컴퓨터공학과 201702081 최재범 10주차 과제

- 구현 코드 및 설명

1. Cifar100 Train, Test 데이터를 받아옴 : Train 50000 / Test 10000

```
[12] import tensorflow as tf
     import numpy as np

     (x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar100.load_data()

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

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

```
train_data
(50000, 32, 32, 3)
(50000, 1)
test_data
(10000, 32, 32, 3)
(10000, 1)
```

2. Train 데이터를 Validation과 Train 데이터로 나누고, 섞음 : Train 40000 / Validation 10000

```
[32] 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
(40000, 32, 32, 3)
(40000, 1)
validation data
(10000, 32, 32, 3)
(10000, 1)
```

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

```
[33] # 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
(40000, 32, 32, 3)
(40000, 100)
valid data
(10000, 32, 32, 3)
(10000, 100)
test data
(10000, 32, 32, 3)
(10000, 100)
```

4. 데이터셋이 랜덤하게 섞였는지 확인함

```
[52] # 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
[391. 408. 388. 398. 402. 402. 401. 406. 407. 406. 393. 395. 416. 395.
 407. 395. 405. 395. 403. 411. 399. 380. 406. 383. 390. 399. 398. 407.
 393. 379. 405. 408. 405. 409. 403. 400. 413. 402. 402. 387. 398. 409.
 403. 387. 410. 413. 404. 407. 386. 402. 389. 419. 400. 398. 392. 387.
 412. 402. 402. 408. 416. 405. 418. 382. 397. 405. 398. 403. 396. 405.
 385. 390. 388. 381. 393. 411. 393. 410. 411. 380. 407. 421. 399. 401.
 406. 394. 389. 405. 386. 402. 409. 414. 388. 398. 405. 402. 385. 395.
 405. 402.]
validation data
[109.  92. 112. 102.  98.  98.  99.  94.  93.  94. 107. 105.  84. 105.
  93. 105.  95. 105.  97.  89. 101. 120.  94. 117. 110. 101. 102.  93.
 107. 121.  95.  92.  95.  91.  97. 100.  87.  98.  98. 113. 102.  91.
  97. 113.  90.  87.  96.  93. 114.  98. 111.  81. 100. 102. 108. 113.
  88.  98.  98.  92.  84.  95.  82. 118. 103.  95. 102.  97. 104.  95.
 115. 110. 112. 119. 107.  89. 107.  90.  89. 120.  93.  79. 101.  99.
  94. 106. 111.  95. 114.  98.  91.  86. 112. 102.  95.  98. 115. 105.
  95.  98.]
```

5. ResNet50 모델 설정 : Fully Connected Layer의 마지막 층을 데이터셋의 분류 수인 100으로 변경함. Learning Rate는 0.0007로 설정

```
[85] # 모델 설정

     base_model = tf.keras.applications.ResNet50(weights=None, input_shape=(32, 32, 3))
     base_model = tf.keras.models.Model(base_model.inputs, base_model.layers[-2].output)   # Output : 끝에서 2번째 까지 ~ FC Layer 출력 변경위함
     x = base_model.output

     # 출력 100개로 변경
     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=0.0007)
     model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['acc'])

     model.summary()
```

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

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

     Epoch 1/25
     1250/1250 [==============================] - 47s 37ms/step - loss: 5.0147 - acc: 0.0532 - val_loss: 52.4427 - val_acc: 0.0458
     Epoch 2/25
     1250/1250 [==============================] - 46s 36ms/step - loss: 4.5335 - acc: 0.0733 - val_loss: 4.9436 - val_acc: 0.1020
     Epoch 3/25
     1250/1250 [==============================] - 46s 36ms/step - loss: 4.2188 - acc: 0.1092 - val_loss: 4.9494 - val_acc: 0.1170
     Epoch 4/25
     1250/1250 [==============================] - 46s 36ms/step - loss: 4.1091 - acc: 0.1137 - val_loss: 3.9840 - val_acc: 0.1242
     Epoch 5/25
     1250/1250 [==============================] - 46s 37ms/step - loss: 4.0575 - acc: 0.1213 - val_loss: 7.3847 - val_acc: 0.1298
     Epoch 6/25
     1250/1250 [==============================] - 46s 37ms/step - loss: 3.8324 - acc: 0.1507 - val_loss: 8.8591 - val_acc: 0.1395
     Epoch 7/25
     1250/1250 [==============================] - 46s 36ms/step - loss: 3.8175 - acc: 0.1473 - val_loss: 3.5210 - val_acc: 0.1664
     Epoch 8/25
     1250/1250 [==============================] - 46s 37ms/step - loss: 3.6026 - acc: 0.1728 - val_loss: 4.2604 - val_acc: 0.1128
     Epoch 9/25
     1250/1250 [==============================] - 46s 37ms/step - loss: 3.6631 - acc: 0.1590 - val_loss: 4.8054 - val_acc: 0.1907
```

7. Accuracy, Loss 변화를 그래프로 출력

```
[87] # 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()
```

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

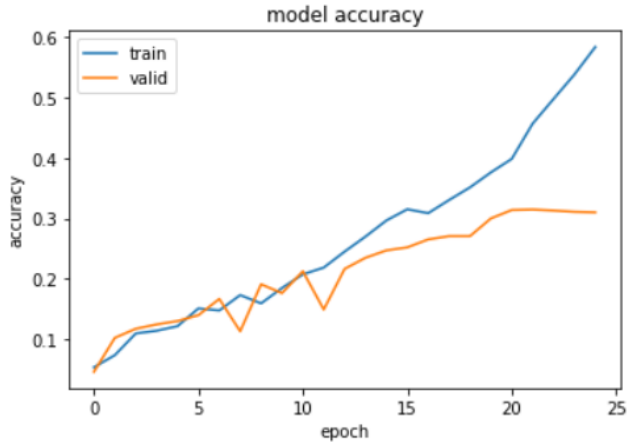
   Test 데이터를 기준으로 한 정확도 출력

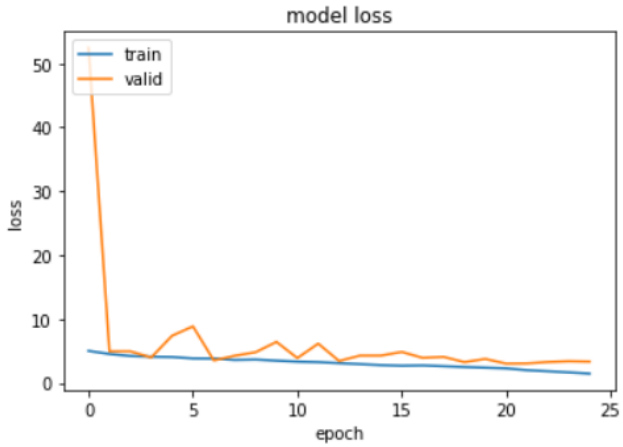
```
[88] # Validation 정확도
     print('validation accuracy')
     print(history.history['val_acc'][-1])
     print(np.max(history.history['val_acc']))

     validation accuracy
     0.30979999899864197
     0.31450000405311584
```

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

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

     313/313 [==============================] - 4s 12ms/step - loss: 3.2264 - acc: 0.3205
     test accuracy
     0.320499986410141
```

   테스트 데이터 기준 약 32%의 정확도가 나옴

- 그래프, 학습 결과





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

```
Epoch 1/25
1250/1250 [==============================] - 47s 37ms/step - loss: 5.0147 - acc: 0.0532 - val_loss: 52.4427 - val_acc: 0.0458
Epoch 2/25
1250/1250 [==============================] - 46s 36ms/step - loss: 4.5335 - acc: 0.0733 - val_loss: 4.9436 - val_acc: 0.1020
Epoch 3/25
1250/1250 [==============================] - 46s 36ms/step - loss: 4.2188 - acc: 0.1092 - val_loss: 4.9494 - val_acc: 0.1170
Epoch 4/25
1250/1250 [==============================] - 46s 36ms/step - loss: 4.1091 - acc: 0.1137 - val_loss: 3.9840 - val_acc: 0.1242
Epoch 5/25
1250/1250 [==============================] - 46s 37ms/step - loss: 4.0575 - acc: 0.1213 - val_loss: 7.3847 - val_acc: 0.1298
Epoch 6/25
1250/1250 [==============================] - 46s 37ms/step - loss: 3.8324 - acc: 0.1507 - val_loss: 8.8591 - val_acc: 0.1395
Epoch 7/25
1250/1250 [==============================] - 46s 36ms/step - loss: 3.8175 - acc: 0.1473 - val_loss: 3.5210 - val_acc: 0.1664
Epoch 8/25
1250/1250 [==============================] - 46s 37ms/step - loss: 3.6026 - acc: 0.1728 - val_loss: 4.2604 - val_acc: 0.1128
Epoch 9/25
1250/1250 [==============================] - 46s 37ms/step - loss: 3.6631 - acc: 0.1590 - val_loss: 4.8054 - val_acc: 0.1907
Epoch 10/25
1250/1250 [==============================] - 46s 37ms/step - loss: 3.4860 - acc: 0.1845 - val_loss: 6.4272 - val_acc: 0.1760
Epoch 11/25
1250/1250 [==============================] - 46s 37ms/step - loss: 3.3319 - acc: 0.2071 - val_loss: 3.8933 - val_acc: 0.2122
Epoch 12/25
1250/1250 [==============================] - 46s 37ms/step - loss: 3.2541 - acc: 0.2183 - val_loss: 6.1552 - val_acc: 0.1489
Epoch 13/25
1250/1250 [==============================] - 46s 37ms/step - loss: 3.0950 - acc: 0.2448 - val_loss: 3.4349 - val_acc: 0.2162
Epoch 14/25
1250/1250 [==============================] - 46s 37ms/step - loss: 2.9463 - acc: 0.2697 - val_loss: 4.2790 - val_acc: 0.2346
Epoch 15/25
1250/1250 [==============================] - 46s 37ms/step - loss: 2.7890 - acc: 0.2964 - val_loss: 4.2791 - val_acc: 0.2468
Epoch 16/25
1250/1250 [==============================] - 46s 37ms/step - loss: 2.6986 - acc: 0.3149 - val_loss: 4.8532 - val_acc: 0.2518
Epoch 17/25
1250/1250 [==============================] - 46s 37ms/step - loss: 2.7417 - acc: 0.3084 - val_loss: 3.9278 - val_acc: 0.2650
Epoch 18/25
1250/1250 [==============================] - 46s 37ms/step - loss: 2.6180 - acc: 0.3302 - val_loss: 4.0809 - val_acc: 0.2704
Epoch 19/25
1250/1250 [==============================] - 46s 37ms/step - loss: 2.4969 - acc: 0.3512 - val_loss: 3.2565 - val_acc: 0.2704
Epoch 20/25
1250/1250 [==============================] - 46s 37ms/step - loss: 2.3916 - acc: 0.3760 - val_loss: 3.7554 - val_acc: 0.2997
Epoch 21/25
1250/1250 [==============================] - 46s 37ms/step - loss: 2.2797 - acc: 0.3984 - val_loss: 3.0234 - val_acc: 0.3137
Epoch 22/25
1250/1250 [==============================] - 46s 37ms/step - loss: 1.9957 - acc: 0.4568 - val_loss: 3.0574 - val_acc: 0.3145
Epoch 23/25
1250/1250 [==============================] - 46s 37ms/step - loss: 1.8244 - acc: 0.4976 - val_loss: 3.2757 - val_acc: 0.3128
Epoch 24/25
1250/1250 [==============================] - 46s 37ms/step - loss: 1.6497 - acc: 0.5381 - val_loss: 3.3955 - val_acc: 0.3107
Epoch 25/25
1250/1250 [==============================] - 46s 37ms/step - loss: 1.4617 - acc: 0.5837 - val_loss: 3.3357 - val_acc: 0.3098
```

22 epoch 에서 Validation accuracy 가 최대가 되었다. 중간중간 정확도가 요동치는 경우가 있어 넉넉하게 설정하였다

- 느낀 점

주식차트 보는거같다

- 과제 난이도

하이퍼파라미터의 범위에 대한 척도가 없어서 시간이 좀 오래 걸렸다