2. 컴퓨터 비전 소개

2.1 의류 아이템 인식하기

MNIST 데이터셋

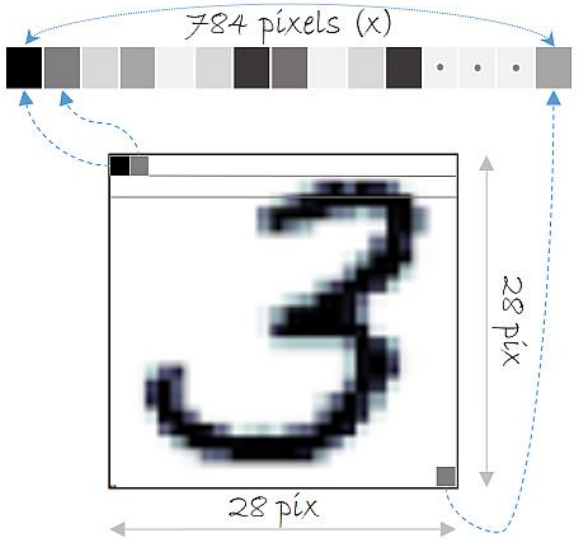
MNIST는 인공지능 연구의 권위자 LeCun교수가 만든 데이터 셋이고 현재 딥러닝을 공부할 때 반드시 거쳐야할 Hello, World같은 존재입니다. MNIST는 60,000개의 트레이닝 셋과 10,000개의 테스트 셋으로 이루어져 있고 이중 트레이닝 셋을 학습데이터로 사용하고 테스트 셋을 신경망을 검증하는 데에 사용합니다.
MNIST는 간단한 컴퓨터 비전 데이터 세트로, 아래와 같이 손으로 쓰여진 이미지들로 구성되어 있습니다. 숫자는 0에서 1까지의 값을 갖는 고정 크기 이미지 (28x28 픽셀)로 크기 표준화되고 중심에 배치되었습니다. 간단히 하기 위해 각 이미지는 평평하게되어 784 피쳐의 1-D numpy 배열로 변환되었습니다 (28 * 28).

1. 구하기 쉽습니다.
2. 모델의 성능을 확인하기 쉽습니다.
   MNIST 데이터셋으로 예측한 결과는 눈으로 확인하고 좋은지 나쁜지, 혹은 얼마나 좋은지를 파악하기가 용이합니다.
   성능이 좋은 모델인지 아닌지는 그저 눈으로 확인한 test image가 읽기 쉬운 숫자인지 아닌지를 판단해보면 알 수 있기 때문입니다.
3. 좋은 성능이 쉽게 나옵니다.
   간단한 모델 구성으로도 비교적 좋은 성능이 나오는 데다가 모델 구성별로 대략의 baseline 성능도 알려져 있기 때문에 그래프가 정상으로 작성되었는지, 하이퍼 파라미터는 적당했는지, 쉽게 판단이 가능합니다. 각 모델의 성능은 아래와 같습니다.
   - Linear Classfier : 92.4%
   - Neural Network(2 layer) : 98.4%
   - Neural Network(6 layer) : 99.65%
   - Convolution Neural Network(6 layer) : 99.79%

2.1 의류 아이템 인식하기

MNIST 이미지 데이터

MNIST 데이타셋에는 총 60,000개의 데이터가 있는데, 이 데이터는 크게 아래와 같이 세 종류의 데이터셋으로 나눠 집니다. 모델 학습을 위한 학습용 데이터인 mnist.train 그리고, 학습된 모델을 테스트하기 위한 테스트 데이터셋은 minst.test, 그리고 모델을 확인하기 위한 mnist.validation 데이터로 구별됩니다. 각 데이터는 아래와 같이 학습용 데이터 55000개, 테스트용 10,000개, 그리고, 확인용 데이터 5000개로 구성되어 있습니다.



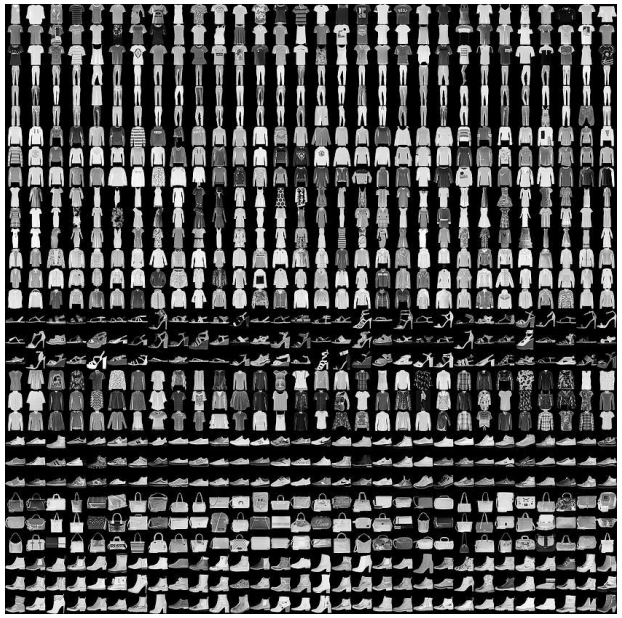
파일	목적
train-images-idx3-ubyte.gz	학습 셋 이미지 - 55000개의 트레이닝 이미지, 5000개의 검증 이미지
train-labels-idx1-ubyte.gz	이미지와 매칭되는 학습 셋 레이블
t10k-images-idx3-ubyte.gz	테스트 셋 이미지 - 10000개의 이미지
t10k-labels-idx1-ubyte.gz	이미지와 매칭되는 테스트 셋 레이블

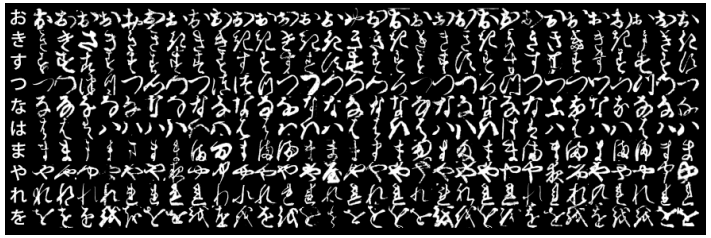
2.1 의류 아이템 인식하기

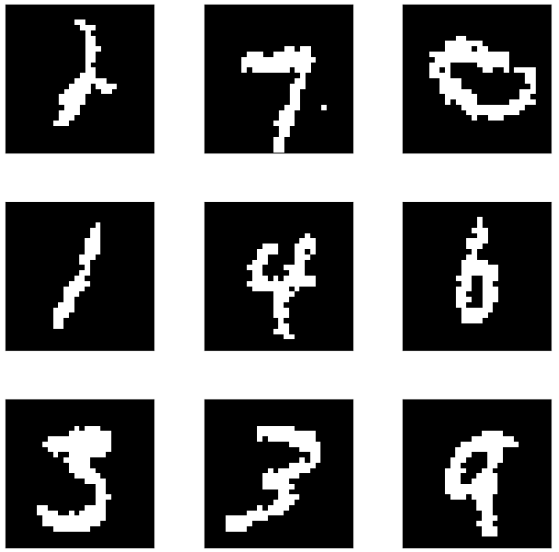
MNIST 데이터셋의 변형

기본 MNIST 데이터셋 외에도 예제를 위한 다양한 변형 MNIST 데이터셋이 존재합니다.

Fashion-MNIST 데이터셋은 아래와 같이 28×28 크기의 10개의 패션 아이템들을 모아놓은 데이터셋.
Kuzushiji-MNIST 데이터셋은 일본어 문자에 대한 MNIST 데이터셋 형태의 데이터셋.
Binarized-MNIST 데이터셋은 MNIST 데이터셋의 픽셀 밝기값(Pixel Intensity)를 0과 1로 이진화(Binarization)한 데이터셋.



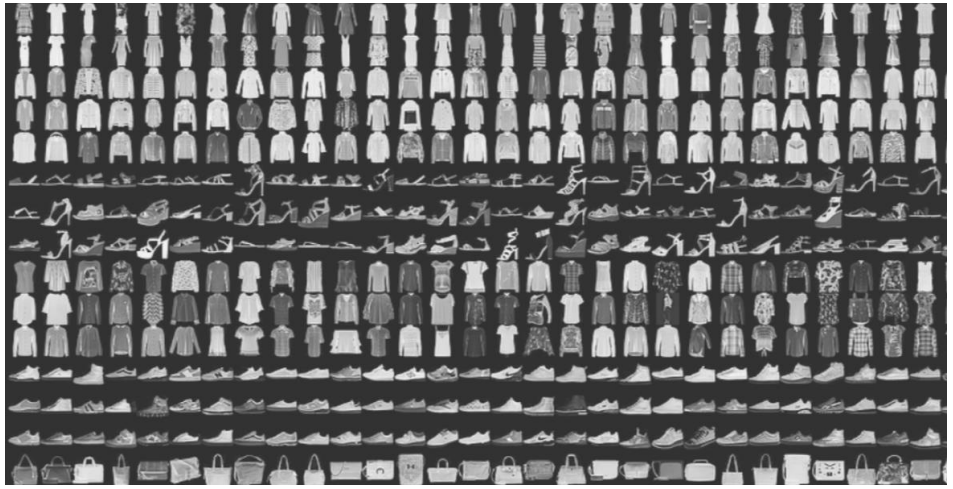


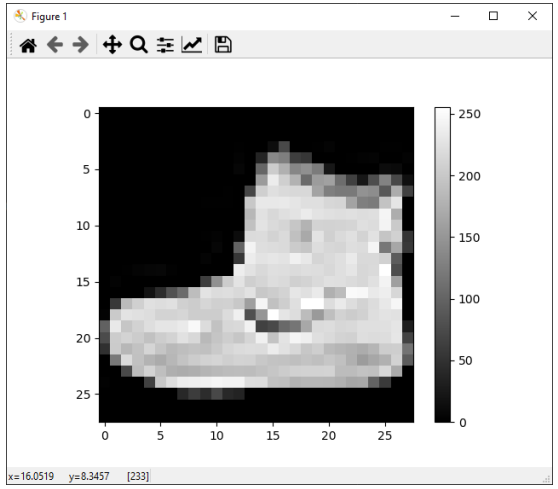


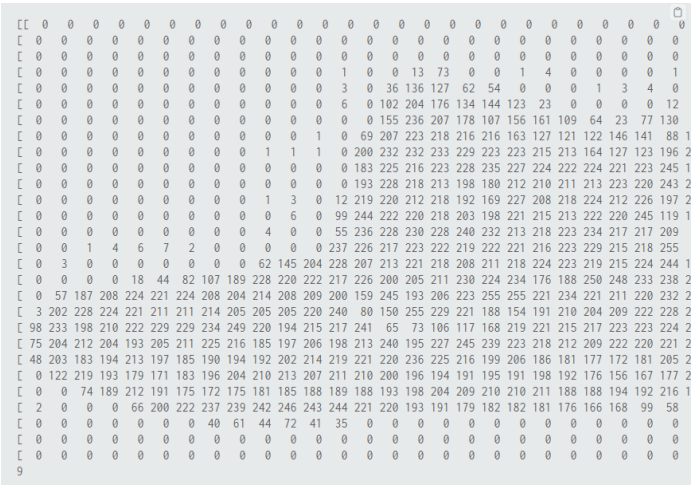
2.1 의류 아이템 인식하기

패션 MNIST 데이터셋

패션 MNIST는 MNIST를 그대로 대체할 수 있도록 설계되었습니다. 즉, 패션 MNIST의 샘플 개수와 이미지 크기, 클래스 개수가 MNIST와 동일합니다. 패션 MNIST에서는 0에서 9까지의 숫자 이미지 대신 10개의 의류 아이템 이미지를 가지고 있습니다.
의류 아이템에는 다양한 셔츠, 바지, 드레스, 여러 가지 종류의 신발이 있습니다. 그림에 나타나 있듯이 흑백 이미지이므로 각 사진은 0과 255 사이의 픽셀 값으로 구성됩니다. 덕분에 데이터셋을 쉽게 다룰 수 있습니다.







2.2 컴퓨터 비전을 위한 뉴런

패션 MNIST 데이터셋

fashion_mnist 모듈은 데이터셋을 반환하는 load_data() 함수를 포함하는데, load_data() 함수를 호출하면 NumPy 어레이의 튜플을 반환합니다.
train_images와 train_labels는 Neural Network 모델의 훈련 (training)에 사용되고, test_images와 test_labels는 테스트 (test)에 사용됩니다.

2.2 컴퓨터 비전을 위한 뉴런

전체 코드

```
import tensorflow as tf

data = tf.keras.datasets.fashion_mnist(training_images,
training_labels), (test_images, test_labels) = data.load_data()

training_images  = training_images / 255.0
test_images = test_images / 255.0

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
model.fit(training_images, training_labels, epochs=5)

model.evaluate(test_images, test_labels)

classifications = model.predict(test_images)
print(classifications[0])
print(test_labels[0])
```

2.3 신경망 설계

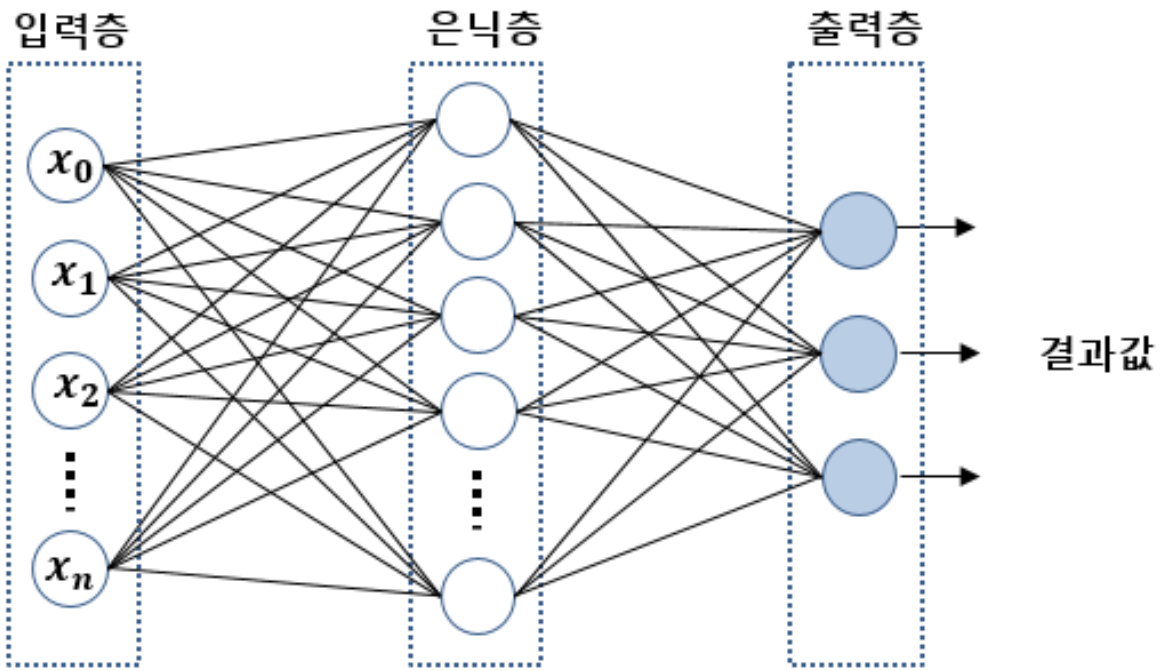
```
data = tf.keras.datasets.fashion_mnist
(training_images, training_labels), (test_images, test_labels) = data.load_data()
```

패션 MNIST는 6만 개의 훈련 이미지와 1만 개의 테스트 이미지로 구성됩니다. 따라서 data.load_data에서 반환받은 training_images는 6만 개의 28 X 28픽셀 배열이고 training_labels는 6만 개의 원소(0~9)로 이루어진 배열입니다. 비슷하게 test_images는 1만개의 28 X 28픽셀 배열이고 test_labels는 0에서 9 사이 1만 개의 값을 담은 배열입니다.

```
training_images  = training_images / 255.0
test_images = test_images / 255.0
```

이미지 픽셀이 모두 흑백이므로 0에서 255사이의 값을 가집니다. 255로 나누면 각 픽셀을 0에서 1 사이의 값으로 나타낼 수 있습니다. 이런 과정을 정규화(normalization)라고 합니다.
텐서플로에서 신경망을 훈련할 때 정규화가 성능을 높인다는 것을 기억하세요. 정규화되지 않은 데이터를 다루면 종종 신경망이 학습하지 못하거나 오류가 발생하기 쉽습니다.

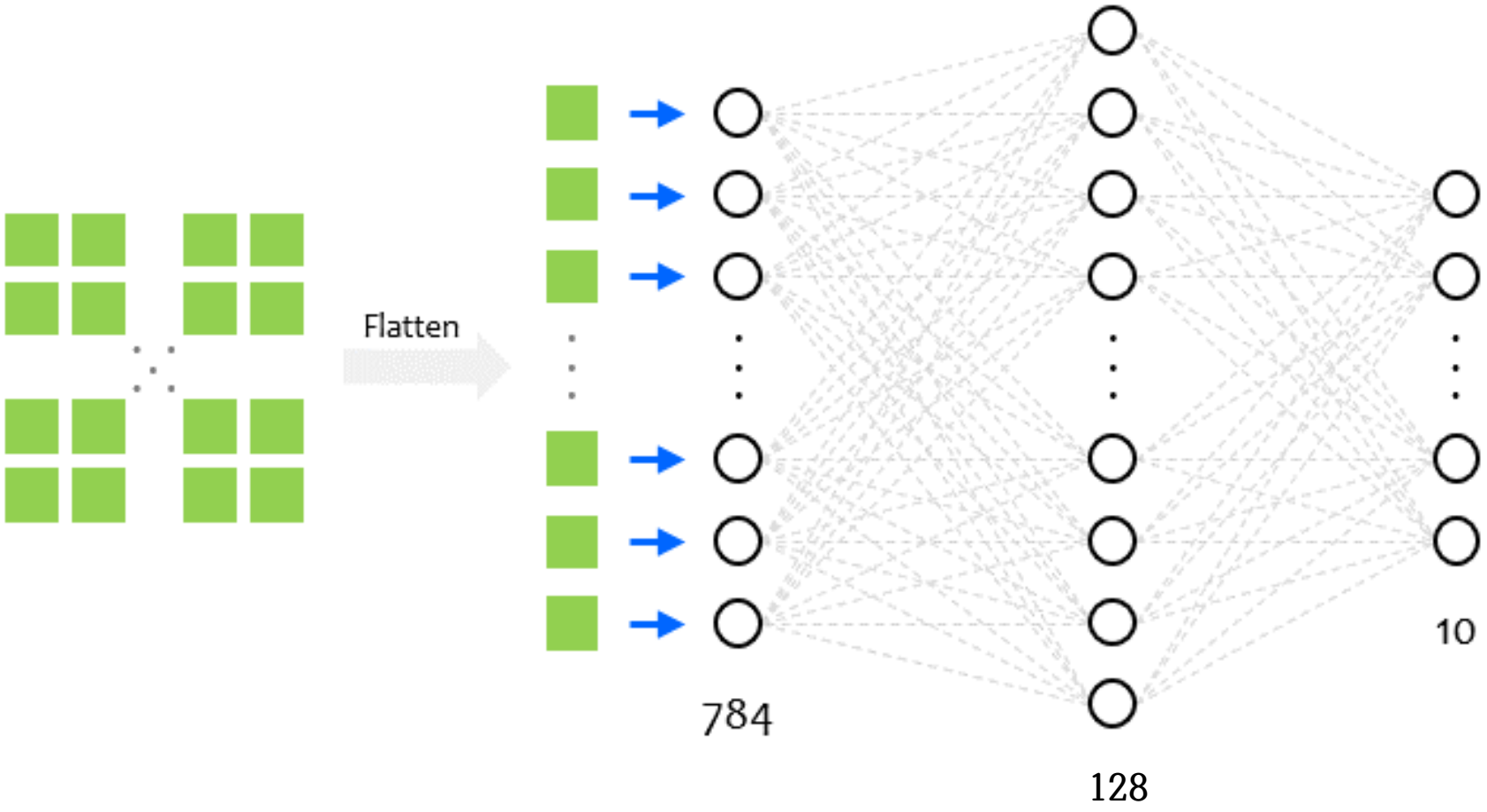
2.3 신경망 설계




```
model = tf.keras.models.Sequential([
            tf.keras.layers.Flatten(input_shape=(28, 28)),
            tf.keras.layers.Dense(128, activation=tf.nn.relu),
            tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

2.3 신경망 설계

첫 번째 Flatten은 뉴런의 층이 아니라 입력을 위한 크기를 지정합니다. 이 예제의 입력은 28 X 28 크기의 이미지이지만 한 줄로 펼친 숫자처럼 다루어야 합니다. Flatten 층은 2D 배열인 행렬을 1D 배열인 벡터로 변환합니다.



2.3 신경망 설계

그 다음 Dense는 뉴런의 층입니다. 여기서는 뉴런 128개를 지정했습니다. 이 부분이 바로 은닉층입니다. 입력과 출력 사이에 위치해 밖에서는 보이지 않으므로 '은닉'되었다고 말합니다. 뉴런 128개의 파라미터는 랜덤하게 초기화 됩니다. 이는 전적으로 임의로 지정된 숫자입니다. 사용할 뉴런의 개수를 지정하는 일정한 규칙은 없습니다. 층을 설계할 때 모델이 충분히 학습할 수 있는 적절한 뉴런의 개수를 선정해야 합니다. 이때 뉴런의 개수가 많으면 파라미터가 더 많아지므로 실행 속도가 느려집니다. 뉴런이 많으면 훈련 데이터를 잘 학습할 수 있는 신경망이 만들어지지만, 본 적이 없는 데이터 처리는 버거워 할 수 있습니다. 이를 **과대접합**이라 부르며 반대로 뉴런 개수가 적으면 모델의 학습에 필요한 파라미터가 부족하다는 의미입니다.
적절한 값을 선택하려면 약간의 실험이 필요합니다. 이런 과정을 **하이퍼파라미터 튜닝**이라고 부릅니다. 머신러닝에서 하이퍼파라미터는 훈련을 제어하는 데 사용되는 값입니다.
층에는 활성화 함수도 있습니다. 활성화 함수는 층의 각 뉴런에 적용되는 함수입니다. 텐서플로는 여러 가지 활성화 함수를 제공하지만 중간 층에 널리 사용되는 함수 중 하나는 렐루(ReLU)입니다. 이 함수는 단순히 0보다 큰 값을 반환하는 간단한 함수입니다. 여기에서는 다음 층의 계산에 음수값을 전달하고 싶지 않으므로 if~then 코드를 쓰는 대신 ReLU 활성화 함수를 사용할 수 있습니다.
마지막으로 또 다른 Dense 층이 있습니다. 이를 **출력 층**이라고 합니다. 클래스가 10개 이므로 10개의 뉴런을 둡니다. 각 뉴런은 입력 픽셀이 해당 클래스에 속할 확률을 출력합니다. 따라서 가장 높은 값을 가진 뉴런을 찾아야 합니다. 이 값을 찾기 위해 루프를 반복할 수 있지만 softmax 활성화 함수가 이를 대신합니다.

2.3 신경망 설계

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

모델을 컴파일할 때 손실 함수와 옵티마이저를 지정합니다. 이 손실 함수를 **희소한 범주형 크로스 엔트로피**라고 부릅니다. 텐서플로에서 제공하는 손실 함수 중 하나입니다. 사용할 손실 함수를 고르는 것은 하나의 예술입니다. 레이블이 정수(0~9)일 때는 sparse_categorical_crossentropy를 사용합니다. 원-핫 인코딩된 레이블의 경우는 categorical_crossentropy 손실 함수를 사용합니다.
의류 아이템은 10개의 카테고리 중 하나에 속해 있기 때문에 범주형 손실 함수를 사용합니다. 이번 예제에서는 희소한 범주형 크로스 엔트로피가 좋은 선택입니다.
옵티마이저를 선택할 때도 동일합니다. adam 옵티마이저는 앞에서 사용한 확률적 경사 하강법 옵티마이저가 진화한 것으로 더 빠르고 효율적이라고 알려져 있습니다. 6만 개의 훈련 이미지를 다루기 때문에 훈련 속도를 높이는 것이 도움이 되므로 adam 옵티마이저를 선택했습니다.
이 코드에서 측정값을 리포트하기 위해 지정하는 새로운 줄을 볼 수 있습니다. 여기서는 훈련하는 동안 신경망의 정확도를 레포트하려고 합니다. 앞에서는 손실만 리포트했습니다. 손실이 감소되는 현상을 보며 신경망이 학습 중이라고 판단했습니다. 여기서는 신경망의 학습을 확인하기 위해 정확도를 살펴봅니다. 정확도는 입력 픽셀을 출력 레이블에 얼마나 정확하게 매핑하는지를 알려줍니다.

2.4 신경망 훈련하기

```
model.fit(training_images, training_labels, epochs=5)
model.evaluate(test_images, test_labels)
```

훈련 이미지를 훈련 레이블에 매핑하도록 5번의 에폭 동안 신경망을 훈련합니다. 그 다음 모델을 평가하는 새로운 작업을 할 수 있습니다. 테스트를 위해 준비한 1만 개의 이미지와 레이블을 훈련된 모델에 전달해 각 이미지의 출력을 얻고 실제 레이블과 비교하며 결과를 확인합니다.
훈련 데이터에서보다 테스트 데이터에서 정확도가 낮은 이유가 궁금할 수 있습니다. 이는 매우 흔한 현상이며 이유를 곰곰이 생각해보면 이해할 수 있습니다. 신경망은 훈련에 사용한 이미지를 출력에 매핑하는 방법만 알고 있습니다. 데이터가 충분히 제공되면 훈련에 사용된 샘픔을 일반화 할 수 있으리라 기대합니다. 즉 신발이나 드레스처럼 보이는 것을 학습합니다. 하지만 본 적 없고 혼동하기 쉬운 다른 샘플도 항상 존재합니다.
예를 들어 스니커즈만 보고 자라서 스니커즈가 신발의 전부라고 알고 있다면 하이힐을 처음 보았을 때 조금 당황할 수 있습니다. 경험상 신발이라고 추측하지만 확신하지 못합니다. 이와 비슷한 개념입니다.

2.5 모델 출력 살펴보기

```
classifications = model.predict(test_images)
print(classifications[0])
print(test_labels[0])
```

model.predict 메서드에 테스트 이미지를 전달해 분류 결과를 얻습니다. 그 다음 첫 번째 분류 결과를 출력해 테스트 레이블과 비교합니다.
분류 결과는 배열로 전달됩니다. 이 값은 10개의 뉴런에서 출력한 것입니다. 첫 번째 의류 아이템의 실제 레이블은 9입니다. 배열을 살펴보면 다른 값들은 매우 작고 마지막 값인 배열 인덱스 9가 가장 크다는 것을 확인할 수 있습니다. 이 값은 이미지가 특정 인덱스의 레이블에 매칭될 확률입니다. 따라서 이 신경망은 인덱스 0에 있는 의류 아이템이 91.4%의 확률로 레이블 9라고 출력한 것입니다. 이 이미지는 레이블 9이므로 정확하게 맞추었습니다.

2.6 더 오래 훈련하기: 과대 적합

앞에서는 5번의 에폭 동안만 훈려했습니다. 즉 뉴런을 랜덤하게 초기화한 후에 훈련 세트를 모델에 전달해 그 결과를 레이블과 비교하고, 손실 함수로 성능을 측정하고, 그 다음 옵티마이저를 업데이트하는 전체 훈련 과정을 5번 수행했습니다. 이로써 얻은 결과는 꽤 훌륭했습니다.
훈련 세트에서 89%의 정확도, 테스트 세트에서는 87%의 정화도를 달성했습니다. 어 오래 훈련을 하면 어떻게 될까요?
에폭 수를 5에서 50으로 늘려 훈련해보죠. 기본보다 훨씬 높은 정확도를 달성했습니다. 훈련 세트의 정확도는 크게 높아졌지만 테스트 세트에 대한 정확도는 조금만 향상되었습니다. 오래 훈련했기 때문에 더 높은 성능이 나오지만 항상 이러한 결과가 나오지는 않습니다.
이 신경망은 훈련 세트에서 훨씬 높은 성능을 달성했지만 월등하게 더 나은 모델은 아닙니다. 사실 정확도 차이가 더 커졌으므로 모델이 훈련 세트에 특화되었음을 보여줍니다. 이를 과대적합이라고 부릅니다.
신경망을 만들 때 주의해야 할 문제입니다.

2.7 훈련 조기 종료

지금까지 훈련할 에폭 횟수를 하드코딩했습니다. 이런 방법도 가능하지만 원하는 성능이 나올 때까지 에폭 수를 바꾸고 훈련을 반복하는 대신, 원하는 정확도에 도달하면 훈련을 멈추게 하는 것이 더 좋습니다. 예를 들어 몇 번의 에폭이 필요할지 정확히는 모르겠지만 모델이 훈련 세트에서 95%의 정확도에 도달할 때까지 훈련하고 싶다면 어떻게 해야 할까요? 가장 쉬운 방법은 훈련하는 동안 콜백을 사용하는 것입니다.

```
import tensorflow as tf

class myCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if(logs.get('accuracy')>0.95):
            print("₩n정확도 95%에 도달하여 훈련을 멈춥니다!")
            self.model.stop_training = True
callbacks = myCallback()
mnist = tf.keras.datasets.fashion_mnist

data = tf.keras.datasets.fashion_mnist
(training_images, training_labels), (test_images, test_labels) =
data.load_data()
```

2.7 훈련 조기 종료

```
training_images  = training_images / 255.0
test_images = test_images / 255.0

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(training_images, training_labels, epochs=50, callbacks=[callbacks])
```

2.7 훈련 조기 종료

먼저 myCallback 클래스를 만들었습니다. 이 클래스는 tf.keras.callbacks.Callback 클래스를 상속합니다. myCallback 클래스 안에 on_epoch_end 함수를 정의합니다. 이 함수로 에폭에 대한 자세한 로그가 제공됩니다. 이 로그에는 정확도가 포함되어 있기 때문에 이 값이 0.95(95%)보다 큰지 확인할 수 있습니다. 정확도가 0.95보다 크면 self.model.stop_training = True로 지정해 훈련을 멈춥니다.
myCallback 클래스를 정의한 후 이 클래스의 인스턴스 callbacks 객체를 만듭니다.
이제 model.fit 호출을 확인해보죠. 에폭 횟수를 50으로 업데이트했고 callbacks 매개변수를 추가했습니다. 이 매개변수에 callbacks 객체를 전달합니다.
훈련할 때 매 에폭 끝에서 콜백 함수가 호출됩니다. 에폭이 종료될 때마다 정확도를 확인하고 약 34번째 에폭에서 훈련 정확도가 95%에 도달해 훈련이 종료됩니다(랜덤한 초기화로 인해 여러분의 실행 결과가 다를 수 있지만 비슷한 에폭 횟수에서 종료될 것입니다.)

이번 장에서는 하나의 뉴런을 넘어서 첫 번째 매우 기본적인 컴퓨터 비전 신경망을 만드는 법을 배웠습니다. 데이터 때문에 약간의 제한이 있었습니다. 모든 이미지는 28 X 28 크기 흑백이고 의류 아이템은 사진 중앙에 놓여 있습니다. 시작하기에는 좋지만 매우 통제된 예제입니다. 비전 작업을 더 잘 수행하려면 컴퓨터가 워본 픽셀 대신에 이미지의 특징을 학습해야 합니다.

Reference

개발자를 위한 머신러닝&딥러닝, Laurence Moroney, 박해선, 한빛미디어
https://github.com/rickiepark/aiml4coders
외 본문 링크