9장. 케라스 튜너

요약 정리

1. 이 장에서 다루는 내용은 **텐서플로우 공식 문서 튜토리얼**에서도 찾아볼 수 있습니다. 번갈아 가면서 공부하면, 효과는 두 배입니다.
2. 우리가 직면한 특정 문제를 해결하려면 **커스터마이제이션 방법**이 필요합니다.
3. **가중치의 학습 여부**(Y/N)에 따라 적합한 층(**Lambda층/Layer 클래스를 상속한 커스텀 케라스층**)을 사용합니다.
4. 2019년에 발표된 **최적화 함수(RAdam), 활성화 함수(Mish)**를 다뤄보았습니다.
5. 실제값(y_true), 모델이 예측한 값(y_pred)을 적절히 조정하면 쉽게 커스텀 손실 함수를 정의할 수 있습니다.
6. 1x1 컨볼루션은 **모델 파라미터 감소, 비선형성 증가, 채널 수 조절의 장점**을 제공합니다.
7. Dense층을 사용하지 않아도, 모델을 구성할 수 있습니다.
8. **딥러닝은 아직 경험적으로 결정되는 요소가 매우 많습니다.**
9. 객체 탐지, 객체 분할, 딥페이크, 질의응답, 이상 현상 탐지 등을 살펴보았습니다.
10. 이 책을 공부하고 나서의 다음 방향으로 **(1) 개인 프로젝트 (2) 논문 구현**을 추천합니다.
11. 실제 환경에서는 **데이터가 매우, 매우, 매우 중요**합니다.
12. 딥러닝이 모든 문제를 해결할 것이라는 생각은 아직 위험합니다. 때론, **기존에 사용되고 있던 정통 모델을 고려해볼 줄 알아야합니다.**

9장의 내용?

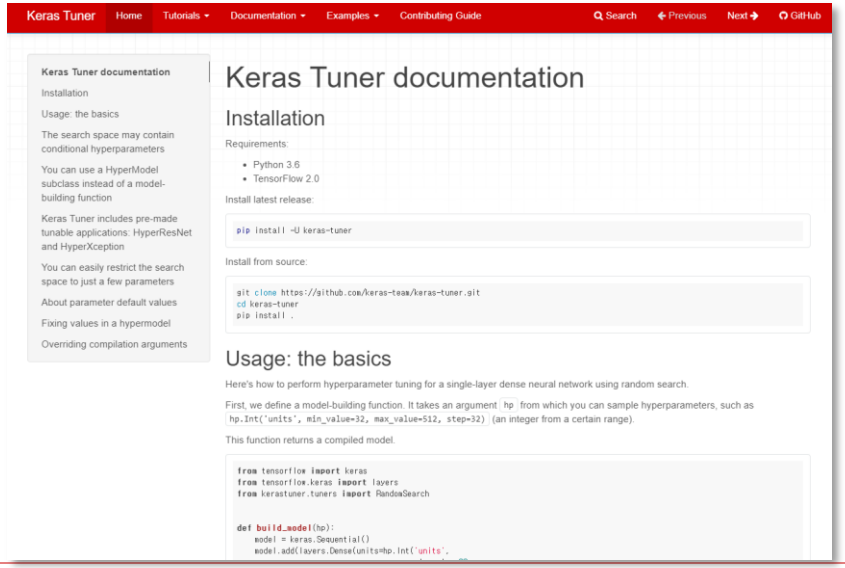
- 하이퍼파라미터 튜닝은 매우 어렵고, 반드시 거쳐야할 과정
  - 학습률, 배치 크기, 층의 개수, 컨볼루션 필터 개수 등을 임의로 정함
  - 예) Dense층의 은닉 유닛 개수는 주로 2 제곱수(16, 32, 64, ...)를 사용
  - 하지만 상황에 따라 은닉 유닛 개수는 10개, 100개일 수 있음

    ☞ 사막에서 바늘 찾기

- 누가 대신 찾아줄 사람 없을까?

  - Grid Search, Random Search, Bayesian Optimization

- 이번 장에서는 케라스 튜너를 다뤄봅니다.



탐색해야 할 하이퍼파라미터

```
01  from tensorflow.keras.models import Model
02  from tensorflow.keras.layers import Conv2D, MaxPooling2D,
                                        GlobalAveragePooling2D, Dense
03  from tensorflow.keras.layers import Input, Dropout
04  from tensorflow.keras.optimizers import Adam
05
06  # 각 층이 가지는 여러 가지 하이퍼파라미터에 집중합시다.
07  inputs = Input(shape = (28, 28, 1))
08  x = Conv2D(32, (3, 3), activation = 'relu')(inputs)  # 필터 개수
09  x = Conv2D(32, (3, 3), activation = 'relu')(x)
10  x = MaxPooling2D(strides = 2)(x)
11  x = GlobalAveragePooling2D()(x)  # 최대 풀링? 평균 풀링?
12  x = Dense(30, activation = 'softmax')(x)  # 은닉층의 개수
13  x = Dropout(0.5)(x)  # 드롭아웃률
14  x = Dense(10, activation = 'softmax')(x)
15
16  model = Model(inputs = inputs, outputs = x)
17
18  model.compile(optimizer = Adam(learning_rate = 0.001),  # 학습률
19                loss = 'categorical_crossentropy',
20                metrics = ['acc'])
```

- Conv2D층의 컨볼루션 필터 개수
- Dense층의 은닉 유닛 개수
- 드롭아웃률
- GAP 또는 GMP의 사용
  - GlobalMaxPooling
- Adam 옵티마이저의 학습률
- …

어떤 것을
탐색해야 할까요?

케라스 튜너 사용하기

- 여러 번의 실험을 대신해줄 존재: **케라스 튜너(Keras Tuner)**
- 케라스 튜너 설치 https://github.com/keras-team/keras-tuner

```
01  pip install -U keras-tuner
```

- HyperParameters() 객체를 사용하여 다양한 메서드를 사용

```
06  def build_model(hp):
07    inputs = Input(shape = (28, 28, 1))
08    x = inputs
09
10    for i in range(hp.Int('n_layers', 1, 3)):
11      # 필터 개수를 탐색합니다.
12      x = Conv2D(filters = hp.Int('filters ' + str(i), 4, 64, step = 8, default = 16),
13                 kernel_size = (3, 3), activation = 'relu',
14                 padding = 'same')(x)
15    x = MaxPooling2D(strides = 2)(x)
16
17    # GAP? GMP?
18    if hp.Choice('global_pooling', ['max', 'avg']) == 'avg':
19        x = GlobalAveragePooling2D()(x)
20    else:
21        x = GlobalMaxPooling2D()(x)
```

케라스 튜너 사용하기

- HyperParameters() 객체를 사용하여 다양한 메서드를 사용

```
23  # 은닉층의 개수를 탐색합니다.
24  x = Dense(units = hp.Int('units',
25                           min_value = 16,
26                           max_value = 128,
27                           step = 16),
28            activation = 'relu')(x)
29  # 드롭아웃률을 탐색합니다.
30  x = Dropout(hp.Choice('dropout_rate', values = [0.2, 0.3, 0.5]))(x)
31  x = Dense(10, activation = 'softmax')(x)
32
33  model = Model(inputs = inputs, outputs = x)
34  model.compile(optimizer = Adam(hp.Choice('learning_rate',
35                                           values = [1e-3, 1e-4, 1e-5])),
36                loss = 'categorical_crossentropy',
37                metrics = ['acc'])
38
39  return model
```

- 다양한 메서드를 제공
  - hp.Float() or hp.Int()
  - hp.Choice()
  - hp.Fixed()
  - ...

케라스 튜너 사용하기(실습)

- 제공되는 코드를 통해 꼭 실습하세요

- RandomSearch() 함수를 활용한 탐색

```
01  from kerastuner.tuners import RandomSearch
02
03  tuner = RandomSearch(build_model,
04                       objective='val_acc',            # 모니터링할 평가지표입니다.
05                       max_trials=5,                   # 5번의 실험을 진행합니다.
06                       executions_per_trial=3,
07                       # directory = 'my_path',        # 경로를 지정합니다.
08                       # project_name = 'helloworld')  # 프로젝트명을 지정합니다.
09
```

- 탐색할 하이퍼파라미터 살펴보기

```
01  tuner.search_space_summary()
```

```
Search space summary
|-Default search space size: 6
conv2D (Int)
|-default: None
|-max_value: 3
|-min_value: 1
|-sampling: None
|-step: 1
... 생략 ...
dropout_rate (Choice)
|-default: 0.2
|-ordered: True
|-values: [0.2, 0.3, 0.5]
learning_rate (Choice)
|-default: 0.001
|-ordered: True
|-values: [0.001, 0.0001, 1e-05]
```

케라스 튜너 사용하기

- 탐색 진행
  - 여유를 가지고 기다리기

```
01  tuner.search(x=x_train,
02               y=y_train,
03               epochs=10,
04               validation_data=(x_val, y_val))
```

```
... 생략 ...
Trial complete
Trial summary
Hp values:
|-dropout_rate: 0.2
|-filters_0: 36
|-filters_1: 12
|-filters_2: 16
|-global_pooling: max
|-learning_rate: 1e-05
|-n_layers: 3
|-units: 128
|-Score: 0.5371111035346985
|-Best step: 0
```

- 결과 요약
  - 어떤 하이퍼파라미터가 사용되었는지 확인 가능!

```
01  tuner.results_summary()
```

```
... 생략 ...
Trial summary
|-Trial ID: b9e7fd958310d707852873ea077d1abe
|-Score: 0.962055504322052
|-Best step: 0
Hyperparameters:
|-dropout_rate: 0.2
|-filters_0: 60
|-filters_1: 16
|-filters_2: 16
|-global_pooling: max
|-learning_rate: 0.001
|-n_layers: 3
|-units: 64
```

케라스 튜너 사용하기

- 가장 좋은 성능 모델 불러오기

```
01  best_model = tuner.get_best_models()[0]
02  best_model.summary()
```

- 모델 하이퍼파라미터 확인하기

```
01  best_hp = tuner.get_best_hyperparameters()[0].values
02  best_hp
```

```
{'dropout_rate': 0.2,
 'filters_0': 60,
 'filters_1': 16,
 'filters_2': 16,
 'global_pooling': 'max',
 'learning_rate': 0.001,
 'n_layers': 3,
 'units': 64}
```

케라스 튜너 사용하기

- 60개 또는 16개 컨볼루션 필터를 생각할 수 있었을까요?
  - 60개는 지금까지 사용해본적 없는 수치입니다.

- 하이퍼파라미터는 수많은 시간과 자원이 필요
  - 별도의 실험을 수작업으로 진행해야 하는 번거로움은 피할 수 있지만,
  - 그만큼 시간과 비용이 발생!

- **신중하게 탐색 범위를 결정해야만, 시간을 절약하면서도 성능이 좋은 모델을 얻을 수 있음**
  - 데이터셋에 적합한 범위, 모델 깊이, 구조 등 고려
  - **탐색해야 할 범위가 늘어날 수록 실험 시간도 길어질 것**

케라스 튜너 사용하기(실습)

- ResNet과 Xception 모델 구조를 데이터셋에 적합하게 조절해줄 수는 없을까?
  - 모델 깊이, 핵심 구조의 반복 등
  - 이를 위해 케라스 튜너가 **HyperResNet, HyperXception**을 제공

```
01  from kerastuner.tuners import Hyperband
02  from kerastuner.applications import HyperResNet
03
04  hypermodel = HyperResNet(input_shape=(28, 28, 1), classes=10)
05
06  tuner = Hyperband(hypermodel,
07                    objective = 'val_accuracy',
08                    max_epochs = 10,
09                    # directory='my_path,
10                    # project_name='helloworld2'
11  )
12
13  tuner.search(x_train, y_train, epochs = 10, validation_data = (x_val, y_val))
```

요약 정리

1. 하이퍼파라미터 튜닝은 어렵지만 반드시 거쳐야 할 과정입니다.
2. 케라스는 하이퍼파라미터 튜닝을 위한 **케라스 튜너**를 제공하고 있습니다.
3. HyperResNet과 HyperXception은 ResNet과 Xception 모델을 튜닝할 수 있도록 도와줍니다.
4. 이 장에서 살펴볼 튜닝 방법 또는 과정에서 사용된 함수들은 케라스 튜너 공식 홈페이지에 더 자세하게 설명되어 있습니다.
5. **오토케라스(AutoKeras)**