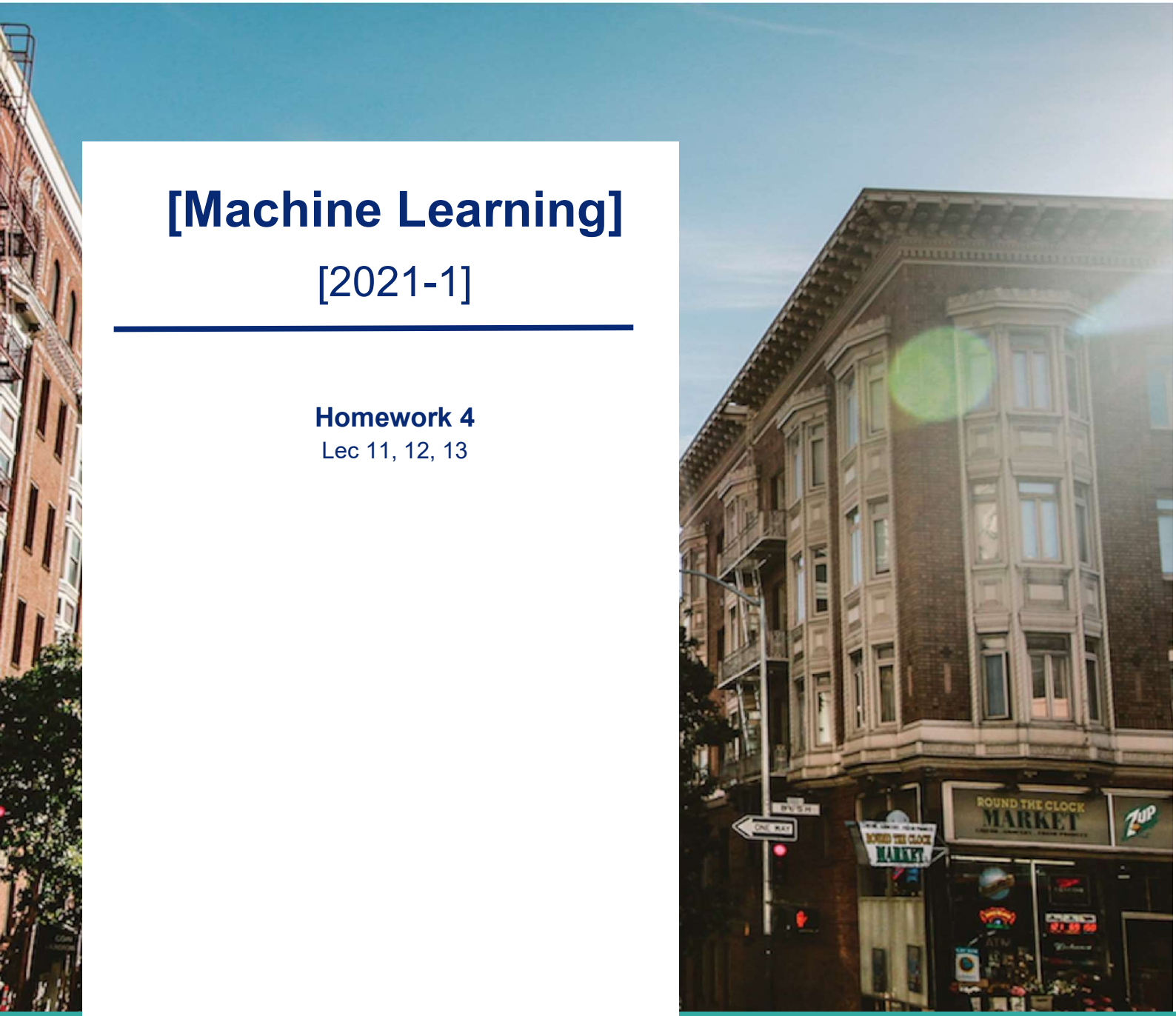
[Machine Learning]

[2021-1]

Homework 4

Lec 11, 12, 13



[DATE] 2020.06.04

Student ID : 2016112158

Name : 김희수

Professor : Juntae Kim

1. Explain the differences between *K-means* and *DBSCAN,* and discuss the advantages and disadvantages. (10pts)

Your Answer
Kmeans 는 중심기반 클러스터링으로 “유사한 데이터들은 중심점 주위에 분포할 것이다”는 가정이 깔려있다. k 개의 중심점이 주어졌을 때, 각 데이터와 중심점 간의 거리를 계산하고, 거리가 최소화되도록 중심점을 이동시킨다. 즉, Kmeans 는 초기에 중심점의 개수를 지정해줘야 한다. DBSCAN 은 밀도기반 클러스터링으로 “유사한 데이터들은 서로 가까이에 분포한다”는 가정이 깔려있다. 따라서 각 데이터들이 얼마나 서로 가까이 있으면 유사한 데이터라고 판단할지를 결정하는 eps 가 필요하며 그 eps 안에 최소 어느정도의 데이터가 있어야 그것을 군집이라 판단할 것인지를 결정할 minPts 가 필요하다. K-means 는 데이터에 대해 미리 알 필요 없이, 관찰 데이터 간의 거리만을 활용해 군집을 형성하므로 굉장히 쉽다는 장점이 있다. 반면 이상치가 전체 거리에 영향을 주어 중심점을 잘못 업데이트 할 수 있다는 단점이 있다. 또한, 초기 중심점의 개수를 사전에 알아둬야한다. DBSCAN 은 군집의 수를 설정할 필요가 없고, 노이즈를 판별할 수 있어 이상치에도 대응이 가능하다. 하지만 밀도가 높은 곳에 집중하기 때문에 밀도가 낮은 곳의 데이터는 하나의 군집으로 인식하지 못하는 경우가 존재한다.

2. Explain what the dropout is in deep neural network model, and what kind of effect you can expect by applying dropout. (10pts)

Your Answer
드랍아웃은 학습을 진행할때, 신경망내에서 무작위로 일부 뉴런만 사용하고, 다른 일부 뉴런은 학습에 이용하지 않는 즉, off 해놓는 방법이다. 드랍아웃을 적용함으로써 모델은 좀더 robust 해지고 그 결과 오버피팅을 방지할 수 있다.

3. Describe how the convolution layer works in CNN. Compute the total number of parameters in CNN with following architecture. Stride=1, no padding. Show how you calculate it. (10pts)

- input: 32 x 32 x 3 image
- conv layer 1: 8 filters of 3 x 3 size + 2 x 2 max pooling
- conv layer 2: 16 filters of 3 x 3 size + 2 x 2 max pooling

- dense layer: 10 outputs

Your Answer
합성곱 레이어는 이미지의 일부와 filter 를 점곱하고 그 결과를 feature map 에 저장한다. filter 는 stride 만큼 이동하면서 점곱의 대상이 되는 이미지의 일부를 변경한다. 그렇게 이동하면서 점곱한 결과들이 feature map 에 저장된다. 따라서 filter 의 개수는 feature map 의 개수와 동일하게 된다. 첫번째 Conv layer : (kernel_size * input_channel + 1) * (kernel 개수 units) = (3 * 3 * 3 + 1) * 8 = 224 두번째 Conv layer : (3 * 3 * 8 + 1) * 16 = 1168 마지막 dense layer : (앞 layer 의 kernel 개수 units + 1) * (dense layer output units 수) = (16 + 1) * 10 = 170 따라서 전체 파라미터 수는 = 224 + 1168 + 170 = 1562

4. Describe how the backpropagation learning works in RNN. Explain the main problem of the RNN, and how LSTM can solve the problem. (10pts)

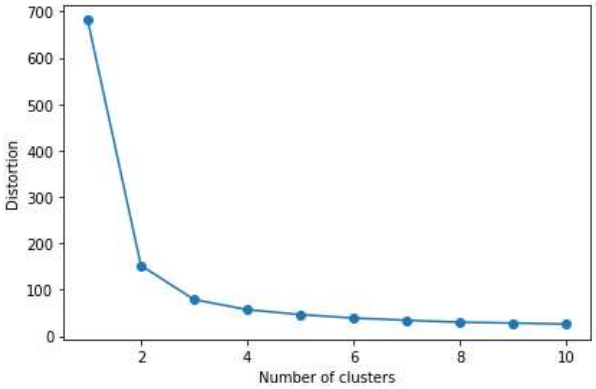
Your Answer
RNN 에서 backpropagation 이 진행될 때 Loss 의 W 에 대한 편미분값의 계산은 다음과 같다. $\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$. 이때 $\frac{\partial L_t}{\partial W} = \frac{\partial L_t}{\partial h_t} * (\prod_t \tanh` * W)$ 에서 tanh 함수의 gradient 값이 vanishing 되는 문제가 존재한다. LSTM 에선 cell state 와 들어온 정보들(이전 cell 의 cell state 값, 이전 cell 의 h 값, input x 값)을 선택적으로 사용하는 여러 게이트들(input gate, forget gate, output gate)의 도입으로 인해 이러한 문제를 해결할 수 있었다.

5. Unsupervised Learning (30pts)

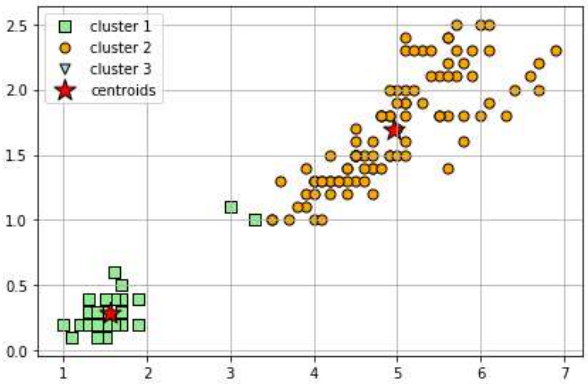
5-1. Perform k-means clustering on the iris dataset. You should choose features [2,3] – Petal length and Petal width.

Expected Output

- Distortions Plotting

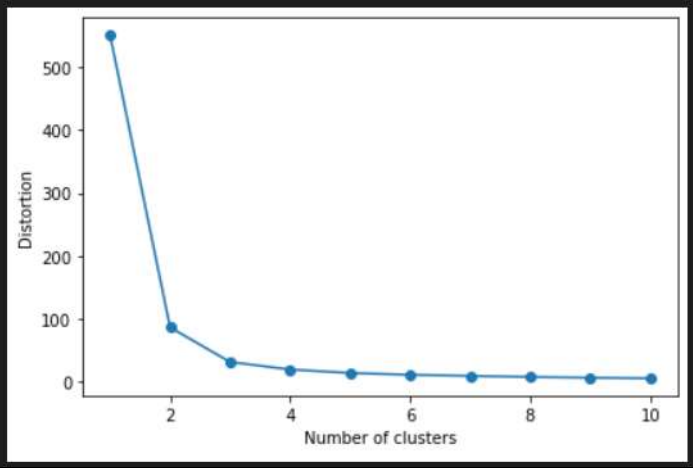


- Clustering Result

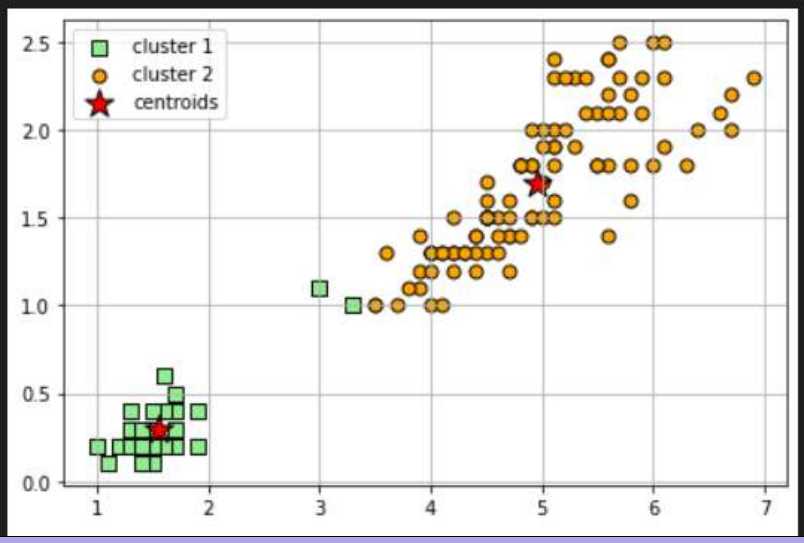


Code
hw4-5.ipynb 참고
Result(Captured images)

```python
# plotting distortions for k = 1 to 11
import matplotlib.pyplot as plt
distortions = []
for i in range(1, 11):
    km = KMeans(n_clusters=i, init="k-means++", max_iter=300, random_state=0)
    km.fit(df_X)
    distortions.append(km.inertia_)

plt.plot(range(1, 11), distortions, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Distortion')
plt.tight_layout()

plt.show()
```



```
# plotting clusters and centers
plt.scatter(df_X[y_km == 0, 0],
            df_X[y_km == 0, 1],
            s=50, c='lightgreen',
            marker='s', edgecolor='black',
            label='cluster 1')
plt.scatter(df_X[y_km == 1, 0],
            df_X[y_km == 1, 1],
            s=50, c='orange',
            marker='o', edgecolor='black',
            label='cluster 2')
plt.scatter(km.cluster_centers_[:, 2],
            km.cluster_centers_[:, 3],
            s=250, marker='*',
            c='red', edgecolor='black',
            label='centroids')

plt.legend(scatterpoints=1)
plt.grid()
plt.tight_layout()

plt.show()
```



Description

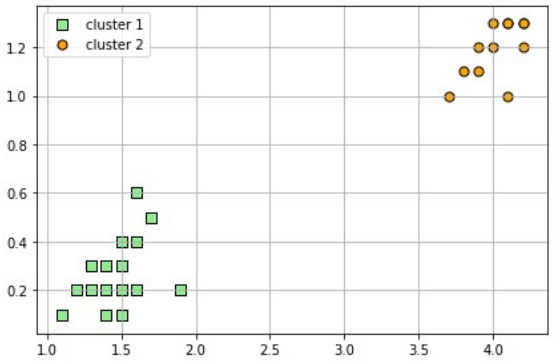
sklearn.datasets 의 load_iris()로 iris 데이터를 불러오고 petal length, petal width 에 해당하는 2 번째, 3 번째 column 에 해당하는 데이터를 df_X 에 저장하였다. 먼저 df_X 를 시각화하여 그 분포를 살펴보았다. 그 후 KMeans 함수의 가장 적절한 cluster 수를 찾기 위해 distortion 을 살펴보고, n_cluster=2 일때가 가장 적절하다는걸 확인했다. 그 후 n_cluster=2 인 KMeans

객체를 생성해서 predict 한 결과를 y_km 에 저장하였다. y_km 을 바탕으로 군집에 맞게 색과 모양을 달리하여 시각화하였다.

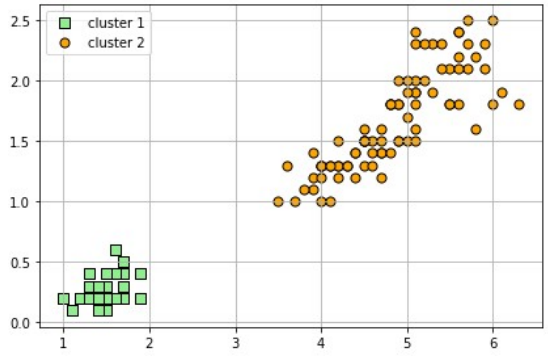
5-2. Perform DBSCAN clustering on the iris dataset. Consider what hyperparameters (epsilon, minPts) you should choose to get results similar to the above K-means results.

Expected Output

- Case example : Wrong hyperparameters are selected

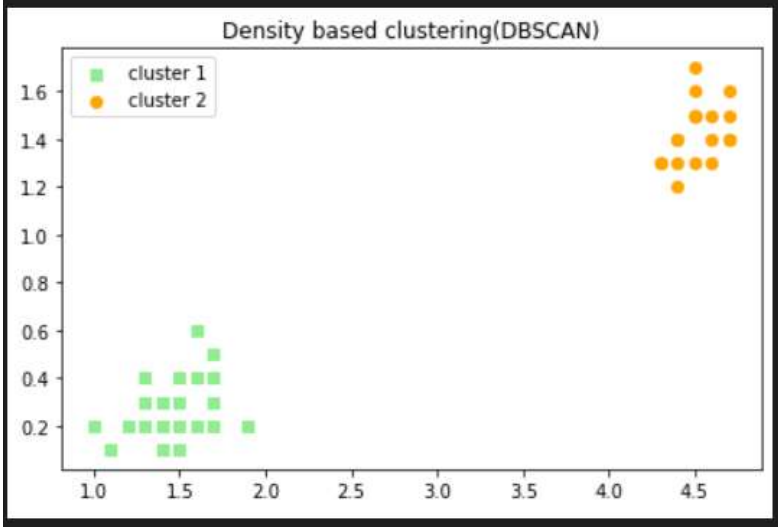


- Case example : Good hyperparameters are selected

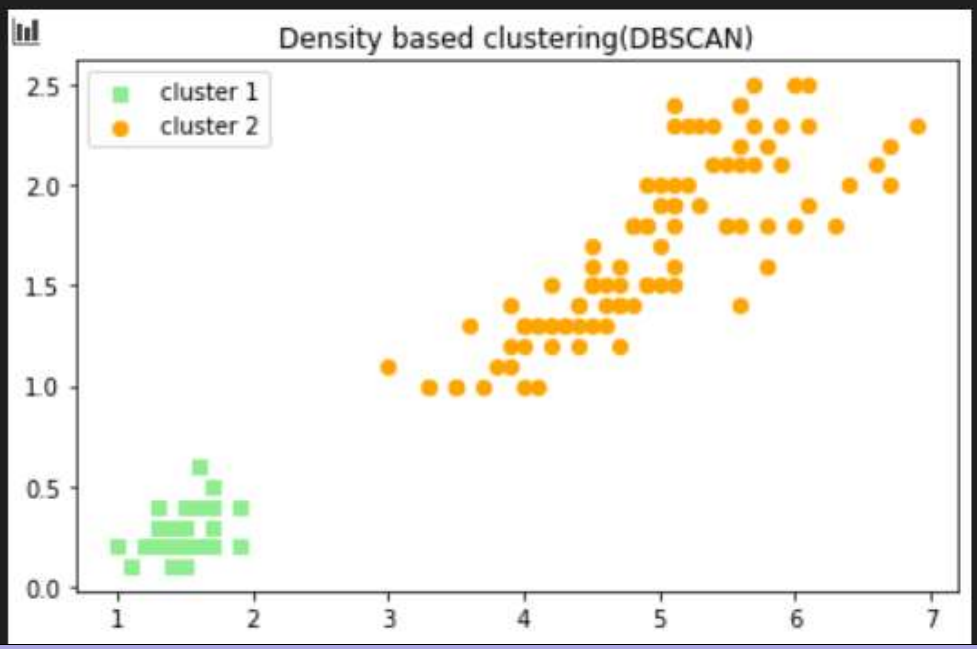


Code
hw4-5.ipynb 참고
Result(Captured images)

```
from sklearn.cluster import DBSCAN


wrogngdb = DBSCAN(eps=0.2, min_samples=10)
wrong_y_db = wrogngdb.fit_predict(df_X)
plt.scatter(df_X[wrong_y_db == 0, 0], df_X[wrong_y_db == 0, 1],
            c='lightgreen', marker='s', s=40,
            label='cluster 1')

plt.scatter(df_X[wrong_y_db == 1, 0], df_X[wrong_y_db == 1, 1],
            c='orange', marker='o', s=40,
            label='cluster 2')
plt.legend()
plt.title('Density based clustering(DBSCAN)')
plt.tight_layout()
```



```
from sklearn.cluster import DBSCAN


db = DBSCAN(eps=0.5, min_samples=5)
y_db = db.fit_predict(df_X)


plt.scatter(df_X[y_db == 0, 0], df_X[y_db == 0, 1],
            c='lightgreen', marker='s', s=40,
            label='cluster 1')

plt.scatter(df_X[y_db == 1, 0], df_X[y_db == 1, 1],
            c='orange', marker='o', s=40,
            label='cluster 2')
plt.legend()
plt.title('Density based clustering(DBSCAN)')
plt.tight_layout()
```



Description

sklearn.datasets 의 load_iris()로 iris 데이터를 불러오고 petal length, petal width 에 해당하는 2 번째, 3 번째 column 에 해당하는 데이터를 df_X 에 저장하였다. 먼저 df_X 를 시각화하여 그 분포를 살펴보았다. 여기까진 앞서

행한 KMeans 와 동일하다. 저장하였다. y_km 을 바탕으로 군집에 맞게 색과 모양을 달리하여 시각화하였다. eps=0.2, min_samples=10 으로 놓고 생성한 DBSCAN 객체로 y 를 예측하여 군집에 맞게 시각화 한 결과, cluster2 에 속한 데이터들이 너무 적어지는 문제가 발생하였다. 그후 eps=0.5, min_samples=5 로 놓고 DBSCAN 객체를 생성하여 y 를 예측한 결과 군이 제대로 형성되었음을 볼 수 있다.

6. Training CNN (30pts)

Build the following CNN model and train it using the CIFAR-10 dataset. Also, build a dropout model and see how the train accuracy and test accuracy differ. Finally, test the 10 new images given with the dropout model.

CIFAR-10(Plotting) : https://www.cs.toronto.edu/~kriz/cifar.html

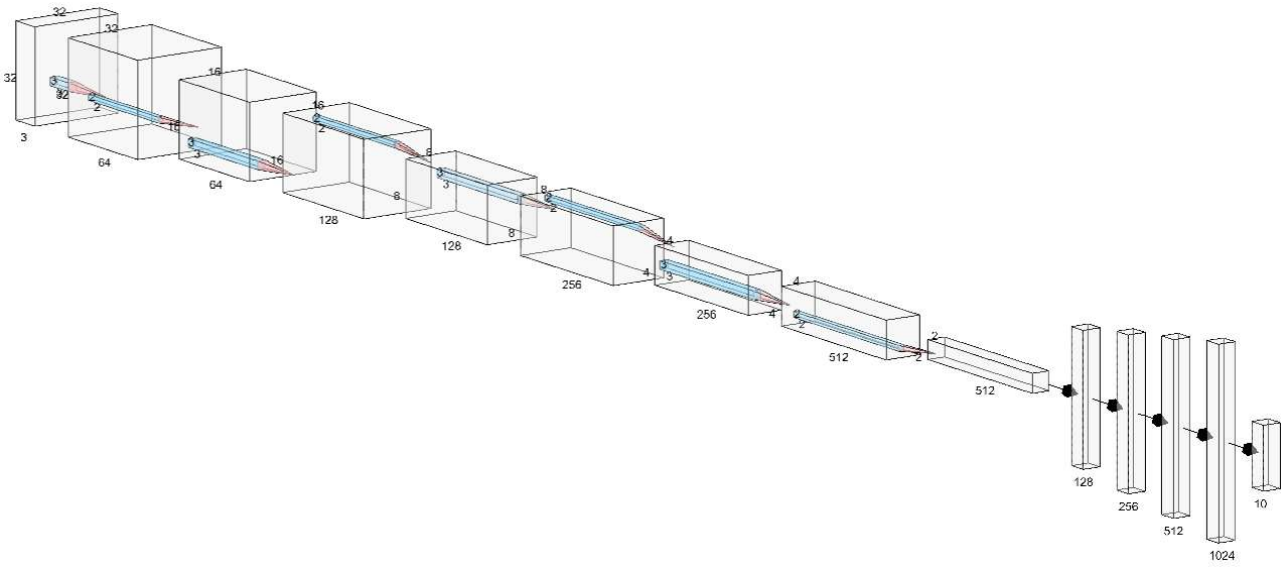
```
import tensorflow as tf
import numpy as np
cifar10 = tf.keras.datasets.cifar10
(x_train, y_train),(x_test, y_test) = cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
num_classes = 10

print("Number of train images: {}".format(len(x_train)))
print("Number of train labels: {}".format(len(y_train)))
print("Number of test images: {}".format(len(x_test)))
print("Number of test labels: {}".format(len(y_test)))
```

```
Number of train images: 50000
Number of train labels: 50000
Number of test images: 10000
Number of test labels: 10000
```

```
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
from IPython.core.pylabtools import figsize
matplotlib.rc('font', family='NanumGothic')  # Linumx
def display_cifar(images, size):
    n = len(images)
    plt.figure()
    plt.gca().set_axis_off()
    im = np.vstack([np.hstack([images[np.random.choice(n)] for i in range(size)])
                    for i in range(size)])
    plt.imshow(im)
```

```
    plt.show()

figsize(15, 7)
display_cifar(x_train, 10)
```

Architecture



Layer	Output Shape	Parameters #
Conv2D(3x3 filter)	(None, 32, 32, 64)	1792
MaxPooling2D(2x2)	(None, 16, 16, 64)	0
Conv2D(3x3 filter)	(None, 16, 16, 128)	73856
MaxPooling2D(2x2)	(None, 8, 8, 128)	0
Conv2D(3x3 filter)	(None, 8, 8, 256)	295168
MaxPooling2D(2x2)	(None, 4, 4, 256)	0
Conv2D(3x3 filter)	(None, 4, 4, 512)	1180160
MaxPooling2D(2x2)	(None, 2, 2, 512)	0
Flatten	(None, 2048)	0
Dense	(None, 128)	262272
Dense	(None, 256)	33024
Dense	(None, 512)	131584
Dense	(None, 1024)	525312
Dense(softmax)	(None, 10)	10250

Expected Output

Test the model

```
model.evaluate(x_train, y_train)
50000/50000 [==============================] - 4s 77us/step
[0.2962545334267616, 0.89866]

model.evaluate(x_test, y_test)
10000/10000 [==============================] - 1s 76us/step
[0.9534229537010193, 0.7268]
```

Test the dropout model

```
dropout_model.evaluate(x_train, y_train)
50000/50000 [==============================] - 4s 80us/step
[0.3286701273083687, 0.89726]

dropout_model.evaluate(x_test, y_test)
10000/10000 [==============================] - 1s 83us/step
[0.905360974931717, 0.7282]
```

Test your own data!

```
from PIL import Image
import glob

image_list = []
for filename in glob.glob('new test images/*.jpg'):
    img = Image.open(filename)
    image_list.append(img)
```

```
for i in range(len(image_list)):
    image_list[i] = np.asarray(image_list[i].resize((32, 32)))
    image_list[i] = np.resize(image_list[i], (1, 32, 32, 3))
    print(dropout_model.predict(image_list[i]))
```

```
[[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
[[0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]]
[[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]]
[[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]]
[[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]]
[[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]]
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]]
[[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]]
[[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]]
```

Code
hw4-6.ipynb 참고
Result(Captured images)

```
model.evaluate(x_train, y_train)
```

```
50000/50000 [==============================] - 4s 84us/step
[0.29223259724378586, 0.9073399901390076]
```

```
model.evaluate(x_test,y_test)
```

```
10000/10000 [==============================] - 1s 89us/step
[0.9864373187065124, 0.7175999879837036]
```

```
dropout_model.evaluate(x_train, y_train)
```

```
50000/50000 [==============================] - 5s 90us/step
[0.3773729834127426, 0.876479983329773]
```

```
dropout_model.evaluate(x_test, y_test)
```

```
10000/10000 [==============================] - 1s 93us/step
[0.9547297867774963, 0.7208999991416931]
```

```
for i in range(len(image_list)):
    image_list[i] = np.asarray(image_list[i].resize((32,32)))
    image_list[i] = np.resize(image_list[i], (1,32,32,3))
    print(model.predict(image_list[i]))
```

```
[[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
[[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
[[0.0000000e+00 0.0000000e+00 0.0000000e+00 1.0000000e+00 0.0000000e+00
  2.0952073e-19 0.0000000e+00 1.4510338e-30 0.0000000e+00 0.0000000e+00]]
[[0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]]
[[0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]]
[[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
[[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
[[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]]
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]]
[[0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]]
```

Description

주어진 summary 대로 tf.keras.models.Sequential 을 이용해서 모델을 구성하자. 그후 model.compile 모델의 loss 와 optimizer 를 설정하고 model.fit 로 학습을 시켜주자. 학습시 VRAM 의 크기가 충분치 않으면 batch_size(기본 32)로 데이터셋을 나눈 후 학습을 수행하게 된다. 학습이 완료되었으면 model.evaluate 로 평가를 해주자. 드랍아웃이 포함된 모델도 위 과정을 거치면 된다. 테스트는 x_test 에서 랜덤하게 데이터들을 추출해서 newImages 폴더에

저장하고 그 데이터들에 대해 predict 를 수행하였다. predict 결과는 onehot encoding 된 결과이다.

Note

1. Summit the file to e-class as pdf.
2. Specify your pdf file name as “hw4_<StudentID>_<Name>.pdf”