**컴퓨터공학과 201702081 최재범 11주차 과제**

- 구현 코드 및 설명

1. Cifar100 Train, Test 데이터를 받아옴 (메모리 한도로 인하여 일부만 가져옴)

   : Train 24000 / Test 6000

```
[4]  import tensorflow as tf
     (x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar100.load_data()

     # 적당히 떼서 사용 : train 24000, test 6000
     x_train = x_train[:24000]
     y_train = y_train[:24000]
     x_test = x_test[:6000]
     y_test = y_test[:6000]

     print('train_data')
     print(x_train.shape)
     print(y_train.shape)

     print('test_data')
     print(x_test.shape)
     print(y_test.shape)
```

```
train_data
(24000, 32, 32, 3)
(24000, 1)
test_data
(6000, 32, 32, 3)
(6000, 1)
```

2. Imagenet 사이즈에 맞추기 위해, (224, 224) 로 데이터셋 크기 변경

```
[5]  # 이미지 리사이징 : (32, 32) -> (224, 224)
     import cv2
     import numpy as np

     x_train_resized = []
     for idx in range(len(x_train)):
       print('\r start ', idx+1, '/ ', len(x_train), end='')
       img = x_train[idx]
       img = cv2.resize(img, (224, 224))
       x_train_resized.append(img)
     print()

     x_test_resized = []
     for idx in range(len(x_test)):
       print('\r start ', idx+1, '/ ', len(x_test), end='')
       img = x_test[idx]
       img = cv2.resize(img, (224, 224))
       x_test_resized.append(img)
     print()

     x_train_resized = np.array(x_train_resized)
     x_test_resized = np.array(x_test_resized)
     x_train = x_train_resized
     x_test = x_test_resized

     print(x_train.shape)
     print(x_test.shape)
```

```
 start  24000 /  24000
 start  6000 /  6000
(24000, 224, 224, 3)
(6000, 224, 224, 3)
```

3. Train 데이터를 Validation과 Train 데이터로 나누고, 섞음 : Train 19200 / Validation 4800

```
[6] from sklearn.model_selection import train_test_split

    # Train을 Train, Validation 데이터로 나누기 (섞음)
    x_train, x_valid, y_train, y_valid = train_test_split(x_train, y_train, test_size=0.2, random_state=123)

    print('train data')
    print(x_train.shape)
    print(y_train.shape)

    print('validation data')
    print(x_valid.shape)
    print(y_valid.shape)
```

```
train data
(19200, 224, 224, 3)
(19200, 1)
validation data
(4800, 224, 224, 3)
(4800, 1)
```

4. 레이블이 스칼라 형태이기 때문에, 이를 One-hot 인코딩 형태로 변환함

```
[7] # Scalar 형태의 레이블을 One-hot Encoding 형태로 변환
    y_train = tf.squeeze(tf.one_hot(y_train, 100), axis=1)
    y_valid = tf.squeeze(tf.one_hot(y_valid, 100), axis=1)
    y_test = tf.squeeze(tf.one_hot(y_test, 100), axis=1)

    print('train data')
    print(x_train.shape)
    print(y_train.shape)

    print('valid data')
    print(x_valid.shape)
    print(y_valid.shape)

    print('test data')
    print(x_test.shape)
    print(y_test.shape)
```

```
train data
(19200, 224, 224, 3)
(19200, 100)
valid data
(4800, 224, 224, 3)
(4800, 100)
test data
(6000, 224, 224, 3)
(6000, 100)
```

5. 데이터셋이 랜덤하게 섞였는지 확인함

```
[8] # 랜덤하게 섞인 데이터셋 수 확인 (Test는 나누지 않았으므로 안섞임)
    print('train data')
    print(np.sum(y_train, axis=0))
    print('validation data')
    print(np.sum(y_valid, axis=0))
```

```
train data
[177. 191. 182. 189. 195. 195. 169. 184. 216. 191. 195. 188. 173. 164.
 202. 194. 210. 179. 195. 206. 191. 199. 195. 197. 166. 189. 196. 201.
 201. 171. 215. 192. 176. 213. 201. 183. 177. 195. 189. 187. 212. 227.
 187. 188. 199. 198. 183. 201. 185. 192. 192. 189. 183. 196. 179. 187.
 202. 203. 192. 184. 200. 190. 188. 201. 173. 225. 197. 207. 199. 201.
 194. 199. 183. 208. 168. 174. 197. 197. 206. 180. 187. 206. 180. 188.
 191. 196. 192. 198. 179. 208. 182. 195. 186. 190. 200. 179. 183. 187.
 193. 185.]
validation data
[46. 48. 46. 47. 39. 47. 38. 45. 53. 40. 30. 51. 54. 51. 47. 51. 57. 47.
 44. 54. 40. 58. 48. 41. 59. 48. 43. 44. 44. 50. 36. 49. 61. 48. 50. 51.
 60. 39. 58. 50. 53. 42. 50. 58. 37. 32. 57. 42. 48. 54. 40. 56. 40. 38.
 59. 51. 39. 50. 48. 65. 31. 58. 58. 59. 54. 60. 36. 56. 48. 42. 47. 53. 54.
 58. 48. 49. 33. 47. 44. 46. 57. 53. 43. 45. 55. 41. 45. 41. 48. 45. 46.
 67. 46. 33. 44. 48. 55. 40. 57. 39. 58.]
```

6. ResNet50 모델 설정 : 모델에 Imagenet 의 데이터를 미리 학습시킴. 따라서 마지막 Fully Connected Layer를 제외한 부분을 학습하지 않도록 설정. 마지막 Fully Connected Layer를 설정할 때, 출력을 데이터셋의 분류 수인 100으로 변경함. Learning Rate는 0.0001로 설정

➔ Trainable Parameter의 개수를 통해, 마지막의 Layer만 학습 가능함을 확인

```
[11] # 모델 설정

     base_model = tf.keras.applications.ResNet50(weights='imagenet', input_shape=(224, 224, 3))
     base_model = tf.keras.models.Model(base_model.inputs, base_model.layers[-2].output)   # Output : 끝에서 2번째 까지 ~ FC Layer 출력 변경위함

     # Imagenet 으로 이미 학습함, 추가학습 X
     base_model.trainable = False

     # 마지막 Fully Connected Layer 수정 : 출력 100개로 변경
     # 이 Layer만 학습하게 됨
     x = base_model.output
     pred = tf.keras.layers.Dense(100, activation='softmax')(x)
     model = tf.keras.models.Model(inputs=base_model.input, outputs=pred)

     opt = tf.keras.optimizers.Adam(learning_rate=0.0001)
     model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['acc'])

     model.summary()
```

```
 conv5_block3_add (Add)          (None, 7, 7, 2048)   0           conv5_block2_out[0][0]
                                                                  conv5_block3_3_bn[0][0]
__________________________________________________________________________________________
 conv5_block3_out (Activation)   (None, 7, 7, 2048)   0           conv5_block3_add[0][0]
__________________________________________________________________________________________
 avg_pool (GlobalAveragePooling2 (None, 2048)         0           conv5_block3_out[0][0]
__________________________________________________________________________________________
 dense_1 (Dense)                 (None, 100)          204900      avg_pool[0][0]
==========================================================================================
 Total params: 23,792,612
 Trainable params: 204,900
 Non-trainable params: 23,587,712
__________________________________________________________________________________________
```

7. Train 데이터를 기반으로 학습하고, Validation 데이터를 기준으로도 정확도 측정. Batch size는 32, epoch 수는 25로 설정

```
[13] # 학습
     history = model.fit(x=x_train, y=y_train, batch_size=32, epochs=25, validation_data=(x_valid, y_valid))
```

8. Accuracy, Loss 변화를 그래프로 출력

```
[14] # Accuracy, Loss 그래프 출력
     import matplotlib.pyplot as plt
     plt.plot(history.history['acc'])
     plt.plot(history.history['val_acc'])
     plt.title('model accuracy')
     plt.ylabel('accuracy')
     plt.xlabel('epoch')
     plt.legend(['train', 'valid'], loc='upper left')
     plt.show()

     plt.plot(history.history['loss'])
     plt.plot(history.history['val_loss'])
     plt.title('model loss')
     plt.ylabel('loss')
     plt.xlabel('epoch')
     plt.legend(['train', 'valid'], loc='upper left')
     plt.show()
```

9. Validation 데이터를 기준으로 한 최고 정확도와, 마지막 Epoch 에서의 정확도 출력

   Test 데이터를 기준으로 한 정확도 출력

```
[15] # Validation 정확도
     print('validation accuracy')
     print(history.history['val_acc'][-1])
     print(np.max(history.history['val_acc']))
```

```
validation accuracy
0.6254166960716248
0.6254166960716248
```

```
[16] # Test 정확도
     results = model.evaluate(x_test, y_test, batch_size=32)

     print('test accuracy')
     print(results[1])
```

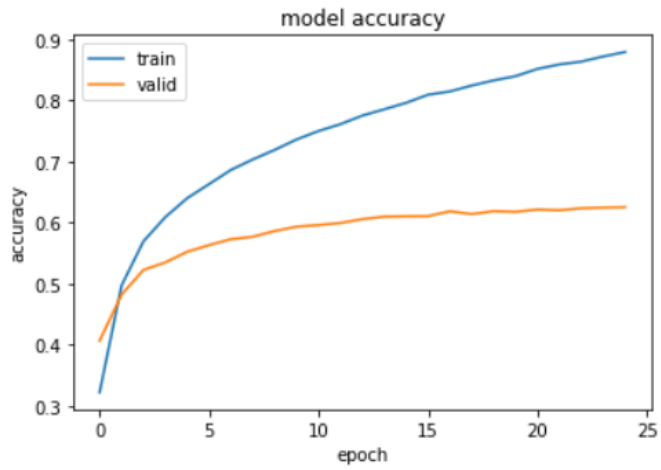
```
188/188 [==============================] - 11s 57ms/step - loss: 1.3505 - acc: 0.6283
test accuracy
0.628333330154419
```

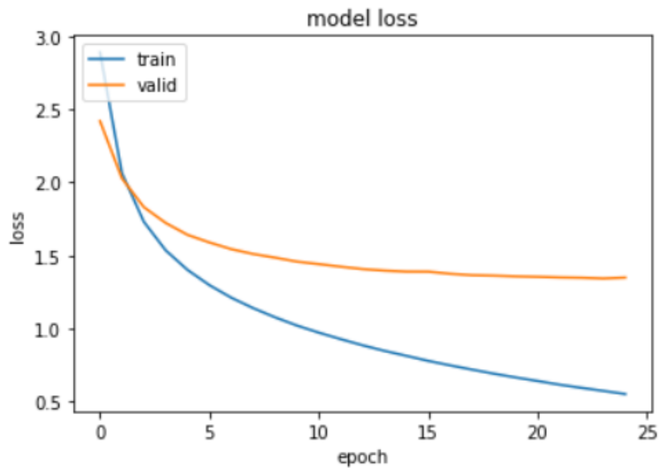
테스트 데이터 기준 약 62.8%의 정확도가 나옴

● 그래프, 학습 결과

```
[13] Epoch 1/25
       2/600 [..............................] - ETA: 31s - loss: 3.6669 - acc: 0.1406WARNING:tensorflow:Callbacks method `on_train_batch_end`
     600/600 [==============================] - ETA: 0s - loss: 2.8893 - acc: 0.3224WARNING:tensorflow:Callbacks method `on_test_batch_end` i
     600/600 [==============================] - 42s 70ms/step - loss: 2.8893 - acc: 0.3224 - val_loss: 2.4189 - val_acc: 0.4069
     Epoch 2/25
     600/600 [==============================] - 41s 68ms/step - loss: 2.0636 - acc: 0.4971 - val_loss: 2.0282 - val_acc: 0.4821
     Epoch 3/25
     600/600 [==============================] - 41s 68ms/step - loss: 1.7315 - acc: 0.5695 - val_loss: 1.8315 - val_acc: 0.5227
     Epoch 4/25
     600/600 [==============================] - 41s 68ms/step - loss: 1.5357 - acc: 0.6092 - val_loss: 1.7231 - val_acc: 0.5350
     Epoch 5/25
     600/600 [==============================] - 41s 68ms/step - loss: 1.4016 - acc: 0.6402 - val_loss: 1.6416 - val_acc: 0.5525
     Epoch 6/25
     600/600 [==============================] - 41s 68ms/step - loss: 1.2976 - acc: 0.6633 - val_loss: 1.5896 - val_acc: 0.5633
     Epoch 7/25
     600/600 [==============================] - 41s 68ms/step - loss: 1.2116 - acc: 0.6866 - val_loss: 1.5439 - val_acc: 0.5731
     Epoch 8/25
     600/600 [==============================] - 41s 68ms/step - loss: 1.1403 - acc: 0.7037 - val_loss: 1.5108 - val_acc: 0.5771
     Epoch 9/25
     600/600 [==============================] - 41s 68ms/step - loss: 1.0777 - acc: 0.7194 - val_loss: 1.4858 - val_acc: 0.5865
     Epoch 10/25
     600/600 [==============================] - 41s 68ms/step - loss: 1.0213 - acc: 0.7364 - val_loss: 1.4586 - val_acc: 0.5935
     Epoch 11/25
     600/600 [==============================] - 41s 68ms/step - loss: 0.9727 - acc: 0.7502 - val_loss: 1.4416 - val_acc: 0.5962
     Epoch 12/25
     600/600 [==============================] - 41s 68ms/step - loss: 0.9284 - acc: 0.7614 - val_loss: 1.4239 - val_acc: 0.5996
     Epoch 13/25
     600/600 [==============================] - 41s 68ms/step - loss: 0.8863 - acc: 0.7754 - val_loss: 1.4076 - val_acc: 0.6058
     Epoch 14/25
     600/600 [==============================] - 41s 68ms/step - loss: 0.8481 - acc: 0.7857 - val_loss: 1.3972 - val_acc: 0.6100
     Epoch 15/25
     600/600 [==============================] - 41s 68ms/step - loss: 0.8142 - acc: 0.7964 - val_loss: 1.3903 - val_acc: 0.6106
     Epoch 16/25
     600/600 [==============================] - 41s 68ms/step - loss: 0.7796 - acc: 0.8096 - val_loss: 1.3899 - val_acc: 0.6108
     Epoch 17/25
     600/600 [==============================] - 41s 68ms/step - loss: 0.7490 - acc: 0.8150 - val_loss: 1.3754 - val_acc: 0.6187
     Epoch 18/25
     600/600 [==============================] - 41s 68ms/step - loss: 0.7201 - acc: 0.8247 - val_loss: 1.3659 - val_acc: 0.6142
     Epoch 19/25
     600/600 [==============================] - 41s 68ms/step - loss: 0.6919 - acc: 0.8331 - val_loss: 1.3629 - val_acc: 0.6190
     Epoch 20/25
     600/600 [==============================] - 41s 68ms/step - loss: 0.6663 - acc: 0.8400 - val_loss: 1.3577 - val_acc: 0.6179
     Epoch 21/25
     600/600 [==============================] - 41s 68ms/step - loss: 0.6419 - acc: 0.8519 - val_loss: 1.3547 - val_acc: 0.6217
     Epoch 22/25
     600/600 [==============================] - 41s 68ms/step - loss: 0.6169 - acc: 0.8592 - val_loss: 1.3506 - val_acc: 0.6204
     Epoch 23/25
     600/600 [==============================] - 41s 69ms/step - loss: 0.5952 - acc: 0.8638 - val_loss: 1.3484 - val_acc: 0.6237
     Epoch 24/25
     600/600 [==============================] - 41s 68ms/step - loss: 0.5744 - acc: 0.8722 - val_loss: 1.3434 - val_acc: 0.6248
     Epoch 25/25
     600/600 [==============================] - 41s 68ms/step - loss: 0.5531 - acc: 0.8795 - val_loss: 1.3496 - val_acc: 0.6254
```

마지막 epoch 에서 Validation accuracy 가 최대가 되었다.




- 느낀 점

당일에 실행할 땐 Epoch당 46초가 걸렸는데, 다음날에 점검할 겸 다시 해보니까 2분씩 걸려서 좀 그랬다

- 과제 난이도

저번 주차에서 가져온 뒤에 약간의 수정만 해서 부담이 덜 했다.