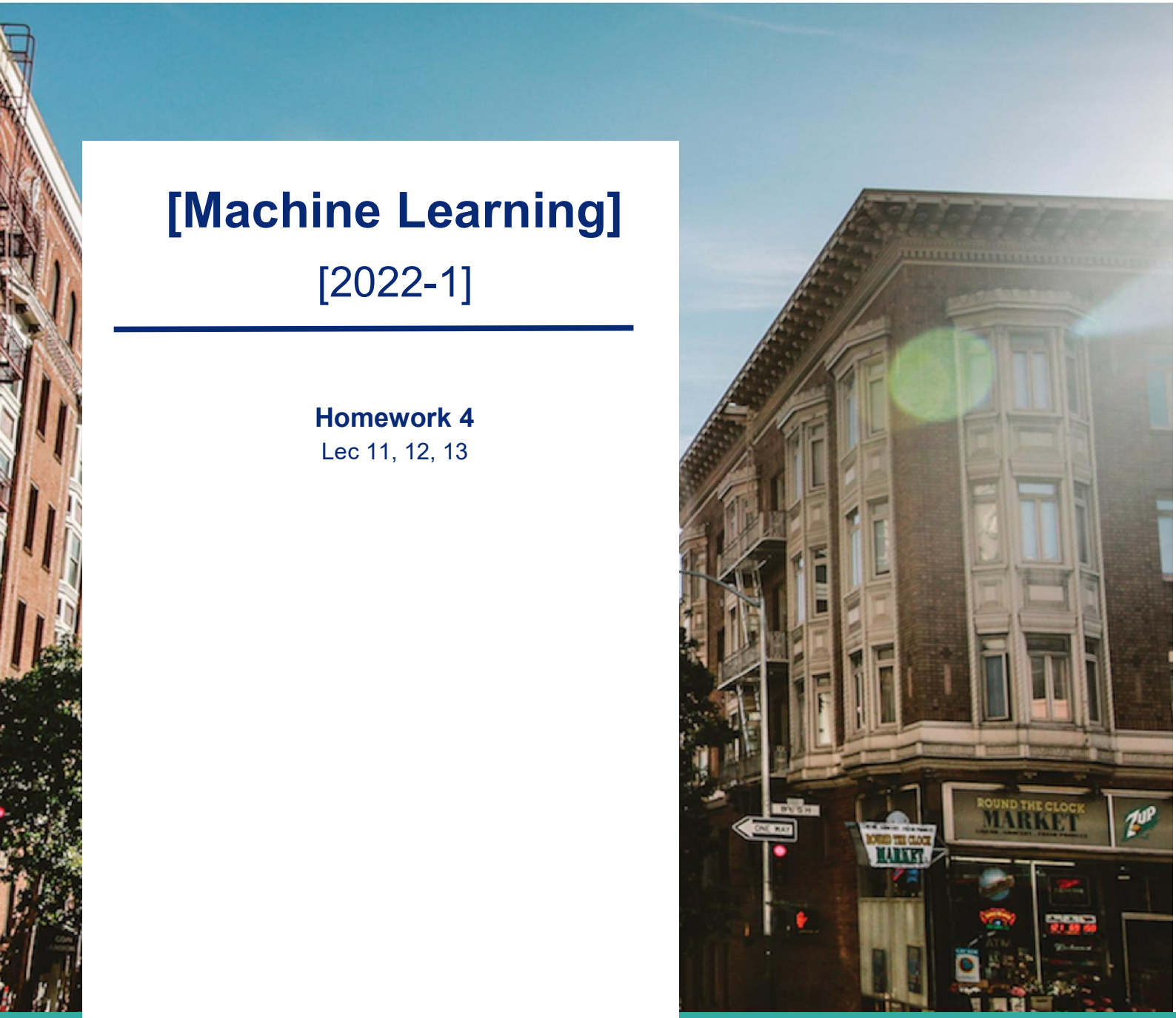


[Machine Learning]

[2022-1]

Homework 4

Lec 11, 12, 13

[DATE] 2022.06.11

Student ID : 2017112138

Name : 정여준

Professor : Juntae Kim

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

Your Answer
K-means 는 효율적이고 간단하다. K-means 는 군집 K 의 수를 지정해야 한다. K-means 는 노이즈나 이상치를 찾아낼 수 없다. DBSCAN 은 임의의 모양의 군집을 발견할 수 있다. DBSCAN 은 노이즈나 이상치를 찾아낼 수 있다. DBSCAN 은 Parameter 입실론과 MinPts 를 선택하는 것이 쉽지 않다.

2. Explain what the dropout and batch normalization are in deep neural network models, and what kind of effect you can expect by applying these techniques. (10pts)

Your Answer
Dropout 은 neural network 에서 overfitting 을 줄이기 위한 regularization 기법 중 하나이다. 학습 중에 랜덤하게 유닛을 생략하고 진행하는 방법이다. Dropout 의 효과는 특정 feature 에 집중되는 것을 예방할 수 있고 Ensemble method 처럼 다양한 모델을 배우는 효과를 줘서 overfitting 을 줄일 수 있다. Batch normalization 은 학습 과정에서 각 배치 단위 별로 데이터가 다양한 분포를 가지더라도 각 배치별로 평균과 분산을 이용해 정규화 하는 것을 뜻한다. Batch normalization 의 효과는 한 layer 의 parameter 의 작은 변화가 다른 layer 로 전파되지 않고 이는 더 높은 learning rate 를 사용할 수 있게 되어 빠른 학습이 되도록 도와준다. 또한 정규화 효과를 통해 overfitting 을 줄일 수 있다.

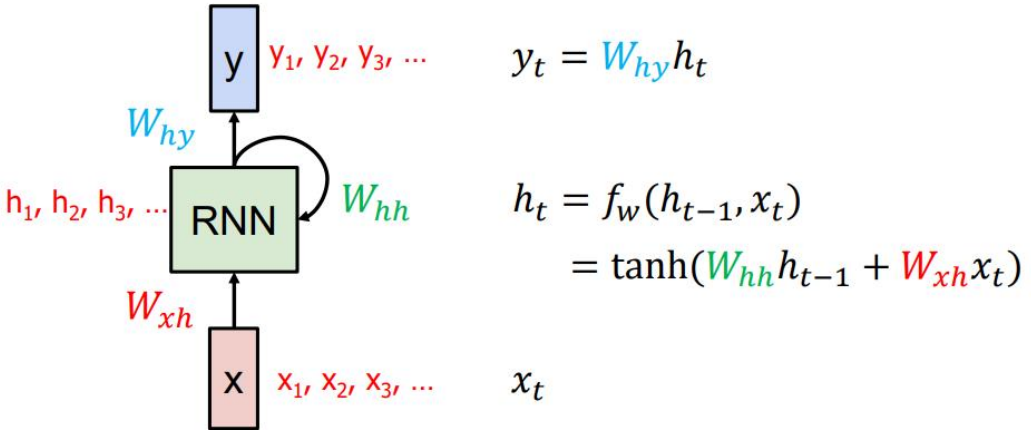
3. Explain what the *tanh* function, and why it is used in RNN model instead of *sigmoid* function. (10pts)

Your Answer
tanh = $\frac{e^x - e^{-x}}{e^x + e^{-x}}$ sigmoid 는 미분했을 때 값의 범위가 0~0.25 다. tanh 는 미분했을 때 값의 범위가 0~1 이다. RNN 에서 sigmoid 대신 tanh 를 사용하는 이유는 RNN 이 Sequential Data 를 다루기 때문에 데이터의 길이가 길어질 수 있는데 그런 상황에서 Gradient 가 누적되면서 곱해졌을 때 Gradient 의 값이 작으면 Gradient 가 너무 작아질 수 있다. 이런 현상을 완화하기 위해서 Gradient 값의 범위가 sigmoid 보다 큰 tanh 를 사용한다.

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

Your Answer
Backward: $\frac{\delta h_t}{\delta h_{t-1}} = tanh'(W_{hh}h_{t-1} + W_{xh}x_t)W_{hh}$ $\frac{\partial L_t}{\partial W} = \frac{\partial L_t}{\partial h_t}\frac{\partial h_t}{\partial h_{t-1}}\cdots\frac{\partial h_1}{\partial W} = \frac{\partial L_t}{\partial h_t} \times \left(\prod_t \tanh' \cdot W\right)$ RNN 의 큰 문제점은 길이가 길어질 때 Vanishing Gradient 가 발생한다는 것이다. 이는 Sequence 의 길이가 커질 때 작은 Gradient 가 계속 곱해지면서 발생하는 문제점이다. LSTM 은 Cell state 가 추가되었는데 이 Cell state 를 추가적으로 계산하는 방법을 통해서 Sequence 가 길어져도 정보가 남아있도록 만든다. Cell state 가 일종의 컨베이어 벨트 역할을 하면서 state 가 꽤 오래 경과하더라도 gradient 가 비교적으로 잘 전파되도록 한다. 즉, Cell state 는 정보가 바뀌지 않고 그대로 흐르도록 하는 역할을 수행한다. 또한 input gate, forget gate, output gate 가 추가됐다. forget, input gate 를 통해 어떤 정보를 버릴지 cell state 에 저장할지를 정하는 과정을 통해 RNN 의 문제를 해결한다.

5. Briefly suggest a deep learning model for the machine translation. Describe how you can train the model. (10pts)

Your Answer
Machine translation 을 위한 딥러닝 모델을 Recurrent Neural Nets (RNN)이 있다. 모델을 학습시키는 방법을 그림으로 나타내면 다음과 같다.    tanh 를 사용한다. 현재 상태 값에 대한 함수의 구성은 (이전 값*W)과 (입력 값*W)의 합에 tanh 를 처리한 값이 된다. 즉, 이전의 상태 값과 입력 값을 이용하여 학습하는 것이다.

6. Compute the total number of parameters in CNN with following architecture. Stride=1, no padding. Show each layer's output shape and # of parameters. Show how you calculate them. (20pts)

- input: 100 x 100 x 3 image
- conv layer 1: 16 filters of 5 x 5 size + 2 x 2 max pooling
- conv layer 2: 32 filters of 3 x 3 size + 2 x 2 max pooling
- dense layer: 256 outputs
- dense layer: 10 outputs

Your Answer

Layer	Output Shape	Parameter #
Conv layer 1	(96, 96, 16)	(5*5*3+1)*16 = 1216
2x2 max pooling	(48, 48, 16)	0
Conv layer 2	(46, 46, 32)	(3*3*16+1)*32 = 4640
2x2 max pooling	(23, 23, 32)	0
dense layer 1	(23, 23, 256)	(32+1)*256 = 8448
dense layer 2	(23, 23, 10)	(256+1)*10 = 2570

7. Training CNN for image classification (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.
(optional: Try to improve the test accuracy by modifying the model and hyperparameters, etc.)

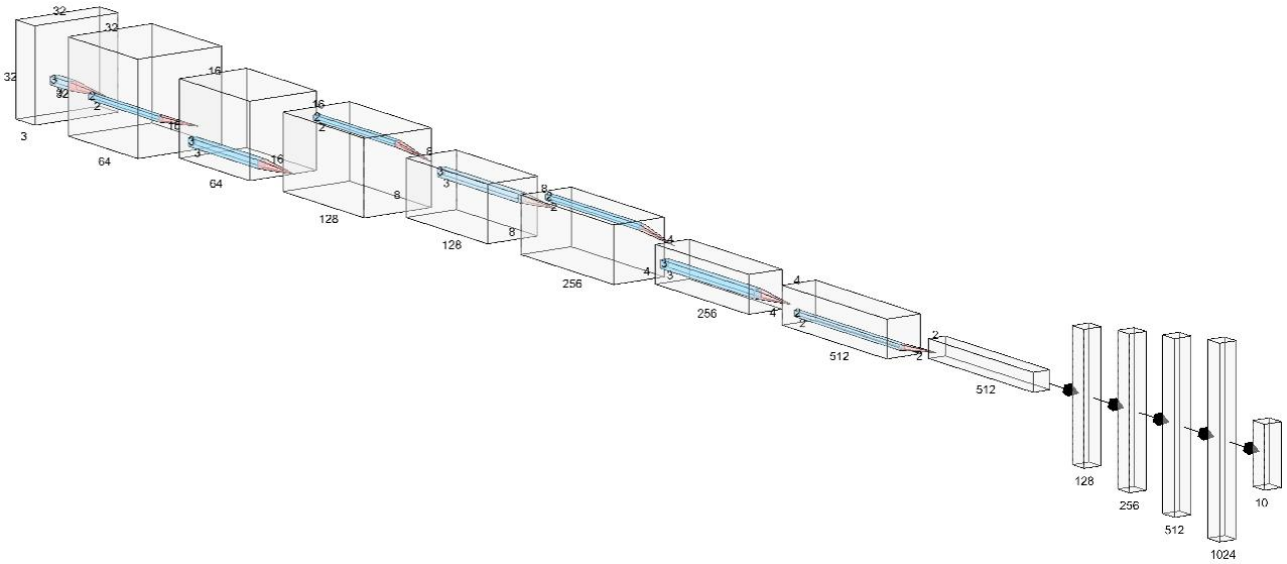
CIFAR-10(Plotting) : https://www.cs.toronto.edu/~kriz/cifar.html

```
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

```
layers = tf.keras.layers

model = tf.keras.models.Sequential([
        layers.Conv2D(64, kernel_size=(3, 3),padding="same",
                      input_shape = (32, 32, 3), activation="relu"),
        layers.MaxPool2D(pool_size=(2,2)),
        layers.Conv2D(128, kernel_size=(3, 3), padding="same",
                      input_shape=(16,16,64), activation="relu"),
        layers.MaxPool2D(pool_size=(2,2)),
        layers.Conv2D(256, kernel_size=(3, 3), padding="same",
                      input_shape=(8,8,256), activation="relu"),
        layers.MaxPool2D(pool_size=(2,2)),
        layers.Conv2D(512, kernel_size=(3, 3), padding="same",
                      input_shape=(4,4,256), activation="relu"),
        layers.MaxPool2D(pool_size=(2,2)),
        layers.Flatten(),
        layers.Dense(128, activation="relu"),
        layers.Dense(256, activation="relu"),
        layers.Dense(512, activation="relu"),
        layers.Dense(1024, activation="relu"),
        layers.Dense(10, activation="softmax")
])
model.summary()

model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metr
ics=['accuracy'])
model_history = model.fit(x_train,y_train, epochs=10)
```

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

dropout_model = tf.keras.models.Sequential([
        layers.Conv2D(64, kernel_size=(3, 3),padding="same",
                      input_shape = (32, 32, 3), activation="relu"),
        layers.MaxPool2D(pool_size=(2,2)),
        layers.Conv2D(128, kernel_size=(3, 3), padding="same",
                      input_shape=(16,16,64), activation="relu"),
        layers.MaxPool2D(pool_size=(2,2)),
        layers.Conv2D(256, kernel_size=(3, 3), padding="same",
                      input_shape=(8,8,256), activation="relu"),
        layers.MaxPool2D(pool_size=(2,2)),
        layers.Conv2D(512, kernel_size=(3, 3), padding="same",
                      input_shape=(4,4,256), activation="relu"),
        layers.MaxPool2D(pool_size=(2,2)),
        layers.Flatten(),
        layers.Dense(128, activation="relu"),
        layers.Dense(256, activation="relu"),
        layers.Dense(512, activation="relu"),
        layers.Dense(1024, activation="relu"),
        layers.Dropout(0.5),
        layers.Dense(10, activation="softmax")
])

dropout_model.summary()
dropout_model.compile(loss='sparse_categorical_crossentropy',
                      optimizer='adam', metrics=['accuracy'])
dropout_history = dropout_model.fit(x_train,y_train, epochs=10)
dropout_model.evaluate(x_train, y_train)
dropout_model.evaluate(x_test, y_test)
```

```
from PIL import Image
import glob

image_list = []
for filename in glob.glob('/content/drive/MyDrive/test_data/*.png'):
  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]))
```

Result(Captured images)

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 32, 32, 64)        1792

 max_pooling2d_4 (MaxPooling  (None, 16, 16, 64)       0
 2D)

 conv2d_5 (Conv2D)           (None, 16, 16, 128)       73856

 max_pooling2d_5 (MaxPooling  (None, 8, 8, 128)        0
 2D)

 conv2d_6 (Conv2D)           (None, 8, 8, 256)         295168

 max_pooling2d_6 (MaxPooling  (None, 4, 4, 256)        0
 2D)

 conv2d_7 (Conv2D)           (None, 4, 4, 512)         1180160

 max_pooling2d_7 (MaxPooling  (None, 2, 2, 512)        0
 2D)

 flatten_1 (Flatten)         (None, 2048)              0

 dense_4 (Dense)             (None, 128)               262272

 dense_5 (Dense)             (None, 256)               33024

 dense_6 (Dense)             (None, 512)               131584

 dense_7 (Dense)             (None, 1024)              525312

 dense_8 (Dense)             (None, 10)                10250

=================================================================
Total params: 2,513,418
Trainable params: 2,513,418
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
1563/1563 [==============================] - 22s 7ms/step - loss: 1.6287 - accuracy: 0.3776
Epoch 2/10
1563/1563 [==============================] - 10s 7ms/step - loss: 1.1357 - accuracy: 0.5929
Epoch 3/10
1563/1563 [==============================] - 11s 7ms/step - loss: 0.9233 - accuracy: 0.6768
Epoch 4/10
1563/1563 [==============================] - 11s 7ms/step - loss: 0.7839 - accuracy: 0.7261
Epoch 5/10
1563/1563 [==============================] - 11s 7ms/step - loss: 0.6805 - accuracy: 0.7643
Epoch 6/10
1563/1563 [==============================] - 11s 7ms/step - loss: 0.6003 - accuracy: 0.7923
Epoch 7/10
1563/1563 [==============================] - 10s 7ms/step - loss: 0.5302 - accuracy: 0.8168
Epoch 8/10
1563/1563 [==============================] - 10s 7ms/step - loss: 0.4682 - accuracy: 0.8395
Epoch 9/10
1563/1563 [==============================] - 10s 7ms/step - loss: 0.4159 - accuracy: 0.8573
Epoch 10/10
1563/1563 [==============================] - 11s 7ms/step - loss: 0.3716 - accuracy: 0.8742
```

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

```
1563/1563 [==============================] - 6s 4ms/step - loss: 0.3087 - accuracy: 0.8919
[0.30869969725608826, 0.8919000029563904]
```

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

```
313/313 [==============================] - 1s 4ms/step - loss: 0.9853 - accuracy: 0.7299
[0.9853163361549377, 0.7299000024795532]
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)           (None, 32, 32, 64)        1792

 max_pooling2d_8 (MaxPooling  (None, 16, 16, 64)       0
 2D)

 conv2d_9 (Conv2D)           (None, 16, 16, 128)       73856

 max_pooling2d_9 (MaxPooling  (None, 8, 8, 128)        0
 2D)

 conv2d_10 (Conv2D)          (None, 8, 8, 256)         295168

 max_pooling2d_10 (MaxPoolin  (None, 4, 4, 256)        0
 g2D)

 conv2d_11 (Conv2D)          (None, 4, 4, 512)         1180160

 max_pooling2d_11 (MaxPoolin  (None, 2, 2, 512)        0
 g2D)

 flatten_2 (Flatten)         (None, 2048)              0

 dense_9 (Dense)             (None, 128)               262272

 dense_10 (Dense)            (None, 256)               33024

 dense_11 (Dense)            (None, 512)               131584

 dense_12 (Dense)            (None, 1024)              525312

 dropout (Dropout)           (None, 1024)              0

 dense_13 (Dense)            (None, 10)                10250

=================================================================
Total params: 2,513,418
Trainable params: 2,513,418
Non-trainable params: 0
_________________________________________________________________
```

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

```
1563/1563 [==============================] - 7s 5ms/step - loss: 0.4170 - accuracy: 0.8623
[0.4170307815074920?, 0.862339973449707]
```

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

```
313/313 [==============================] - 1s 4ms/step - loss: 0.8908 - accuracy: 0.7176
[0.8907642364501953, 0.7175999879837036]
```

```
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]))
```

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

Description

model 에서 input 과 output shape 의 차이가 없는 Conv layer 에서 padding=”same”을 사용했다. kernel_size 는 filter 가 3x3 이기 때문에 (3,3)으로 지정했다. 또한 활성함수는 relu 를 사용했다. Dense layer 는 문제에서 요구하는 output 에 맞춰서 지정하였다. dropout_model 에서는 model 과 모두 동일하지만 마지막 Dense layer 앞에 Dropout(0.5) 처리했다. compile 할 때는 loss=’sparse_categorical_crossentropy’를 사용했고 optimizer=’adam’을 사용하였다. epoch 는 10 으로 지정했는데 20 으로 했을 때 더 높은 정확도가 나오는 것을 확인했다.

Note

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