2020년 컴퓨터그래픽스

- HW 10 -

제출일자	2020.12.02.
이 름	장수훈
학 번	201402414
분 반	00

구현코드

```
def vgg16(input_shape = (244,244,3)):
  model = tf.keras.models.Sequential()

  model.add(tf.keras.layers.Conv2D(input_shape=input_shape, filters = 64, kernel_size=(3,3), strides = (1,1), activation= 'relu', padding = 'same'))
  model.add(tf.keras.layers.Conv2D(filters= 64, kernel_size = (3,3), strides= (1,1), activation='relu', padding='same'))
  model.add(tf.keras.layers.MaxPooling2D(pool_size=(2,2)))

  model.add(tf.keras.layers.Conv2D(filters= 128, kernel_size = (3,3), strides= (1,1), activation='relu', padding='same'))
  model.add(tf.keras.layers.Conv2D(filters= 128, kernel_size = (3,3), strides= (1,1), activation='relu', padding='same'))
  model.add(tf.keras.layers.MaxPooling2D(pool_size=(2,2)))

  model.add(tf.keras.layers.Conv2D(filters= 256, kernel_size = (3,3), strides= (1,1), activation='relu', padding='same'))
  model.add(tf.keras.layers.Conv2D(filters= 256, kernel_size = (3,3), strides= (1,1), activation='relu', padding='same'))
  model.add(tf.keras.layers.Conv2D(filters= 256, kernel_size = (3,3), strides= (1,1), activation='relu', padding='same'))
  model.add(tf.keras.layers.MaxPooling2D(pool_size=(2,2)))

  model.add(tf.keras.layers.Conv2D(filters= 512, kernel_size = (3,3), strides= (1,1), activation='relu', padding='same'))
  model.add(tf.keras.layers.Conv2D(filters= 512, kernel_size = (3,3), strides= (1,1), activation='relu', padding='same'))
  model.add(tf.keras.layers.Conv2D(filters= 512, kernel_size = (3,3), strides= (1,1), activation='relu', padding='same'))
  model.add(tf.keras.layers.MaxPooling2D(pool_size=(2,2)))

  model.add(tf.keras.layers.Conv2D(filters= 512, kernel_size = (3,3), strides= (1,1), activation='relu', padding='same'))
  model.add(tf.keras.layers.Conv2D(filters= 512, kernel_size = (3,3), strides= (1,1), activation='relu', padding='same'))
  model.add(tf.keras.layers.Conv2D(filters= 512, kernel_size = (3,3), strides= (1,1), activation='relu', padding='same'))
  model.add(tf.keras.layers.MaxPooling2D(pool_size=(2,2)))

  model.add(tf.keras.layers.Flatten())
  model.add(tf.keras.layers.Dense(4096, activation='relu'))
  model.add(tf.keras.layers.Dense(4096, activation='relu'))
  model.add(tf.keras.layers.Dense(10, activation='softmax'))

  return model
```

vgg16모델이다. 실습에서는 직접 구현하는 방법으로 사용하였다. shape를 인풋으로 받고 layer의 개수가 16개가 되도록 하였다.

```
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, 10), axis=1)
y_valid = tf.squeeze(tf.one_hot(y_valid, 10), axis=1)
y_test = tf.squeeze(tf.one_hot(y_test, 10), axis=1)
```

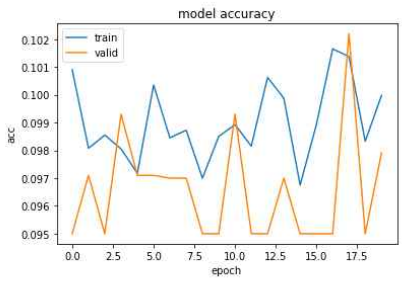
```
history = vgg_cifar10.fit(x_train, y_train, batch_size=40, epochs=20, validation_data=(x_valid, y_valid))
```

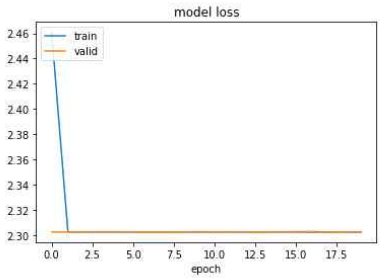
학습할 데이터를 split 함수를 이용해 train과 valid로 나누었다.
train은 모델을 학습하는데 사용하고 validation은 모델의 성능을 평가하는데 사용하였다.

```
history = vgg_cifar10.fit(x_train, y_train, batch_size=40, epochs=20, validation_data=(x_valid, y_valid))

Epoch 1/20
   1/1000 [..............................] - ETA: 0s - loss: 2.2975 - acc: 0.1000WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared to t
 999/1000 [============================>.] - ETA: 0s - loss: 2.4612 - acc: 0.1009WARNING:tensorflow:Callbacks method `on_test_batch_end` is slow compared to th
1000/1000 [==============================] - 38s 38ms/step - loss: 2.4610 - acc: 0.1009 - val_loss: 2.3028 - val_acc: 0.0950
Epoch 2/20
1000/1000 [==============================] - 37s 37ms/step - loss: 2.3028 - acc: 0.0981 - val_loss: 2.3027 - val_acc: 0.0971
Epoch 3/20
1000/1000 [==============================] - 37s 37ms/step - loss: 2.3027 - acc: 0.0985 - val_loss: 2.3030 - val_acc: 0.0950
Epoch 4/20
1000/1000 [==============================] - 37s 37ms/step - loss: 2.3028 - acc: 0.0980 - val_loss: 2.3030 - val_acc: 0.0993
Epoch 5/20
1000/1000 [==============================] - 37s 37ms/step - loss: 2.3027 - acc: 0.0972 - val_loss: 2.3030 - val_acc: 0.0971
Epoch 6/20
1000/1000 [==============================] - 37s 37ms/step - loss: 2.3027 - acc: 0.1003 - val_loss: 2.3029 - val_acc: 0.0971
Epoch 7/20
1000/1000 [==============================] - 37s 37ms/step - loss: 2.3027 - acc: 0.0984 - val_loss: 2.3027 - val_acc: 0.0970
Epoch 8/20
1000/1000 [==============================] - 37s 37ms/step - loss: 2.3027 - acc: 0.0987 - val_loss: 2.3028 - val_acc: 0.0970
Epoch 9/20
1000/1000 [==============================] - 37s 37ms/step - loss: 2.3027 - acc: 0.0970 - val_loss: 2.3028 - val_acc: 0.0950
Epoch 10/20
1000/1000 [==============================] - 37s 37ms/step - loss: 2.3027 - acc: 0.0985 - val_loss: 2.3030 - val_acc: 0.0950
Epoch 11/20
1000/1000 [==============================] - 37s 37ms/step - loss: 2.3028 - acc: 0.0989 - val_loss: 2.3029 - val_acc: 0.0993
Epoch 12/20
1000/1000 [==============================] - 37s 37ms/step - loss: 2.3028 - acc: 0.0981 - val_loss: 2.3030 - val_acc: 0.0950
Epoch 13/20
1000/1000 [==============================] - 37s 37ms/step - loss: 2.3027 - acc: 0.1006 - val_loss: 2.3029 - val_acc: 0.0950
Epoch 14/20
1000/1000 [==============================] - 37s 37ms/step - loss: 2.3027 - acc: 0.0999 - val_loss: 2.3028 - val_acc: 0.0970
Epoch 15/20
1000/1000 [==============================] - 37s 37ms/step - loss: 2.3028 - acc: 0.0967 - val_loss: 2.3030 - val_acc: 0.0950
Epoch 16/20
1000/1000 [==============================] - 37s 37ms/step - loss: 2.3027 - acc: 0.0989 - val_loss: 2.3030 - val_acc: 0.0950
Epoch 17/20
1000/1000 [==============================] - 37s 37ms/step - loss: 2.3027 - acc: 0.1016 - val_loss: 2.3032 - val_acc: 0.0950
Epoch 18/20
1000/1000 [==============================] - 37s 37ms/step - loss: 2.3027 - acc: 0.1014 - val_loss: 2.3027 - val_acc: 0.1022
Epoch 19/20
1000/1000 [==============================] - 37s 37ms/step - loss: 2.3027 - acc: 0.0983 - val_loss: 2.3028 - val_acc: 0.0950
Epoch 20/20
1000/1000 [==============================] - 37s 37ms/step - loss: 2.3027 - acc: 0.1000 - val_loss: 2.3029 - val_acc: 0.0979
```

batch size를 40, epochs를 20으로하여 학습을 시켜보았다.

```
import matplotlib.pyplot as plt

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





위의 학습 결과 시각화

```
vgg_cifar10 = vgg16(input_shape = (32, 32, 3))

opt = tf.keras.optimizers.Adam(learning_rate=0.05)
vgg_cifar10.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['acc'])

vgg_cifar10.summary()
```

cifar-10 dataset으로 학습하는데 learning late를 적절히 조절하여 정확도를 높혀보았다.

```
import numpy as np
from google.colab import drive
drive.mount('/content/drive')

imgs = np.load('drive/MyDrive/imgs.npy')
labels = np.load('drive/MyDrive/labels.npy')

print(imgs.shape)
print(labels.shape)
```

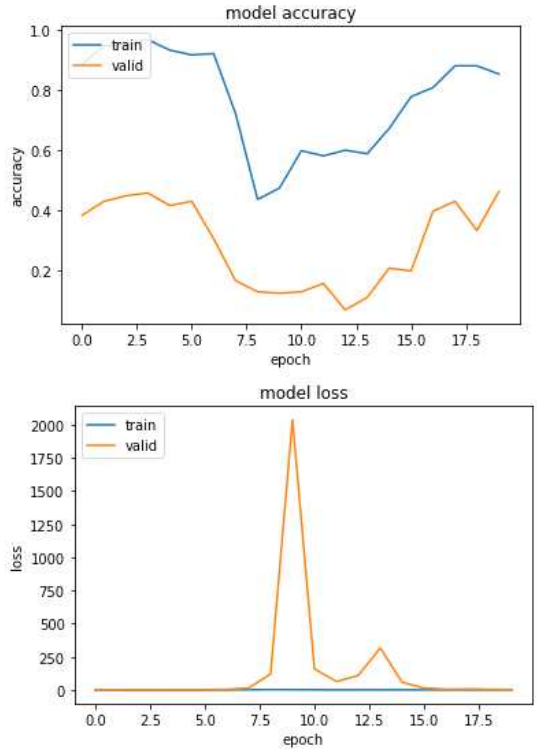
```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
(1080, 299, 299, 3)
(1080,)
```

```
import cv2
print(cv2.__version__)

#resize (299, 299) -> (32, 32)
x_train = []
for idx in range(len(imgs)):
  print('₩r start', idx+1, '/', len(imgs), end='')
  img = imgs[idx]
  img = cv2.resize(img, (32, 32))
  x_train.append(img)
print() #end

x_train = np.array(x_train)
print(x_train.shape)
```

```
4.1.2
 start 1080 / 1080
(1080, 32, 32, 3)
```

ResNet을 실습하기위해 dataset을 다운받아 불러와 사이즈를 resize 해주었다.

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




```
results = model.evaluate(x_test, y_test, batch_size=40)
```

```
5/5 [==============================] - 0s 8ms/step - loss: 2.6788 - acc: 0.4451
```

모델을 직접 구현하지 않고 tensorflow에 있는 모델을 사용하여 학습 시킨 결과이다.

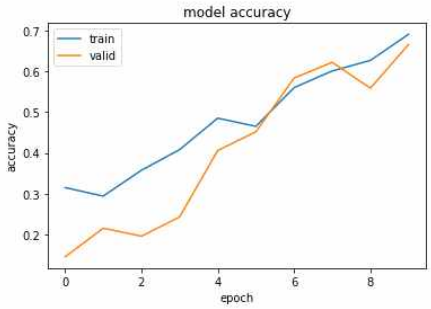
44.51%로 30%를 넘겼다.

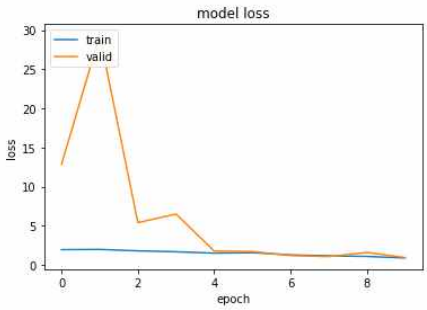
```
x_data = []
for i in range(len(x_train)):
  img = cv2.resize(x_train[i], (75, 75))
  x_data.append(img)

x_train = np.array(x_data)

x_data = []
for i in range(len(x_valid)):
  img = cv2.resize(x_valid[i], (75, 75))
  x_data.append(img)

x_valid = np.array(x_data)

x_data = []
for i in range(len(x_test)):
  img = cv2.resize(x_test[i], (75, 75))
  x_data.append(img)

x_test = np.array(x_data)
```

```
base_model = tf.keras.applications.InceptionV3(weights=None, input_shape=(75, 75, 3))
base_model = tf.keras.models.Model(base_model.inputs, base_model.layers[-2].output)
x = base_model.output
pred = tf.keras.layers.Dense(10, activation='softmax')(x)
model = tf.keras.models.Model(inputs=base_model.input, outputs=pred)

opt = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['acc'])

model.summary()
```

Inception network 실습이다. 입력 사이즈를 75x75 로 바꿔주기 위한 코드이다
activation 함수를 softmax로 이용하여 학습시킨 결과

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






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

313/313 [==============================] - 5s 15ms/step - loss: 0.9372 - acc: 0.6701
```

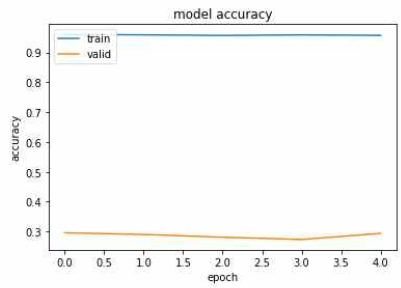
위의 결과를 시각화 하였다. 67프로인 준수한 성능인 것 같다.

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

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-100-python.tar.gz
169009152/169001437 [==============================] - 3s 0us/step
```

```
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=0.001)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['acc'])

model.summary()
```

ResNet50 네트워크 학습시키는 과제이다. cifar-100 데이터셋을 학습시켜 정확도를 30%이상 나오게 하는 것이 목적이다.




```
import numpy as np

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

results = model.evaluate(x_test, y_test, batch_size=40)

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

```
validation accuracy
0.2939000129699707
0.2960000381469727
250/250 [==============================] - 2s 10ms/step - loss: 8.0751 - acc: 0.2952
test accuracy
0.295199990272522
```

정확도가 이것밖에 나오지 않았다..

느낀점

컴퓨터 그래픽스, 기계학습 전부 이번학기에 처음 접하게 된 과목이고, 이 두 수업을 같이 듣길 잘했다는 생각을 하게되었다.

난이도

무난했던 것 같다.