프로젝트 4 설명

서강대학교 컴퓨터공학과
박운상 교수

1 프로젝트 4 개요

기본적인 이미지 분류 예제로 유명한 MNIST dataset을 가지고 이미지 분류를 하는 머신 러닝 모델을 개발하는 방법을 살펴본다.

MNIST 예제를 바탕으로 프로젝트 4에서는 Google Colab에서 Keras를 이용하여 이미지 분류를 하는 머신 러닝 모델을 개발한다. CIFAR-10이라는 dataset을 이용하여 이미지 분류 성능을 측정하고 직접 모델의 파라미터들을 바꿔가며 테스트 셋에 대한 정확도 80% 이상을 얻는 것을 목표로 한다.

2 MNIST

MNIST Dataset이란 디지털 영상 처리와 이미지 분류 시스템을 학습하기 위해서 많이 사용되는 데이터셋으로 아래 그림과 같이 숫자들을 직접 손으로 쓴 이미지를 모아둔 것으로 학습 데이터는 60,000개, 테스트 데이터는 10,000개로 이루어져 있다.

2 MNIST

MNIST 데이터 셋은 http://yann.lecun.com/exdb/mnist/에서 다운 받을 수 있다. 해당 사이트에서 MNIST 데이터 셋에 대한 구체적인 설명이 제공된다.

그러나 Keras를 사용하면 직접 데이터 셋을 다운 받지 않아도 Keras를 통해서 MNIST 데이터 셋을 쉽게 가져올 수 있다.

Keras를 가지고 어떻게 MNIST을 가지고 오는지, 머신 러닝 모델을 어떻게 만들고, 학습하는 지에 대한 코드가 https://colab.research.google.com/drive/1buEeIG5WrFXedTnX6zA52jygK2YAqHZp

에서 제공된다.

머신 러닝을 처음 접하는 학생들은 소스 코드만 봤을 때 어떤 내용인지 쉽게 알 수 없기 때문에 위 링크에 있는 코드에 대한 설명을 먼저 하도록 한다.

3 Import modules and tuning parameters

```
[ ] import keras
    from keras.datasets import mnist
    from keras.models import Sequential
    from keras.layers import Dense, Dropout, Flatten, Activation
    from keras.layers import Conv2D, MaxPooling2D
    from keras import backend as K
    from keras.utils.vis_utils import model_to_dot
    from IPython.display import SVG
    %matplotlib inline
    import matplotlib.pyplot as plt
    from sklearn.metrics import confusion_matrix
    import pandas as pd
    import seaborn as sns
```

```
[ ] epochs = 10
    learning_rate=0.01
```

```
[ ] batch_size = 128
    num_classes = 10
```

- 첫번째 cell은 모델을 개발하는 데에 있어서 필요한 것들을 import하는 코드이다.
- 두번째 cell에서 epochs은 전체 학습 데이터 셋에 대해서 몇 번 학습을 시킬 것인지를 의미하는데 10이라고 하면 10번 학습을 시킨다고 보면 된다.
- epochs의 횟수가 적절하지 않으면 underfitting, overfitting 현상이 발생할 수 있다.
- learning_rate는 Machine Learning 슬라이드에 나오는 η에 해당하며 학습 시 가중치가 업데이트되는 정도를 나타낸다.
- 세번째 cell에서 batch_size는 입력 데이터로 사용되는 소그룹에 포함되는 샘플의 개수를 나타낸다. 하나의 batch에 있는 샘플들로 forward 연산을 수행하고, 에러의 평균치를 가지고 backprob 연산을 수행한다.
- Num_classes는 분류해야 하는 클래스의 개수로 MNIST는 0부터 9까지의 숫자가 있기 때문에 10이 된다.

3 Import modules and tuning parameters

본 코드는 MINST에 있는 이미지 일부를 가지고 와서 이미지 그림과 이미지가 어떤 클래스에 속해 있는지를 출력해주는 코드이다.

파라미터로 size가 (5,5)이면 총 5*5 = 25개의 임의의 데이터를 가져와서 보여주게 된다.

학습이 끝나고 분류를 수행할 때 분류 결과도 보여준다. (Pred가 분류 결과를 나타낸다)

```
[ ]  def plot_images(x, y_true, y_pred=None, size=(5, 5)):
         assert len(x) == len(y_true) == size[0] * size[1]

         fig, axes = plt.subplots(size[0], size[1])
         fig.subplots_adjust(hspace=0.5, wspace=0.1)

         for i, ax in enumerate(axes.flat):
             if x[i].shape[-1] == 1:
               ax.imshow(x[i].reshape(x[i].shape[0], x[i].shape[1]))
             else:
               ax.imshow(x[i])

             if y_pred is None:
                 xlabel = "True: {0}".format(y_true[i].argmax())
             else:
                 xlabel = "True: {0}, Pred: {1}".format(y_true[i].argmax(),
                                                        y_pred[i].argmax())

             ax.set_xlabel(xlabel)

             ax.set_xticks([])
             ax.set_yticks([])

         plt.show()
```

4 Loading the data

Import 부분을 보면 from keras.datasets import mnist를 확인할 수 있는데 이는 kreas.datasets에서 mnist에 관한 module를 가져온다고 보면 되는데 가져온 modul에서 mnist 데이터를 가져오는 과정이다. 처음 실행했을 때는 mnist에 대한 data가 없기 때문에 2번째 그림처럼 다운로드를 받게 되고 그 이후에 다시 실행하면 data가 있기 때문에 다운 받지 않고 넘어가게 된다.

Load dataset

```
[77] (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
```

5 Reshaping the data

- 첫 번째 셀은 학습을 위해 MNIST 데이터를 전처리 한다
  - mnist 이미지를 담은 배열의 rank를 4로 맞춰준다
  - 리포트에 관련 내용 설명할 것
- 두 번째 cell은 Normalization을 진행하는 과정으로 픽셀은 0부터 255까지의 값을 갖기 때문에 255로 나눈다
- 마지막 cell은 One-hot encoding을 적용하는 것으로 MNIST 데이터는 0~9의 총 10개의 클래스가 있는데 이를 숫자 하나로 매칭하는 것이 아니라 10차원 벡터로 정의한다. 만약 데이터가 클래스 2에 매칭이 된다면 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]로 나타나게 된다.

Reshaping the data

```
[63] if len(x_train.shape) < 4:
       x_train = x_train.reshape(x_train.shape[0], x_train.shape[1], x_train.shape[2], 1)
       x_test = x_test.reshape(x_test.shape[0], x_train.shape[1], x_train.shape[2], 1)
```

```
[64] x_train = x_train.astype('float32')
     x_test = x_test.astype('float32')
     x_train /= 255
     x_test /= 255
     print('x_train shape:', x_train.shape)
     print(x_train.shape[0], 'train samples')
     print(x_test.shape[0], 'test samples')
```

```
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
```

Applying One hot encoding for the data

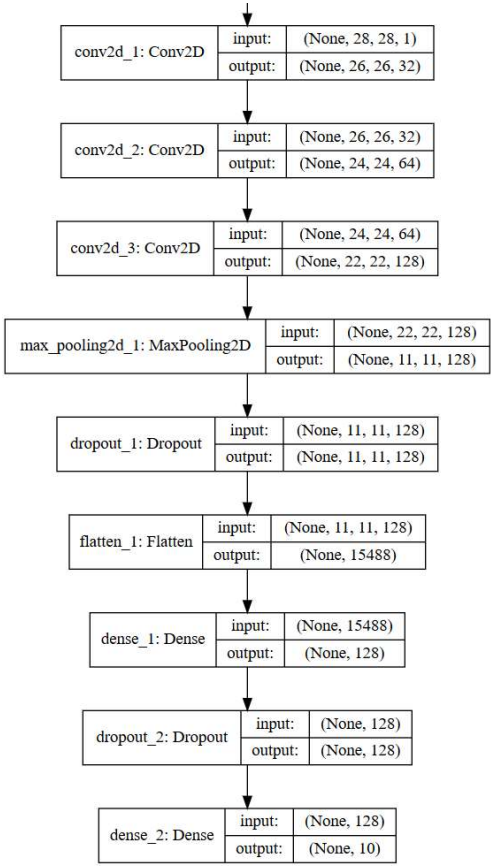
```
[65] y_train = keras.utils.to_categorical(y_train, num_classes)
     y_test = keras.utils.to_categorical(y_test, num_classes)
```

6 Creating the model

- Model = Sequential()라고 하면 layer들을 순차적으로 쌓아서 모델을 만든다.
  - Model.add(…) 를 꼭 써야 함
- 두번째 cell를 보면 layer들을 위에서 아래로 추가한 순서대로 오른쪽 그림에서 똑같이 나타나고 있음을 볼 수 있다.
- Layer들에 대한 의미는 다음 슬라이드에서 설명하겠다.
- 리포트에 본인이 구성한 최종 Layer를 오른쪽과 같이 그려 넣고, Layer 구조에 대한 내용 설명할 것 (특히 input, output dimension에 대한 부분)

```
[ ]  model = Sequential()
```

```
[ ]  model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
                      input_shape=x_train.shape[1:]))
     model.add(Conv2D(64, (3, 3), activation='relu'))
     model.add(Conv2D(128, (3, 3), activation='relu'))
     model.add(MaxPooling2D(pool_size=(2, 2)))
     model.add(Dropout(0.25))
     model.add(Flatten())
     model.add(Dense(128, activation='relu'))
     model.add(Dropout(0.5))
     model.add(Dense(num_classes, activation='softmax'))
```



6 Creating the model

```
model.add(Conv2D(64, (3, 3), activation='relu'))
```

Activation function으로 ReLU를 사용하는 64개의 3×3 kernel size의 convolution filters가 있는 layer를 추가하는 것을 의미한다. 추가적으로 strides의 범위와 padding 개수를 설정할 수 있다.

```
model.add(MaxPooling2D(pool_size=(2, 2))
```

2×2 kernel size의 max pooling하는 layer를 추가하는 것을 의미한다.
추가적으로 strides의 범위와 padding 개수를 설정할 수 있다.

```
model.add(Dense(128, activation='relu'))
```

Activation function으로 ReLU를 사용하는 K to 128인 fully connected layer를 사용하는 것을 의미한다.
여기서 K 는 바로 앞의 layer output size를 의미한다.

7 Compiling and training the model

- 입력 샘플을 랜덤하게 선택하는 SGD (Stochastic Gradient Descent) 방식을 사용한다.
- Optimizer까지 설정했으면 model.compile 로 모델을 컴파일한다. (여기서 loss function은 cross Entropy를 사용하는데 클래스 개수가 10개이므로 categorical_crossentropy를 사용한다.)
  - 리포트에 loss function에 관한 설명 추가
- 마지막으로 model.fit을 하면 만든 모델과 설정한 파라미터들을 가지고 학습을 진행하게 된다. 그 결과는 오른쪽 그림에 나타나 있다.

128 128

128 batch | batch2

60,000

epoch 1

test set evaluation

train

val

test

test set "

```
[ ]  optimizer = keras.optimizers.SGD(lr=learning_rate)
```

```
[ ]  model.compile(loss=keras.losses.categorical_crossentropy,
                   optimizer=optimizer,
                   metrics=['accuracy'])
```

```
[ ]  model.fit(x_train, y_train,
               batch_size=batch_size,
               epochs=epochs,
               verbose=1,
               validation_data=(x_test, y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 21s 346us/step - loss: 1.0488 - acc: 0.6583 - val_loss: 0.2787 - val_acc: 0.9176
Epoch 2/10
60000/60000 [==============================] - 16s 260us/step - loss: 0.3841 - acc: 0.8827 - val_loss: 0.1955 - val_acc: 0.9439
Epoch 3/10
60000/60000 [==============================] - 16s 260us/step - loss: 0.3117 - acc: 0.9061 - val_loss: 0.1772 - val_acc: 0.9465
Epoch 4/10
60000/60000 [==============================] - 16s 259us/step - loss: 0.2573 - acc: 0.9223 - val_loss: 0.1290 - val_acc: 0.9620
Epoch 5/10
60000/60000 [==============================] - 15s 256us/step - loss: 0.2223 - acc: 0.9344 - val_loss: 0.1118 - val_acc: 0.9664
Epoch 6/10
60000/60000 [==============================] - 15s 256us/step - loss: 0.1886 - acc: 0.9439 - val_loss: 0.0942 - val_acc: 0.9711
Epoch 7/10
60000/60000 [==============================] - 15s 257us/step - loss: 0.1658 - acc: 0.9505 - val_loss: 0.0806 - val_acc: 0.9753
Epoch 8/10
60000/60000 [==============================] - 15s 255us/step - loss: 0.1440 - acc: 0.9571 - val_loss: 0.0695 - val_acc: 0.9784
Epoch 9/10
60000/60000 [==============================] - 15s 254us/step - loss: 0.1307 - acc: 0.9616 - val_loss: 0.0662 - val_acc: 0.9785
Epoch 10/10
60000/60000 [==============================] - 15s 254us/step - loss: 0.1195 - acc: 0.9648 - val_loss: 0.0593 - val_acc: 0.9802
```

8 Evaluating and prediction the model

- 학습이 끝난 후 학습에 사용되지 않은 test 셋에 대해서 평가를 진행하면 loss와 정확도를 얻을 수 있다. 이 때 loss는 모델을 컴파일 할 때 정의한 loss function을 가지고 계산을 한다.
- 분류 결과를 보기 위해서 맨 처음 정의한 plot_images 함수를 사용해서 볼 수 있는데 모델이 분류한 결과를 y_pred에 저장하고 plot_images 함수를 호출하면 진짜 클래스와 모델이 분류한 결과와 함께 그림 이미지를 함께 볼 수 있다.
- 또한 맨 오른쪽처럼 표로도 결과를 확인할 수 있다.

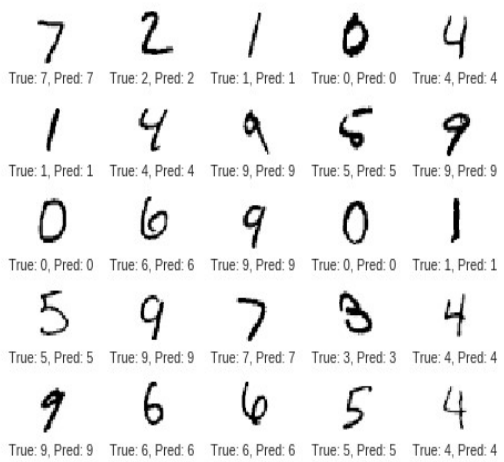
```
[ ]  score = model.evaluate(x_test, y_test, verbose=0)
     print('Test loss:', score[0])
     print('Test accuracy:', score[1])

[ ]  y_pred = model.predict(x_test)

[ ]  plot_images(x=x_test[:25], y_true=y_test[:25], y_pred=y_pred[:25])

[ ]  y_result = confusion_matrix(y_test.argmax(axis=1), y_pred.argmax(axis=1))
     sns.heatmap(pd.DataFrame(y_result, range(10), range(10)), annot=True, fmt='g')
```

```
Test loss: 0.05925932562798262
Test accuracy: 0.9802
```



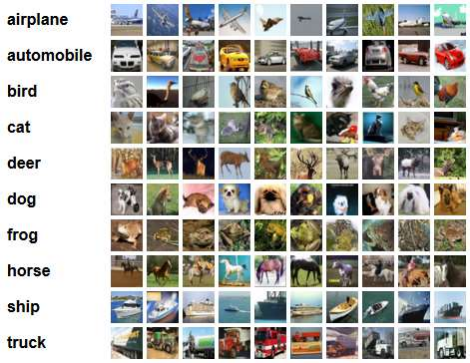
	0	1	2	3	4	5	6	7	8	9
0	971	0	1	0	0	0	3	1	4	0
1	0	1124	3	2	0	0	3	1	2	0
2	5	2	1009	1	2	0	1	8	4	0
3	0	0	3	997	0	2	0	5	3	0
4	1	0	3	0	956	0	5	2	2	13
5	2	0	0	10	0	872	4	1	2	1
6	5	3	0	1	3	5	940	0	1	0
7	1	3	10	1	0	0	0	1009	1	3
8	7	1	4	4	2	0	2	2	950	2
9	7	6	0	6	5	1	0	6	4	974

CIFAR-10

1 CIFAR-10

CIFAR-10 데이터 셋은 60000개의 32*32 픽셀의 컬러이미지로 클래스당 6000개의 이미지로 총 10개의 클래스가 있다.

MNIST와 마찬가지로 50000개의 학습 데이터와 10000개의 테스트 데이터로 이루어져 있다. 클래스당 10개의 임의의 이미지가 옆 그림에 있다.



하나의 이미지에 대해서 2개 이상의 클래스에 들어가 있는 경우는 없다. 예를 들면 클래스 중에서 automobile과 truck이 있는데 truck은 큰 트럭만 의미하고 automobile은 그 이외를 의미 하기 때문에 어떤 차에 대한 이미지가 automobile, truck 2개의 클래스에 들어가지 않고 2개 중에 하나의 클래스로만 mapping이 된다.

1 CIFAR-10

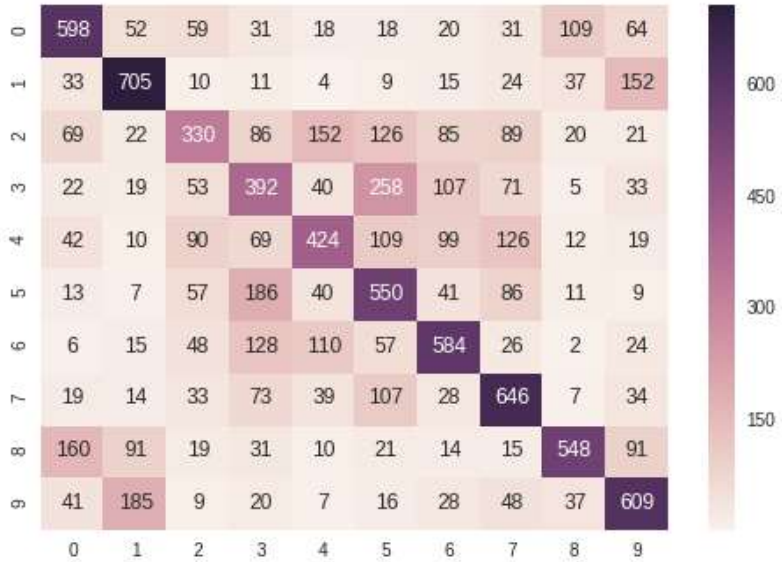
The CIFAR-10에 대한 정보는 https://www.cs.toronto.edu/~kriz/cifar.html에 있으므로 프로젝트를 수행하기 위해서 CIFAR-10에 대한 정보를 얻거나 직접 데이터를 다운 받으려면 위 URL에 들어가면 된다.

이번 프로젝트에서 만들어야 하는 모델에 대한 기본 BASE가 되는 모델에 대한 코드는 https://colab.research.google.com/drive/1FJPtyBIW02NVytn7eYpE-R530Wae-mgS에 있으므로 이 코드를 참고해서 프로젝트를 수행하면 된다.

2 CIFAR-10 with the previous method

https://colab.research.google.com/drive/1FJPtyBIW02NVytn7eYpE-R530Wae-mgS에 있는 코드를 그대로 수행하게 되면 아래 그림과 같은 결과가 나오는데 정확도가 53.86%이므로 제공한 코드를 그대로 제출하면 안된다. 이 코드는 MNIST 데이터 셋에 최적화 되어 있는 코드이기 때문이다. 그러므로 각자의 방법으로 모델을 개선하거나 새로 만들어서 정확도가 80%이상으로 나올 수 있는 모델을 만들어야 한다.

```
Test loss: 1.3167947158813476
Test accuracy: 0.5386
```



Project

1 Goal

본 프로젝트의 목표는 CIFAR-10 데이터 셋에 대해서 80% 이상의 정확도를 얻을 수 있는 모델을 개발하는 것이다.

성능 향상을 위해서 다음과 같은 시도들을 해볼 수 있다.

1. **좀 더 많은 layer층을 쌓는다.** (프로젝트 목표를 달성하기 위한 핵심 방법이다.)
   1)Core Layers: https://keras.io/layers/core/
   2)Convolution Layers: https://keras.io/layers/convolutional/
   3)Pooling Layers: https://keras.io/layers/pooling/
2. 다른 optimizer를 사용해 본다. (https://keras.io/optimizers/에 optimizer에 대한 설명이 있다)
3. 다른 activation function을 사용해 본다. (https://keras.io/activations/에 activation function에 대한 설명이 있다.)
4. Learning rate를 바꾸면서 학습을 시켜본다.

기타 여러 가지 방법을 동원해서 성능을 향상시키면 된다.

그러나 epoch 횟수는 바꾸면 안된다. epoch 횟수는 50으로 고정해야 한다.

2 Submission

제출해야 할 것

(1) 아웃풋 결과가 있는 colab notebook file

- 결과를 지우면 안된다. 결과가 있는 notebook file을 제출해야 한다.
- Colab에서 File 〉 Download .ipynb으로 다운받을 수 있다.

(2) 프로젝트 보고서 파일

- 어떻게 프로그램을 실행할 수 있는지에 대한 방법이 있어야 한다.
- 어떤 방법으로 모델을 만들었고, 어떻게 모델이 구성되어 있는지(9 page에 있는 모델 그림과 똑같은 그림을 리포트에 추가할 것), 모델 향상을 어떻게 했는지에 대한 구체적인 기술이 있어야 한다.(9 page 참고할 것)
- Confusion matrix 그림이 있어야 한다. (https://ko.wikipedia.org/wiki/%EC%BB%A8%ED%93%A8%EC%A0%84_%ED%96%89%EB%A0%AC 참고 할 것)
- 모델을 컴파일 할 때 사용한 Loss function에 대한 설명이 있어야 한다.
- 전처리 과정에서 학습 및 테스트 데이터의 Rank를 4로 맞추는지에 대한 설명이 있어야 한다.
- 프로젝트 1~3에서 사용했던 프로젝트 보고서 양식을 수정해서 사용해도 된다.

2 Submission

Instructions on Submission

- Sp(자신의 학번)_proj4라는 폴더를 만든다.
- Ipynb, document 파일을 폴더에 넣고 폴더를 zip 또는 tar로 압축을 한다. (압축하고 제대로 압축이 되었는지 꼭 확인해볼 것)
- notebook file 이름을 "**자신의 학번**.ipynb" .으로 바꿔야 한다.
- Tar 파일로 압축할 때, z option을 사용하면 안된다.

예:

sp20201234_proj4/

document.docx

20201234.ipynb

2 Submission

sp20201234_proj4.tar or sp20201234_proj5.zip 파일을 사이버 캠퍼스 프로젝트 4 과제란에 제출하면 됩니다.

제출 날짜: 6/21(일), 11:59분

Late Submission

Late submissions are accepted for five days after the deadline. 10% of the points are deducted For each day. Submissions are not accepted after five days.