**Deep Learning 수행평가 – 김현석**

**1. Fashion-MNIST**

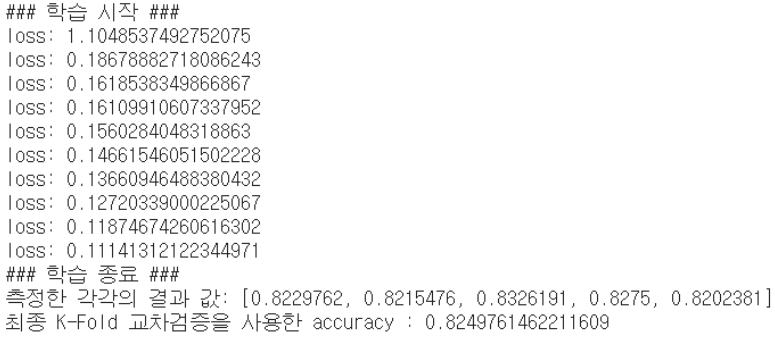
- 데이터 전처리

|  |
| --- |
| import numpy as np  import pandas as pd  import tensorflow as tf  import matplotlib.pyplot as plt  from sklearn.preprocessing import MinMaxScaler  from sklearn.model\_selection import train\_test\_split  from sklearn.model\_selection import KFold  from sklearn.metrics import classification\_report  # tf.reset\_default\_graph()  ## Raw Data Loading  df = pd.read\_csv('/content/drive/MyDrive/machine learning colab/archive/fashion-mnist\_train.csv')  display(df.head(), df.shape)  ## 데이터 확인  img\_data = df.drop('label', axis=1, inplace=False).values  fig = plt.figure()  fig\_arr = list()  for n in range(10):  fig\_arr.append(fig.add\_subplot(2,5,n+1))  fig\_arr[n].imshow(img\_data[n].reshape(28,28), cmap='gray')    plt.tight\_layout()  plt.show()  ## Data Split  x\_data\_train, x\_data\_test, t\_data\_train, t\_data\_test = \  train\_test\_split(df.drop('label', axis=1, inplace=False),  df['label'],  test\_size=0.3,  random\_state=0)  ## Normalization  scaler = MinMaxScaler()  scaler.fit(x\_data\_train)  x\_data\_train\_norm = scaler.transform(x\_data\_train)  x\_data\_test\_norm = scaler.transform(x\_data\_test)  ## one-hot encoding  sess = tf.Session()  t\_data\_train\_onehot = sess.run(tf.one\_hot(t\_data\_train, depth=10))  t\_data\_test\_onehot = sess.run(tf.one\_hot(t\_data\_test, depth=10)) |



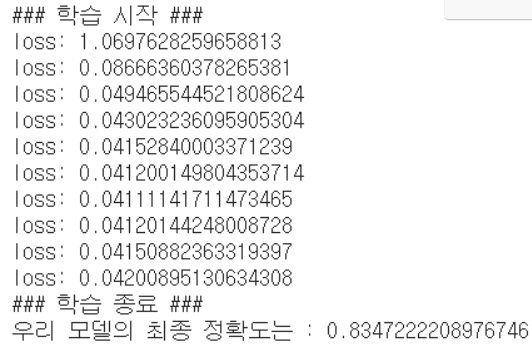
**1) Multinomial Classification 구현 후 정확도 측정**

|  |
| --- |
| # tensorflow 구현  ## placeholder  X = tf.placeholder(shape=[None,784], dtype=tf.float32)  T = tf.placeholder(shape=[None,10], dtype=tf.float32)  ## Weight & bias  W = tf.Variable(tf.random.normal([784,10]), name='weight')  b = tf.Variable(tf.random.normal([10]), name='bias')  ## Hypothesis  logit = tf.matmul(X,W) + b  H = tf.nn.softmax(logit)  ## loss function  loss = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits\_v2(logits=logit,  labels=T))  ## Train  train = tf.train.GradientDescentOptimizer(learning\_rate=1e-1).minimize(loss)  ## parameter  num\_of\_epoch = 100  batch\_size = 10  ## 학습용 함수  def run\_train(sess, train\_x, train\_t):  print('### 학습 시작 ###')  sess.run(tf.global\_variables\_initializer())  for step in range(num\_of\_epoch):  total\_batch = int(train\_x.shape[0] / batch\_size)  for i in range(total\_batch):  batch\_x = train\_x[i\*batch\_size:(i+1)\*batch\_size]  batch\_t = train\_t[i\*batch\_size:(i+1)\*batch\_size]  \_,loss\_val = sess.run([train,loss],  feed\_dict={X:batch\_x,  T:batch\_t})    if step % 10 == 0:  print(f'loss: {loss\_val}')  print('### 학습 종료 ###')  ## Accuracy 측정  predict = tf.argmax(H,1)  correct = tf.equal(predict, tf.argmax(T,1))  accuracy = tf.reduce\_mean(tf.cast(correct, dtype=tf.float32))  # run\_train(sess, x\_data\_train\_norm, t\_data\_train\_onehot)  ## K-Fold Cross validation  cv = 5  results = []  kf = KFold(n\_splits=cv, shuffle=True)  for training\_idx, validation\_idx in kf.split(x\_data\_train\_norm):  training\_x = x\_data\_train\_norm[training\_idx]  training\_t = t\_data\_train\_onehot[training\_idx]  val\_x = x\_data\_train\_norm[validation\_idx]  val\_t = t\_data\_train\_onehot[validation\_idx]  run\_train(sess, training\_x, training\_t)  results.append(sess.run(accuracy, feed\_dict={X:val\_x,  T:val\_t}))    print(f'측정한 각각의 결과 값: {results}')  print(f'최종 K-Fold 교차검증을 사용한 accuracy : {np.mean(results)}') |



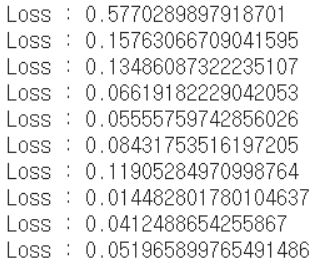
- 예측

|  |
| --- |
| # testing  run\_train(sess, x\_data\_train\_norm, t\_data\_train\_onehot)  final\_accurancy = sess.run(accuracy, feed\_dict={X:x\_data\_test\_norm,  T:t\_data\_test\_onehot})  print(f'우리 모델의 최종 정확도는 : {final\_accurancy}') |



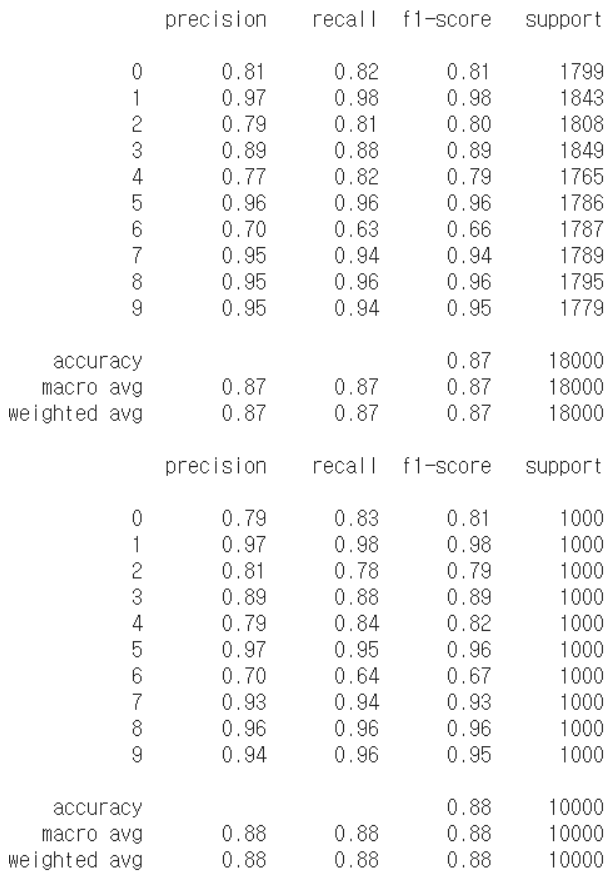
**2) DNN으로 구현**

|  |
| --- |
| # DNN 구현  # Placeholder  X = tf.placeholder(shape=[None,784], dtype=tf.float32)  T = tf.placeholder(shape=[None,10], dtype=tf.float32)  # Weight & bias  W2 = tf.get\_variable('weight2', shape=[784,256],  initializer=tf.contrib.layers.variance\_scaling\_initializer())  b2 = tf.Variable(tf.random.normal([256]))  \_layer2 = tf.nn.relu(tf.matmul(X,W2) + b2)  layer2 = tf.nn.dropout(\_layer2, rate=0.4)  W3 = tf.get\_variable('weight3', shape=[256,128],  initializer=tf.contrib.layers.variance\_scaling\_initializer())  b3 = tf.Variable(tf.random.normal([128]))  \_layer3 = tf.nn.relu(tf.matmul(layer2,W3) + b3)  layer3 = tf.nn.dropout(\_layer3, rate=0.4)  W4 = tf.get\_variable('weight4', shape=[128,10],  initializer=tf.contrib.layers.variance\_scaling\_initializer())  b4 = tf.Variable(tf.random.normal([10]))  # Hypothesis  logit = tf.matmul(layer3,W4) + b4  H = tf.nn.softmax(logit)  # loss  loss = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits\_v2(logits=logit,  labels=T))  # train  train = tf.train.AdamOptimizer(learning\_rate=1e-3).minimize(loss)  num\_of\_epoch = 1000  batch\_size = 100  # session & 초기화  sess = tf.Session()  sess.run(tf.global\_variables\_initializer())  # 반복학습  for step in range(num\_of\_epoch):    total\_batch = int(x\_data\_train\_norm.shape[0] / batch\_size)    for i in range(total\_batch):  batch\_x = x\_data\_train\_norm[i\*batch\_size:(i+1)\*batch\_size]  batch\_t = t\_data\_train\_onehot[i\*batch\_size:(i+1)\*batch\_size]  \_, loss\_val = sess.run([train, loss], feed\_dict={X:batch\_x,  T:batch\_t})  if step % 100 == 0:  print('Loss : {}'.format(loss\_val)) |



- 예측

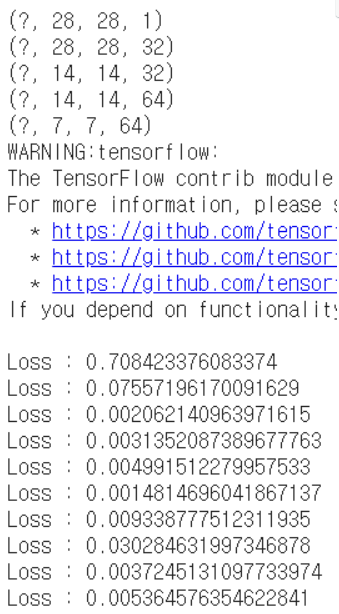
|  |
| --- |
| # model의 정확도 구학기  predict = tf.argmax(H, 1)  print(classification\_report(t\_data\_test,  sess.run(predict, feed\_dict={X:x\_data\_test\_norm})))  # test 데이터로 최종 결과 출력  final\_df = pd.read\_csv('/content/drive/MyDrive/Machine Learning Colab/Fashion-MNIST/fashion-mnist\_test.csv')  final\_x\_data = final\_df.drop('label', axis=1, inplace=False).values  final\_t\_data = final\_df['label'].values  final\_scaler = MinMaxScaler()  final\_scaler.fit(final\_x\_data)  final\_x\_data\_norm = final\_scaler.transform(final\_x\_data)  print(classification\_report(final\_t\_data,  sess.run(predict, feed\_dict={X:final\_x\_data\_norm}))) |



모델의 최종 정확도는 0.88

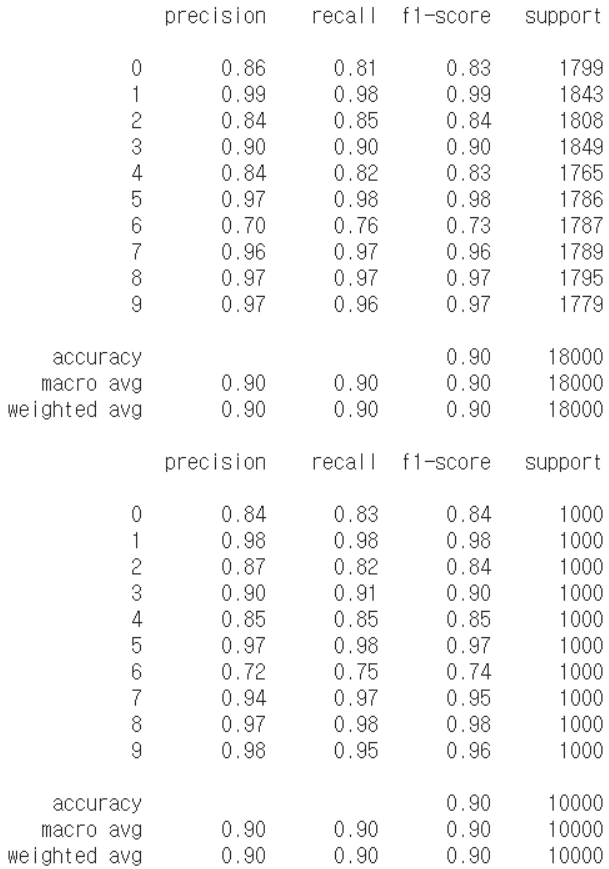
**3) CNN으로 구현**

|  |
| --- |
| # CNN 구현  # Placeholder  X = tf.placeholder(shape=[None,784], dtype=tf.float32)  T = tf.placeholder(shape=[None,10], dtype=tf.float32)  # Convolution  # 입력데이터의 형태부터 수정해야 해요! 4차원으로.  # (이미지 개수, 높이, 너비, channel)  x\_img = tf.reshape(X, [-1,28,28,1])  print(x\_img.shape) # (?, 28, 28, 1)  # filter(kernel) => (filter height, filter width, filter channel, filter 개수)  W1 = tf.Variable(tf.random.normal([3,3,1,32]))  L1 = tf.nn.conv2d(x\_img,  W1,  strides=[1,1,1,1],  padding='SAME')  L1 = tf.nn.relu(L1)  print(L1.shape) # (?, 28, 28, 32)  P1 = tf.nn.max\_pool(L1,  ksize=[1,2,2,1],  strides=[1,2,2,1],  padding='SAME')  print(P1.shape) # (?, 14, 14, 32)  W2 = tf.Variable(tf.random.normal([3,3,32,64]))  L2 = tf.nn.conv2d(P1,  W2,  strides=[1,1,1,1],  padding='SAME')  L2 = tf.nn.relu(L2)  print(L2.shape) # (?, 14, 14, 64)  P2 = tf.nn.max\_pool(L2,  ksize=[1,2,2,1],  strides=[1,2,2,1],  padding='SAME')  print(P2.shape) # (?, 7, 7, 64)  # P2의 shape은 4차원  P2 = tf.reshape(P2, [-1,7\*7\*64])  # Weight & bias  W3 = tf.get\_variable('weight3', shape=[7\*7\*64,256],  initializer=tf.contrib.layers.variance\_scaling\_initializer())  b3 = tf.Variable(tf.random.normal([256]))  \_layer3 = tf.nn.relu(tf.matmul(P2,W3) + b3)  layer3 = tf.nn.dropout(\_layer3, rate=0.4)  W4 = tf.get\_variable('weight4', shape=[256,10],  initializer=tf.contrib.layers.variance\_scaling\_initializer())  b4 = tf.Variable(tf.random.normal([10]))  # Hypothesis  logit = tf.matmul(layer3,W4) + b4  H = tf.nn.softmax(logit) # softmax activation function  # # loss  loss = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits\_v2(logits=logit,  labels=T))  # # train  train = tf.train.AdamOptimizer(learning\_rate=1e-3).minimize(loss)  num\_of\_epoch = 1000  batch\_size = 100  # session & 초기화  sess = tf.Session()  sess.run(tf.global\_variables\_initializer())  # 반복학습  for step in range(num\_of\_epoch):    total\_batch = int(x\_data\_train\_norm.shape[0] / batch\_size)    for i in range(total\_batch):  batch\_x = x\_data\_train\_norm[i\*batch\_size:(i+1)\*batch\_size]  batch\_t = t\_data\_train\_onehot[i\*batch\_size:(i+1)\*batch\_size]  \_, loss\_val = sess.run([train, loss], feed\_dict={X:batch\_x,  T:batch\_t})  if step % 100 == 0:  print('Loss : {}'.format(loss\_val)) |



- 예측

|  |
| --- |
| # model의 정확도 구학기  predict = tf.argmax(H, 1)  print(classification\_report(t\_data\_test,  sess.run(predict, feed\_dict={X:x\_data\_test\_norm})))  # test 데이터로 최종 결과 출력  final\_df = pd.read\_csv('/content/drive/MyDrive/machine learning colab/archive/fashion-mnist\_test.csv')  final\_x\_data = final\_df.drop('label', axis=1, inplace=False).values  final\_t\_data = final\_df['label'].values  final\_scaler = MinMaxScaler()  final\_scaler.fit(final\_x\_data)  final\_x\_data\_norm = final\_scaler.transform(final\_x\_data)  print(classification\_report(final\_t\_data,  sess.run(predict, feed\_dict={X:final\_x\_data\_norm}))) |



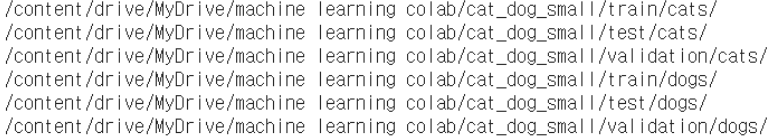
모델의 최종 정확도는 0.90, DNN 사용보다 0.03 오름

**2. Dogs vs. Cats 분류문제**

**1) Dogs vs. Cats 분류문제 TF2.x로 구현 후 정확도 측정**

- 데이터 전처리

|  |
| --- |
| import os  import numpy as np  import matplotlib.pyplot as plt  import pandas as pd  import keras  os.listdir()  # loading img  import PIL.Image as Image  p = '/content/drive/MyDrive/machine learning colab/cat\_dog\_small'  os.chdir(p)  def resize\_img(img,size):  return img.resize(size)  def load\_img(file\_path):  data = []  print(p + file\_path[1:] + '/')  for f in os.listdir(file\_path):  data.append(resize\_img(Image.open(p + file\_path[1:] + '/' + f) , (64,64)))  return data  train\_cats = load\_img('./train/cats') # img\_data\_list / element is image not nparray  test\_cats = load\_img('./test/cats')  val\_cats = load\_img('./validation/cats')  train\_dogs = load\_img('./train/dogs')  test\_dogs = load\_img('./test/dogs')  val\_dogs = load\_img('./validation/dogs') |

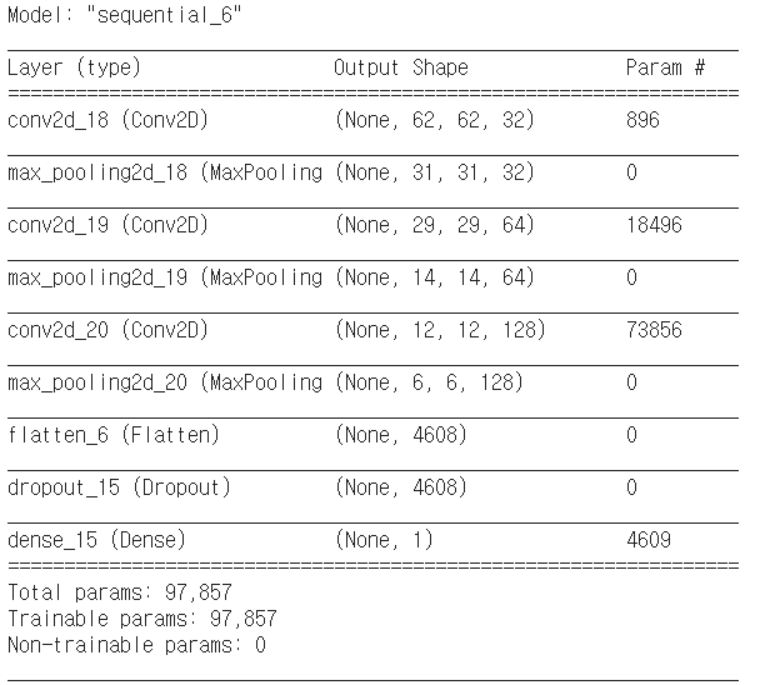


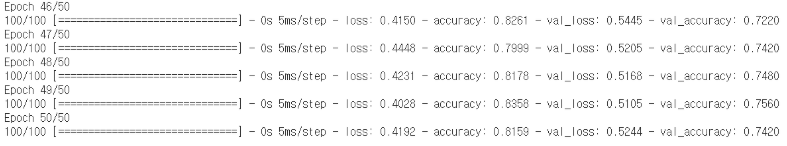
|  |
| --- |
| print(len(train\_cats),len(test\_cats),len(val\_cats),len(train\_dogs),len(test\_dogs),len(val\_dogs))  # 결과 값  # (1000 500 500 1000 500 500) |

|  |
| --- |
| # We have to img -> array & labeling solution  def img\_to\_array(img):  return np.array(img, dtype = 'float32')/255.0  train\_dogs\_arr,train\_dogs\_sol = np.array([img\_to\_array(dog) for dog in train\_dogs]), np.array([1] \* len(train\_dogs))  test\_dogs\_arr,test\_dogs\_sol = np.array([img\_to\_array(dog) for dog in test\_dogs]), np.array([1] \* len(test\_dogs))  val\_dogs\_arr,val\_dogs\_sol = np.array([img\_to\_array(dog) for dog in val\_dogs]), np.array([1] \* len(val\_dogs))  train\_cats\_arr,train\_cats\_sol = np.array([img\_to\_array(cat) for cat in train\_cats]), np.array([0] \* len(train\_cats))  test\_cats\_arr,test\_cats\_sol = np.array([img\_to\_array(cat) for cat in test\_cats]), np.array([0] \* len(test\_cats))  val\_cats\_arr,val\_cats\_sol = np.array([img\_to\_array(cat) for cat in val\_cats]),np.array([0]\*len(val\_cats))  train\_img,train\_sol = np.concatenate((train\_dogs\_arr,train\_cats\_arr)),np.concatenate((train\_dogs\_sol,train\_cats\_sol))  test\_img,test\_sol = np.concatenate((test\_dogs\_arr,test\_cats\_arr)),np.concatenate((test\_dogs\_sol,test\_cats\_sol))  val\_img,val\_sol = np.concatenate((val\_dogs\_arr,val\_cats\_arr)),np.concatenate((val\_dogs\_sol,val\_cats\_sol)) |

- 모델 생성

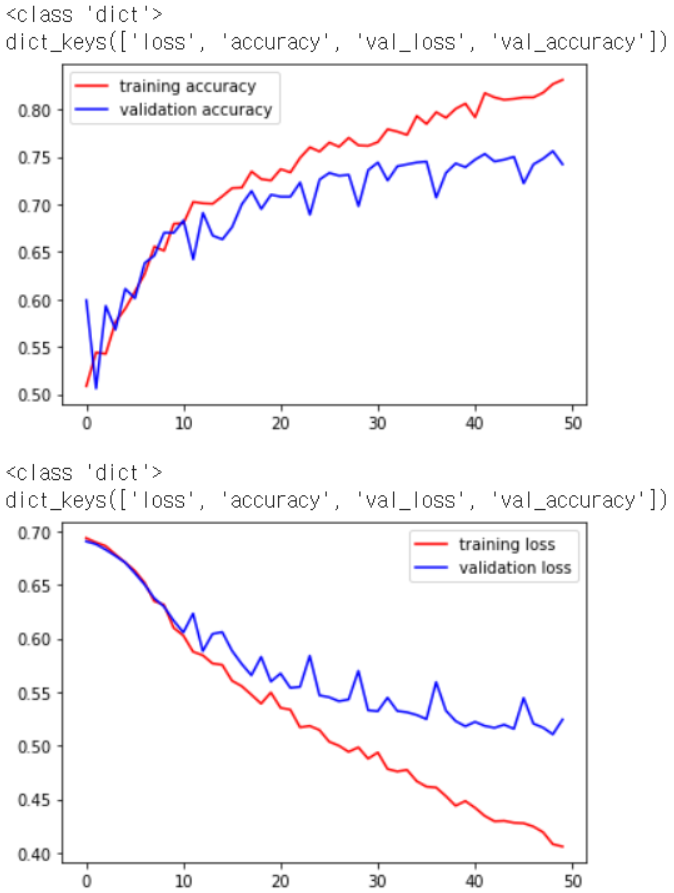
|  |
| --- |
| import numpy as np  import tensorflow as tf  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Conv2D, Flatten, MaxPooling2D, Dense, Dropout  from tensorflow.keras.optimizers import Adam  model = Sequential()  model.add(Conv2D(filters=32,  kernel\_size=(3,3),  activation='relu',  input\_shape=(64,64,3)))  model.add(MaxPooling2D(pool\_size=(2,2)))  model.add(Conv2D(filters=64,  kernel\_size=(3,3),  activation='relu'))  model.add(MaxPooling2D(pool\_size=(2,2)))  model.add(Conv2D(filters=128,  kernel\_size=(3,3),  activation='relu'))  model.add(MaxPooling2D(pool\_size=(2,2)))  model.add(Flatten())  model.add(Dropout(rate=0.5))  model.add(Dense(units=1,  activation='sigmoid'))  print(model.summary())  model.compile(optimizer=Adam(learning\_rate=1e-4),  loss='binary\_crossentropy',  metrics=['accuracy'])  history = model.fit(x=train\_img,  y=train\_sol,  steps\_per\_epoch=100,  epochs=50,  validation\_data=(val\_img,val\_sol),  validation\_steps=50) |





- 그래프 출력

|  |
| --- |
| # history 객체를 통해 과적합인지 확인할 수 있어요  print(type(history.history))  print(history.history.keys())  train\_acc = history.history['accuracy']  val\_acc = history.history['val\_accuracy']  train\_loss = history.history['loss']  val\_loss = history.history['val\_loss']  plt.plot(train\_acc, color='r', label='training accuracy')  plt.plot(val\_acc, color='b', label='validation accuracy')  plt.legend()  plt.show()  # plt.plot(train\_loss, color='r', label='training loss')  # plt.plot(val\_loss, color='b', label='validation loss')  # plt.legend()  # plt.show() |



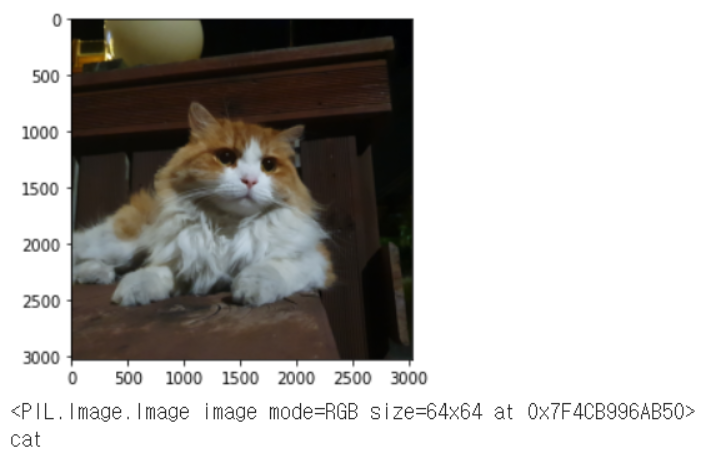
- test 데이터로 성능 평가

|  |
| --- |
| model.evaluate(test\_img,test\_sol,batch\_size = 128)  # 최종 정확도는 72.5% |



- 실제 이미지로 예측

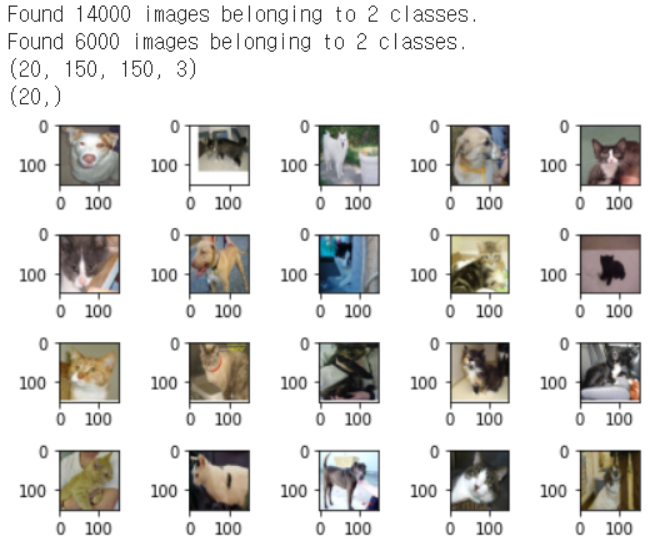
|  |
| --- |
| import PIL.Image as Image  def resize\_img(img,size):  return img.resize(size)  def load\_img(file\_path):  data = []  print(p + file\_path[1:] + '/')  for f in os.listdir(file\_path):  data.append(resize\_img(Image.open(p + file\_path[1:] + '/' + f) , (64,64)))  return data  def classify\_model(arr):  pre = model.predict(arr.reshape(1,64,64,3))  if pre > 0.5:  return 'dog'  else:  return 'cat'  img = '/content/drive/MyDrive/machine learning colab/test\_cat1.jpg'  nd\_img = Image.open(img)  plt.imshow(nd\_img)  plt.show()  nd\_img = resize\_img(nd\_img, (64,64))  print(nd\_img)  nd\_img = np.array(img\_to\_array(nd\_img))  print(classify\_model(nd\_img)) |



**2) 전체 데이터(25,000개)에 ImageDataGenerator를 이용하여 모델 구현 후 Accuracy 측정**

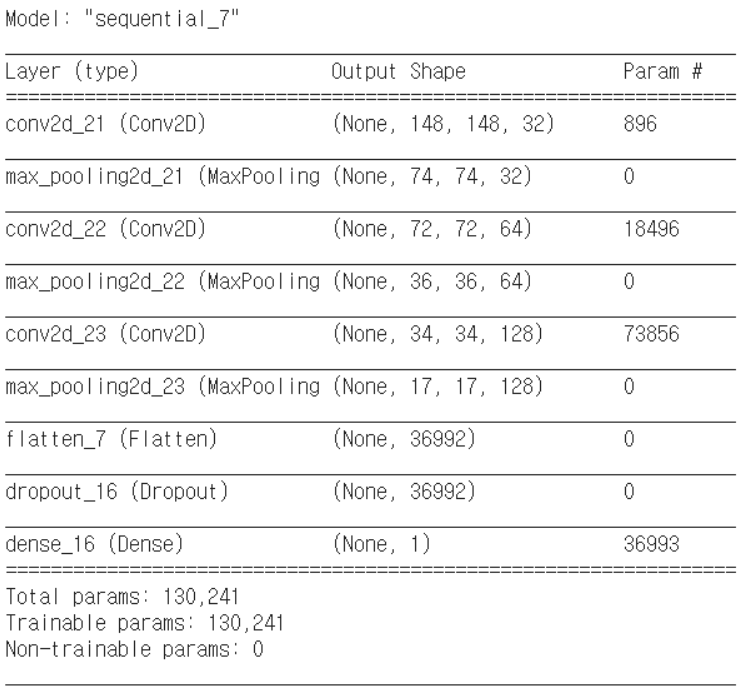
- 전처리

|  |
| --- |
| # %reset  import os  from tensorflow.keras.preprocessing.image import ImageDataGenerator  import matplotlib.pyplot as plt  train\_dir = '/content/drive/MyDrive/machine learning colab/cat\_dog\_full/train'  validation\_dir = '/content/drive/MyDrive/machine learning colab/cat\_dog\_full/validation'  # ImageDataGenerator 생성  train\_datagen = ImageDataGenerator(rescale=1/255)  validation\_datagen = ImageDataGenerator(rescale=1/255)  train\_generator = train\_datagen.flow\_from\_directory(  train\_dir,  classes=['cats','dogs'],  target\_size=(150,150),  batch\_size=20,  class\_mode='binary'  )  validation\_generator = validation\_datagen.flow\_from\_directory(  validation\_dir,  classes=['cats','dogs'],  target\_size=(150,150),  batch\_size=20,  class\_mode='binary'  )  for x\_data, t\_data in train\_generator:  print(x\_data.shape)  print(t\_data.shape)  break;  fig = plt.figure()  fig\_arr = []  for i in range(20):  fig\_arr.append(fig.add\_subplot(4,5,i+1))  for data\_batch, label\_batch in train\_generator:  for idx, img\_data in enumerate(data\_batch):  fig\_arr[idx].imshow(img\_data)  break;  fig.tight\_layout()  plt.show() |



- 모델 구현 및 학습

|  |
| --- |
| import numpy as np  import tensorflow as tf  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Conv2D, Flatten, MaxPooling2D, Dense, Dropout  from tensorflow.keras.optimizers import Adam  model = Sequential()  model.add(Conv2D(filters=32,  kernel\_size=(3,3),  activation='relu',  input\_shape=(150,150,3)))  model.add(MaxPooling2D(pool\_size=(2,2)))  model.add(Conv2D(filters=64,  kernel\_size=(3,3),  activation='relu'))  model.add(MaxPooling2D(pool\_size=(2,2)))  model.add(Conv2D(filters=128,  kernel\_size=(3,3),  activation='relu'))  model.add(MaxPooling2D(pool\_size=(2,2)))  model.add(Flatten())  model.add(Dropout(rate=0.5))  model.add(Dense(units=1,  activation='sigmoid'))  print(model.summary())  model.compile(optimizer=Adam(learning\_rate=1e-4),  loss='binary\_crossentropy',  metrics=['accuracy'])  history = model.fit(train\_generator,  steps\_per\_epoch=100,  epochs=50,  validation\_data=validation\_generator,  validation\_steps=50)  model.save('/content/drive/MyDrive/machine learning colab/cat\_dog\_full\_cnn\_tf2.4.h5') |



- 모델 불러오고 정확도 출력

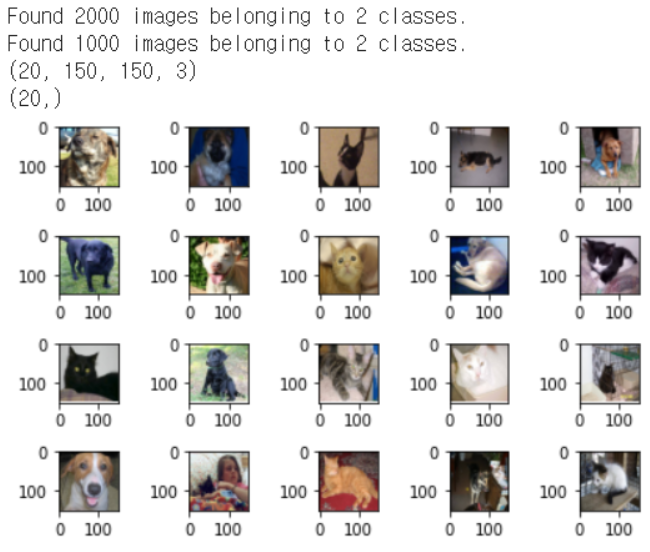
|  |
| --- |
| import os  from tensorflow.keras.preprocessing.image import ImageDataGenerator  import matplotlib.pyplot as plt  import numpy as np  import tensorflow as tf  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Conv2D, Flatten, MaxPooling2D, Dense, Dropout  from tensorflow.keras.optimizers import Adam  from keras.models import load\_model  model = load\_model('/content/drive/MyDrive/machine learning colab/cat\_dog\_full\_cnn\_tf2.4.h5')  test\_dir = '/content/drive/MyDrive/machine learning colab/cat\_dog\_full/test'  testGenerator = ImageDataGenerator(  rescale=1./255  )  testGen = testGenerator.flow\_from\_directory(  test\_dir,  target\_size=(150, 150),  )  model.evaluate\_generator(testGen) |



**3) 일부데이터로 ImageDataGenerator 이용해 예측**

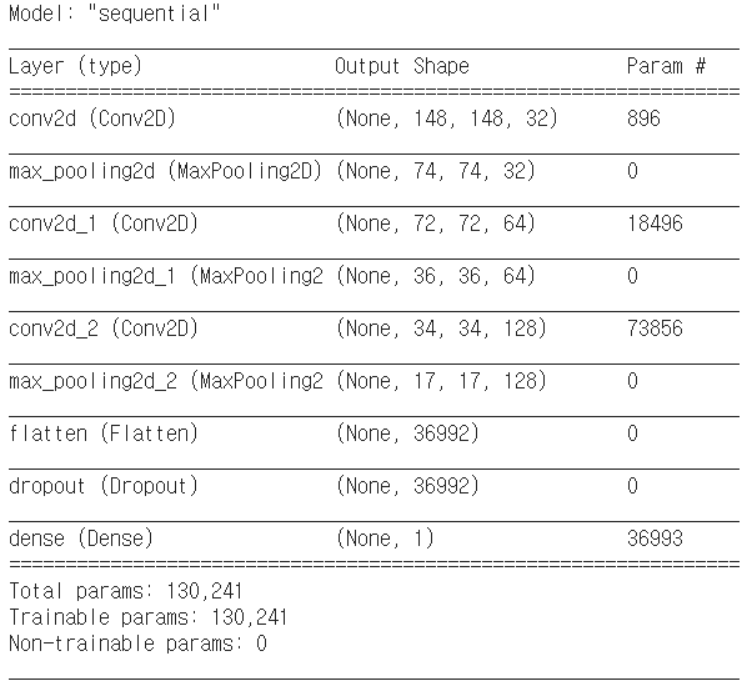
- 전처리

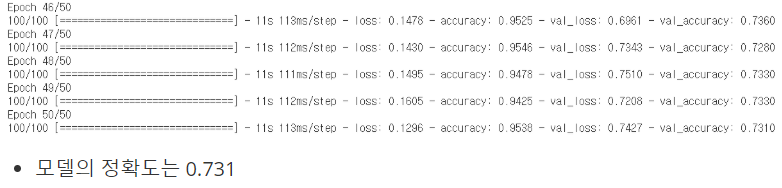
|  |
| --- |
| ## 3. 일부데이터를 ImageDataGenerator로 구현  # %reset  import os  from tensorflow.keras.preprocessing.image import ImageDataGenerator  import matplotlib.pyplot as plt  train\_dir = '/content/drive/MyDrive/machine learning colab/cat\_dog\_small/train'  validation\_dir = '/content/drive/MyDrive/machine learning colab/cat\_dog\_small/validation'  # ImageDataGenerator 생성  train\_datagen = ImageDataGenerator(rescale=1/255)  validation\_datagen = ImageDataGenerator(rescale=1/255)  train\_generator = train\_datagen.flow\_from\_directory(  train\_dir,  classes=['cats','dogs'],  target\_size=(150,150),  batch\_size=20,  class\_mode='binary')  validation\_generator = validation\_datagen.flow\_from\_directory(  validation\_dir,  classes=['cats','dogs'],  target\_size=(150,150),  batch\_size=20,  class\_mode='binary')  # generator를 이용  for x\_data, t\_data in train\_generator:  print(x\_data.shape) # (20,150,150,3)  print(t\_data.shape) # (20,)  break;  # 그림 이미지를 matplotlib을 이용해서 출력  fig = plt.figure()  fig\_arr = []  for i in range(20):  fig\_arr.append(fig.add\_subplot(4,5,i+1))  for data\_batch, label\_batch in train\_generator:  for idx, img\_data in enumerate(data\_batch):  fig\_arr[idx].imshow(img\_data)  break;  fig.tight\_layout()  plt.show() |



- 모델 생성 및 학습

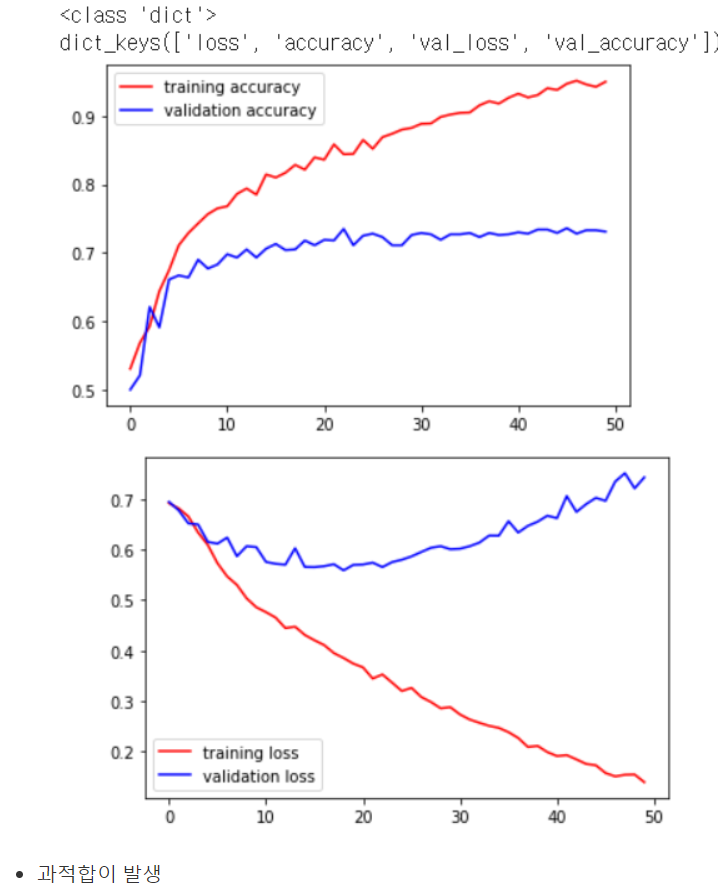
|  |
| --- |
| import numpy as np  import tensorflow as tf  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Conv2D, Flatten, MaxPooling2D, Dense, Dropout  from tensorflow.keras.optimizers import Adam  model = Sequential()  model.add(Conv2D(filters=32,  kernel\_size=(3,3),  activation='relu',  input\_shape=(150,150,3)))  model.add(MaxPooling2D(pool\_size=(2,2)))  model.add(Conv2D(filters=64,  kernel\_size=(3,3),  activation='relu'))  model.add(MaxPooling2D(pool\_size=(2,2)))  model.add(Conv2D(filters=128,  kernel\_size=(3,3),  activation='relu'))  model.add(MaxPooling2D(pool\_size=(2,2)))  model.add(Flatten())  model.add(Dropout(rate=0.5))  model.add(Dense(units=1,  activation='sigmoid'))  print(model.summary())  model.compile(optimizer=Adam(learning\_rate=1e-4),  loss='binary\_crossentropy',  metrics=['accuracy'])  history = model.fit(train\_generator,  steps\_per\_epoch=100,  epochs=50,  validation\_data=validation\_generator,  validation\_steps=50) |





- 그래프 출력

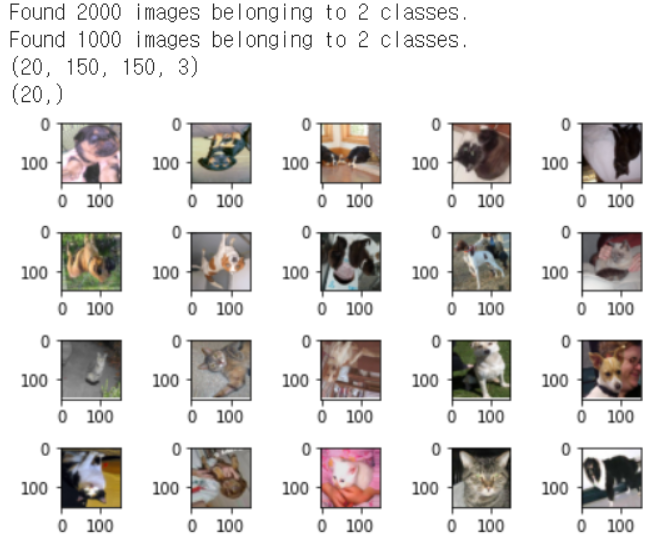
|  |
| --- |
| # history 객체를 통해 과적합인지 확인할 수 있어요  print(type(history.history))  print(history.history.keys())  train\_acc = history.history['accuracy']  val\_acc = history.history['val\_accuracy']  train\_loss = history.history['loss']  val\_loss = history.history['val\_loss']  plt.plot(train\_acc, color='r', label='training accuracy')  plt.plot(val\_acc, color='b', label='validation accuracy')  plt.legend()  plt.show()  # plt.plot(train\_loss, color='r', label='training loss')  # plt.plot(val\_loss, color='b', label='validation loss')  # plt.legend()  # plt.show() |



**4) overfitting을 피하기 위해 증식으로 모델 생성 후 예측**

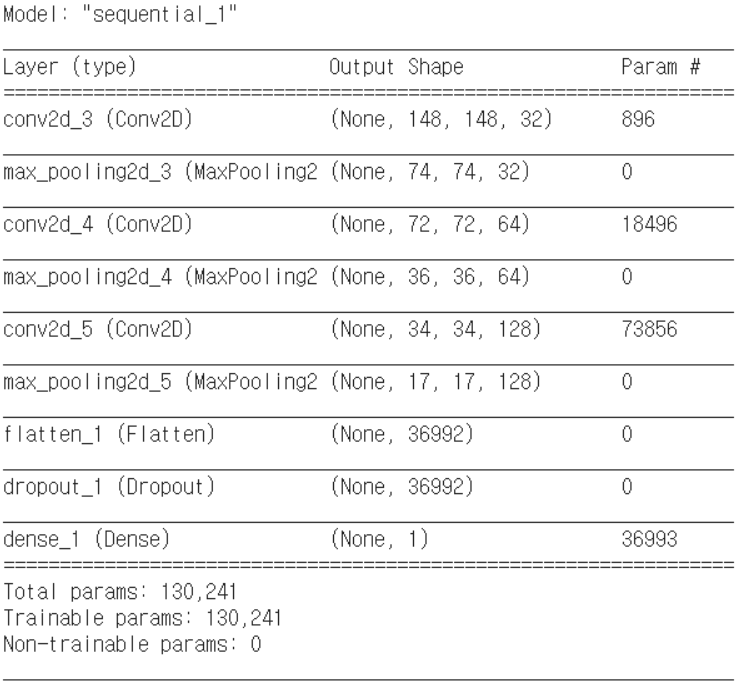
- 증식을 이용한 전처리

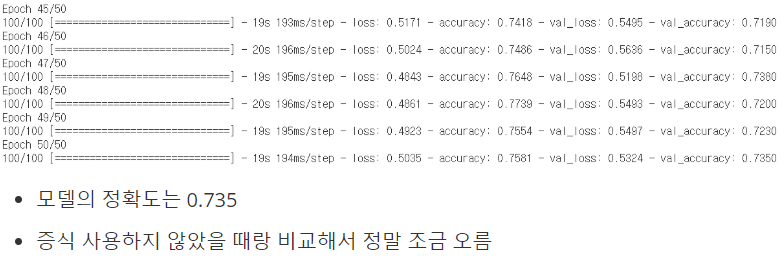
|  |
| --- |
| # %reset  # 증식을 이용한 코드  import os  from tensorflow.keras.preprocessing.image import ImageDataGenerator  import matplotlib.pyplot as plt  train\_dir = '/content/drive/MyDrive/machine learning colab/cat\_dog\_small/train'  validation\_dir = '/content/drive/MyDrive/machine learning colab/cat\_dog\_small/validation'  # ImageDataGenerator 생성  train\_datagen = ImageDataGenerator(rescale=1/255,  rotation\_range=20,  width\_shift\_range=0.1,  zoom\_range=0.1,  horizontal\_flip=True,  vertical\_flip=True)  validation\_datagen = ImageDataGenerator(rescale=1/255)  train\_generator = train\_datagen.flow\_from\_directory(  train\_dir,  classes=['cats','dogs'],  target\_size=(150,150),  batch\_size=20,  class\_mode='binary'  )  validation\_generator = validation\_datagen.flow\_from\_directory(  validation\_dir,  classes=['cats','dogs'],  target\_size=(150,150),  batch\_size=20,  class\_mode='binary'  )  # generator를 이용  for x\_data, t\_data in train\_generator:  print(x\_data.shape) # (20,150,150,3)  print(t\_data.shape) # (20,)  break;  # 그림 이미지를 matplotlib을 이용해서 출력  fig = plt.figure()  fig\_arr = []  for i in range(20):  fig\_arr.append(fig.add\_subplot(4,5,i+1))  for data\_batch, label\_batch in train\_generator:  for idx, img\_data in enumerate(data\_batch):  fig\_arr[idx].imshow(img\_data)  break;  fig.tight\_layout()  plt.show()  # 다음 단계는 CNN을 구성하고 학습을 진행  # MNIST와 유사, Convolution Layer와 Pooling Layer 추가 |



- 모델 생성 및 학습

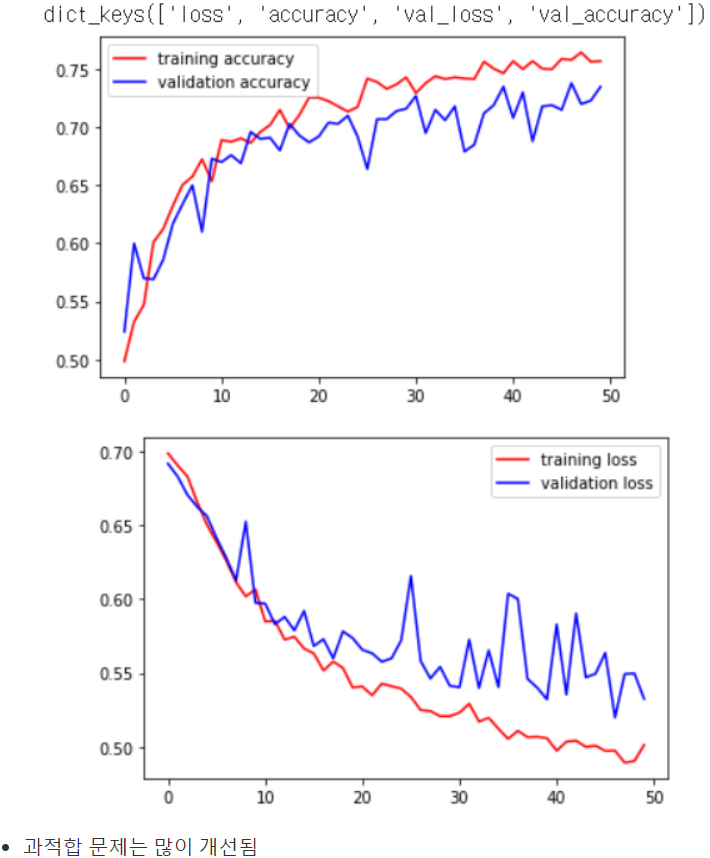
|  |
| --- |
| import numpy as np  import tensorflow as tf  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Conv2D, Flatten, MaxPooling2D, Dense, Dropout  from tensorflow.keras.optimizers import Adam  model = Sequential()  model.add(Conv2D(filters=32,  kernel\_size=(3,3),  activation='relu',  input\_shape=(150,150,3)))  model.add(MaxPooling2D(pool\_size=(2,2)))  model.add(Conv2D(filters=64,  kernel\_size=(3,3),  activation='relu'))  model.add(MaxPooling2D(pool\_size=(2,2)))  model.add(Conv2D(filters=128,  kernel\_size=(3,3),  activation='relu'))  model.add(MaxPooling2D(pool\_size=(2,2)))  model.add(Flatten())  model.add(Dropout(rate=0.5))  model.add(Dense(units=1,  activation='sigmoid'))  print(model.summary())  model.compile(optimizer=Adam(learning\_rate=1e-4),  loss='binary\_crossentropy',  metrics=['accuracy'])  history = model.fit(train\_generator,  steps\_per\_epoch=100,  epochs=50,  validation\_data=validation\_generator,  validation\_steps=50) |





- 그래프 출력

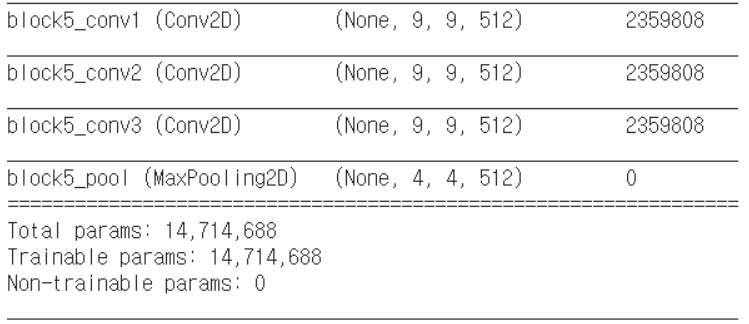
|  |
| --- |
| # history 객체를 통해 과적합인지 확인할 수 있어요  print(type(history.history))  print(history.history.keys())  train\_acc = history.history['accuracy']  val\_acc = history.history['val\_accuracy']  train\_loss = history.history['loss']  val\_loss = history.history['val\_loss']  plt.plot(train\_acc, color='r', label='training accuracy')  plt.plot(val\_acc, color='b', label='validation accuracy')  plt.legend()  plt.show()  # plt.plot(train\_loss, color='r', label='training loss')  # plt.plot(val\_loss, color='b', label='validation loss')  # plt.legend()  # plt.show() |



**5) Pretrained Network**

- Pretrained network 생셩

|  |
| --- |
| # %reset  # pretrained network 생성  from tensorflow.keras.applications import VGG16  model\_base = VGG16(weights='imagenet',  include\_top=False,  input\_shape=(150,150,3))  model\_base.summary() |



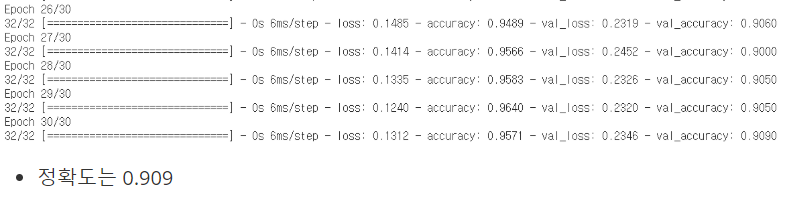
- feature map 추출

|  |
| --- |
| # 개와 고양이 training data set에 대한 feature map을 추출  import os  import numpy as np  from tensorflow.keras.preprocessing.image import ImageDataGenerator  base\_dir = '/content/drive/MyDrive/machine learning colab/cat\_dog\_small'  train\_dir = os.path.join(base\_dir,'train')  validation\_dir = os.path.join(base\_dir,'validation')  datagen = ImageDataGenerator(rescale=1/255)  batch\_size=20  def extract\_feature(directory, sample\_count):  features = np.zeros(shape=(sample\_count,4,4,512))  labels=np.zeros(shape=(sample\_count,))  generator = datagen.flow\_from\_directory(  directory,  target\_size=(150,150),  batch\_size=batch\_size,  class\_mode='binary'  )    i = 0  for x\_data\_batch, t\_data\_batch in generator:  feature\_batch = model\_base.predict(x\_data\_batch)  features[i\*batch\_size:(i+1)\*batch\_size] = feature\_batch  labels[i\*batch\_size:(i+1)\*batch\_size] = t\_data\_batch  i += 1  if i\*batch\_size >= sample\_count:  break  return features, labels  train\_features, train\_labels = extract\_feature(train\_dir, 2000)  validation\_features, validation\_labels = extract\_feature(validation\_dir, 1000) |



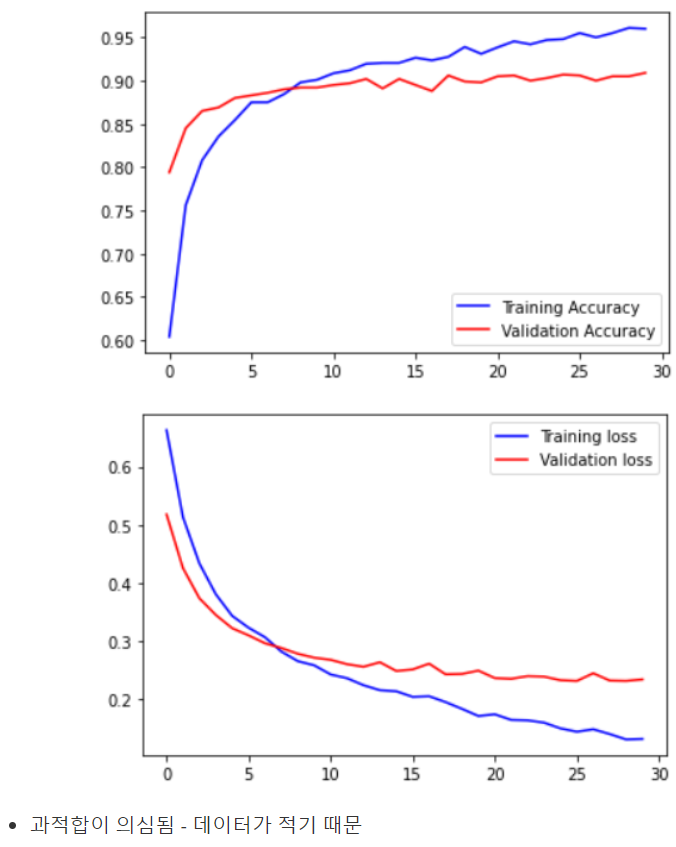
- 모델 생성 및 학습

|  |
| --- |
| # classifier  train\_features = np.reshape(train\_features, (2000,4\*4\*512))  validation\_features = np.reshape(validation\_features, (1000,4\*4\*512))  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense, Dropout  from tensorflow.keras.optimizers import Adam, RMSprop  model = Sequential()  model.add(Dense(256,  activation='relu',  input\_shape=(4\*4\*512,)))  model.add(Dropout(0.5))  model.add(Dense(1,  activation='sigmoid'))  model.compile(optimizer=RMSprop(learning\_rate=2e-5),  loss='binary\_crossentropy',  metrics=['accuracy'])  history = model.fit(train\_features,  train\_labels,  epochs=30,  batch\_size=64,  validation\_data=(validation\_features,validation\_labels)) |



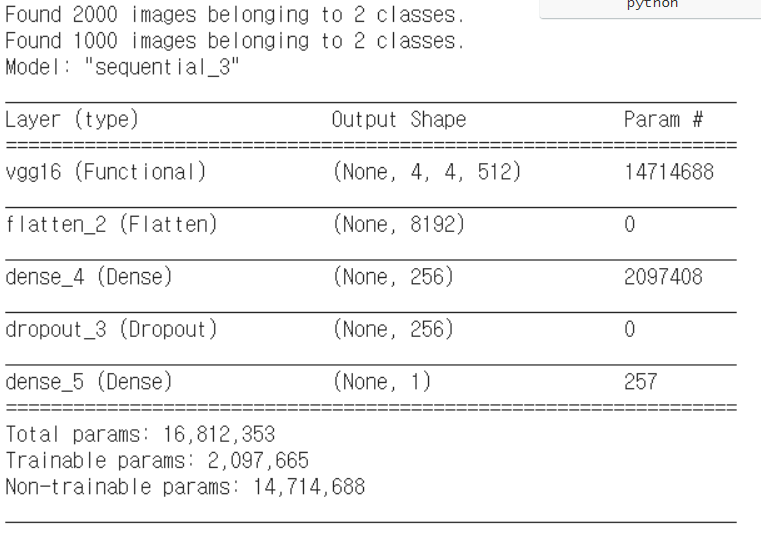
- 그래프 출력

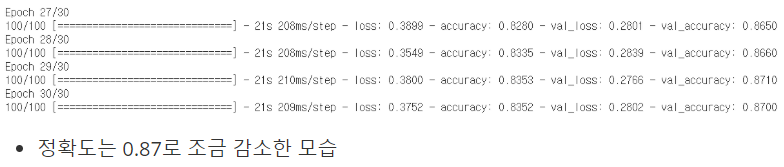
|  |
| --- |
| import matplotlib.pyplot as plt  train\_acc = history.history['accuracy']  val\_acc = history.history['val\_accuracy']  train\_loss = history.history['loss']  val\_loss = history.history['val\_loss']  plt.plot(train\_acc, color='b', label='Training Accuracy')  plt.plot(val\_acc, color='r', label='Validation Accuracy')  plt.legend()  plt.show()  # plt.plot(train\_loss, color='b', label='Training loss')  # plt.plot(val\_loss, color='r', label='Validation loss')  # plt.legend()  # plt.show() |



- 증식을 이용해서 재학습

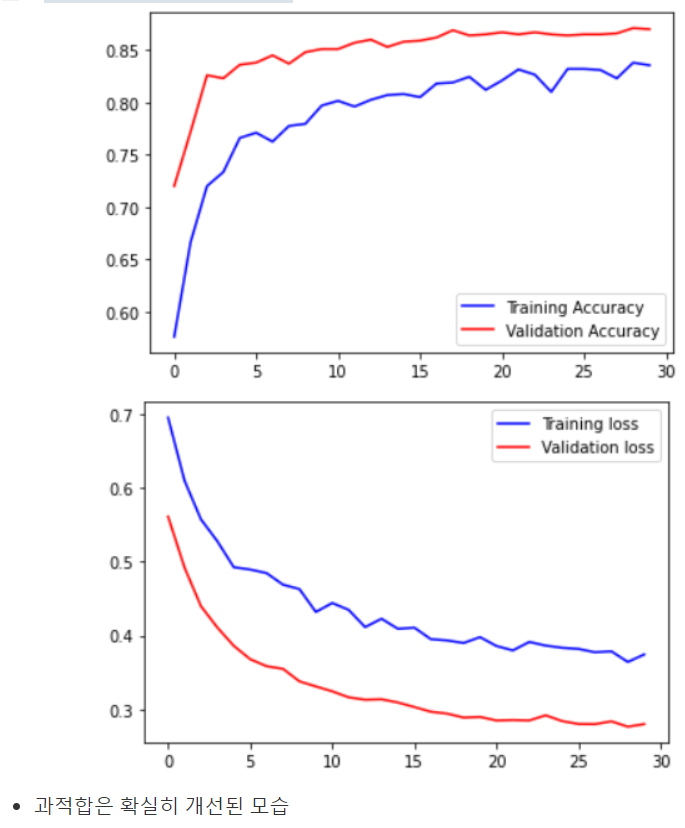
|  |
| --- |
| # 데이터가 많아지면 결과가 나아질 것  # 증식을 포함  # pretrained network와 classifier을 합쳐서 모델 생성  # 기학습된 모델을 따로 쓰는게 아니라 모델 안에 삽입 데이터가 많아지면 결과가 나아질 것  import os  import numpy as np  from tensorflow.keras.applications import VGG16  from tensorflow.keras.preprocessing.image import ImageDataGenerator  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense, Dropout, Flatten  from tensorflow.keras.optimizers import RMSprop  import matplotlib.pyplot as plt  base\_dir = '/content/drive/MyDrive/machine learning colab/cat\_dog\_small'  train\_dir = os.path.join(base\_dir,'train')  validation\_dir = os.path.join(base\_dir,'validation')  train\_datagen = ImageDataGenerator(rescale=1/255,  rotation\_range=40,  width\_shift\_range=0.1,  height\_shift\_range=0.1,  zoom\_range=0.2,  horizontal\_flip=True,  vertical\_flip=True)  validation\_datagen = ImageDataGenerator(rescale=1/255)  train\_generator = train\_datagen.flow\_from\_directory(  train\_dir,  classes=['cats', 'dogs'],  target\_size=(150,150),  batch\_size=20,  class\_mode='binary'  )  validation\_generator = validation\_datagen.flow\_from\_directory(  validation\_dir,  classes=['cats', 'dogs'],  target\_size=(150,150),  batch\_size=20,  class\_mode='binary'  )  # pretrained network  model\_base = VGG16(weights='imagenet',  include\_top=False,  input\_shape=(150,150,3))  model\_base.trainable=False # Convolution Layer를 동결  model = Sequential()  model.add(model\_base)  model.add(Flatten(input\_shape=(4\*4\*512,)))  model.add(Dense(units=256,  activation='relu'))  model.add(Dropout(0.5))  model.add(Dense(units=1,  activation='sigmoid'))  model.summary()  model.compile(optimizer=RMSprop(learning\_rate=2e-5),  loss='binary\_crossentropy',  metrics=['accuracy'])  history = model.fit(train\_generator,  steps\_per\_epoch=100,  epochs=30,  validation\_data=validation\_generator,  validation\_steps=50) |





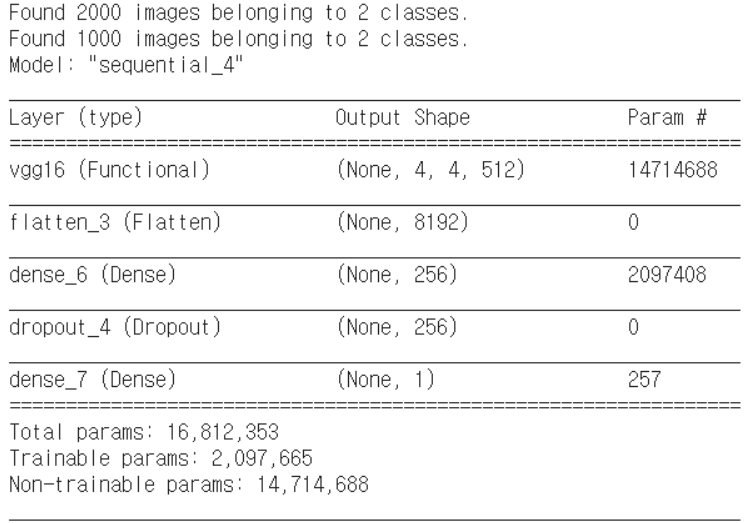
- 그래프 출력

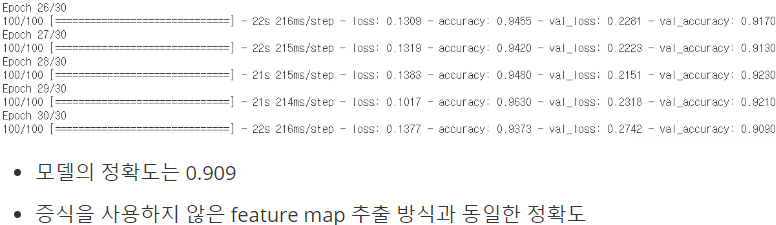
|  |
| --- |
| import matplotlib.pyplot as plt  train\_acc = history.history['accuracy']  val\_acc = history.history['val\_accuracy']  train\_loss = history.history['loss']  val\_loss = history.history['val\_loss']  plt.plot(train\_acc, color='b', label='Training Accuracy')  plt.plot(val\_acc, color='r', label='Validation Accuracy')  plt.legend()  plt.show()  # plt.plot(train\_loss, color='b', label='Training loss')  # plt.plot(val\_loss, color='r', label='Validation loss')  # plt.legend()  # plt.show() |



- Fine Tuning 기법을 이용해 구현

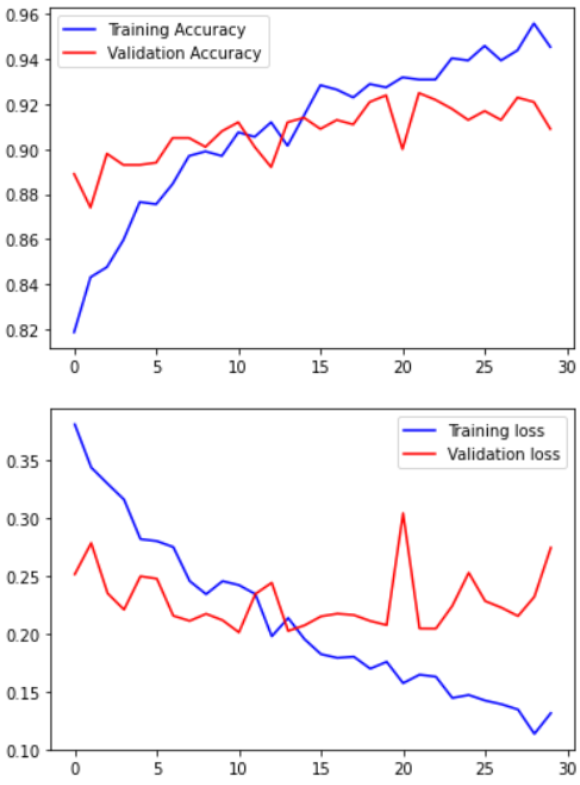
|  |
| --- |
| # fine tuning  import os  import numpy as np  from tensorflow.keras.applications import VGG16  from tensorflow.keras.preprocessing.image import ImageDataGenerator  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense, Dropout, Flatten  from tensorflow.keras.optimizers import RMSprop  import matplotlib.pyplot as plt  base\_dir = '/content/drive/MyDrive/machine learning colab/cat\_dog\_small'  train\_dir = os.path.join(base\_dir,'train')  validation\_dir = os.path.join(base\_dir,'validation')  train\_datagen = ImageDataGenerator(rescale=1/255,  rotation\_range=40,  width\_shift\_range=0.1,  height\_shift\_range=0.1,  zoom\_range=0.2,  horizontal\_flip=True,  vertical\_flip=True)  validation\_datagen = ImageDataGenerator(rescale=1/255)  train\_generator = train\_datagen.flow\_from\_directory(  train\_dir,  classes=['cats', 'dogs'],  target\_size=(150,150),  batch\_size=20,  class\_mode='binary'  )  validation\_generator = validation\_datagen.flow\_from\_directory(  validation\_dir,  classes=['cats', 'dogs'],  target\_size=(150,150),  batch\_size=20,  class\_mode='binary'  )  # pretrained network  model\_base = VGG16(weights='imagenet',  include\_top=False,  input\_shape=(150,150,3))  model\_base.trainable=False # Convolution Layer를 동결  model = Sequential()  model.add(model\_base)  model.add(Flatten(input\_shape=(4\*4\*512,)))  model.add(Dense(units=256,  activation='relu'))  model.add(Dropout(0.5))  model.add(Dense(units=1,  activation='sigmoid'))  model.summary()  model.compile(optimizer=RMSprop(learning\_rate=2e-5),  loss='binary\_crossentropy',  metrics=['accuracy'])  history = model.fit(train\_generator,  steps\_per\_epoch=100,  epochs=30,  validation\_data=validation\_generator,  validation\_steps=50)  model\_base.trainable = True  for layer in model\_base.layers:  if layer.name in ['block5\_conv1', 'block5\_conv2', 'block5\_conv3']:  layer.trainable = True  else:  layer.trainable = False  # 일반적으로 러닝 레이트를 더 작게 설정  model.compile(optimizer=RMSprop(learning\_rate=1e-5),  loss='binary\_crossentropy',  metrics=['accuracy'])  history = model.fit(train\_generator,  steps\_per\_epoch=100,  epochs=30,  validation\_data=validation\_generator,  validation\_steps=50) |





- 그래프 출력

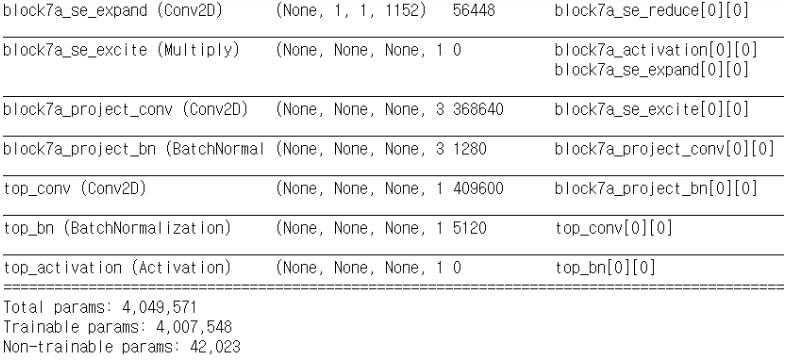
|  |
| --- |
| import matplotlib.pyplot as plt  train\_acc = history.history['accuracy']  val\_acc = history.history['val\_accuracy']  train\_loss = history.history['loss']  val\_loss = history.history['val\_loss']  plt.plot(train\_acc, color='b', label='Training Accuracy')  plt.plot(val\_acc, color='r', label='Validation Accuracy')  plt.legend()  plt.show()  # plt.plot(train\_loss, color='b', label='Training loss')  # plt.plot(val\_loss, color='r', label='Validation loss')  # plt.legend()  # plt.show() |



**6) EfficientNet 이용**

- pretrained network 생성

|  |
| --- |
| %reset  # pretrained network 생성  from tensorflow.keras.applications import EfficientNetB0  model\_base = EfficientNetB0(weights='imagenet',  include\_top=False)  model\_base.summary() |



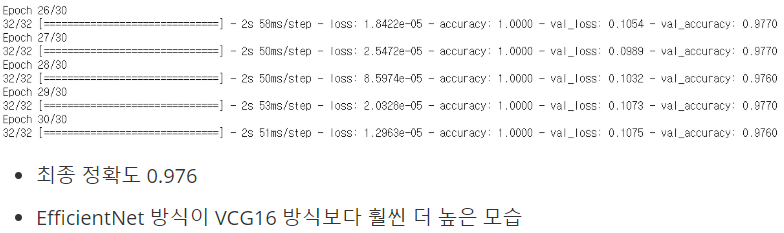
- feature map 추출

|  |
| --- |
| # 개와 고양이 training data set에 대한 feature map을 추출  import os  import numpy as np  from tensorflow.keras.preprocessing.image import ImageDataGenerator  base\_dir = './data/cat\_dog\_small'  train\_dir = os.path.join(base\_dir,'train')  validation\_dir = os.path.join(base\_dir,'validation')  datagen = ImageDataGenerator()  batch\_size=20  def extract\_feature(directory, sample\_count):  features = np.zeros(shape=(sample\_count,4,4,1280))  labels=np.zeros(shape=(sample\_count,))  generator = datagen.flow\_from\_directory(  directory,  target\_size=(150,150),  batch\_size=batch\_size,  class\_mode='binary'  )    i = 0  for x\_data\_batch, t\_data\_batch in generator:  feature\_batch = model\_base.predict(x\_data\_batch)  features[i\*batch\_size:(i+1)\*batch\_size] = feature\_batch  labels[i\*batch\_size:(i+1)\*batch\_size] = t\_data\_batch  i += 1  if i\*batch\_size >= sample\_count:  break  return features, labels  train\_features, train\_labels = extract\_feature(train\_dir, 2000)  validation\_features, validation\_labels = extract\_feature(validation\_dir, 1000) |



- 학습 및 정확도 출력

|  |
| --- |
| # classifier  train\_features = np.reshape(train\_features, (2000,4\*4\*1280))  validation\_features = np.reshape(validation\_features, (1000,4\*4\*1280))  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense, Dropout  from tensorflow.keras.optimizers import Adam, RMSprop  model = Sequential()  model.add(Dense(256,  activation='relu',  input\_shape=(4\*4\*1280,)))  model.add(Dropout(0.5))  model.add(Dense(1,  activation='sigmoid'))  model.compile(optimizer=RMSprop(learning\_rate=2e-5),  loss='binary\_crossentropy',  metrics=['accuracy'])  history = model.fit(train\_features,  train\_labels,  epochs=30,  batch\_size=64,  validation\_data=(validation\_features,validation\_labels)) |



- fine tuning 방식으로 구현

|  |
| --- |
| # fine tuning  import os  import numpy as np  from tensorflow.keras.applications import EfficientNetB0  from tensorflow.keras.preprocessing.image import ImageDataGenerator  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense, Dropout, Flatten  from tensorflow.keras.optimizers import RMSprop  import matplotlib.pyplot as plt  base\_dir = './data/cat\_dog\_small'  train\_dir = os.path.join(base\_dir,'train')  validation\_dir = os.path.join(base\_dir,'validation')  train\_datagen = ImageDataGenerator(rotation\_range=40,  width\_shift\_range=0.1,  height\_shift\_range=0.1,  zoom\_range=0.2,  horizontal\_flip=True,  vertical\_flip=True)  validation\_datagen = ImageDataGenerator()  train\_generator = train\_datagen.flow\_from\_directory(  train\_dir,  classes=['cats', 'dogs'],  target\_size=(150,150),  batch\_size=20,  class\_mode='binary'  )  validation\_generator = validation\_datagen.flow\_from\_directory(  validation\_dir,  classes=['cats', 'dogs'],  target\_size=(150,150),  batch\_size=20,  class\_mode='binary'  )  # pretrained network  model\_base = EfficientNetB0(weights='imagenet',  include\_top=False,  input\_shape=(150,150,3))  model\_base.trainable=False # Convolution Layer를 동결  model = Sequential()  model.add(model\_base)  model.add(Flatten(input\_shape=(4\*4\*1280,)))  model.add(Dense(units=256,  activation='relu'))  model.add(Dropout(0.5))  model.add(Dense(units=1,  activation='sigmoid'))  model.summary()  model.compile(optimizer=RMSprop(learning\_rate=2e-5),  loss='binary\_crossentropy',  metrics=['accuracy'])  history = model.fit(train\_generator,  steps\_per\_epoch=100,  epochs=30,  validation\_data=validation\_generator,  validation\_steps=50)  model\_base.trainable = True  for layer in model\_base.layers:  if layer.name in ['block5\_conv1', 'block5\_conv2', 'block5\_conv3']:  layer.trainable = True  else:  layer.trainable = False  # 일반적으로 러닝 레이트를 더 작게 설정  model.compile(optimizer=RMSprop(learning\_rate=1e-5),  loss='binary\_crossentropy',  metrics=['accuracy'])  history = model.fit(train\_generator,  steps\_per\_epoch=100,  epochs=30,  validation\_data=validation\_generator,  validation\_steps=50) |

0