**EIEN 443 Artificial Neural Networks and Deep Learning (2021 Spring)**

Homework #3 (Due: May 10. 2021)

Name: \_\_\_\_\_\_\_\_\_\_박정수\_\_\_\_\_\_\_\_\_\_ Student ID: \_\_\_\_\_2016270431\_\_\_\_\_

**Q1 (2 points)**

|  |
| --- |
| from tensorflow.keras.preprocessing.image import ImageDataGenerator  import matplotlib.pyplot as plt  from tensorflow.keras import models, layers, optimizers  from tensorflow.keras.applications import VGG16  from tensorflow.keras.layers import BatchNormalization, Activation  from tensorflow.keras.models import load\_model  train\_dir = '/home/oms315/Desktop/HW3/chest\_xray/train/'  test\_dir = '/home/oms315/Desktop/HW3/chest\_xray/test/'  validation\_dir = '/home/oms315/Desktop/HW3/chest\_xray/val/'  train\_datagen = ImageDataGenerator(rescale = 1. / 255)  test\_datagen = ImageDataGenerator(rescale = 1. / 255)  validation\_datagen = ImageDataGenerator(rescale = 1. / 255)  train\_generator = train\_datagen.flow\_from\_directory(  train\_dir,  target\_size = (128, 128),  batch\_size = 20,  class\_mode = 'binary')  test\_generator = test\_datagen.flow\_from\_directory(  test\_dir,  target\_size = (128, 128),  batch\_size = 20,  class\_mode = 'binary')  validation\_generator = validation\_datagen.flow\_from\_directory(  validation\_dir,  target\_size = (128, 128),  batch\_size = 20,  class\_mode = 'binary')  for data\_batch, labels\_batch in train\_generator:  print('data\_size:', data\_batch.shape)  print('labels\_size:', labels\_batch.shape)  break  input\_shape = [128, 128, 3]  def build\_model():  model = models.Sequential()  conv\_base = VGG16(weights = 'imagenet',  include\_top = False,  input\_shape = input\_shape)  conv\_base.trainable = False  model.add(conv\_base)  model.add(layers.GlobalAveragePooling2D())  model.add(layers.Dense(512))  model.add(BatchNormalization())  model.add(Activation('relu'))  model.add(layers.Dense(128, activation = 'relu'))  model.add(layers.Dense(1, activation = 'sigmoid'))  model.compile(optimizer = optimizers.RMSprop(lr = 1e-5),  loss = 'binary\_crossentropy', metrics = ['accuracy'])  return model  import time  starttime = time.time();  num\_epochs = 100  model = build\_model()  history = model.fit\_generator(train\_generator,  epochs = num\_epochs, steps\_per\_epoch = 100,  validation\_data = validation\_generator, validation\_steps = 50)  model.save('hw3\_model.h5')  train\_loss, train\_acc = model.evaluate\_generator(train\_generator)  test\_loss, test\_acc = model.evaluate\_generator(test\_generator)  print('train\_loss:', train\_loss)  print('train\_acc:', train\_acc)  print('test\_loss:', test\_loss)  print('test\_acc:', test\_acc)  print(‘elapsed time: ‘, time.time() - starttime)  model = load\_model('hw3\_1\_model.h5')  conv\_base = model.layers[0]  for layer in conv\_base.layers:  if layer.name.startswith('block5'):  layer.trainable = True  model.compile(optimizer = optimizers.RMSprop(lr = 1e-5),  loss = 'binary\_crossentropy', metrics = ['accuracy'])  import time  starttime = time.time()  num\_epochs = 50  history = model.fit\_generator(train\_generator,  epochs = num\_epochs, steps\_per\_epoch = 100,  validation\_data = validation\_generator, validation\_steps = 50)  model.save('hw3\_fine\_model.h5')  train\_loss, train\_acc = model.evaluate\_generator(train\_generator)  test\_loss, test\_acc = model.evaluate(test\_generator)  print('train\_finetune\_loss:', train\_loss)  print('train\_finetune\_acc:', train\_acc)  print('test\_finetune\_loss:', test\_loss)  print('test\_finetune\_acc:', test\_acc)  print(‘elapsed time: ‘, time.time() - starttime)  def plot\_acc(h, title="accuracy"):  plt.plot(h.history['acc'])  plt.plot(h.history['val\_acc'])  plt.title(title)  plt.ylabel('Accuracy')  plt.xlabel('Epoch')  plt.legend(['Training', 'Validation'], loc = 0)  def plot\_loss(h, title='loss'):  plt.plot(h.history['loss'])  plt.plot(h.history['val\_loss'])  plt.title(title)  plt.ylabel('Loss')  plt.xlabel('Epoch')  plt.legend(['Training', 'Validation'], loc = 0)  plot\_loss(history)  plt.savefig('loss.png')  plt.clf()  plot\_acc(history)  plt.savefig('acc.png') |

|  |  |  |
| --- | --- | --- |
|  | Before FT | After FT |
| Training  Loss | 0.0406 | 2.4159e-05 |
| Training  Accuracy | 0.9713 | 1 |
| Test  Loss | 0.6934 | 2.2499 |
| Test  Accuracy | 0.7639 | 0.7548 |

|  |  |
| --- | --- |
| Loss Graph | Accuracy Graph |

**Your answer:**

**Q2 (1 point)**

Filling the table and Attaching graphs is worthy of 1 point.

아래 표와 그래프를 첨부하는 것이 0.5점입니다.

|  |  |  |
| --- | --- | --- |
|  | Before FT | After FT |
| Training  Loss | 0.2365 | 6.2973e-05 |
| Training  Accuracy | 0.8993 | 1 |
| Test  Loss | 0.3029 | 1.5668 |
| Test  Accuracy | 0.8782 | 0.8061 |

|  |  |
| --- | --- |
| Loss Graph  C:\Users\OhMinSeop\AppData\Local\Microsoft\Windows\INetCache\Content.Word\LOSS.JPG | Accuracy Graph |

**Your answer:** (The answer is 0.5 points worthy!)

Do you think that overfitting is reduced?

그렇다. Q1과 같은 Epoch 수를 기준으로 보았을 때 validation loss가 급격하게 튀는 구간이 덜 발생한다.

Is it improved compared to the results of Q1?  
그렇다. Test Accuracy가 Q1의 결과보다 증가했기 때문이다.

**Q3 (1 point)**

Filling the table and Attaching graphs is worthy of 0.5 point.

아래 표와 그래프를 첨부하는 것이 0.5점입니다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | [256, 256]  Before FT | [256, 256]  After FT | [512, 512]  Before FT | [512, 512]  After FT |
| Training  Loss | 0.0327 | 1.9190e-06 | 0.0558 | 0.0129 |
| Training  Accuracy | 0.9896 | 1 | 0.9804 | 0.9933 |
| Test  Loss | 0.5736 | 2.9503 | 0.5737 | 0.7353 |
| Test  Accuracy | 0.8349 | 0.7621 | 0.8205 | 0.8622 |

|  |  |
| --- | --- |
| Loss Graph | Accuracy Graph |

**Your answer:** (The answer is 0.5 points worthy!)

Which one is the best among results among Q1 Q2, and Q3? Why?

Q3([512,512])가 best model 이다. 그 이유는 Q3의 Test Accuracy가 가장 크기 때문이다.

**Q4 (2 points)**

|  |
| --- |
| from tensorflow.keras.applications.vgg16 import VGG16, preprocess\_input, decode\_predictions  from tensorflow.keras import backend as K  from tensorflow.keras.preprocessing import image  import matplotlib.pyplot as plt  import numpy as np  import cv2  img\_path= '/home/oms315/Desktop/HW3/chest\_xray/chest\_xray/train/PNEUMONIA/person877\_virus\_1525.jpeg'  img = image.load\_img(img\_path, target\_size=(224, 224))  img\_tensor = image.img\_to\_array(img)  img\_tensor = np.expand\_dims(img\_tensor, axis=0)  img\_tensor = preprocess\_input(img\_tensor)  model = VGG16(weights='imagenet')  def deprocess\_image(x):  x -= x.mean()  x /= (x.std() + 1e-5)  x \*= 0.1  x += 0.5  x = np.clip(x, 0, 1)  x \*= 255  x = np.clip(x, 0, 255).astype('uint8')  return x  def draw\_activation(activation, figure\_name):  images\_per\_row = 16  n\_features = activation.shape[-1]  size = activation.shape[1]  n\_cols = n\_features // images\_per\_row  display\_grid = np.zeros((size\*n\_cols, images\_per\_row\*size))  for col in range(n\_cols):  for row in range(images\_per\_row):  channel\_image = activation[0, :, :, col\*images\_per\_row+row]  channel\_image = deprocess\_image(channel\_image)  display\_grid[col\*size:(col+1)\*size, row\*size:(row+1)\*size] = channel\_image  scale = 1./size  plt.figure(figsize=(scale\*display\_grid.shape[1], scale\*display\_grid.shape[0]))  plt.title(figure\_name)  plt.grid(False)  plt.imshow(display\_grid, aspect='auto', cmap='viridis')  plt.show()  def gradCAM(model, x):  preds = model.predict(x)  print('Predicted: ', decode\_predictions(preds, top=3)[0])  max\_output = model.output[:, np.argmax(preds)]  last\_conv\_layer = model.get\_layer('block5\_conv3')  grads = K.gradients(max\_output, last\_conv\_layer.output)[0]  pooled\_grads = K.mean(grads, axis=(0, 1, 2))  iterate = K.function([model.input], [pooled\_grads, last\_conv\_layer.output[0]])  pooled\_grads\_value, conv\_layer\_output\_value = iterate([x])  for i in range(512):  conv\_layer\_output\_value[:, :, i] \*= pooled\_grads\_value[i]  heatmap = np.mean(conv\_layer\_output\_value, axis=-1)  heatmap = np.maximum(heatmap, 0)  heatmap /= np.max(heatmap)  return heatmap, conv\_layer\_output\_value, pooled\_grads\_value  heatmap, conv\_output, pooled\_grads = gradCAM(model, img\_tensor)  img = cv2.imread(img\_path)  heatmap = cv2.resize(heatmap, (img.shape[1], img.shape[0]))  heatmap = np.uint8(255\*heatmap)  heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP\_JET)  superimposed\_img = heatmap\*0.4 + img  cv2.imwrite('HW3.jpg', superimposed\_img)  draw\_no = range(256, 256+32, 1)  conv\_activation = np.expand\_dims(conv\_output[:, :, draw\_no], axis=0)  draw\_activation(conv\_activation, 'last\_conv')  plt.matshow(pooled\_grads[draw\_no].reshape(-1, 16), cmap='viridis')  plt.savefig('HW3CAM.jpg') |

|  |
| --- |
|  |

**QE1 (Extra 0.5 points)**

|  |  |  |
| --- | --- | --- |
|  | [None, None]  Before FT | [None, None]  After FT |
| Training  Loss | 0.0323 | 0.7001e-06 |
| Training  Accuracy | 0.9906 | 1 |
| Test  Loss | 0.6501 | 2.742 |
| Test  Accuracy | 0.8109 | 0.7532 |

|  |  |
| --- | --- |
| Loss Graph | Accuracy Graph |

**Your answer:**

Run the code. Does it work?

네, 해당 코드는 정상적으로 동작합니다.

Replace GlobalAveragePooling2D with Flatten. Run the code, does it work?

동작하지 않습니다, Flatten을 사용할때는 Input shape이 None이 아니어야 하기때문입니다.

Does it work better than the best model of Q3? If so, why? If not, why?

Image를 [512,512]로 resizing한 모델이 QE1의 모델보다 Test Accuracy가 훨씬 높다. 따라서 QE1은 best model of Q3보다 좋지 않은 모델이다.