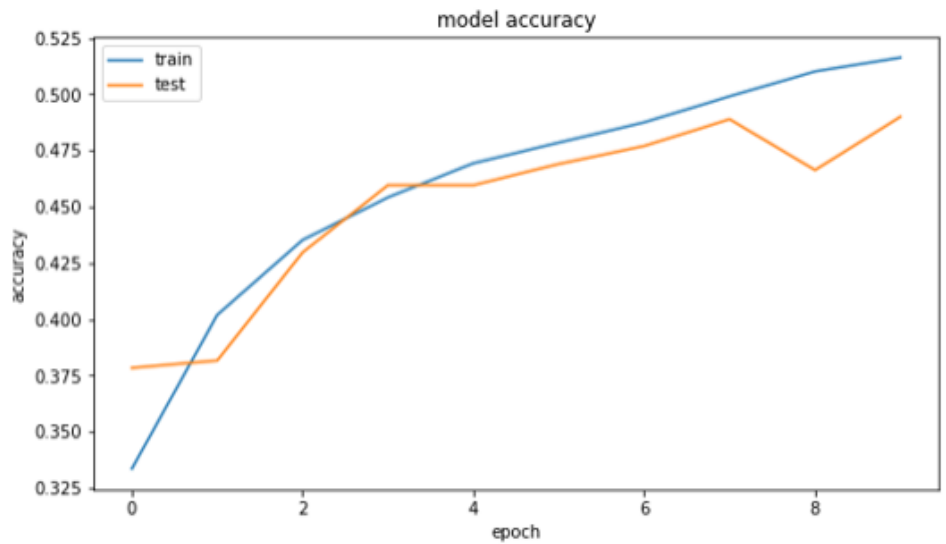
모델 훈련

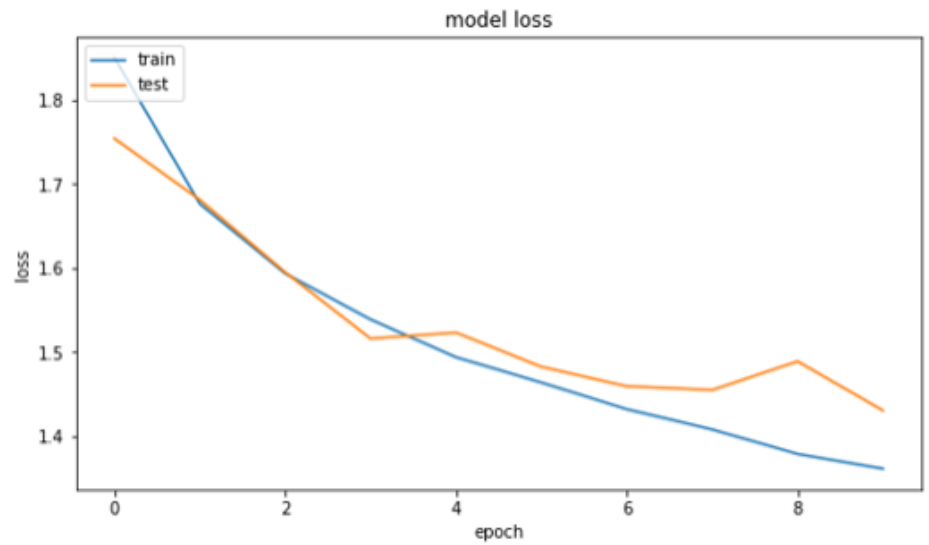
In [11]:
```
history = model.fit(x_train
                    , y_train
                    , batch_size=50
                    , epochs=10
                    , shuffle=True
                    , validation_data = (x_test, y_test))
```

```
WARNING:tensorflow:From C:₩Users₩User₩anaconda3₩envs₩testGAN₩lib₩site-packages₩tensorflow₩python₩ops₩math_ops.py:3066: to_int32 (from tenso
rflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [==============================] - 29s 572us/step - loss: 1.8507 - acc: 0.3335 - val_loss: 1.7544 - val_acc: 0.3783
Epoch 2/10
50000/50000 [==============================] - 33s 661us/step - loss: 1.6766 - acc: 0.4019 - val_loss: 1.6815 - val_acc: 0.3816
Epoch 3/10
50000/50000 [==============================] - 24s 482us/step - loss: 1.5937 - acc: 0.4352 - val_loss: 1.5951 - val_acc: 0.4297
Epoch 4/10
50000/50000 [==============================] - 22s 444us/step - loss: 1.5389 - acc: 0.4542 - val_loss: 1.5162 - val_acc: 0.4596
Epoch 5/10
50000/50000 [==============================] - 28s 560us/step - loss: 1.4940 - acc: 0.4693 - val_loss: 1.5234 - val_acc: 0.4596
Epoch 6/10
50000/50000 [==============================] - 28s 567us/step - loss: 1.4639 - acc: 0.4785 - val_loss: 1.4826 - val_acc: 0.4690
Epoch 7/10
50000/50000 [==============================] - 29s 583us/step - loss: 1.4321 - acc: 0.4875 - val_loss: 1.4595 - val_acc: 0.4770
Epoch 8/10
50000/50000 [==============================] - 27s 533us/step - loss: 1.4080 - acc: 0.4991 - val_loss: 1.4550 - val_acc: 0.4889
Epoch 9/10
50000/50000 [==============================] - 26s 526us/step - loss: 1.3788 - acc: 0.5102 - val_loss: 1.4891 - val_acc: 0.4663
Epoch 10/10
50000/50000 [==============================] - 27s 546us/step - loss: 1.3612 - acc: 0.5163 - val_loss: 1.4306 - val_acc: 0.4900
```

모델 성능 그래프 (train accuracy & train loss)

In [12]:
```
plt.figure(figsize=(20, 5))

# summarize history for accuracy
plt.subplot(121)
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')

# summarize history for loss
plt.subplot(122)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')

plt.show()
```

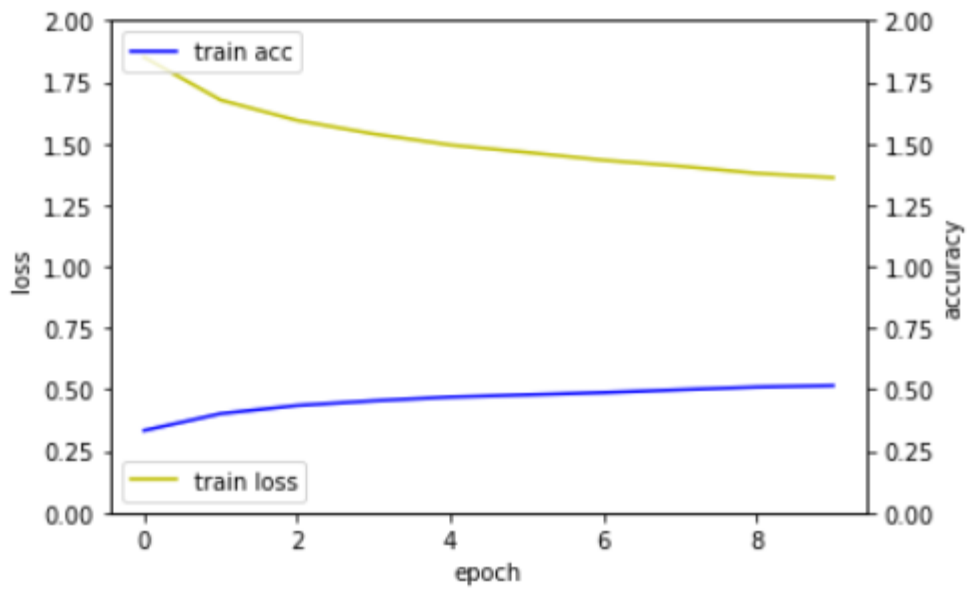




모델 성능 그래프 (train accuracy & train loss)

In [13]:
```
fig, loss_ax = plt.subplots()
acc_ax = loss_ax.twinx()

loss_ax.set_ylim([0.0, 2.0])
acc_ax.set_ylim([0.0, 2.0])

loss_ax.plot(history.history['loss'], 'y', label = 'train loss')
acc_ax.plot(history.history['acc'], 'b', label = 'train acc')

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

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

Out[13]:
```
<matplotlib.legend.Legend at 0x1eb028a6208>
```



모델 평가

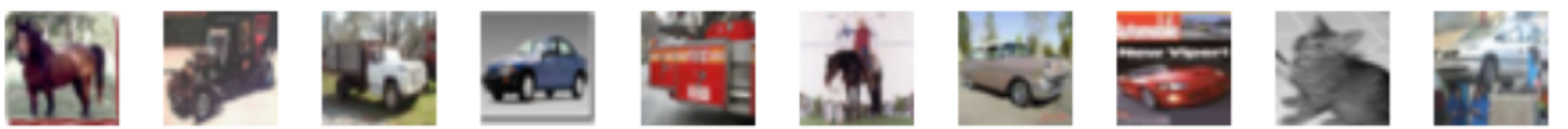
In [14]:
```
model.evaluate(x_test, y_test)
```

```
10000/10000 [==============================] - 1s 120us/step
```

Out[14]: [1.426109185218811, 0.4985] acc = 49.8%

In [15]:
```
CLASSES = np.array(['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'])

preds = model.predict(x_test)
preds_single = CLASSES[np.argmax(preds, axis = -1)]
actual_single = CLASSES[np.argmax(y_test, axis = -1)]
```

In [16]:
```
n_to_show = 10
indices = np.random.choice(range(len(x_test)), n_to_show)

fig = plt.figure(figsize=(15, 3))
fig.subplots_adjust(hspace=0.4, wspace=0.4)

for i, idx in enumerate(indices):
    img = x_test[idx]
    ax = fig.add_subplot(1, n_to_show, i+1)
    ax.axis('off')
    ax.text(0.5, -0.35, 'pred = ' + str(preds_single[idx]), fontsize=10, ha='center', transform=ax.transAxes)
    ax.text(0.5, -0.7, 'act = ' + str(actual_single[idx]), fontsize=10, ha='center', transform=ax.transAxes)
    ax.imshow(img)
```



어떻게 하면 정확도를 끌어 올릴 수 있을까요?

여러 방법이 있지만 책에 나와있는 방법으로는

1. 합성곱 층(Convolution Layer)
2. 배치의 정규화 (Batch Normalization)
3. 드롭아웃 (Drop Out)

의 사용을 예로 들고 있습니다.

먼저 합성곱 층이 무엇인지 가시적(visualize)으로 확인해 보고

이것들을 적용해보고 정말로 정확도가 올라가는지 확인해 봅니다.

이미지의 특징을 잘 나타내기 위해 합성곱 층(convolution layer) 을 사용합니다.

합성곱은 필터를 이미지의 일부분과 픽셀끼리 곱한 후 결과를 더하는 것입니다.

이미지의 영역이 필터와 비슷할수록 큰 양수가 출력되고 필터와 반대일 수록 큰 음수가 출력됩니다.

그림 2-10 합성곱 연산

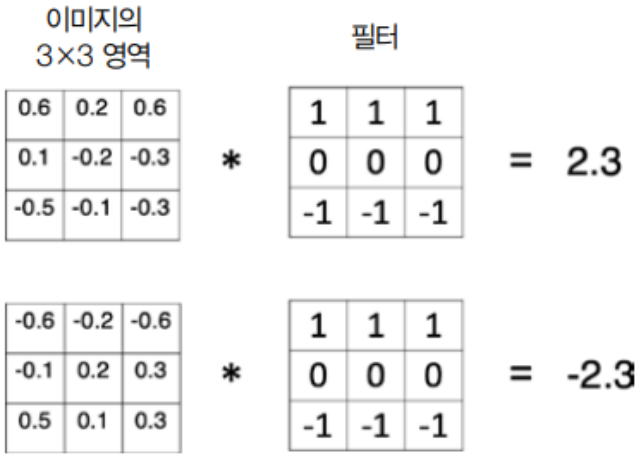
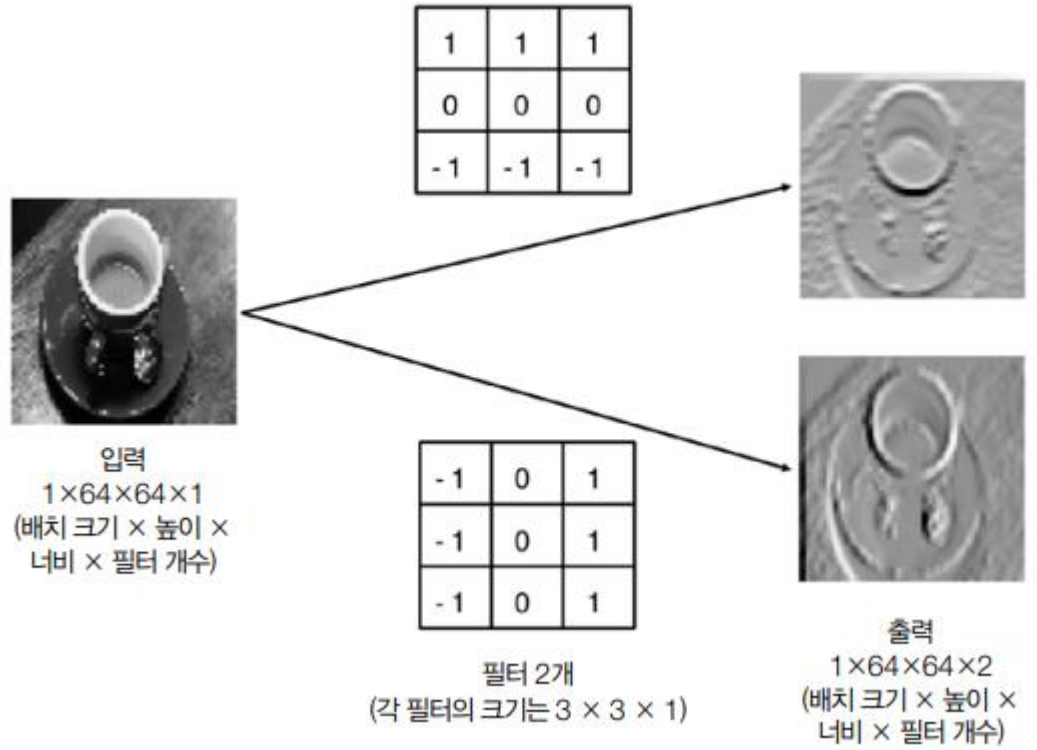


그림 2-11 흑백 이미지에 적용된 두 개의 합성곱 필터



합성곱

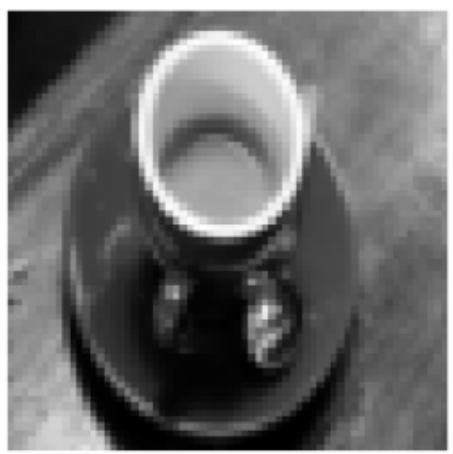
라이브러리 임포트

In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
from scipy.ndimage import correlate
import numpy as np
from skimage import data
from skimage.color import rgb2gray
from skimage.transform import rescale,resize
```

원본 이미지

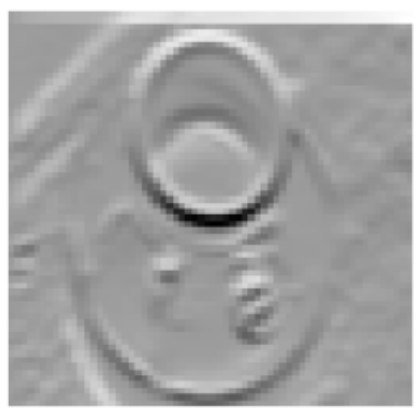
In [2]:
```
im = rgb2gray(data.coffee())
im = resize(im, (64,64))
print(im.shape)

plt.axis('off')
plt.imshow(im, cmap = 'gray');
```

```
(64, 64)
```



수평 모서리 필터

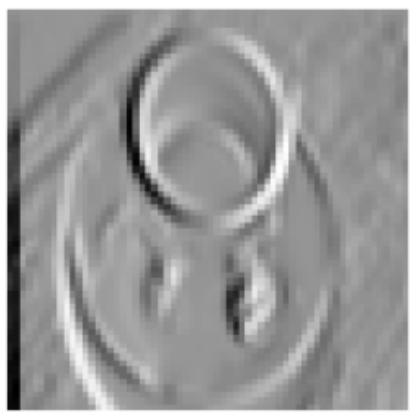
In [3]:
```
filter1 = np.array([
    [ 1,  1,  1],
    [ 0,  0,  0],
    [-1, -1, -1]
])

new_image = np.zeros(im.shape)

im_pad = np.pad(im, 1, 'constant')

for i in range(im.shape[0]):
    for j in range(im.shape[1]):
        try:
            new_image[i,j] = ₩
            im_pad[i-1,j-1] * filter1[0,0] + ₩
            im_pad[i-1,j] * filter1[0,1] + ₩
            im_pad[i-1,j+1] * filter1[0,2] + ₩
            im_pad[i,j-1] * filter1[1,0] + ₩
            im_pad[i,j] * filter1[1,1] + ₩
            im_pad[i,j+1] * filter1[1,2] +₩
            im_pad[i+1,j-1] * filter1[2,0] + ₩
            im_pad[i+1,j] * filter1[2,1] + ₩
            im_pad[i+1,j+1] * filter1[2,2]
        except:
            pass

plt.axis('off')
plt.imshow(new_image, cmap='Greys');
```



수평 부분의 모서리가 강조됩니다.

수직 모서리 필터

In [4]:
```
filter2 = np.array([
    [ -1,  0,  1],
    [ -1,  0,  1],
    [ -1,  0,  1]
])

new_image = np.zeros(im.shape)

im_pad = np.pad(im,1, 'constant')

for i in range(im.shape[0]):
    for j in range(im.shape[1]):
        try:
            new_image[i,j] = ₩
            im_pad[i-1,j-1] * filter2[0,0] + ₩
            im_pad[i-1,j] * filter2[0,1] + ₩
            im_pad[i-1,j+1] * filter2[0,2] + ₩
            im_pad[i,j-1] * filter2[1,0] + ₩
            im_pad[i,j] * filter2[1,1] + ₩
            im_pad[i,j+1] * filter2[1,2] +₩
            im_pad[i+1,j-1] * filter2[2,0] + ₩
            im_pad[i+1,j] * filter2[2,1] + ₩
            im_pad[i+1,j+1] * filter2[2,2]
        except:
            pass

plt.axis('off')
plt.imshow(new_image, cmap='Greys');
```



수직 부분의 모서리가 강조됩니다.

첫 번째 합성곱 신경망 (모델 성능 향상 ver.)

라이브러리 임포트

In [1]:
```
import numpy as np

from keras.layers import Input, Flatten, Dense, Conv2D, BatchNormalization, LeakyReLU, Dropout, Activation
from keras.models import Model
from keras.optimizers import Adam
from keras.utils import to_categorical
import keras.backend as K

from keras.datasets import cifar10
```

```
Using TensorFlow backend.
C:₩Users₩User₩anaconda3₩envs₩testGAN₩lib₩site-packages₩tensorflow₩python₩framework₩dtypes.py:526: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:₩Users₩User₩anaconda3₩envs₩testGAN₩lib₩site-packages₩tensorflow₩python₩framework₩dtypes.py:527: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:₩Users₩User₩anaconda3₩envs₩testGAN₩lib₩site-packages₩tensorflow₩python₩framework₩dtypes.py:528: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:₩Users₩User₩anaconda3₩envs₩testGAN₩lib₩site-packages₩tensorflow₩python₩framework₩dtypes.py:529: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:₩Users₩User₩anaconda3₩envs₩testGAN₩lib₩site-packages₩tensorflow₩python₩framework₩dtypes.py:530: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:₩Users₩User₩anaconda3₩envs₩testGAN₩lib₩site-packages₩tensorflow₩python₩framework₩dtypes.py:535: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

데이터 적재

In [2]:
```
NUM_CLASSES = 10
```

In [3]:
```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

In [4]:
```
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

y_train = to_categorical(y_train, NUM_CLASSES)
y_test = to_categorical(y_test, NUM_CLASSES)
```

모델 만들기

In [5]:
```
input_layer = Input((32,32,3))

x = Conv2D(filters = 32, kernel_size = 3, strides = 1, padding = 'same')(input_layer)
x = BatchNormalization()(x)
x = LeakyReLU()(x)


x = Conv2D(filters = 32, kernel_size = 3, strides = 2, padding = 'same')(x)
x = BatchNormalization()(x)
x = LeakyReLU()(x)


x = Conv2D(filters = 64, kernel_size = 3, strides = 1, padding = 'same')(x)
x = BatchNormalization()(x)
x = LeakyReLU()(x)


x = Conv2D(filters = 64, kernel_size = 3, strides = 2, padding = 'same')(x)
x = BatchNormalization()(x)
x = LeakyReLU()(x)


x = Flatten()(x)

x = Dense(128)(x)
x = BatchNormalization()(x)
x = LeakyReLU()(x)
x = Dropout(rate = 0.5)(x)

x = Dense(NUM_CLASSES)(x)
output_layer = Activation('softmax')(x)

model = Model(input_layer, output_layer)
```

```
WARNING:tensorflow:From C:₩Users₩User₩anaconda3₩envs₩testGAN₩lib₩site-packages₩tensorflow₩python₩framework₩op_def_library.py:263: colocate_
with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From C:₩Users₩User₩anaconda3₩envs₩testGAN₩lib₩site-packages₩keras₩backend₩tensorflow_backend.py:3445: calling dropout (f
rom tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
```

In [6]:
```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 32, 32, 3)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 32)        128
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 32)        9248
_________________________________________________________________
batch_normalization_2 (Batch (None, 16, 16, 32)        128
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
batch_normalization_3 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 64)          36928
_________________________________________________________________
batch_normalization_4 (Batch (None, 8, 8, 64)          256
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 8, 8, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4096)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               524416
_________________________________________________________________
batch_normalization_5 (Batch (None, 128)               512
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 128)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
_________________________________________________________________
activation_1 (Activation)    (None, 10)                0
=================================================================
Total params: 592,554
Trainable params: 591,914
Non-trainable params: 640
_________________________________________________________________
```

모델 훈련

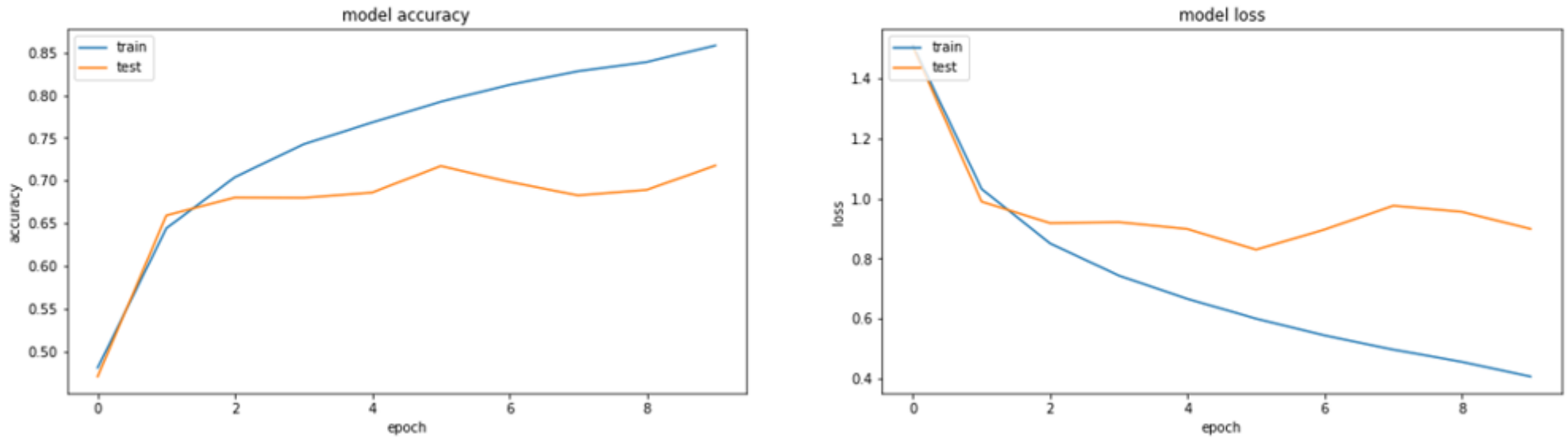
In [7]:
```
opt = Adam(lr=0.0005)
model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
```

In [8]:
```
history = model.fit(x_train
                    , y_train
                    , batch_size=50
                    , epochs=10
                    , shuffle=True
                    , validation_data = (x_test, y_test))
```

```
WARNING:tensorflow:From C:₩Users₩User₩anaconda3₩envs₩testGAN₩lib₩site-packages₩tensorflow₩python₩ops₩math_ops.py:3066: to_int32 (from tenso
rflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [==============================] - 815s 16ms/step - loss: 1.5103 - acc: 0.4810 - val_loss: 1.5046 - val_acc: 0.4707
Epoch 2/10
50000/50000 [==============================] - 693s 14ms/step - loss: 1.0823 - acc: 0.6443 - val_loss: 0.9900 - val_acc: 0.6594
Epoch 3/10
50000/50000 [==============================] - 788s 16ms/step - loss: 0.8505 - acc: 0.7042 - val_loss: 0.9184 - val_acc: 0.6803
Epoch 4/10
50000/50000 [==============================] - 870s 17ms/step - loss: 0.7437 - acc: 0.7426 - val_loss: 0.9215 - val_acc: 0.6800
Epoch 5/10
50000/50000 [==============================] - 900s 18ms/step - loss: 0.6659 - acc: 0.7682 - val_loss: 0.8986 - val_acc: 0.6862
Epoch 6/10
50000/50000 [==============================] - 788s 16ms/step - loss: 0.6001 - acc: 0.7925 - val_loss: 0.8299 - val_acc: 0.7172
Epoch 7/10
50000/50000 [==============================] - 876s 18ms/step - loss: 0.5441 - acc: 0.8122 - val_loss: 0.8970 - val_acc: 0.6986
Epoch 8/10
50000/50000 [==============================] - 1045s 21ms/step - loss: 0.4968 - acc: 0.8282 - val_loss: 0.9762 - val_acc: 0.6829
Epoch 9/10
50000/50000 [==============================] - 1003s 20ms/step - loss: 0.4562 - acc: 0.8389 - val_loss: 0.9560 - val_acc: 0.6893
Epoch 10/10
50000/50000 [==============================] - 1043s 21ms/step - loss: 0.4071 - acc: 0.8583 - val_loss: 0.8989 - val_acc: 0.7177
```

모델 성능 그래프 (train accuracy & train loss)

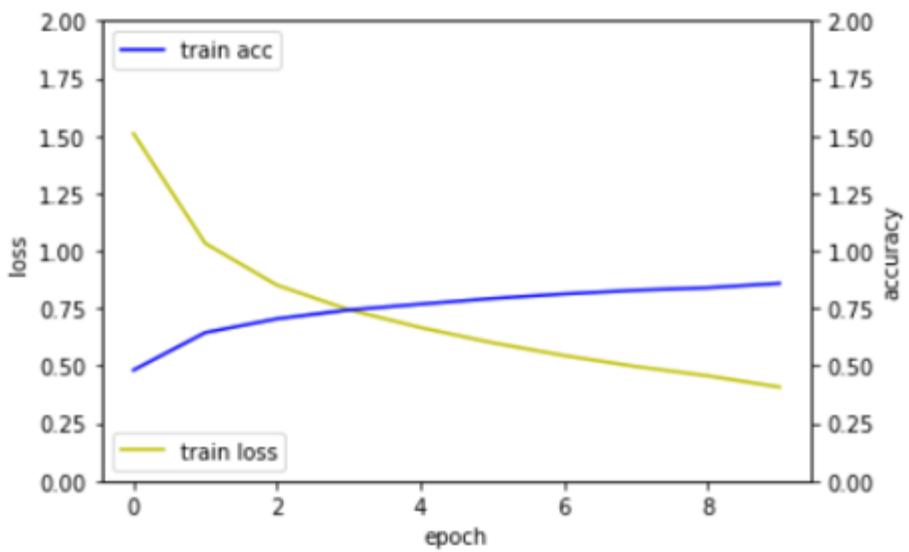
In [9]:
```
plt.figure(figsize=(20, 5))

# summarize history for accuracy
plt.subplot(121)
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')

# summarize history for loss
plt.subplot(122)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')

plt.show()
```



모델 성능 그래프 (train accuracy & train loss)

In [10]:
```
fig, loss_ax = plt.subplots()
acc_ax = loss_ax.twinx()

loss_ax.set_ylim([0.0, 2.0])
acc_ax.set_ylim([0.0, 2.0])

loss_ax.plot(history.history['loss'], 'y', label = 'train loss')
acc_ax.plot(history.history['acc'], 'b', label = 'train acc')

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

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

Out[10]: <matplotlib.legend.Legend at 0x21605aff438>



모델 평가

In [11]:
```
model.evaluate(x_test, y_test, batch_size=1000)
```

```
10000/10000 [==============================] - 130s 13ms/step
```

Out[11]: [0.8988511860370636, 0.7176999986171723] acc = 71.7%

In [12]:
```
CLASSES = np.array(['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'])

preds = model.predict(x_test)
preds_single = CLASSES[np.argmax(preds, axis = -1)]
actual_single = CLASSES[np.argmax(y_test, axis = -1)]
```

In [13]:
```
import matplotlib.pyplot as plt

n_to_show = 10
indices = np.random.choice(range(len(x_test)), n_to_show)

fig = plt.figure(figsize=(15, 3))
fig.subplots_adjust(hspace=0.5, wspace=0.5)

for i, idx in enumerate(indices):
    img = x_test[idx]
    ax = fig.add_subplot(1, n_to_show, i+1)
    ax.axis('off')
    ax.text(0.5, -0.35, 'pred = ' + str(preds_single[idx]), fontsize=10, ha='center', transform=ax.transAxes)
    ax.text(0.5, -0.7, 'act = ' + str(actual_single[idx]), fontsize=10, ha='center', transform=ax.transAxes)
    ax.imshow(img)
```

pred = dog	pred = frog	pred = horse	pred = airplane	pred = dog	pred = cat	pred = airplane	pred = truck	pred = automobile	pred = frog
act = dog	act = bird	act = horse	act = airplane	act = dog	act = frog	act = airplane	act = truck	act = airplane	act = frog

사실 파라미터는 어느 한 개의 변수에 종속되어 변하는 것이 아니어서

어떤 변수를 변경하면 무조건 정확도가 올라간다는 것이 보장 되지 않습니다.

그 이유는 신경망이 어떤 원리로 특징을 찾아내는지 알 수 없는 것에서 찾아 볼 수 있습니다.

따라서 스무고개를 하듯이 파라미터를 변경을 하며 최적을 상태를 찾아갑니다.

이제 본인이 나름대로 파라미터를 변경 해 보았습니다.

모델 만들기

In [5]:
```
input_layer = Input((32,32,3))

x = Conv2D(filters = 32, kernel_size = 3, strides = 1, padding = 'same')(input_layer)
x = BatchNormalization()(x)
x = LeakyReLU()(x)


x = Conv2D(filters = 32, kernel_size = 3, strides = 2, padding = 'same')(x)
x = BatchNormalization()(x)
x = LeakyReLU()(x)


x = Conv2D(filters = 64, kernel_size = 3, strides = 1, padding = 'same')(x)
x = BatchNormalization()(x)
x = LeakyReLU()(x)


x = Conv2D(filters = 64, kernel_size = 3, strides = 2, padding = 'same')(x)
x = BatchNormalization()(x)
x = LeakyReLU()(x)


x = Flatten()(x)

x = Dense(128)(x)
x = BatchNormalization()(x)
x = LeakyReLU()(x)

x = Dense(NUM_CLASSES)(x)
output_layer = Activation('softmax')(x)

model = Model(input_layer, output_layer)
```

```
WARNING:tensorflow:From C:₩Users₩User₩anaconda3₩envs₩testGAN₩lib₩site-packages₩tensorflow₩python₩framework₩op_def_library.py:263: colocate_
with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
```

모델 훈련

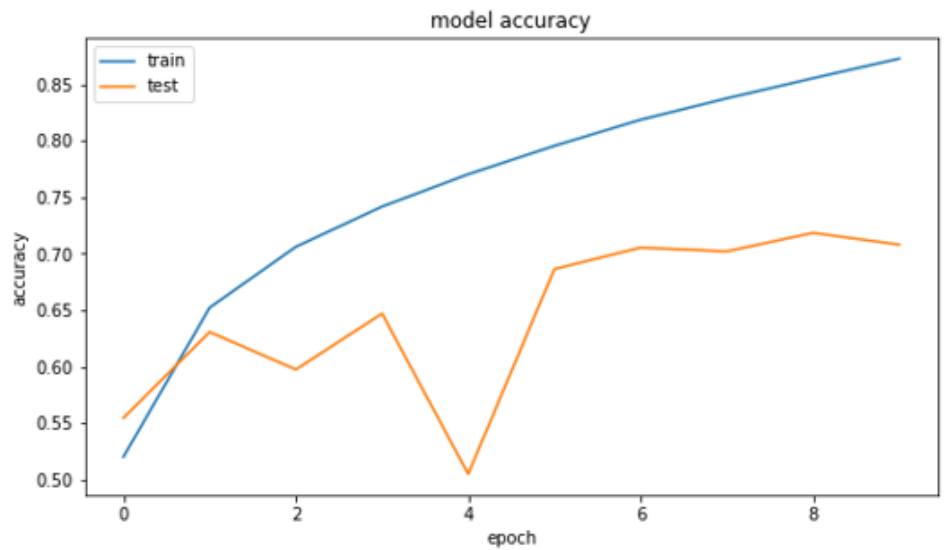
In [7]:
```
opt = Adam(lr=0.0005)
model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
```

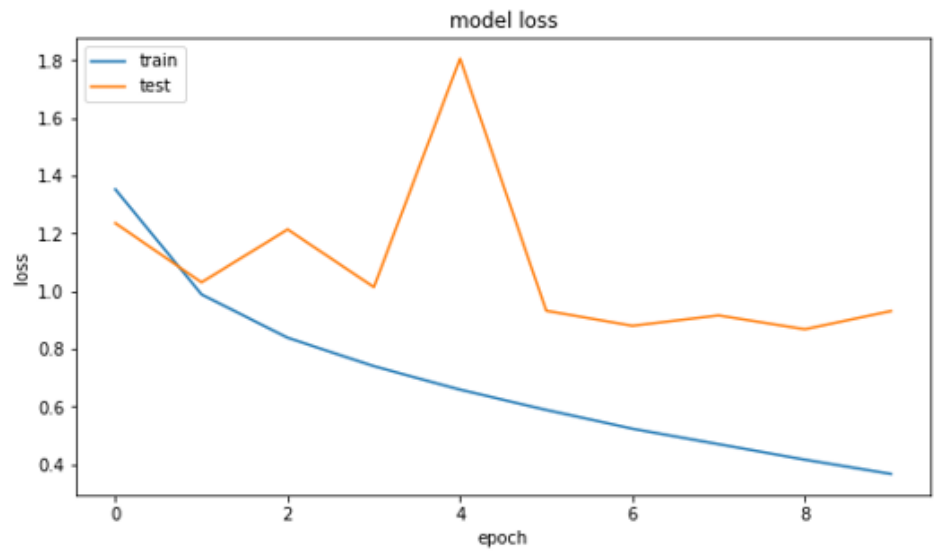
In [8]:
```
history = model.fit(x_train
                    , y_train
                    , batch_size=50
                    , epochs=10
                    , shuffle=True
                    , validation_data = (x_test, y_test))
```

```
WARNING:tensorflow:From C:₩Users₩User₩anaconda3₩envs₩testGAN₩lib₩site-packages₩tensorflow₩python₩ops₩math_ops.py:3066: to_int32 (from tenso
rflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [==============================] - 621s 12ms/step - loss: 1.3529 - acc: 0.5202 - val_loss: 1.2356 - val_acc: 0.5546
Epoch 2/10
50000/50000 [==============================] - 612s 12ms/step - loss: 0.9888 - acc: 0.6522 - val_loss: 1.0306 - val_acc: 0.6306
Epoch 3/10
50000/50000 [==============================] - 600s 12ms/step - loss: 0.8384 - acc: 0.7060 - val_loss: 1.2138 - val_acc: 0.5974
Epoch 4/10
50000/50000 [==============================] - 611s 12ms/step - loss: 0.7400 - acc: 0.7416 - val_loss: 1.0140 - val_acc: 0.6469
Epoch 5/10
50000/50000 [==============================] - 700s 14ms/step - loss: 0.6586 - acc: 0.7702 - val_loss: 1.8058 - val_acc: 0.5049
Epoch 6/10
50000/50000 [==============================] - 691s 14ms/step - loss: 0.5885 - acc: 0.7954 - val_loss: 0.9318 - val_acc: 0.6863
Epoch 7/10
50000/50000 [==============================] - 618s 12ms/step - loss: 0.5234 - acc: 0.8185 - val_loss: 0.8803 - val_acc: 0.7053
Epoch 8/10
50000/50000 [==============================] - 745s 15ms/step - loss: 0.4696 - acc: 0.8374 - val_loss: 0.9158 - val_acc: 0.7020
Epoch 9/10
50000/50000 [==============================] - 868s 17ms/step - loss: 0.4163 - acc: 0.8552 - val_loss: 0.8682 - val_acc: 0.7183
Epoch 10/10
50000/50000 [==============================] - 890s 18ms/step - loss: 0.3664 - acc: 0.8726 - val_loss: 0.9308 - val_acc: 0.7080
```

모델 성능 그래프 (train accuracy & train loss)

In [9]:
```
plt.figure(figsize=(20, 5))

# summarize history for accuracy
plt.subplot(121)
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')

# summarize history for loss
plt.subplot(122)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')

plt.show()
```

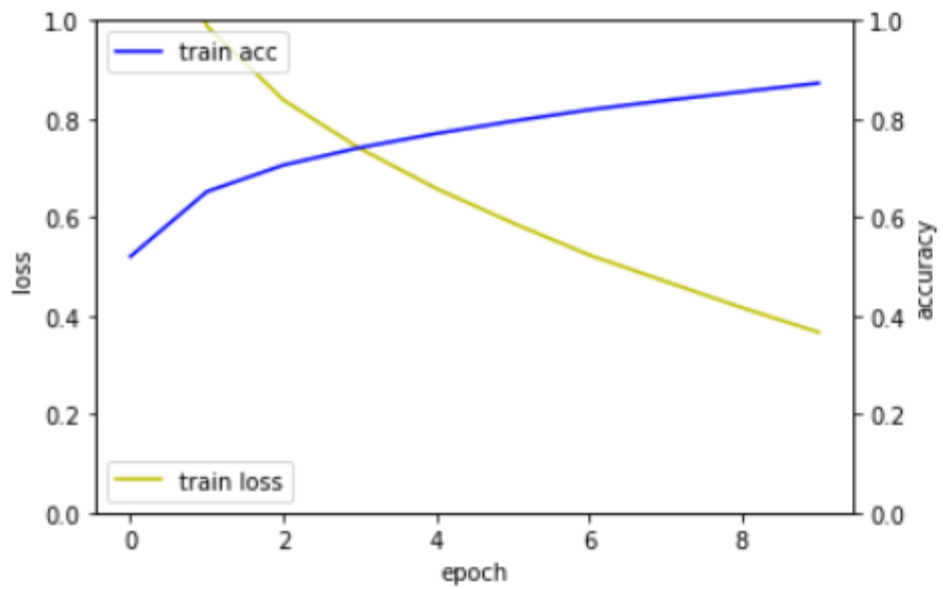




모델 성능 그래프 (train accuracy & train loss)

In [10]:
```
fig, loss_ax = plt.subplots()
acc_ax = loss_ax.twinx()

loss_ax.set_ylim([0.0, 1.0])
acc_ax.set_ylim([0.0, 1.0])

loss_ax.plot(history.history['loss'], 'y', label = 'train loss')
acc_ax.plot(history.history['acc'], 'b', label = 'train acc')

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

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

Out[10]: <matplotlib.legend.Legend at 0x27585555240>



모델 평가

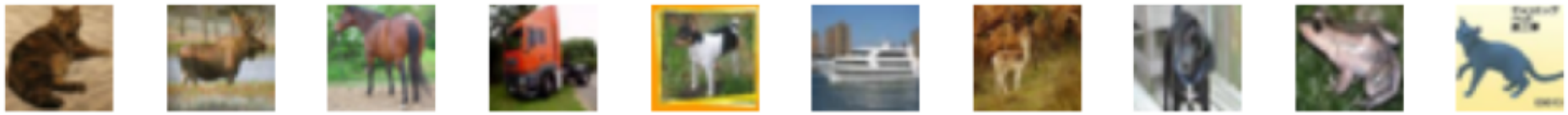
In [11]:
```
model.evaluate(x_test, y_test, batch_size=1000)
```

```
10000/10000 [==============================] - 50s 5ms/step
```

Out[11]: [0.930810272693634, 0.7080000042915344] acc = 70.8%

In [12]:
```
CLASSES = np.array(['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'])

preds = model.predict(x_test)
preds_single = CLASSES[np.argmax(preds, axis = -1)]
actual_single = CLASSES[np.argmax(y_test, axis = -1)]
```

In [13]:
```
import matplotlib.pyplot as plt

n_to_show = 10
indices = np.random.choice(range(len(x_test)), n_to_show)

fig = plt.figure(figsize=(15, 3))
fig.subplots_adjust(hspace=0.5, wspace=0.5)

for i, idx in enumerate(indices):
    img = x_test[idx]
    ax = fig.add_subplot(1, n_to_show, i+1)
    ax.axis('off')
    ax.text(0.5, -0.35, 'pred = ' + str(preds_single[idx]), fontsize=10, ha='center', transform=ax.transAxes)
    ax.text(0.5, -0.7, 'act = ' + str(actual_single[idx]), fontsize=10, ha='center', transform=ax.transAxes)
    ax.imshow(img)
```



pred = cat act = cat | pred = deer act = deer | pred = horse act = horse | pred = truck act = truck | pred = dog act = dog | pred = ship act = ship | pred = dog act = deer | pred = dog act = dog | pred = frog act = frog | pred = bird act = cat

위와 같은 상황(과대 적합; loss = 93%)을 피하기 위하여 드랍 아웃(Drop out)을 사용하기도 합니다.

실제 모델을 사용시에 새로운 이미지를 판별하는 일이 대부분이므로 그다지 좋은 모델이 아닙니다.

모델 만들기

In [5]:
```
input_layer = Input((32,32,3))

x = Conv2D(filters = 32, kernel_size = 3, strides = 1, padding = 'same')(input_layer)
x = BatchNormalization()(x)
x = ReLU()(x)


x = Conv2D(filters = 32, kernel_size = 3, strides = 2, padding = 'same')(x)
x = BatchNormalization()(x)
x = ReLU()(x)


x = Conv2D(filters = 64, kernel_size = 3, strides = 1, padding = 'same')(x)
x = BatchNormalization()(x)
x = ReLU()(x)


x = Conv2D(filters = 64, kernel_size = 3, strides = 2, padding = 'same')(x)
x = BatchNormalization()(x)
x = ReLU()(x)


x = Flatten()(x)

x = Dense(128)(x)
x = BatchNormalization()(x)
x = ReLU()(x)
x = Dropout(rate = 0.5)(x)

x = Dense(NUM_CLASSES)(x)
output_layer = Activation('softmax')(x)

model = Model(input_layer, output_layer)
```

```
WARNING:tensorflow:From C:₩Users₩User₩anaconda3₩envs₩testGAN₩lib₩site-packages₩tensorflow₩python₩framework₩op_def_library.py:263: colocate_
with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From C:₩Users₩User₩anaconda3₩envs₩testGAN₩lib₩site-packages₩keras₩backend₩tensorflow_backend.py:3445: calling dropout (f
rom tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
```

모델 훈련

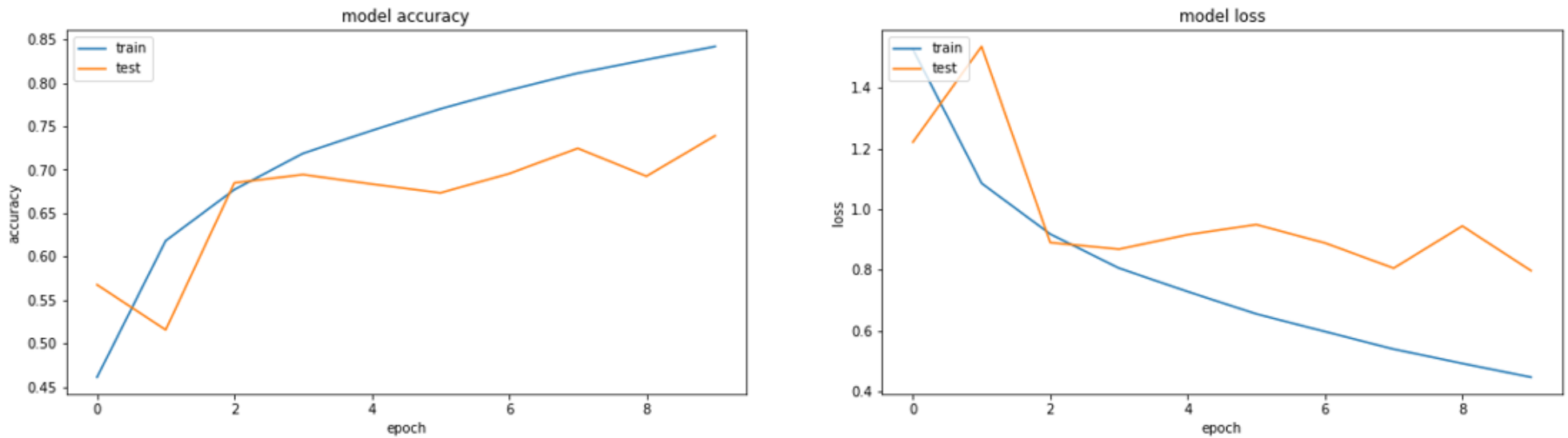
In [7]:
```
opt = Adam(lr=0.0005)
model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
```

In [8]:
```
history = model.fit(x_train
                    , y_train
                    , batch_size=50
                    , epochs=10
                    , shuffle=True
                    , validation_data = (x_test, y_test))
```

```
WARNING:tensorflow:From C:\Users\User\anaconda3\envs\testGAN\lib\site-packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tenso
rflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [==============================] - 636s 13ms/step - loss: 1.5270 - acc: 0.4610 - val_loss: 1.2210 - val_acc: 0.5675
Epoch 2/10
50000/50000 [==============================] - 664s 13ms/step - loss: 1.0854 - acc: 0.6182 - val_loss: 1.5362 - val_acc: 0.5156
Epoch 3/10
50000/50000 [==============================] - 653s 13ms/step - loss: 0.9181 - acc: 0.6771 - val_loss: 0.8900 - val_acc: 0.6848
Epoch 4/10
50000/50000 [==============================] - 839s 17ms/step - loss: 0.8058 - acc: 0.7188 - val_loss: 0.8683 - val_acc: 0.6944
Epoch 5/10
50000/50000 [==============================] - 682s 14ms/step - loss: 0.7285 - acc: 0.7451 - val_loss: 0.9161 - val_acc: 0.6834
Epoch 6/10
50000/50000 [==============================] - 725s 15ms/step - loss: 0.6549 - acc: 0.7700 - val_loss: 0.9491 - val_acc: 0.6733
Epoch 7/10
50000/50000 [==============================] - 641s 13ms/step - loss: 0.5968 - acc: 0.7915 - val_loss: 0.8887 - val_acc: 0.6954
Epoch 8/10
50000/50000 [==============================] - 601s 12ms/step - loss: 0.5388 - acc: 0.8111 - val_loss: 0.8057 - val_acc: 0.7246
Epoch 9/10
50000/50000 [==============================] - 598s 12ms/step - loss: 0.4918 - acc: 0.8266 - val_loss: 0.9445 - val_acc: 0.6925
Epoch 10/10
50000/50000 [==============================] - 603s 12ms/step - loss: 0.4463 - acc: 0.8419 - val_loss: 0.7974 - val_acc: 0.7391
```

모델 성능 그래프 (train accuracy & train loss)

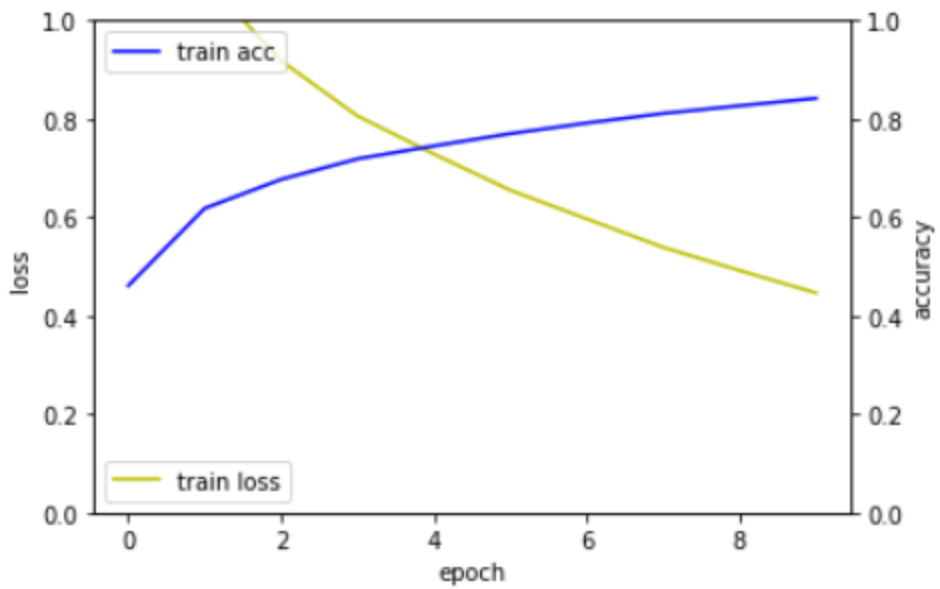
In [9]:
```
plt.figure(figsize=(20, 5))

# summarize history for accuracy
plt.subplot(121)
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')

# summarize history for loss
plt.subplot(122)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')

plt.show()
```



모델 성능 그래프 (train accuracy & train loss)

In [10]:
```
fig, loss_ax = plt.subplots()
acc_ax = loss_ax.twinx()

loss_ax.set_ylim([0.0, 1.0])
acc_ax.set_ylim([0.0, 1.0])

loss_ax.plot(history.history['loss'], 'y', label = 'train loss')
acc_ax.plot(history.history['acc'], 'b', label = 'train acc')

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

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

Out[10]: <matplotlib.legend.Legend at 0x242855ad4a8>



모델 평가

In [11]:
```
model.evaluate(x_test, y_test, batch_size=1000)
```

```
10000/10000 [==============================] - 31s 3ms/step
```

Out[11]: [0.7973899841308594, 0.7390999972820282] acc = 73.9%

In [12]:
```
CLASSES = np.array(['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'])

preds = model.predict(x_test)
preds_single = CLASSES[np.argmax(preds, axis = -1)]
actual_single = CLASSES[np.argmax(y_test, axis = -1)]
```

In [13]:
```
import matplotlib.pyplot as plt

n_to_show = 10
indices = np.random.choice(range(len(x_test)), n_to_show)

fig = plt.figure(figsize=(15, 3))
fig.subplots_adjust(hspace=0.5, wspace=0.5)

for i, idx in enumerate(indices):
    img = x_test[idx]
    ax = fig.add_subplot(1, n_to_show, i+1)
    ax.axis('off')
    ax.text(0.5, -0.35, 'pred = ' + str(preds_single[idx]), fontsize=10, ha='center', transform=ax.transAxes)
    ax.text(0.5, -0.7, 'act = ' + str(actual_single[idx]), fontsize=10, ha='center', transform=ax.transAxes)
    ax.imshow(img)
```

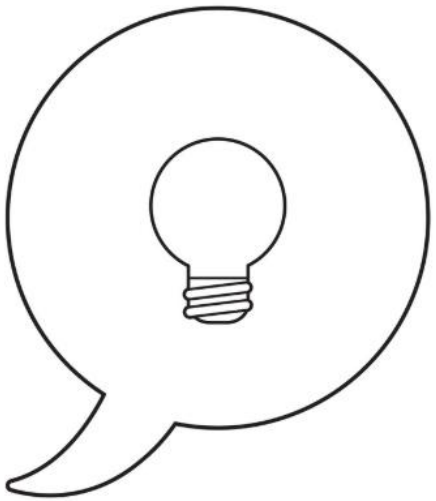
pred = horse	pred = frog	pred = frog	pred = bird	pred = frog	pred = airplane	pred = ship	pred = bird	pred = deer	pred = cat
act = horse	act = frog	act = frog	act = bird	act = frog	act = airplane	act = ship	act = bird	act = deer	act = ship

드랍 아웃(Drop out)을 적용하고 활성화 함수(Activate Function)을 ReLU로 바꾸었습니다.

loss = 79.7% accuracy = 73.9 % 로

과적합도 피할 수 있고 새로운 이미지에 대한 정확도는 73.9%로 여태 실험해 본 모델 중 가장 뛰어납니다.

여러 파라미터를 변경하다 보면 73.9%보다 높아질 지도 모릅니다.



수고하셨습니다.