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

Homework #4 (Due: June 11. 2021)

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

**Q1 (2 points)**

|  |
| --- |
| Codes:  from tensorflow.keras import layers, models  from tensorflow.keras import datasets  import matplotlib.pyplot as plt  from tensorflow.keras.layers import Conv2D, MaxPooling2D, BatchNormalization, UpSampling2D, Activation, Dropout, Concatenate, Input  class UNET(models.Model):  def conv(x, n\_f, mp\_flag=True):  x = MaxPooling2D((2, 2), padding='same')(x) if mp\_flag else x  x = Conv2D(n\_f, (3, 3), padding='same')(x)  x = BatchNormalization()(x)  x = Activation('tanh')(x)  x = Dropout(0.05)(x)  x = Conv2D(n\_f, (3, 3), padding='same')(x)  x = BatchNormalization()(x)  x = Activation('tanh')(x)  return x  def deconv\_unet(x, e, n\_f):  x = UpSampling2D((2, 2))(x)  x = Concatenate(axis=3)([x, e])  x = Conv2D(n\_f, (3, 3), padding='same')(x)  x = BatchNormalization()(x)  x = Activation('tanh')(x)  x = Conv2D(n\_f, (3, 3), padding='same')(x)  x = BatchNormalization()(x)  x = Activation('tanh')(x)  return x  def \_\_init\_\_(self, org\_shape):  original = Input(shape=org\_shape)  c1 = UNET.conv(original, 16, mp\_flag=False)  c2 = UNET.conv(c1, 32)  encoded = UNET.conv(c2, 64)  x = UNET.deconv\_unet(encoded, c2, 32)  y = UNET.deconv\_unet(x, c1, 16)  decoded = layers.Conv2D(3, (3, 3), activation='sigmoid', padding='same')(y)  super().\_\_init\_\_(original, decoded)  self.compile(optimizer='adadelta', loss='mse')  class DATA():  def \_\_init\_\_(self):  (x\_train, y\_train), (x\_test, y\_test) = datasets.cifar10.load\_data()  x\_train = x\_train.astype('float32') / 255  x\_test = x\_test.astype('float32') / 255  self.x\_train\_in = DATA.RGB2gray(x\_train)  self.x\_train\_in = DATA.RGB2gray(x\_test)  self.x\_train\_in = x\_train  self.x\_test\_in = x\_test  self.x\_train\_out = x\_train  self.x\_test\_out = x\_test  img\_rows, img\_cols, n\_ch = self.x\_train\_in.shape[1:]  self.input\_shape = (img\_rows, img\_cols, n\_ch)  def RGB2gray(X):  R = X[..., 0:1]  G = X[..., 1:2]  B = X[..., 2:3]  return 0.299 \* R + 0.587 \* G + 0.114 \* B  def show\_images(data, unet):  x\_test\_in = data.x\_test\_in  x\_test\_out = data.x\_test\_out  decoded\_imgs = unet.predict(x\_test\_in)  n = 10  plt.figure(figsize=(20, 6))  for i in range(n):  ax = plt.subplot(3, n, i + 1)  plt.imshow(x\_test\_in[i, :, :, 0], cmap='gray')  ax.get\_xaxis().set\_visible(False)  ax.get\_yaxis().set\_visible(False)  ax = plt.subplot(3, n, i + 1 + n)  plt.imshow(decoded\_imgs[i])  ax.get\_xaxis().set\_visible(False)  ax.get\_yaxis().set\_visible(False)  ax = plt.subplot(3, n, i + 1 + n + n)  plt.imshow(x\_test\_out[i])  ax.get\_xaxis().set\_visible(False)  ax.get\_yaxis().set\_visible(False)  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)  in\_ch = 1  epochs = 100  batch\_size = 512  data = DATA()  unet = UNET(data.input\_shape)  unet.summary()  history = unet.fit(data.x\_train\_in, data.x\_train\_out, epochs=epochs, batch\_size=batch\_size,  shuffle=True, validation\_data=(data.x\_test\_in, data.x\_test\_out))  plot\_loss(history)  plt.savefig('UNET\_LOSS1.png')  plt.clf()  show\_images(data, unet)  plt.savefig('UNET\_PRED1.png')  plt.show() |
| Results: |

**Q2 (2 points)**

|  |
| --- |
| Code:  from tensorflow.keras import layers, models  from tensorflow.keras import datasets  import matplotlib.pyplot as plt  from tensorflow.keras.layers import Conv2D, MaxPooling2D, BatchNormalization, UpSampling2D, Activation, Dropout, Concatenate, Input  class UNET(models.Model):  def conv(x, n\_f, mp\_flag=True):  x = MaxPooling2D((2, 2), padding='same')(x) if mp\_flag else x  x = Conv2D(n\_f, (3, 3), padding='same')(x)  x = BatchNormalization()(x)  x = Activation('tanh')(x)  x = Dropout(0.05)(x)  x = Conv2D(n\_f, (3, 3), padding='same')(x)  x = BatchNormalization()(x)  x = Activation('tanh')(x)  return x  def deconv\_unet(x, e, n\_f):  x = UpSampling2D((2, 2))(x)  x = Concatenate(axis=3)([x, e])  x = Conv2D(n\_f, (3, 3), padding='same')(x)  x = BatchNormalization()(x)  x = Activation('tanh')(x)  x = Conv2D(n\_f, (3, 3), padding='same')(x)  x = BatchNormalization()(x)  x = Activation('tanh')(x)  return x  def \_\_init\_\_(self, org\_shape):  original = Input(shape=org\_shape)  c1 = UNET.conv(original, 16, mp\_flag=False)  c2 = UNET.conv(c1, 32)  c3 = UNET.conv(c2, 64)  encoded = UNET.conv(c3, 128)  x = UNET.deconv\_unet(encoded, c3, 64)  y = UNET.deconv\_unet(x, c2, 32)  z = UNET.deconv\_unet(y, c1, 16)  decoded = layers.Conv2D(3, (3, 3), activation='sigmoid', padding='same')(z)  super().\_\_init\_\_(original, decoded)  self.compile(optimizer='adadelta', loss='mse')  class DATA():  def \_\_init\_\_(self):  (x\_train, y\_train), (x\_test, y\_test) = datasets.cifar10.load\_data()  x\_train = x\_train.astype('float32') / 255  x\_test = x\_test.astype('float32') / 255  self.x\_train\_in = DATA.RGB2gray(x\_train)  self.x\_train\_in = DATA.RGB2gray(x\_test)  self.x\_train\_in = x\_train  self.x\_test\_in = x\_test  self.x\_train\_out = x\_train  self.x\_test\_out = x\_test  img\_rows, img\_cols, n\_ch = self.x\_train\_in.shape[1:]  self.input\_shape = (img\_rows, img\_cols, n\_ch)  def RGB2gray(X):  R = X[..., 0:1]  G = X[..., 1:2]  B = X[..., 2:3]  return 0.299 \* R + 0.587 \* G + 0.114 \* B  def show\_images(data, unet):  x\_test\_in = data.x\_test\_in  x\_test\_out = data.x\_test\_out  decoded\_imgs = unet.predict(x\_test\_in)  n = 10  plt.figure(figsize=(20, 6))  for i in range(n):  ax = plt.subplot(3, n, i + 1)  plt.imshow(x\_test\_in[i, :, :, 0], cmap='gray')  ax.get\_xaxis().set\_visible(False)  ax.get\_yaxis().set\_visible(False)  ax = plt.subplot(3, n, i + 1 + n)  plt.imshow(decoded\_imgs[i])  ax.get\_xaxis().set\_visible(False)  ax.get\_yaxis().set\_visible(False)  ax = plt.subplot(3, n, i + 1 + n + n)  plt.imshow(x\_test\_out[i])  ax.get\_xaxis().set\_visible(False)  ax.get\_yaxis().set\_visible(False)  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)  in\_ch = 1  epochs = 100  batch\_size = 512  fig = True  data = DATA()  unet = UNET(data.input\_shape)  unet.summary()  history = unet.fit(data.x\_train\_in, data.x\_train\_out, epochs=epochs, batch\_size=batch\_size,  shuffle=True, validation\_data=(data.x\_test\_in, data.x\_test\_out))  plot\_loss(history)  plt.savefig('UNET\_LOSS22.png')  plt.clf()  show\_images(data, unet)  plt.savefig('UNET\_PRED22.png')  plt.show() |
| Results: |

**Q3 (2 points)**

|  |
| --- |
| Codes:  from tensorflow.keras import layers, models  from tensorflow.keras import datasets  import matplotlib.pyplot as plt  from tensorflow.keras.layers import Conv2D, MaxPooling2D, BatchNormalization, UpSampling2D, Activation, Dropout, Concatenate, Input  class SCAE(models.Model):  def conv(x, n\_f, mp\_flag=True):  x = MaxPooling2D((2, 2), padding='same')(x) if mp\_flag else x  x = Conv2D(n\_f, (3, 3), padding='same')(x)  x = BatchNormalization()(x)  x = Activation('tanh')(x)  x = Dropout(0.05)(x)  x = Conv2D(n\_f, (3, 3), padding='same')(x)  x = BatchNormalization()(x)  x = Activation('tanh')(x)  return x  def deconv\_scae(x, n\_f):  x = UpSampling2D((2, 2))(x)  x = Conv2D(n\_f, (3, 3), padding='same')(x)  x = BatchNormalization()(x)  x = Activation('tanh')(x)  x = Conv2D(n\_f, (3, 3), padding='same')(x)  x = BatchNormalization()(x)  x = Activation('tanh')(x)  return x  def \_\_init\_\_(self, org\_shape):  original = Input(shape=org\_shape)  c1 = SCAE.conv(original, 16, mp\_flag=False)  c2 = SCAE.conv(c1, 32)  encoded = SCAE.conv(c2, 64)  x = SCAE.deconv\_scae(encoded, 32)  y = SCAE.deconv\_scae(x, 16)  decoded = layers.Conv2D(3, (3, 3), activation='sigmoid', padding='same')(y)  super().\_\_init\_\_(original, decoded)  self.compile(optimizer='adadelta', loss='mse')  class DATA():  def \_\_init\_\_(self):  (x\_train, y\_train), (x\_test, y\_test) = datasets.cifar10.load\_data()  x\_train = x\_train.astype('float32') / 255  x\_test = x\_test.astype('float32') / 255  self.x\_train\_in = DATA.RGB2gray(x\_train)  self.x\_train\_in = DATA.RGB2gray(x\_test)  self.x\_train\_in = x\_train  self.x\_test\_in = x\_test  self.x\_train\_out = x\_train  self.x\_test\_out = x\_test  img\_rows, img\_cols, n\_ch = self.x\_train\_in.shape[1:]  self.input\_shape = (img\_rows, img\_cols, n\_ch)  def RGB2gray(X):  R = X[..., 0:1]  G = X[..., 1:2]  B = X[..., 2:3]  return 0.299 \* R + 0.587 \* G + 0.114 \* B  def show\_images(data, scae):  x\_test\_in = data.x\_test\_in  x\_test\_out = data.x\_test\_out  decoded\_imgs = scae.predict(x\_test\_in)  n = 10  plt.figure(figsize=(20, 6))  for i in range(n):  ax = plt.subplot(3, n, i + 1)  plt.imshow(x\_test\_in[i, :, :, 0], cmap='gray')  ax.get\_xaxis().set\_visible(False)  ax.get\_yaxis().set\_visible(False)  ax = plt.subplot(3, n, i + 1 + n)  plt.imshow(decoded\_imgs[i])  ax.get\_xaxis().set\_visible(False)  ax.get\_yaxis().set\_visible(False)  ax = plt.subplot(3, n, i + 1 + n + n)  plt.imshow(x\_test\_out[i])  ax.get\_xaxis().set\_visible(False)  ax.get\_yaxis().set\_visible(False)  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)  in\_ch = 1  epochs = 100  batch\_size = 512  fig = True  data = DATA()  scae = SCAE(data.input\_shape)  scae.summary()  history = scae.fit(data.x\_train\_in, data.x\_train\_out, epochs=epochs, batch\_size=batch\_size,  shuffle=True, validation\_data=(data.x\_test\_in, data.x\_test\_out))  plot\_loss(history)  plt.savefig('SCAE\_LOSS.png')  plt.clf()  show\_images(data, scae)  plt.savefig('SCAE\_PRED3.png')  plt.show() |
| Results: |

**EQ1 (1 point)**

**Your answer:**

} Extending Q2, can you add more downsampling encoding blocks infinitely? } If not, why?

DownSampling은 무한히 할 수 없다.

계속해서 DownSampling을 하게되면 이미지의 최소 단위보다 작아지기 때문에 문제가 발생하여 실행이 되지 않기때문이다.