컴퓨터 그래픽스 과제 #10

학과 : 컴퓨터공학과
학번 : 201802161
이름 : 조은빈

1. 구현 코드

```
[10] import tensorflow as tf
     import cv2
     import numpy as np
```

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

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)
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'])
```

```
from sklearn.model_selection import train_test_split

x_train, x_valid, y_train, y_valid = train_test_split(x_train, y_train, test_size=0.2, random_state=123)

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)
```

```
history = model.fit(x=x_train, y=y_train, batch_size=32, epochs=20, validation_data=(x_valid, y_valid))
```

```
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()
```

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

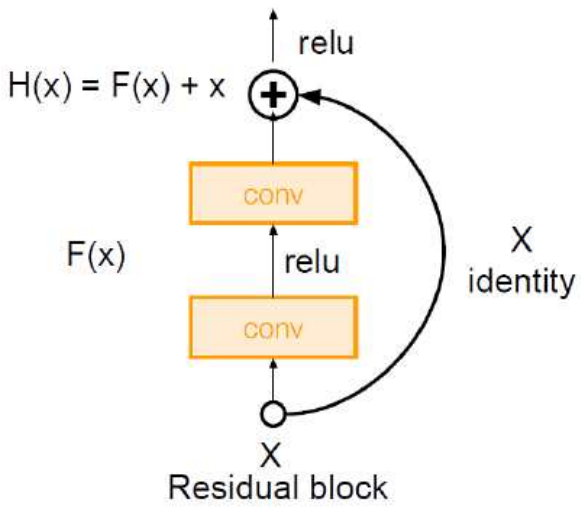
2. 코드 설명

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

Cifar-100 Dataset을 load한다. Cifar-100은 100개의 클래스로 분류되는 이미지들의 집합이다. Dataset은 학습을 위한 train 데이터와 학습된 모델의 정확도를 측정하기 위한 test 데이터로 분류된다.

```
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)
x = base_model.output
pred = tf.keras.layers.Dense(100, activation='softmax')(x)
model = tf.keras.models.Model(inputs=base_model.input, outputs=pred)
```

ResNet50 모델을 불러온다. 예측해야 하는 클래스의 개수가 100개이므로 Dense를 100으로 설정한다. ResNet은 layer의 입력을 출력에 더해주는 구조를 가진다. 출력 H(x)=F(x)+x이므로 F(x)=H(x)-x이다. 출력과 입력의 차이 즉, residual이 0이 되는 방향으로 학습한다.

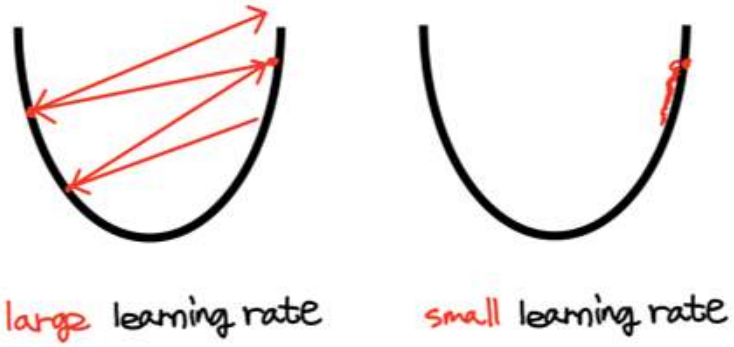



```
opt = tf.keras.optimizers.Adam(learning_rate=0.0001)

model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['acc'])
```

learning_rate를 설정한다. learning rate는 한 번 학습할 때 얼마만큼 학습해야 하는지 학습 양을 의미한다. learning rate가 너무 크면 한 번에 학습하는 양이 커서 학습할 때마다 결과가 크게 변해서 최적의 값에 수렴하지 않고 발산할 수 있다. 반대로 너무 작으면

거의 갱신되지 않기 때문에 학습 속도가 너무 느려지는 문제가 있다. 적절한 learning rate를 찾는게 중요한데 이번 과제에서 0.1, 0.01, 0.001 등을 해봤을 때 learning rate가 클수록 accuracy가 낮게 나왔다. 0.0001까지 낮췄을 때 가장 높은 accuracy를 도출할 수 있었다.




```
x_train, x_valid, y_train, y_valid = train_test_split(x_train, y_train, test_size=0.2, random_state=123)

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)
```

불러온 데이터 셋은 train 데이터와 test 데이터로 나누어져 있다. 학습할 때마다 정확도를 측정해보기 위한 validation 데이터를 train 데이터에서 뽑아낸다.

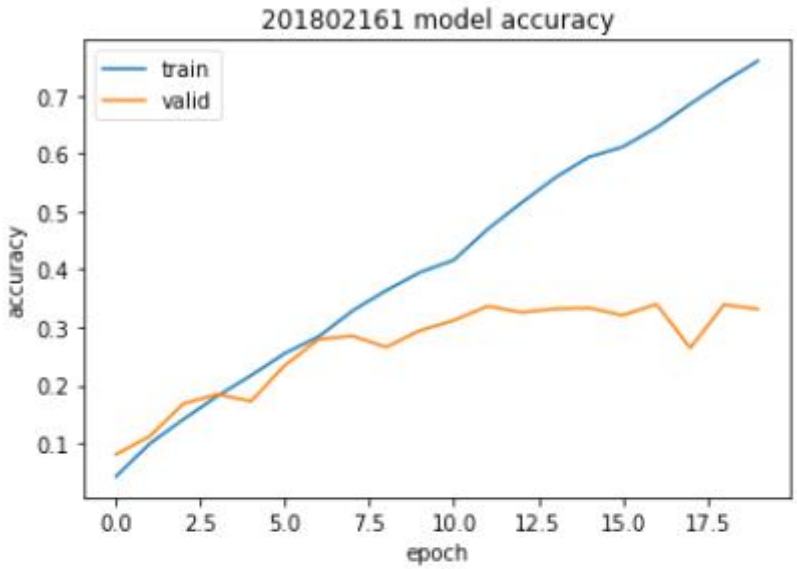
결과 데이터를 one-hot 인코딩한다. 즉. 정답인 클래스만 1이고 나머지는 0인 형태로 바꿔준다.

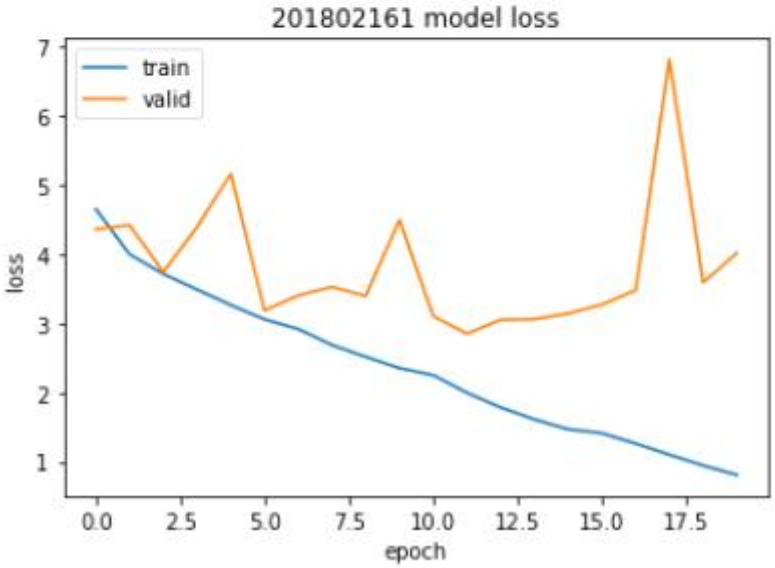
```
history = model.fit(x=x_train, y=y_train, batch_size=20, epochs=20, validation_data=(x_valid, y_valid))
```

batch는 한 번에 확인할 데이터의 양이다. 전체 데이터 셋을 한 번에 학습할 수 없기 때문에 batch size만큼 일정하게 쪼개서 학습한다. 전체 데이터 셋을 epoch만큼 돌면서 모델을 학습시킨다. epoch이 너무 작으면 loss가 줄어들지 않는 underfitting 문제가 발생하고 너무 크면 학습 데이터에 대해서만 학습되는 overfitting 문제가 발생한다. 따라서 적절한 epoch을 설정해줘야 한다. epoch을 10으로 설정했을 땐 학습 데이터에 대한 정답률과 테스트 데이터에 대한 정답률 모두 높지 않게 출력되었다. 50으로 설정했을 땐 학습 데이터에 대한 정답률은 90% 이상이었지만 테스트 데이터에 대한 정답률은 30% 이하인 것을 확인하였다.

3. 이미지

```
Epoch 1/20
2000/2000 [==============================] - 70s 35ms/step - loss: 4.6536 - acc: 0.0424 - val_loss: 4.3612 - val_acc: 0.0804
Epoch 2/20
2000/2000 [==============================] - 69s 34ms/step - loss: 3.9982 - acc: 0.0992 - val_loss: 4.4220 - val_acc: 0.1120
Epoch 3/20
2000/2000 [==============================] - 69s 34ms/step - loss: 3.7153 - acc: 0.1412 - val_loss: 3.7412 - val_acc: 0.1684
Epoch 4/20
2000/2000 [==============================] - 69s 34ms/step - loss: 3.4865 - acc: 0.1812 - val_loss: 4.3935 - val_acc: 0.1843
Epoch 5/20
2000/2000 [==============================] - 68s 34ms/step - loss: 3.2644 - acc: 0.2168 - val_loss: 5.1616 - val_acc: 0.1724
Epoch 6/20
2000/2000 [==============================] - 68s 34ms/step - loss: 3.0577 - acc: 0.2554 - val_loss: 3.1869 - val_acc: 0.2344
Epoch 7/20
2000/2000 [==============================] - 68s 34ms/step - loss: 2.9144 - acc: 0.2839 - val_loss: 3.4004 - val_acc: 0.2792
Epoch 8/20
2000/2000 [==============================] - 68s 34ms/step - loss: 2.6834 - acc: 0.3286 - val_loss: 3.5255 - val_acc: 0.2850
Epoch 9/20
2000/2000 [==============================] - 68s 34ms/step - loss: 2.5149 - acc: 0.3635 - val_loss: 3.3940 - val_acc: 0.2657
Epoch 10/20
2000/2000 [==============================] - 68s 34ms/step - loss: 2.3495 - acc: 0.3953 - val_loss: 4.4963 - val_acc: 0.2946
Epoch 11/20
2000/2000 [==============================] - 68s 34ms/step - loss: 2.2466 - acc: 0.4161 - val_loss: 3.1052 - val_acc: 0.3123
Epoch 12/20
2000/2000 [==============================] - 68s 34ms/step - loss: 1.9953 - acc: 0.4694 - val_loss: 2.8476 - val_acc: 0.3366
Epoch 13/20
2000/2000 [==============================] - 68s 34ms/step - loss: 1.7811 - acc: 0.5150 - val_loss: 3.0514 - val_acc: 0.3260
Epoch 14/20
2000/2000 [==============================] - 68s 34ms/step - loss: 1.6062 - acc: 0.5586 - val_loss: 3.0595 - val_acc: 0.3320
Epoch 15/20
2000/2000 [==============================] - 68s 34ms/step - loss: 1.4651 - acc: 0.5943 - val_loss: 3.1415 - val_acc: 0.3336
Epoch 16/20
2000/2000 [==============================] - 68s 34ms/step - loss: 1.4095 - acc: 0.6119 - val_loss: 3.2707 - val_acc: 0.3210
Epoch 17/20
2000/2000 [==============================] - 68s 34ms/step - loss: 1.2608 - acc: 0.6450 - val_loss: 3.4753 - val_acc: 0.3400
Epoch 18/20
2000/2000 [==============================] - 68s 34ms/step - loss: 1.0957 - acc: 0.6860 - val_loss: 6.8211 - val_acc: 0.2642
Epoch 19/20
2000/2000 [==============================] - 68s 34ms/step - loss: 0.9406 - acc: 0.7245 - val_loss: 3.5882 - val_acc: 0.3393
Epoch 20/20
2000/2000 [==============================] - 68s 34ms/step - loss: 0.8054 - acc: 0.7606 - val_loss: 4.0185 - val_acc: 0.3316
```






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

```
validation accuracy
0.33160001039505005
0.3400000035762787
```

```
[17] results = model.evaluate(x_test, y_test, batch_size=32)

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

```
313/313 [==============================] - 3s 10ms/step - loss: 3.9867 - acc: 0.3465
test accuracy
0.3465000092983246
```

4. 느낀점

이전 과제들은 정답인지 아닌지 판별하기 어려웠는데 이번 과제는 accuracy 수치로 확인할 수 있어서 좋았다.

5. 과제 난이도

모델을 load해서 사용하는 것은 크게 어렵지 않았지만 실제로 모델을 만드는 것은 어려울 것 같다.