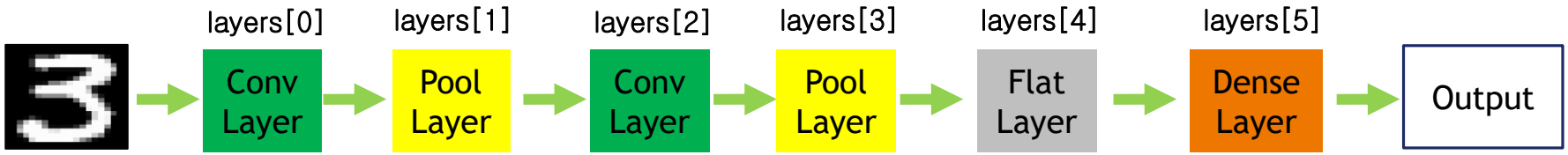
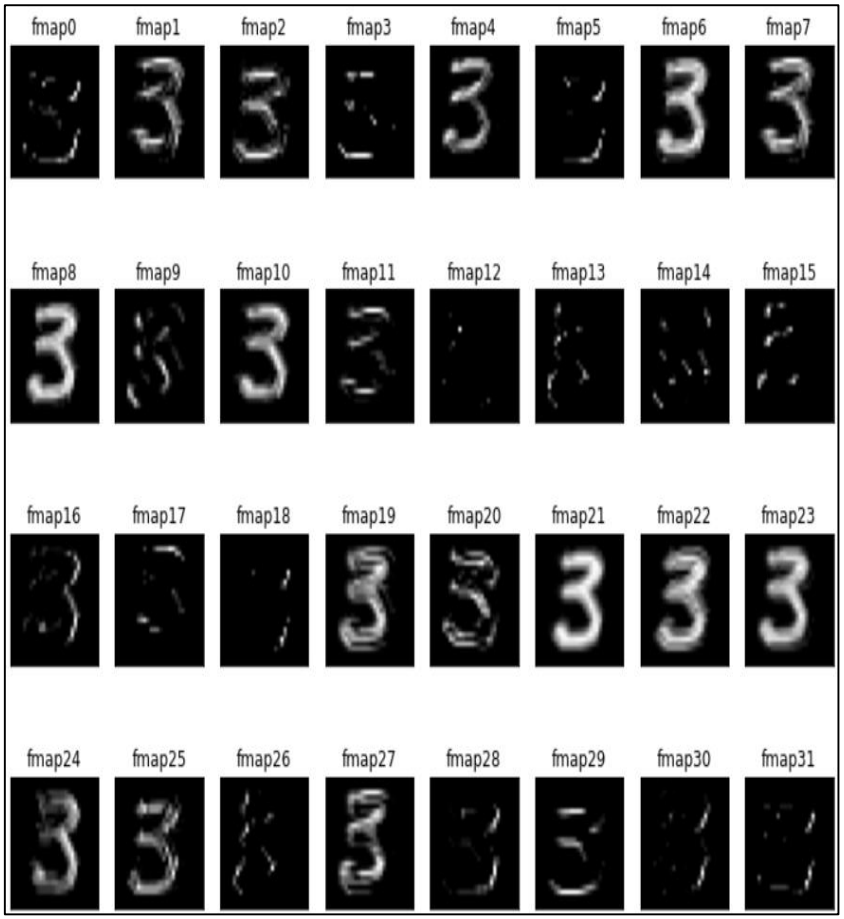
- CNN 특징맵 · 풀링맵 시각화 -

박성호 (neowizard2018@gmail.com)

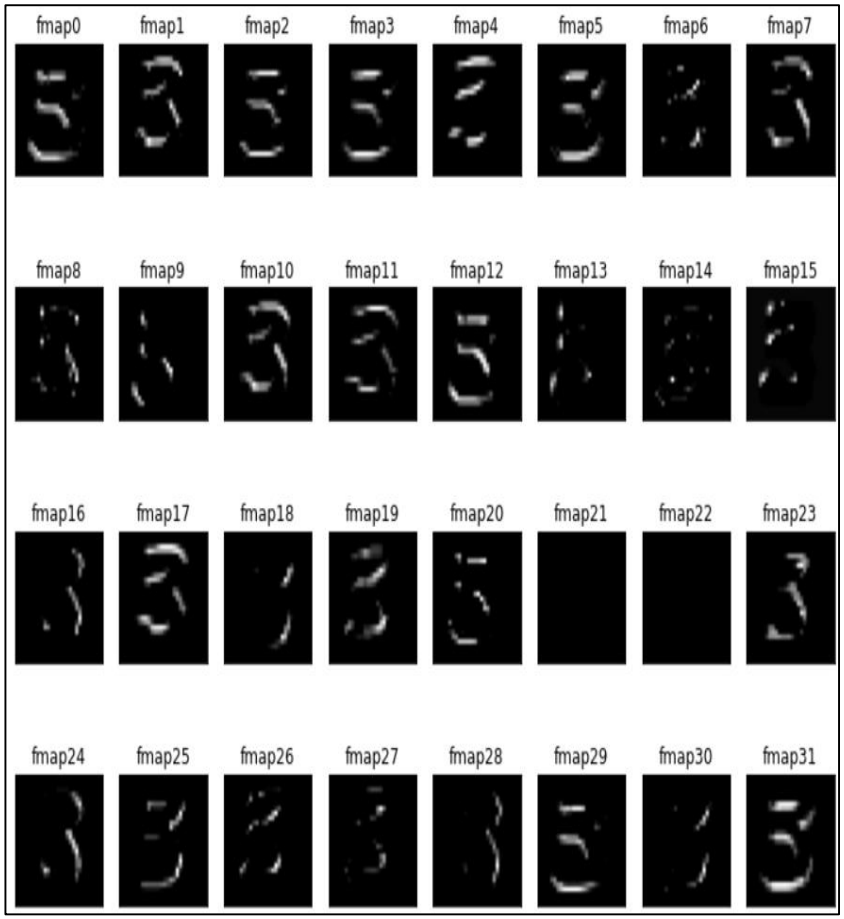
CNN 구조에서 컨볼루션 층의 특징맵(feature map)시각화



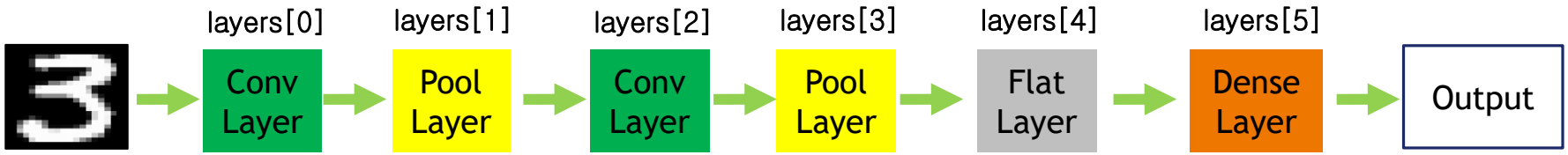
학습전 layer[0] 특징맵 시각화



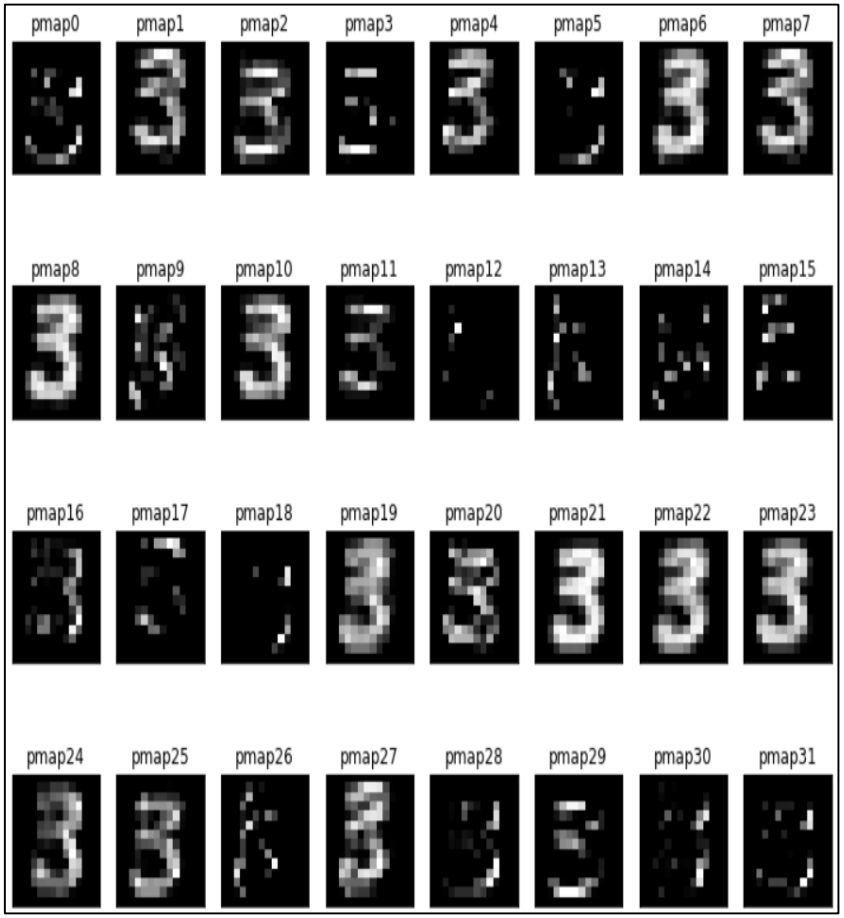
학습후 layer[0] 특징맵 시각화



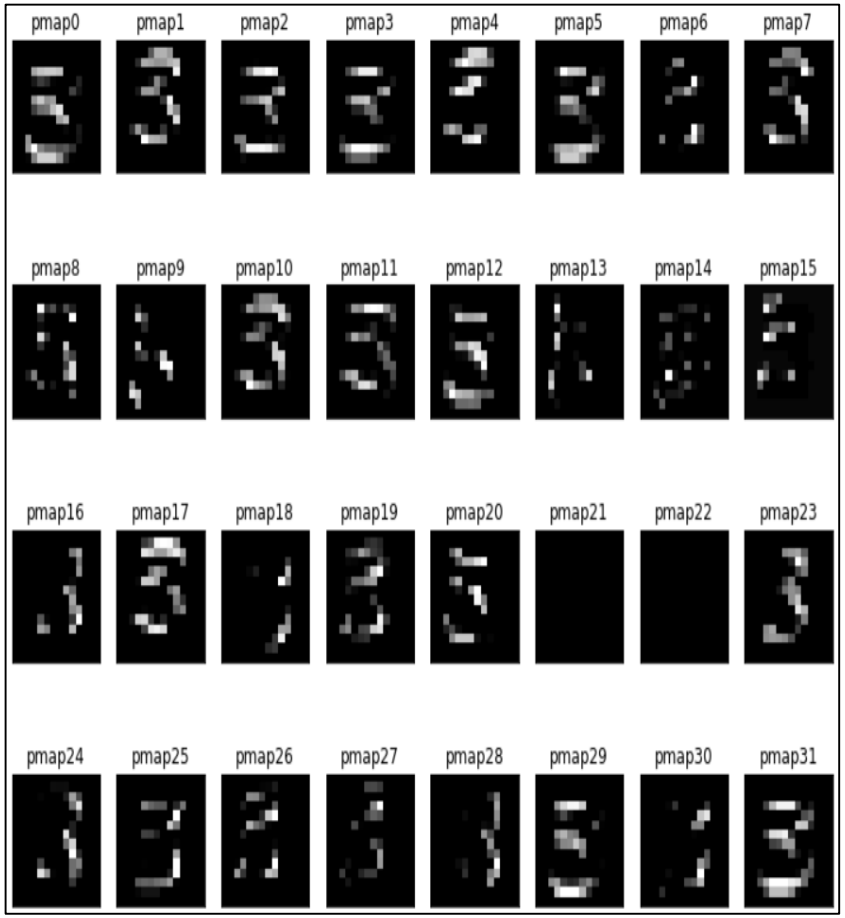
CNN 구조에서 풀링층의 풀링맵(pooling map)시각화



학습전 layer[1] 풀링맵 시각화



학습후 layer[1] 풀링맵 시각화



MNIST Dataset에 대한 CNN 시각화

```
import tensorflow as tf

from tensorflow.keras.layers import Flatten, Dense, Conv2D, MaxPooling2D, Dropout
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.datasets import mnist

import numpy as np
from datetime import datetime
import matplotlib.pyplot as plt
```

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

x_train = x_train / 255.0
x_test = x_test / 255.0

print('x_train.shape = ', x_train.shape, ' , x_test.shape = ', x_test.shape)
print('t_train.shape = ', y_train.shape, ' , t_test.shape = ', y_test.shape)

x_train = x_train.reshape(-1,28,28,1)
x_test = x_test.reshape(-1,28,28,1)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
x_train.shape =  (60000, 28, 28)  , x_test.shape =  (10000, 28, 28)
t_train.shape =  (60000,)  , t_test.shape =  (10000,)
```

MNIST Dataset에 대한 CNN 시각화

MNIST Data를 학습 조건이 다음과 같을 때,

sequential model / functional model 을 각각 구축 하시오

---

feature extractor 조건

1st conv : 3 x 3 x 32 filter, padding='SAME', activation='relu', stride=(1,1)
1st pooling : 2 x 2 maxpooling, padding='SAME'
Dropout(0.25)

2nd conv : 3 x 3 x 64 filter, padding='SAME', activation='relu', stride=(1,1)
2nd pooling : 2 x 2 maxpooling, padding='SAME'
Dropout(0.25)

3rd conv : 3 x 3 x 128 filter, padding='SAME', activation='relu', stride=(1,1)
3rd pooling : 2 x 2 maxpooling, padding='SAME'
Dropout(0.25)

---

compile 조건

optimizer는 Adam(), loss function 은 sparse_categorical_crossentropy

MNIST Dataset에 대한 feature map 시각화

```
for idx in range(len(model.layers)):

    print('model.layers[%d] = %s, %s' % (idx, model.layers[idx].name, model.layers[idx].output.shape))
```

```
model.layers[0] = conv2d, (None, 28, 28, 32)
model.layers[1] = max_pooling2d, (None, 14, 14, 32)
model.layers[2] = dropout, (None, 14, 14, 32)
model.layers[3] = conv2d_1, (None, 14, 14, 64)
model.layers[4] = max_pooling2d_1, (None, 7, 7, 64)
model.layers[5] = dropout_1, (None, 7, 7, 64)
model.layers[6] = conv2d_2, (None, 7, 7, 128)
model.layers[7] = max_pooling2d_2, (None, 4, 4, 128)
model.layers[8] = dropout_2, (None, 4, 4, 128)
model.layers[9] = flatten, (None, 2048)
model.layers[10] = dense, (None, 10)
```

즉 위와 같은 모델이 생성되었을때, 1st conv 층 출력을 가지는 partial_model 을 sequential model 과 functional model을 바탕으로 각각 생성하시오

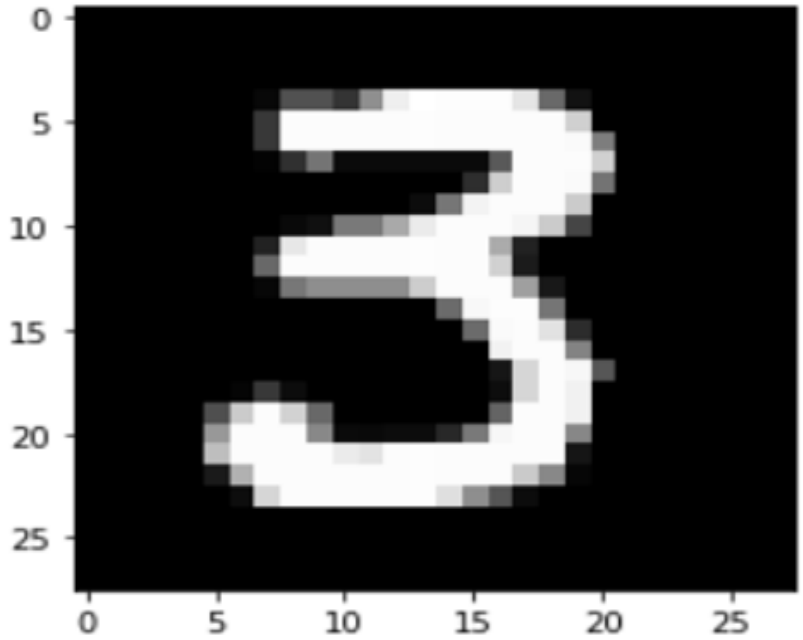
즉 partial_model = Model(inputs=..., outputs=....) 형태로 각각 생성하시오

MNIST Dataset에 대한 CNN 시각화

아래와 같이 임의의 MNIST data 1개를 임의로 선택하시오

```
random_idx = np.random.randint(0, len(x_test))

print(random_idx)
plt.imshow(x_test[random_idx].reshape(28,28), cmap='gray')
```

```
8856
<matplotlib.image.AxesImage at 0x7f395cff4490>
```

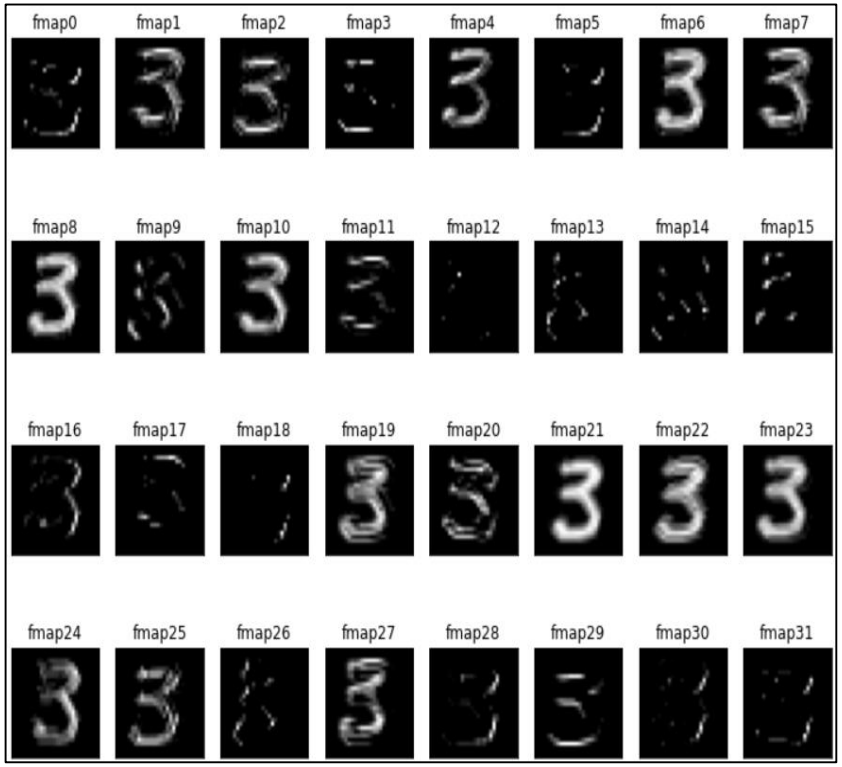


학습 전 layer[0] 특징맵 시각화

```
feature_map = partial_model.predict(x_test[random_idx].reshape(-1,28,28,1))

print(feature_map.shape)

fm = feature_map[0]  # 0번 이미지의 특징 맵을 시각화

print(fm.shape)
```

```
(1, 28, 28, 32)
(28, 28, 32)
```

```
plt.figure(figsize=(10, 8))

for i in range(32): # i번째 특징 맵

    plt.subplot(4,8,i+1)

    plt.imshow(fm[:,:,i], cmap='gray')

    plt.xticks([]); plt.yticks([])
    plt.title("fmap"+str(i))

plt.tight_layout()
plt.show()
```



MNIST Dataset에 대한 pooling map 시각화

```
for idx in range(len(model.layers)):

    print('model.layers[%d] = %s, %s' % (idx, model.layers[idx].name, model.layers[idx].output.shape))
```

```
model.layers[0] = conv2d, (None, 28, 28, 32)
model.layers[1] = max_pooling2d, (None, 14, 14, 32)
model.layers[2] = dropout, (None, 14, 14, 32)
model.layers[3] = conv2d_1, (None, 14, 14, 64)
model.layers[4] = max_pooling2d_1, (None, 7, 7, 64)
model.layers[5] = dropout_1, (None, 7, 7, 64)
model.layers[6] = conv2d_2, (None, 7, 7, 128)
model.layers[7] = max_pooling2d_2, (None, 4, 4, 128)
model.layers[8] = dropout_2, (None, 4, 4, 128)
model.layers[9] = flatten, (None, 2048)
model.layers[10] = dense, (None, 10)
```

즉 위와 같은 모델이 생성되었을때 1st pooling 층 출력을 가지는 partial_model 을 sequential model 과 functional model을 바탕으로 각각 생성하시오

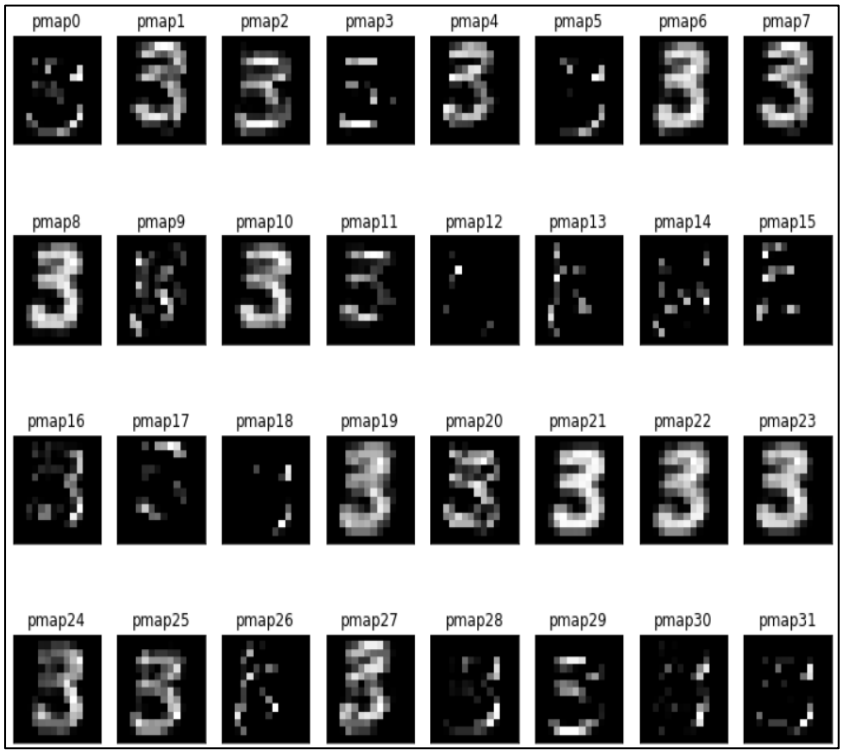
즉 partial_model = Model(inputs=..., outputs=....) 형태로 각각 생성하시오

학습 전 layer[1] 풀링맵 시각화

```
pooling_map = partial_model.predict(x_test[random_idx].reshape(-1,28,28,1))

print(pooling_map.shape)

pm = pooling_map[0]  # 0번 이미지의 풀링 맵을 시각화

print(pm.shape)
```

```
(1, 14, 14, 32)
(14, 14, 32)
```

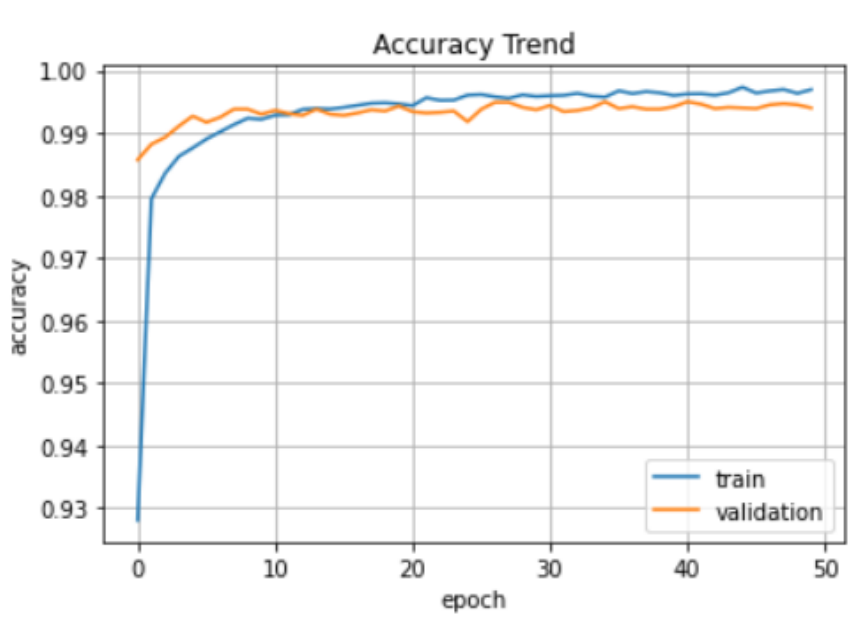
```
plt.figure(figsize=(10, 8))

for i in range(32): # i번째 풀링 맵

    plt.subplot(4,8,i+1)

    plt.imshow(pm[:,:,i], cmap='gray')

    plt.xticks([]); plt.yticks([])
    plt.title("pmap"+str(i))

plt.tight_layout()
plt.show()
```



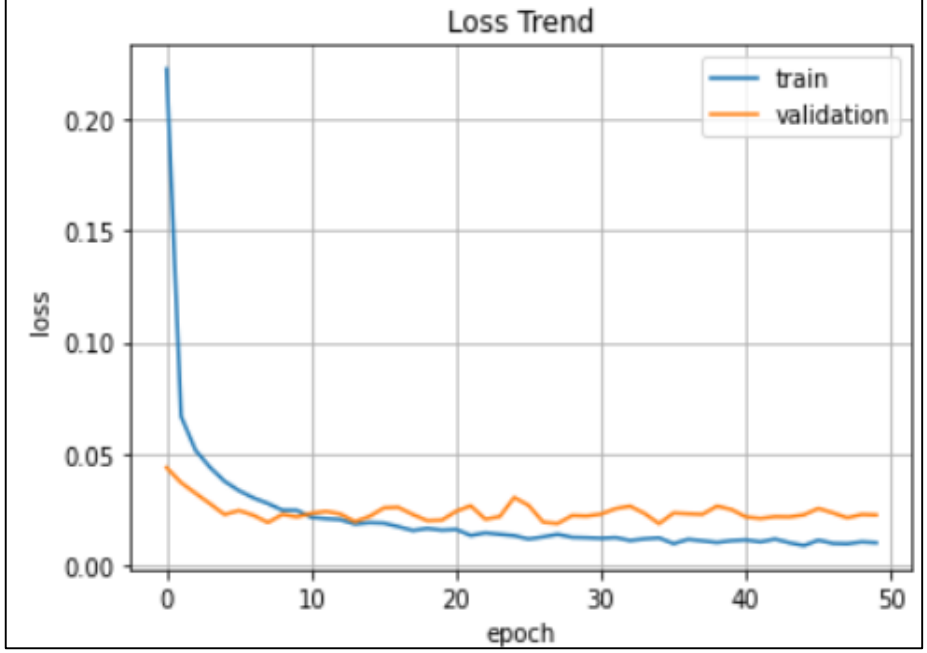
학습 및 오버피팅 확인

```
hist = model.fit(x_train, y_train, batch_size=64, epochs=50, validation_data=(x_test, y_test))
```

```
import matplotlib.pyplot as plt

plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('Accuracy Trend')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','validation'], loc='best')
plt.grid()
plt.show()
```

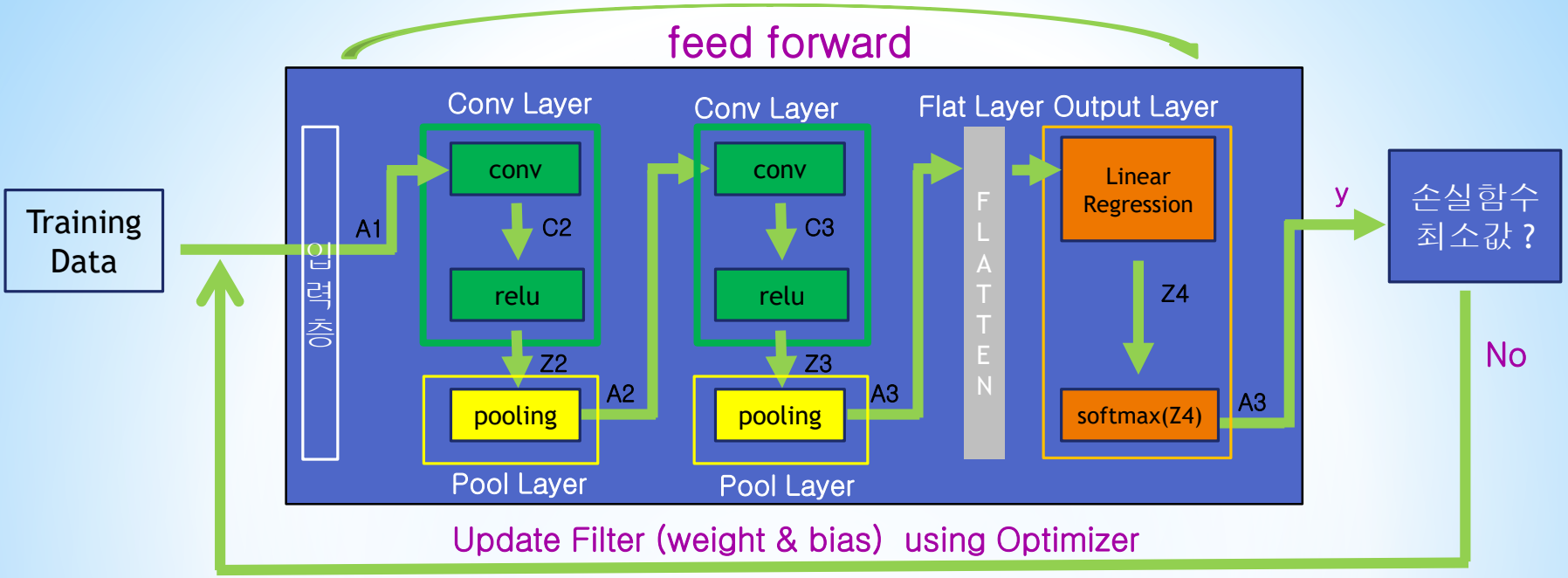



```
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Loss Trend')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','validation'], loc='best')
plt.grid()
plt.show()
```



[Self Study 1] 학습을 모두 마친 후, 1st feature map, 1st pooling map 시각화 하는 코드를 작성하시오

[Self Study 2] 학습을 모두 마친 후, 2nd feature map, 2nd pooling map 시각화 하는 코드를 작성하시오

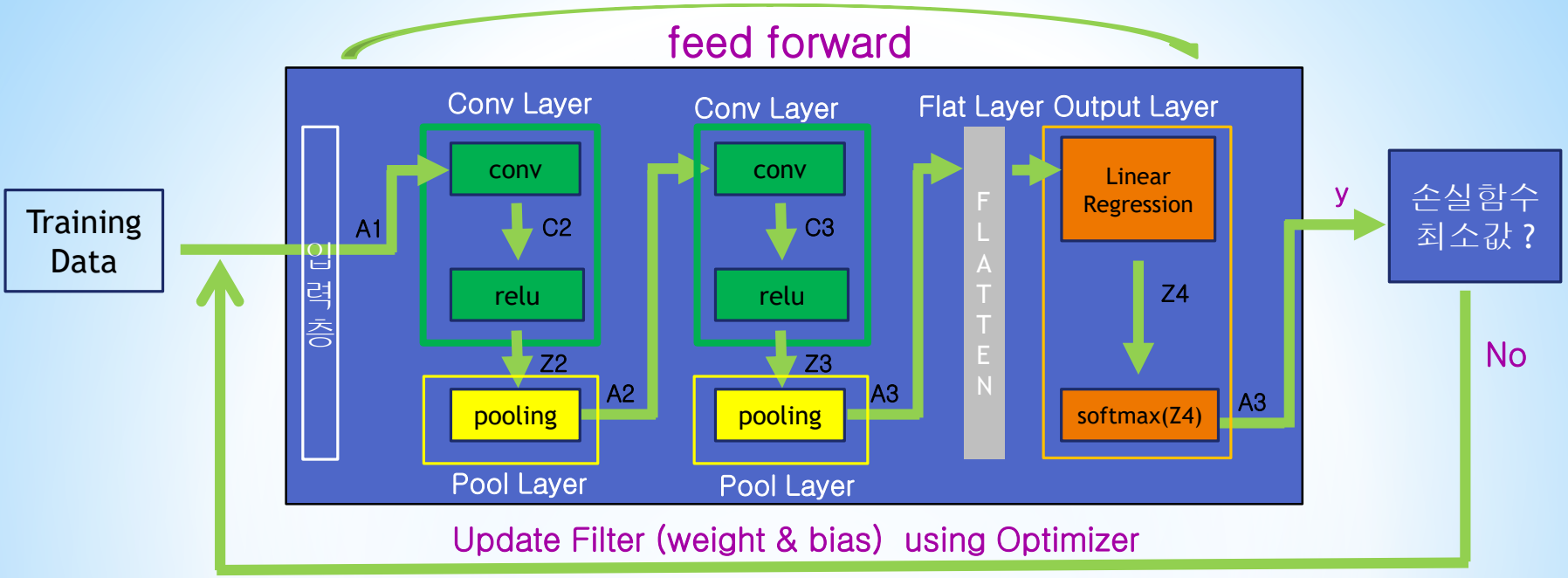


[예제 4_1] 2 conv / 1 flatten CNN 구조를 이용하여 index_label_prediction 값을 출력하는 코드를 구현하시오. 이때 model.summary() 에서 출력되는 전체 파리미터 개수를 검증하시오

[예시]

1st conv => 3 x 3 크기의 32 개 필터, padding 있음, 2 x 2 max pooling, 1stride

2nd conv => 4 x 4 크기의 64 개 필터, padding 있음, 2 x 2 max pooling, 1stride



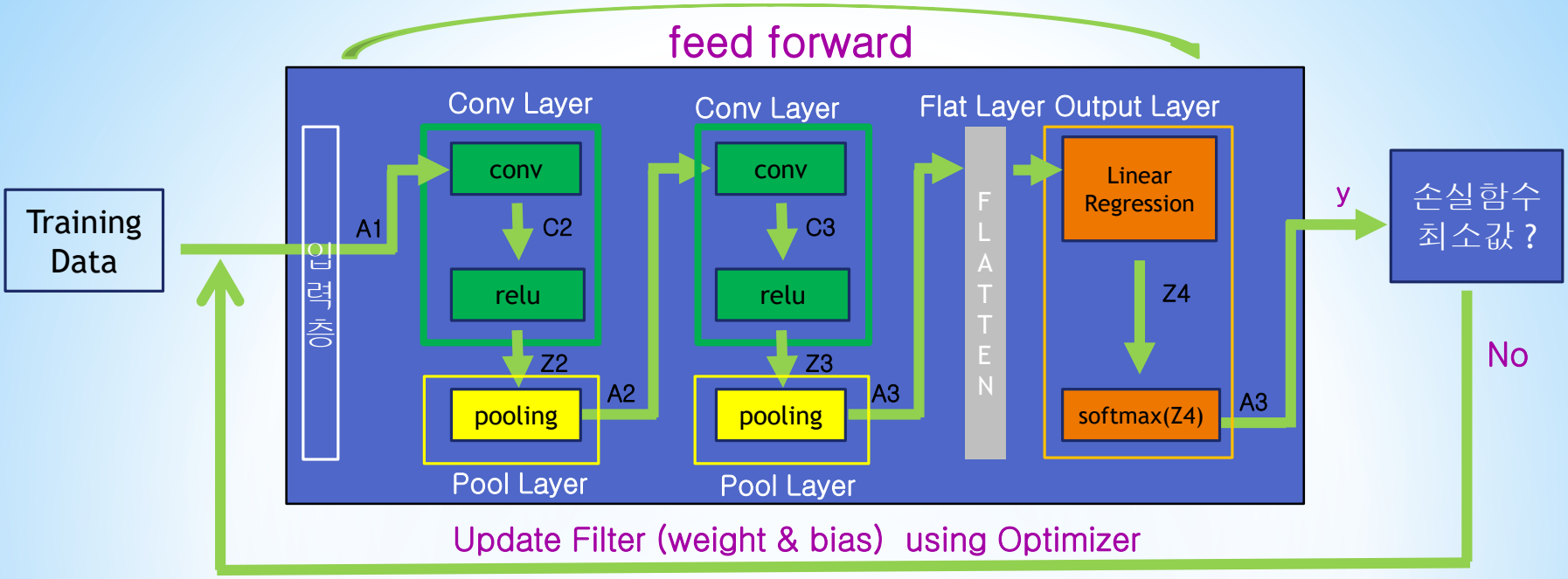
[예제 4_2] 다음과 같은 2 conv / 1 flatten CNN 구조에서 학습대상의 전체 파라미터를 계산하고 오버피팅을 확인할수 있는 그래프를 출력하시오

1st conv layer

⇒ 5 x 5 크기의 32개 필터, 1 stride, 2 x 2 max pooling, padding 있음

2nd conv layer

⇒ 5 x 5 크기의 64개 필터, 1 stride, 2 x 2 max pooling, padding 있음



[예제 4_3] 다음과 같은 2 conv / 1 flatten CNN 구조에서 Dropout 층을 추가하고 validation_split = 0.2 부분을 validation_data=(x_test, t_test) 바꾼 후 오버피팅을 확인할수 있는 그래프를 출력하시오 (여기서 Dropout(0.25) 1개일때와 2개일때 오버피팅을 확인하고, Dropout(0.5) 인 경우도 확인하시오)

1st conv layer

⇒ 5 x 5 크기의 32개 필터, 1 stride, 2 x 2 max pooling, padding 있음

2nd conv layer

⇒ 5 x 5 크기의 64개 필터, 1 stride, 2 x 2 max pooling, padding 있음

[예제 4_6]

아래와 같은 조건으로 3 conv/ 1 flatten CNN 구조를 이용하여 MNIST 정확도를 검증한 후, 전체 파라미터(parameter) 개수를 계산하고 오버피팅을 확인하는그래프 출력 하시오 (예제4_6은 Dropout 없음, 예제4_7 는 Dropout(0.25)각각 추가, 예제4_8은 Dropout(0.5) 각각 추가)

1stconv=> 3 x 3 필터32개, stride 1, convpadding 있음, 2 x 2 max pooling

2ndconv=> 3 x 3 필터64개, stride 1, convpadding 있음, 2 x 2 max pooling

3rdconv=> 3 x 3 필터128개, stride 1, convpadding 있음, 2 x 2 max pooling

[예제 5]

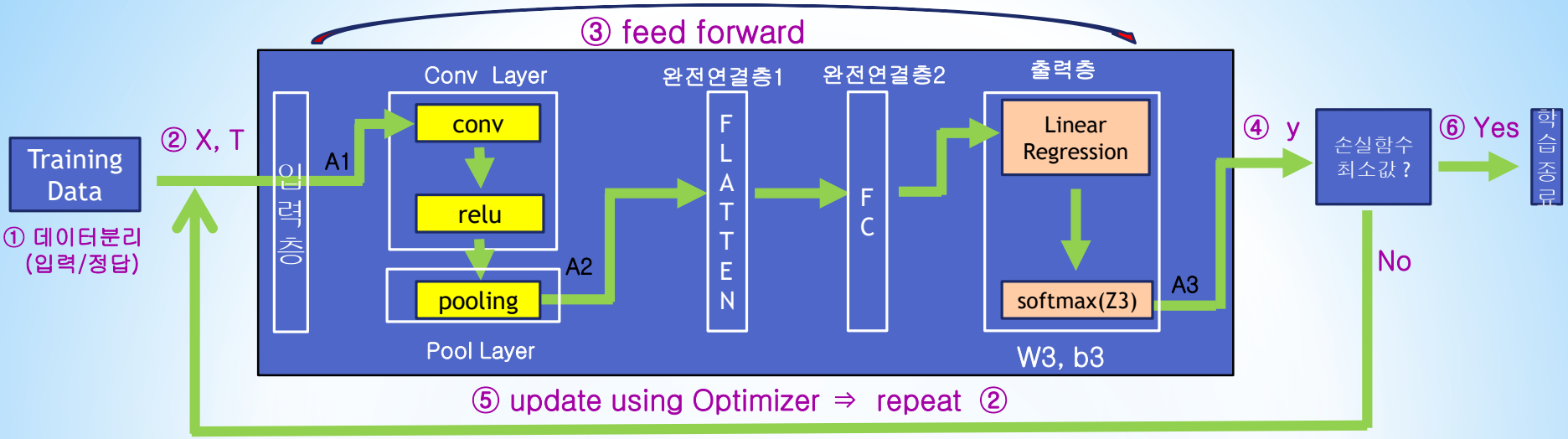
MNIST 데이터에 대해서 , 아래와 같은 CCPCCP 아키텍처로 오버피팅이 거의 발생하지 않고 94% 이상의 정확도가 나오는지 확인하시오 (Dropout 레이어를 적절히 추가하시오)

1st conv => 3 x 3 필터 32개, stride 1, conv padding 있음, 2 x 2 max pooling

2nd conv => 3 x 3 필터 32개, stride 1, conv padding 있음, 2 x 2 max pooling

3rd conv => 3 x 3 필터 64개, stride 1, conv padding 있음, 2 x 2 max pooling

4th conv => 3 x 3 필터 64개, stride 1, conv padding 있음, 2 x 2 max pooling



[예제 6_1]

1conv / 2 FC CNN 구조를 이용하여 정확도를 구하고 오버피팅 발생여부를 그래프로 확인하시오

[예제 6_2]

2 or 3 conv / 2 FC CNN 구조를 이용하여 파라미터 수를 계산하고 오버피팅 발생 여부를 그래프로 확인하시오

CIAFR10 Project

– 다양한 이미지 예측 및 개선점 찾기 –

박성호 (neowizard2018@gmail.com)

[project_2] cifar 10 분류 가능한 실제 이미지 예측

[1] Project1 에서 저장된 80% 이상의 정확도 가진 CNN 모델 load

- 모델 로드 방법 예시

  cnn = tensorflow.keras.models.load_model('./cifar10_accuracy_80.h5')

  cnn.summary()

---

[2] 테스트 이미지 데이터 구한 후 불러오기

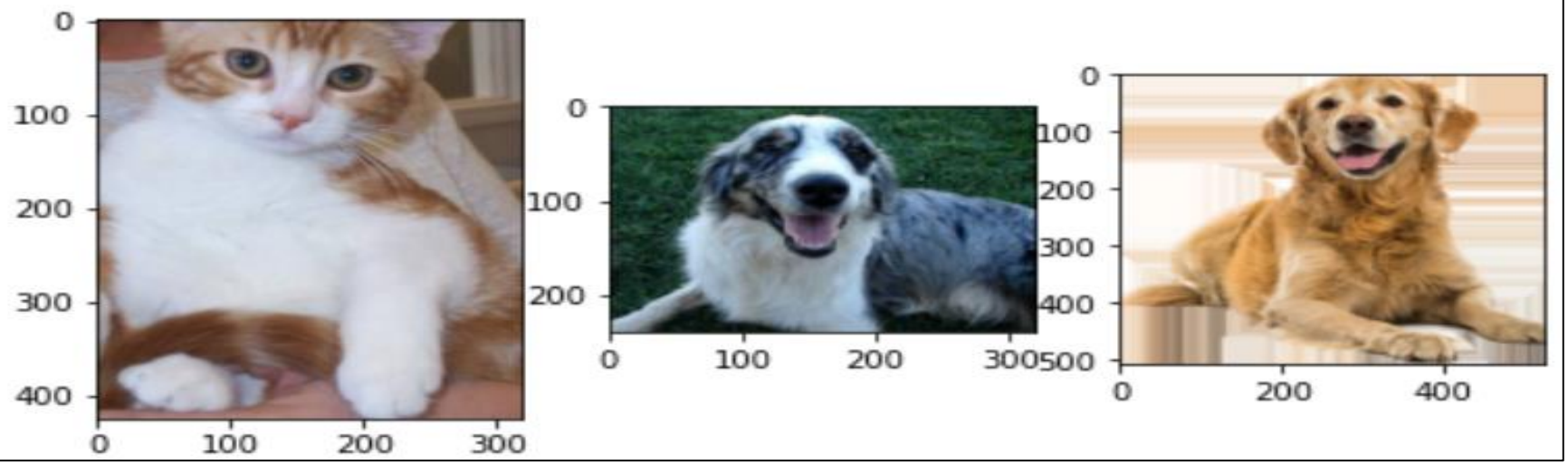
- 테스트에 사용할 이미지는 자체적으로 구한 후 이미지 로드함

```
import cv2

src_img1 = cv2.imread('./cat.9.jpg', cv2.IMREAD_COLOR)
src_img2 = cv2.imread('./dog.6331.jpg', cv2.IMREAD_COLOR)
src_img3 = cv2.imread('./dog.1.png', cv2.IMREAD_COLOR)

dst_img1 = cv2.cvtColor(src_img1, cv2.COLOR_BGR2RGB)
dst_img2 = cv2.cvtColor(src_img2, cv2.COLOR_BGR2RGB)
dst_img3 = cv2.cvtColor(src_img3, cv2.COLOR_BGR2RGB)
```

[project_2] cifar 10 분류 가능한 실제 이미지 예측

[3] 테스트 이미지 확인 (예시)

```
import matplotlib.pyplot as plt

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

plt.subplot(1,3,1)
plt.imshow(dst_img1)
plt.subplot(1,3,2)
plt.imshow(dst_img2)
plt.subplot(1,3,3)
plt.imshow(dst_img3)

plt.show()
```



[project_2] cifar 10 분류 가능한 실제 이미지 예측

[4] 테스트 이미지 데이터 전처리 (resize, normalization, etc)

- cifar10 이미지 사이즈에 맞게 cv2.resize() 이용하여 크기 변경
- 이미지 이므로 0~1 값을 가지도록 정규화 수행

---

[5] predict() 실행을 위한 데이터 shape 변경

- 예측을 하기위해 cnn.predict(test_data) 실행. 이때 입력으로 주어지는
  test_data.shape = (batch_size, width, height, channel) 형태의 4차원 텐서로 구성

---

[6] predict() 결과를 해석하고 정확도 검증

- 예측 pred = cnn.predict(test_data) 실행 후에, 리턴값 pred.shape 확인
  가장 높은 확률의 예측 값 1 개 확인 (np.argmax() 및 np.max() 활용)
  Top 3 예측 결과와 해당 확률 확인 (np.argsort() 활용, 검색활용해 사용법 숙지)

CIAFR10 Project

– 다양한 이미지 예측 및 개선점 찾기 –

박성호 (neowizard2018@gmail.com)

[project_3] cifar 10 분류 가능한 실제 이미지 예측

[1] project_1 에서 저장된 80% 이상의 정확도 가진 CNN 모델 load

- 모델 로드 방법 예시

```
cnn = tensorflow.keras.models.load_model('./cifar10_accuracy_80.h5')
cnn.summary()
```

---

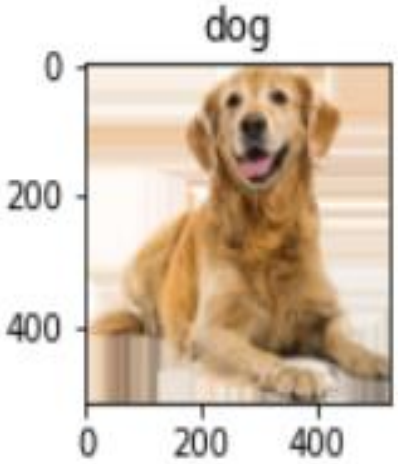
[2] 테스트 이미지 데이터 구한 후 불러오기 (이미지 파일에 정답이 나타나도록 저장)

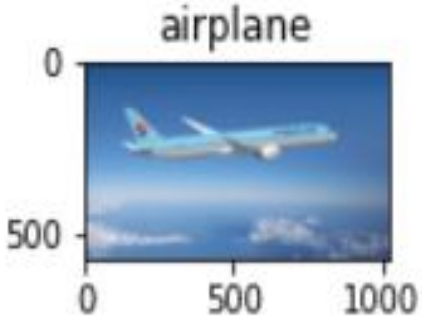
- 테스트에 사용될 이미지 이름은 cat.6.jpg 같이 정답을 나타내는 단어가 가장 먼저 오도록 변경 한 후에, my_test_image 디렉토리에 저장해 놓는다

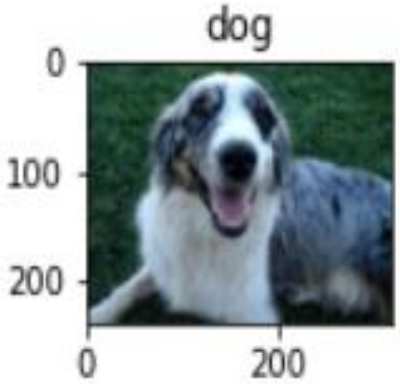
- glob.glob(...) 이용하여 my_test_image 내의 이미지를 읽어서 정답을 추출.
  즉 이미지 파일 이름이 cat.6.jpg 일때 정답으로 cat 추출하여 정답 리스트 구축함

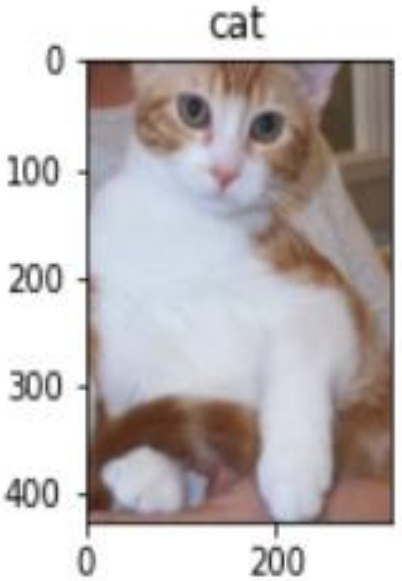
[project_3] cifar 10 분류 가능한 실제 이미지 예측

[3] 테스트 이미지 확인 (정답과 함께 이미지 출력)









[project_3] cifar 10 분류 가능한 실제 이미지 예측

[4] 테스트 이미지 데이터 전처리 (resize, normalization, etc)

- cifar10 이미지 사이즈에 맞게 cv2.resize() 이용하여 크기 변경
- 이미지 이므로 0~1 값을 가지도록 정규화 수행

---

[5] predict() 실행을 위한 데이터 shape 변경

- 예측을 하기위해 cnn.predict(test_data) 실행. 이때 입력으로 주어지는
  test_data.shape = (batch_size, width, height, channel) 형태의 4차원 텐서로 구성

---

[6] predict() 결과를 해석하고 정확도 검증

- 예측 pred = cnn.predict(test_data) 실행 후에, 리턴값 pred.shape 확인
  가장 높은 확률의 예측 값 1 개 확인 (np.argmax() 및 np.max() 활용)
  Top 3 예측 결과와 해당 확률 확인 (np.argsort() 활용, 검색활용해 사용법 숙지)