Tensorflow

https://www.tensorflow.org/

Contents

- Tensorflow?
- 설치
- 학습 - 예제들

Tensorflow 2.0 설치하기

https://www.tensorflow.org/install

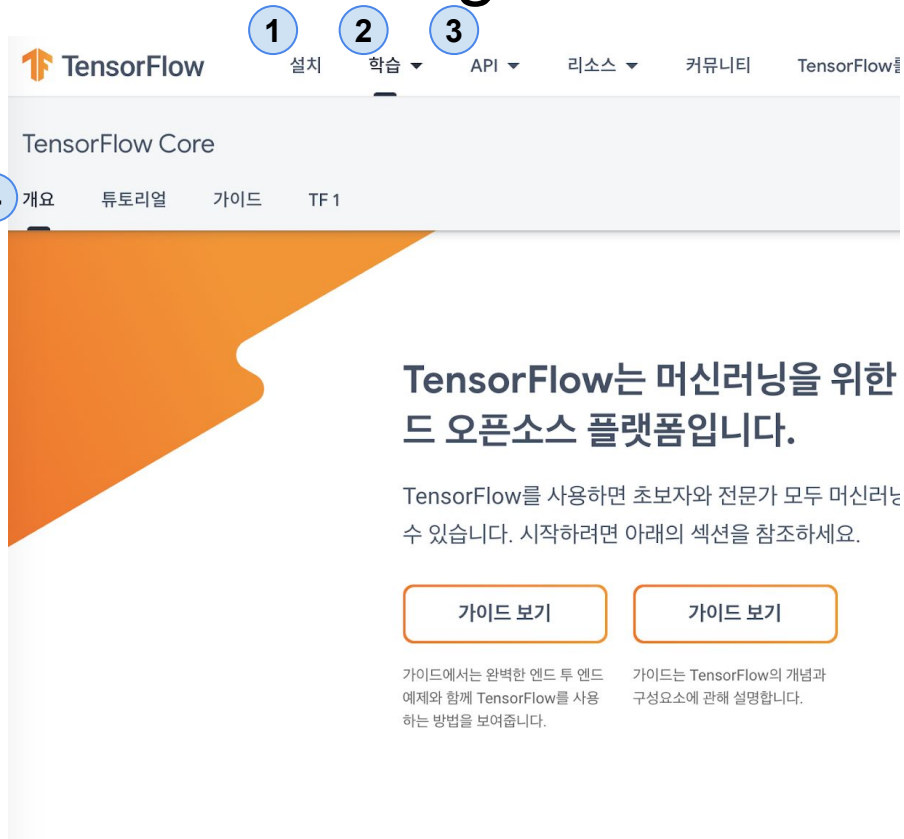
TensorFlow는 다음 64비트 시스템에서 테스트 및 지원됩니다.

- Python 3.5-3.7
- Ubuntu 16.04 이상
- Windows 7 이상
- macOS 10.12.6(Sierra) 이상(GPU 지원 없음)
- Raspbian 9.0 이상

```
# Requires the latest pip
pip install --upgrade pip

# Current stable release for CPU and
GPU
pip install tensorflow

# Or try the preview build (unstable)
pip install tf-nightly
```

colab 에서 설치된 환경 바로 사용 가능

Tensorflow.org 둘러보기



1. Tensorflow 설치 방법
2. Tensorflow 사용법 학습
3. Tensorflow API 문서
4. 2.의 세부 페이지
   a. 개요
   b. 튜토리얼
   다양한 예제 및 설정법 포함
   c. 가이드
   d. TF 1

TF 2.0 바로 시작하기 (MNIST tutorial)

초보자용

사용자에게 친숙한 Sequential API로 시작하는 것이 가장 좋습니다. 구성요소를 연결하여 모델을 만들 수 있습니다. 아래의 'Hello World' 예제를 실행한 다음 가이드를 방문하여 자세한 내용을 알아보세요.

ML에 관해 배워보려면 교육 페이지를 확인하세요. 엄선된 커리큘럼으로 기본적인 ML 분야의 역량을 키워보세요.

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

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

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

지금 코드 실행 Google의 대화형 메모장에서 사용해보기

전문가용

Subclassing API는 고급 연구를 위한 define-by-run 인터페이스를 제공합니다. 모델에 대한 클래스를 만든 다음 명령형으로 순방향 패스를 작성합니다. 맞춤형 레이어, 활성화 및 학습 루프를 쉽게 만들 수 있습니다. 아래의 'Hello World' 예제를 실행한 다음 가이드를 방문하여 자세한 내용을 알아보세요.

```
class MyModel(tf.keras.Model):
  def __init__(self):
    super(MyModel, self).__init__()
    self.conv1 = Conv2D(32, 3, activation='relu')
    self.flatten = Flatten()
    self.d1 = Dense(128, activation='relu')
    self.d2 = Dense(10, activation='softmax')

  def call(self, x):
    x = self.conv1(x)
    x = self.flatten(x)
    x = self.d1(x)
    return self.d2(x)
model = MyModel()

with tf.GradientTape() as tape:
  logits = model(images)
  loss_value = loss(logits, labels)
grads = tape.gradient(loss_value, model.trainable_variable
optimizer.apply_gradients(zip(grads, model.trainable_varia
```

지금 코드 실행 Google의 대화형 메모장에서 사용해보기

필수 문서

TensorFlow 설치

패키지를 설치하거나 소스에서 빌드합니다. CUDA® 지원 카드에 GPU 지원이 제공됩니다.

TensorFlow 2

코드를 이전하기 위한 TensorFlow 2 권장사항 및 도구입니다.

Keras

Keras는 ML 연구자뿐 아니라 초보자도 쉽게 사용할 수 있는 상위 수준 API입니다.

맞춤설정

연구 및 실험을 위한 TensorFlow입니다. 맞춤 레이어 및 모델, 정방향 전달, 학습 루프를 작성합니다.

데이터 입력 파이프라인

`tf.data` API를 사용하면 간단하고 재사용 가능한 조각으로 복잡한 입력 파이프라인을 빌드할 수 있습니다.

에스티메이터

확장 및 비동기 학습을 위해 설계된 완전한 모델을 나타내는 상위 수준 API입니다.

모델 저장

체크포인트 또는 저장된 모델 형식을 사용하여 TensorFlow 모델을 저장합니다.

가속기

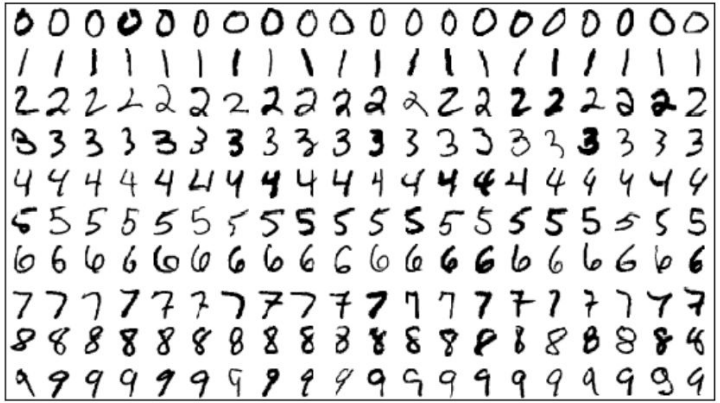
여러 GPU, 여러 머신 또는 TPU에 학습을 배포합니다.

성능

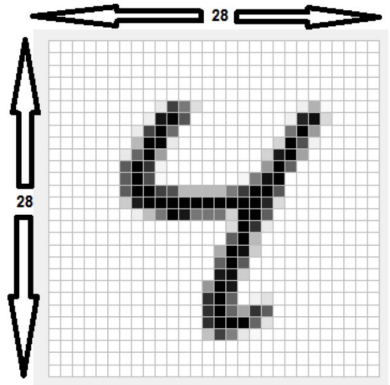
TensorFlow 성능 최적화를 위한 권장사항 및 최적화 기술

MNIST data

MNIST Dataset



28x28x1 image



train-images-idx3-ubyte.gz: training set images (9912422 bytes)
train-labels-idx1-ubyte.gz: training set labels (28881 bytes)
t10k-images-idx3-ubyte.gz: test set images (1648877 bytes)
t10k-labels-idx1-ubyte.gz: test set labels (4542 bytes)

http://yann.lecun.com/exdb/mnist/

TF 2.0 바로 시작하기

지난 시간까지 배운것들..

- 퍼셉트론
- 활성화 함수
- 다차원 배열의 계산
- 신경망 구현하기, 출력층 설계하기
- 데이터 추론
- 손실함수
- 미니배치
- 경사 하강법 (수치 미분), 오차역전파법

TF 2.0 바로 시작하기 (MNIST tutorial)

```python
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

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

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
Epoch 1/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3037 - accuracy: 0.9113
Epoch 2/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1473 - accuracy: 0.9562
Epoch 3/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1129 - accuracy: 0.9662
Epoch 4/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0912 - accuracy: 0.9718
Epoch 5/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0783 - accuracy: 0.9755
313/313 [==============================] - 1s 2ms/step - loss: 0.0822 - accuracy: 0.9763
[0.08216668665409088, 0.9763000011444092]
```

tf.keras.datasets

Modules

`boston_housing` module: Boston housing price regression dataset.

`cifar10` module: CIFAR10 small images classification dataset.

`cifar100` module: CIFAR100 small images classification dataset.

`fashion_mnist` module: Fashion-MNIST dataset.

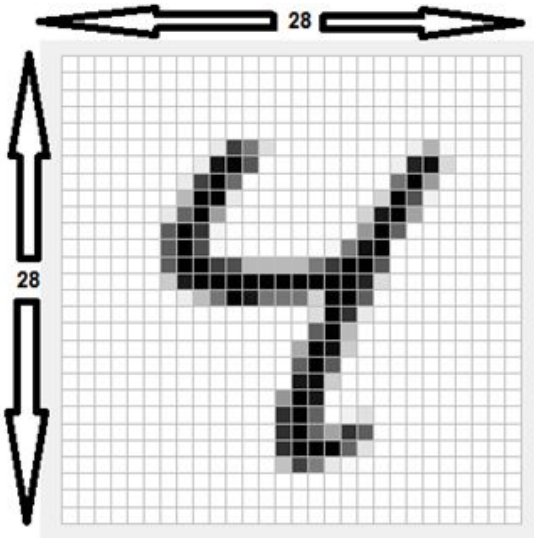
`imdb` module: IMDB sentiment classification dataset.

`mnist` module: MNIST handwritten digits dataset.

`reuters` module: Reuters topic classification dataset.

Training Neural Network

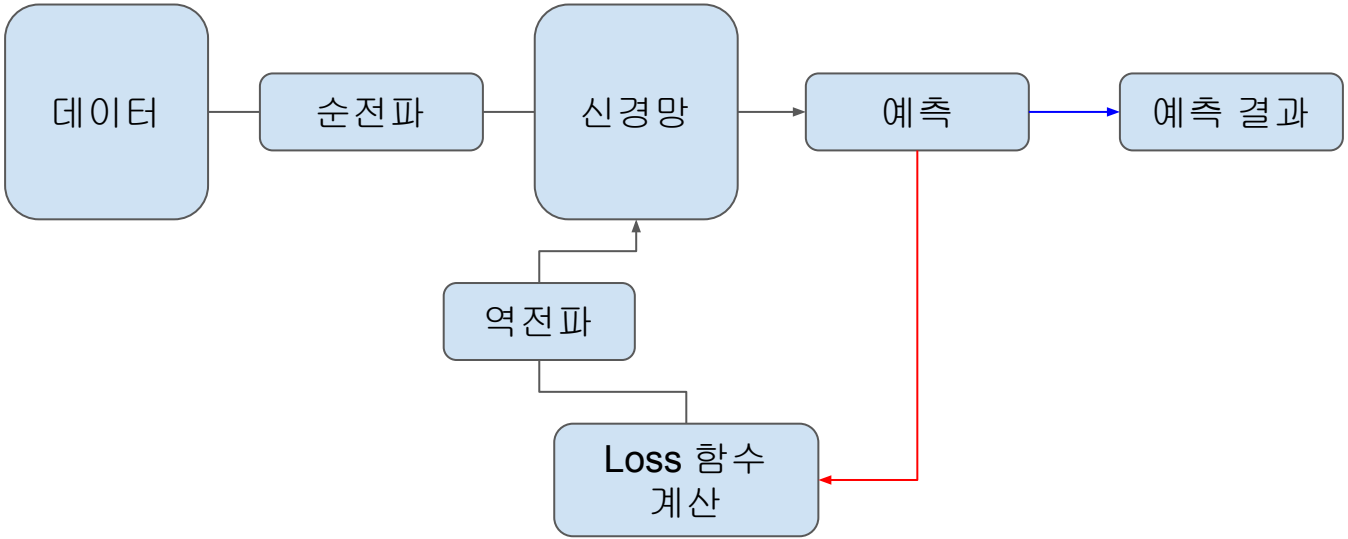
MNIST dataset



Training MNIST data

Training MNIST data

Colab에서 학습해보기



Training MNIST data

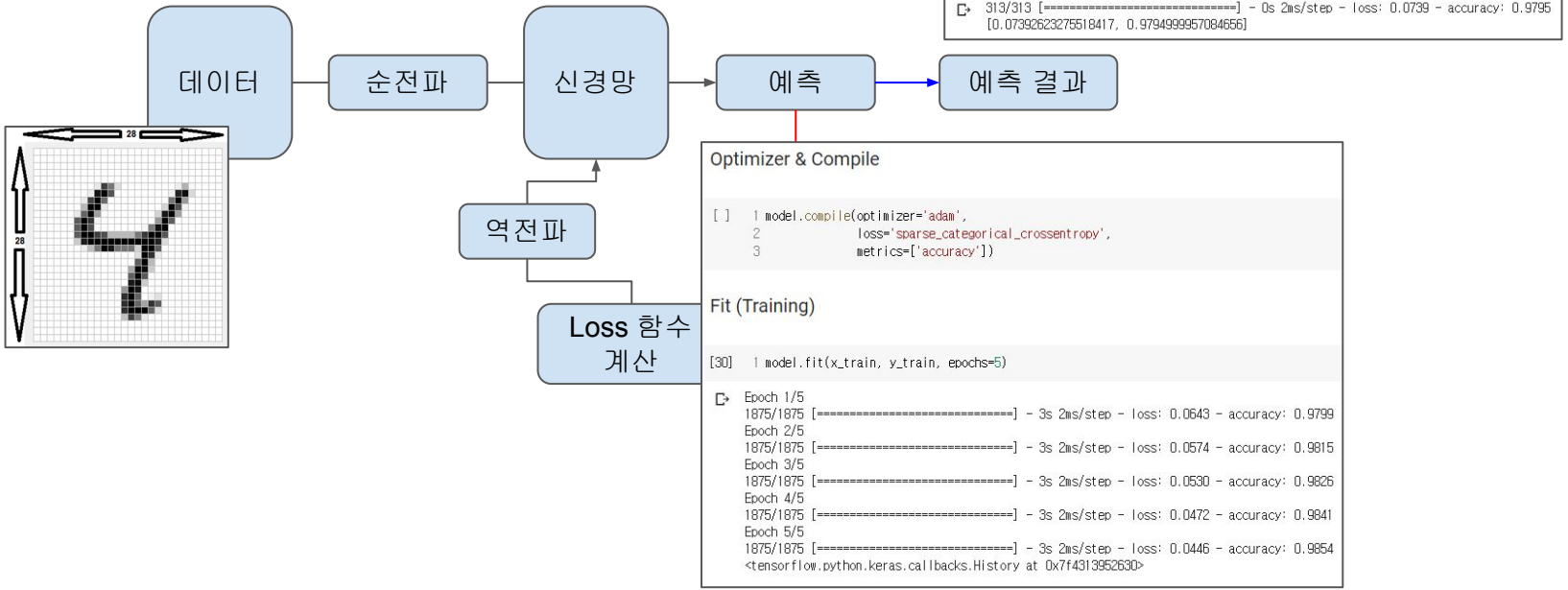
[Colab에서 학습해보기]

Model

```
[ ]  1 model = tf.keras.models.Sequential([
     2   tf.keras.layers.Flatten(input_shape=(28, 28)),
     3   tf.keras.layers.Dense(128, activation='relu'),
     4   tf.keras.layers.Dropout(0.2),
     5   tf.keras.layers.Dense(10, activation='softmax')
     6 ])
```

Evaluate (Testing)

```
[31]  1 model.evaluate(x_test, y_test)

      313/313 [==============================] - 0s 2ms/step - loss: 0.0739 - accuracy: 0.9795
      [0.07392623275518417, 0.9794999957084656]
```



Optimizer & Compile

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

Fit (Training)

```
[30]  1 model.fit(x_train, y_train, epochs=5)

      Epoch 1/5
      1875/1875 [==============================] - 3s 2ms/step - loss: 0.0643 - accuracy: 0.9799
      Epoch 2/5
      1875/1875 [==============================] - 3s 2ms/step - loss: 0.0574 - accuracy: 0.9815
      Epoch 3/5
      1875/1875 [==============================] - 3s 2ms/step - loss: 0.0530 - accuracy: 0.9826
      Epoch 4/5
      1875/1875 [==============================] - 3s 2ms/step - loss: 0.0472 - accuracy: 0.9841
      Epoch 5/5
      1875/1875 [==============================] - 3s 2ms/step - loss: 0.0446 - accuracy: 0.9854
      <tensorflow.python.keras.callbacks.History at 0x7f4313952630>
```

Training MNIST data

Model

```
[ ]  1 model = tf.keras.models.Sequential([
     2   tf.keras.layers.Flatten(input_shape=(28, 28)),
     3   tf.keras.layers.Dense(128, activation='relu'),
     4   tf.keras.layers.Dropout(0.2),
     5   tf.keras.layers.Dense(10, activation='softmax')
     6 ])
```

```
[32]  1 model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 128)               100480
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

Optimizer & Compile

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

Fit (Training)

```
[30]  1 model.fit(x_train, y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0643 - accuracy: 0.9799
Epoch 2/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0574 - accuracy: 0.9815
Epoch 3/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0530 - accuracy: 0.9826
Epoch 4/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0472 - accuracy: 0.9841
Epoch 5/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0446 - accuracy: 0.9854
<tensorflow.python.keras.callbacks.History at 0x7f4313952630>
```

Evaluate (Testing)

```
[31]  1 model.evaluate(x_test, y_test)
```

```
313/313 [==============================] - 0s 2ms/step - loss: 0.0739 - accuracy: 0.9795
[0.07392623275518417, 0.9794999957084656]
```

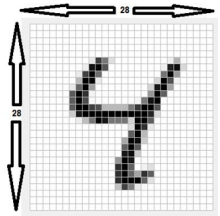
Training MNIST data

Model

```
[ ]  1 model = tf.keras.models.Sequential([
     2   tf.keras.layers.Flatten(input_shape=(28, 28)),
     3   tf.keras.layers.Dense(128, activation='relu'),
     4   tf.keras.layers.Dropout(0.2),
     5   tf.keras.layers.Dense(10, activation='softmax')
     6 ])
```

```
[32]  1 model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 128)               100480
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```



Flatten — input layer

Dense — hidden layer

Dropout — hidden layer

0	0	0	1	0	0	0	0	0	0

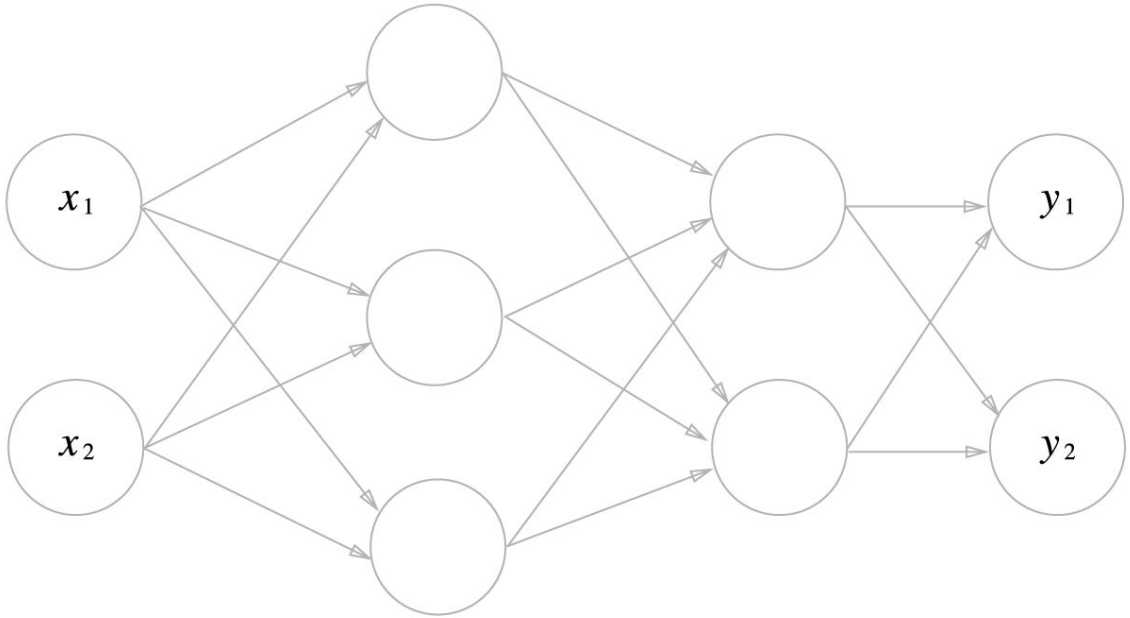
output layer

Training MNIST data

- Dense? Flatten?
- Dropout?
- softmax? relu?
- loss?
- optimizer?
- epochs?

Training MNIST data

- Dense? Flatten?
- Dropout?
- softmax? relu?
- loss?
- optimizer?
- epochs?



epoch / batch / iterations

In the neural network terminology:

one **epoch** = one forward pass and one backward pass of *all* the training examples

batch size = the number of training examples in one forward/backward pass. The higher the batch size, the more memory space you'll need.

number of **iterations** = number of passes, each pass using [batch size] number of examples. To be clear, one pass = one forward pass + one backward pass (we do not count the forward pass and backward pass as two different passes).

Example: if you have *1000 training examples*, and your *batch size is 500*, then it will take 2 iterations to complete 1 epoch.

epoch / batch / iterations

In the neural network terminology:

one **epoch** = one forward pass and one backward pass of *all* the training examples

전체 Training example 단위

batch size = the number of training e gher the batch size, the more memory space you'll need.

한번에 학습 하는 단위

number of **iterations** = number of passes ples. To be clear, one pass = one forward pass + one backward pass (w pass as two different passes).

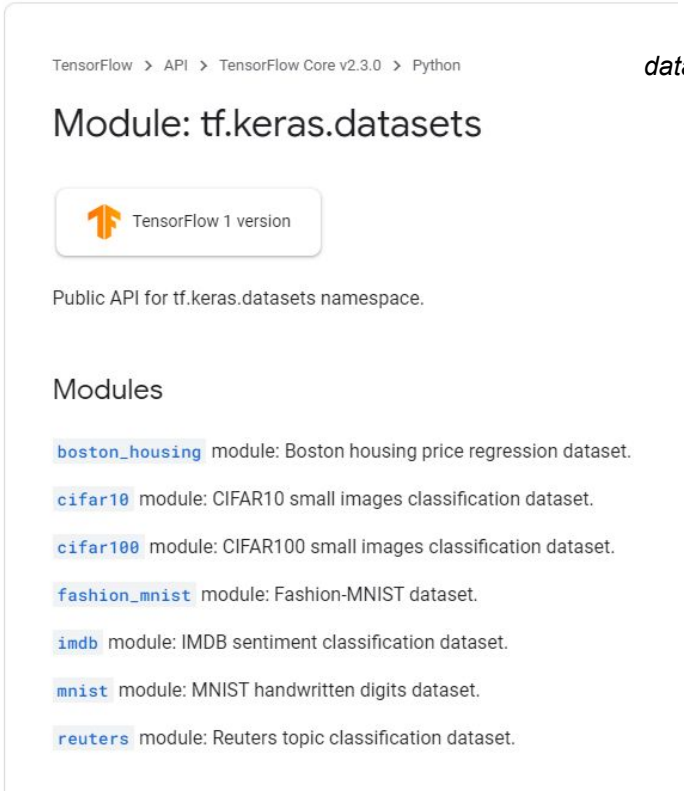
반복하는 회수

Example: if you have *1000 training examples*, and your *batch size is 500*, then it will take 2 iterations to complete 1 epoch.

1000개의 Training Example이 있고, **batch size**가 500이면,
1 **epoch**을 위해 2의 **iteration**이 필요하다

tf.keras.datasets

https://www.tensorflow.org/api_docs/python/tf/keras/datasets

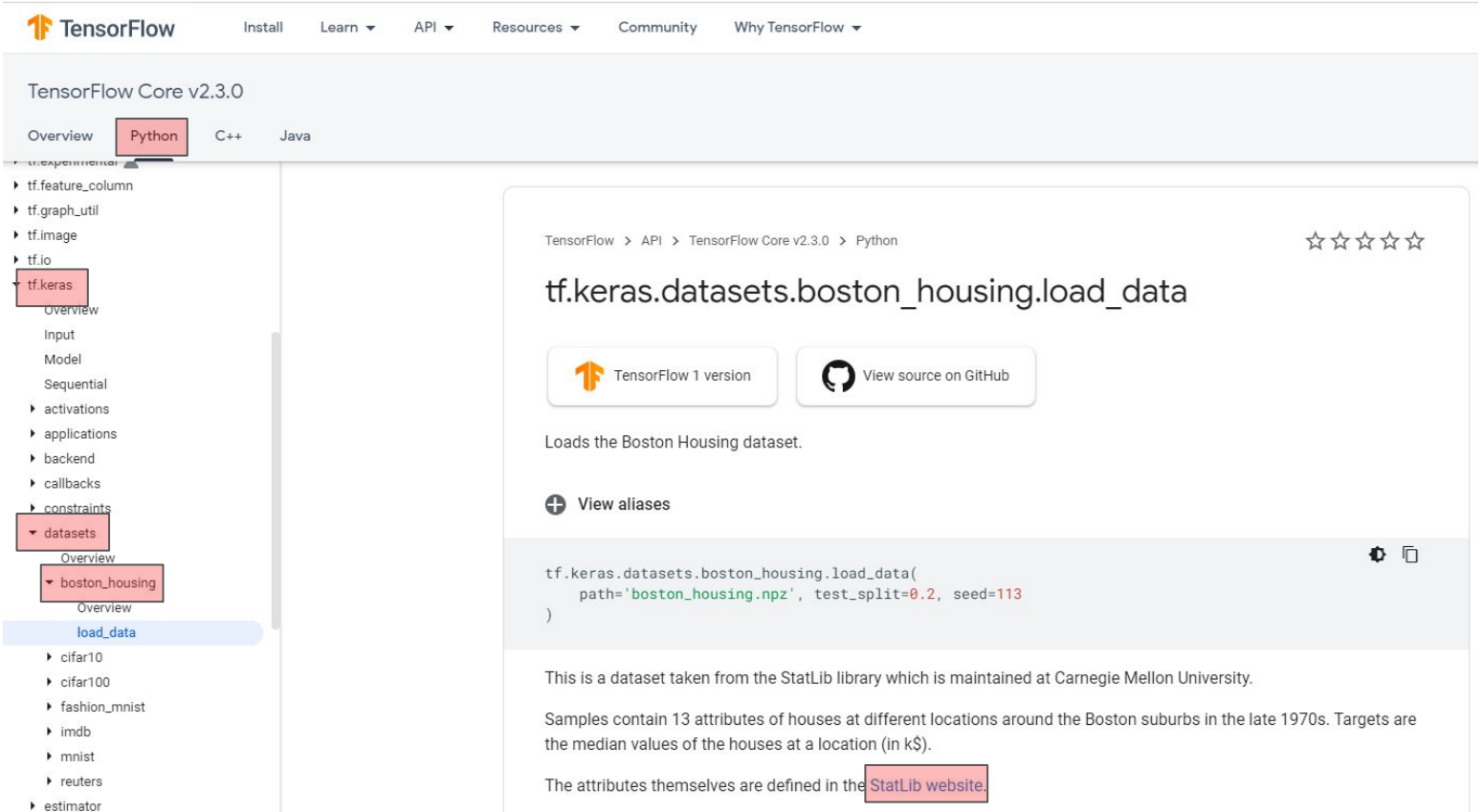


dataframe	`boston_housing` module: Boston housing price **regression** dataset.
image	`cifar10` module: CIFAR10 small **images classification** dataset.
image	`cifar100` module: CIFAR100 small **images classification** dataset.
image	`fashion_mnist` module: Fashion-MNIST dataset.
text	`imdb` module: IMDB **sentiment** classification dataset.
image	`mnist` module: MNIST handwritten **digits** dataset.
text	`reuters` module: Reuters **topic** classification dataset.

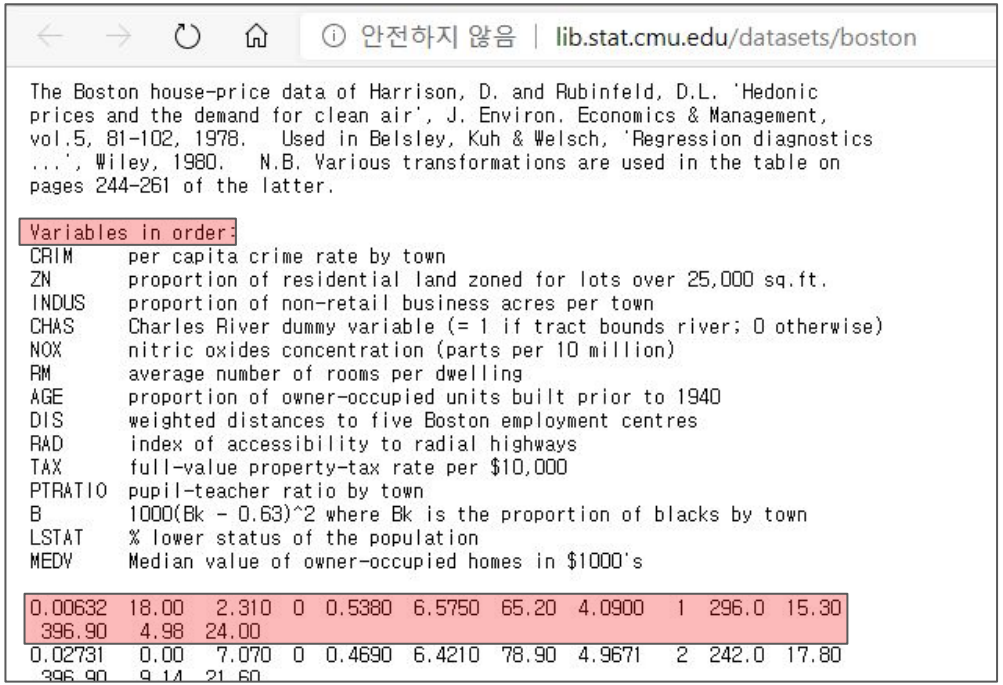
```
1 boston_data = tf.keras.datasets.
2 (x_train, y_train),(x_test, y_test) = boston_data.load_data()
```

module name

dataset 자세히 보기 - boston_housing 예시



dataset 자세히 보기 - boston_housing 예시



안전하지 않음 | lib.stat.cmu.edu/datasets/boston

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic prices and the demand for clean air', J. Environ. Economics & Management, vol.5, 81-102, 1978. Used in Belsley, Kuh & Welsch, 'Regression diagnostics ...', Wiley, 1980. N.B. Various transformations are used in the table on pages 244-261 of the latter.

Variables in order:

CRIM	per capita crime rate by town
ZN	proportion of residential land zoned for lots over 25,000 sq.ft.
INDUS	proportion of non-retail business acres per town
CHAS	Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
NOX	nitric oxides concentration (parts per 10 million)
RM	average number of rooms per dwelling
AGE	proportion of owner-occupied units built prior to 1940
DIS	weighted distances to five Boston employment centres
RAD	index of accessibility to radial highways
TAX	full-value property-tax rate per $10,000
PTRATIO	pupil-teacher ratio by town
B	1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
LSTAT	% lower status of the population
MEDV	Median value of owner-occupied homes in $1000's

0.00632 18.00 2.310 0 0.5380 6.5750 65.20 4.0900 1 296.0 15.30
396.90 4.98 24.00
0.02731 0.00 7.070 0 0.4690 6.4210 78.90 4.9671 2 242.0 17.80
396.90 9.14 21.60

```
1 boston_data = tf.keras.datasets.boston_housing
2 (x_train, y_train),(x_test, y_test) = boston_data.load_data()
```

```
1 x_train[0]
```

```
array([  1.23247,   0.     ,   8.14   ,   0.     ,   0.538  ,   6.142  ,
        91.7    ,   3.9769 ,   4.     , 307.     ,  21.     , 396.9    ,
        18.72   ])
```

Sequential 모델

`Sequential` 모델은 각 레이어에 **정확히 하나의 입력 텐서와 하나의 출력 텐서**가 있는 **일반 레이어 스택**에 적합합니다.

```
# Define Sequential model with 3 layers
model = keras.Sequential(
    [
        layers.Dense(2, activation="relu", name="layer1"),
        layers.Dense(3, activation="relu", name="layer2"),
        layers.Dense(4, name="layer3"),
    ]
)
# Call model on a test input
x = tf.ones((3, 3))
y = model(x)
```

```
# Create 3 layers
layer1 = layers.Dense(2, activation="relu", name="layer1")
layer2 = layers.Dense(3, activation="relu", name="layer2")
layer3 = layers.Dense(4, name="layer3")

# Call layers on a test input
x = tf.ones((3, 3))
y = layer3(layer2(layer1(x)))
```

Sequential 모델

Sequential 모델은 다음의 경우에 **적합하지 않습니다**.

- 모델에 다중 입력 또는 다중 출력이 있습니다
- 레이어에 **다중 입력** 또는 **다중 출력**이 있습니다
- **레이어 공유**를 해야 합니다
- 비선형 토폴로지를 원합니다(예: 잔류 연결, 다중 분기 모델)

Sequential 모델

Layer의 추가와 삭제

`add()` 메서드를 통해 Sequential 모델을 점진적으로 작성할 수도 있습니다.

```
model = keras.Sequential()
model.add(layers.Dense(2, activation="relu"))
model.add(layers.Dense(3, activation="relu"))
model.add(layers.Dense(4))
```

레이어를 제거하는 `pop()` 메서드도 있습니다. Sequential 모델은 레이어의 리스트와 매우 유사하게 동작합니다.

```
model.pop()
print(len(model.layers))  # 2
```

Training CIFAR data

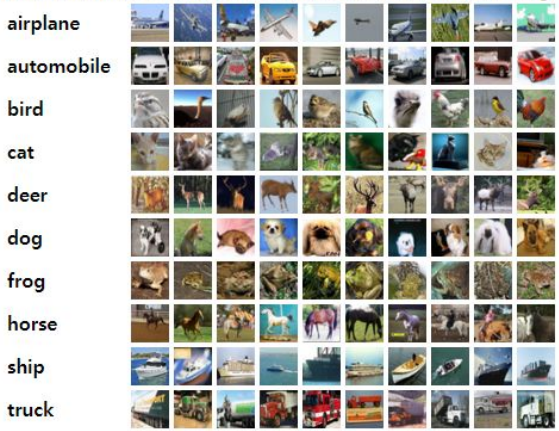
https://www.cs.toronto.edu/~kriz/cifar.html

The CIFAR-10 dataset

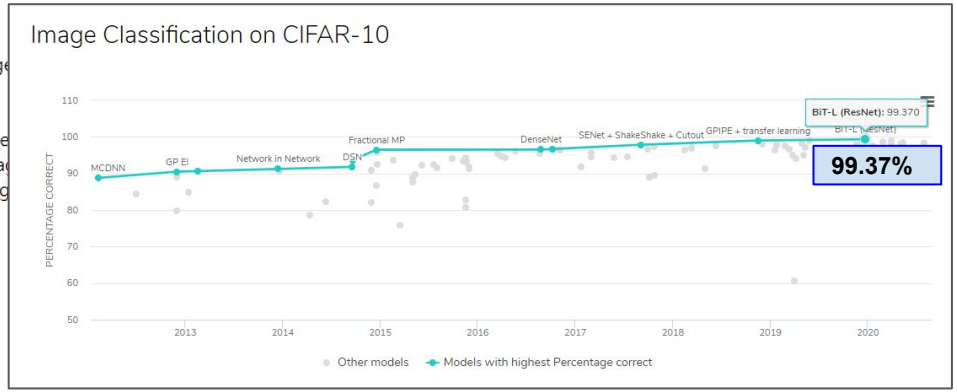
The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 image
images and 10000 test images.

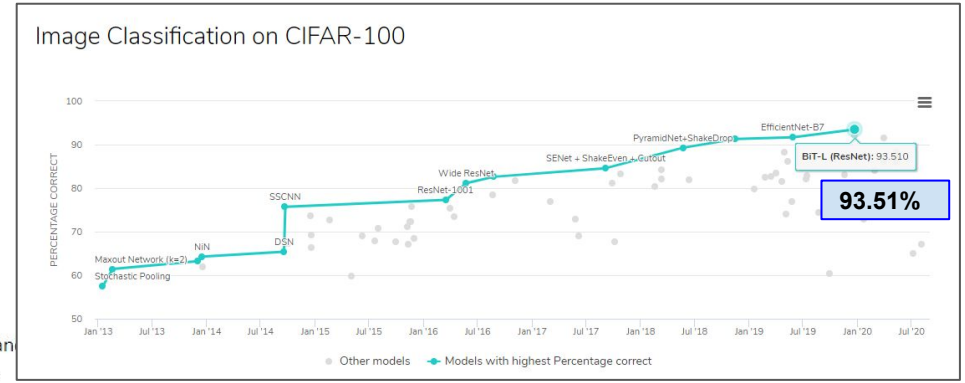
The dataset is divided into five training batches and one test batch, each with 10000 image
randomly-selected images from each class. The training batches contain the remaining ima
batches may contain more images from one class than another. Between them, the training
from each class.

Here are the classes in the dataset, as well as 10 random images from each:



The classes are completely mutually exclusive. There is no overlap between automobiles an
SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.





Training CIFAR-10 data

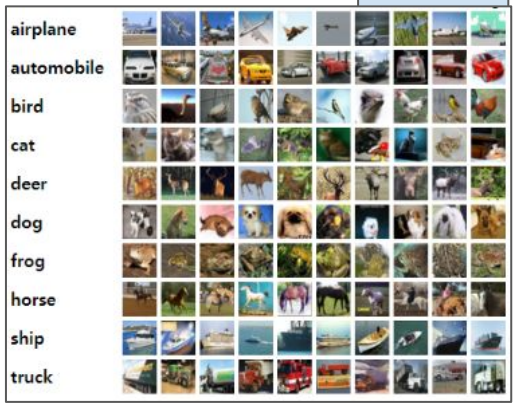
Model

```
[ ]  1 model = tf.keras.models.Sequential([
     2   tf.keras.layers.Flatten(input_shape=(28, 28)),
     3   tf.keras.layers.Dense(128, activation='relu'),
     4   tf.keras.layers.Dropout(0.2),
     5   tf.keras.layers.Dense(10, activation='softmax')
     6 ])
```

Evaluate (Testing)

```
[31]  1 model.evaluate(x_test, y_test)

      313/313 [==============================] - 0s 2ms/step - loss: 0.0739 - accuracy: 0.9795
      [0.07392623275518417, 0.9794999957084656]
```

데이터 → 순전파 → 신경망 → 예측 → 예측 결과



역전파

Loss 함수 계산

Optimizer & Compile

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

Fit (Training)

```
[30]  1 model.fit(x_train, y_train, epochs=5)

      Epoch 1/5
      1875/1875 [==============================] - 3s 2ms/step - loss: 0.0643 - accuracy: 0.9799
      Epoch 2/5
      1875/1875 [==============================] - 3s 2ms/step - loss: 0.0574 - accuracy: 0.9815
      Epoch 3/5
      1875/1875 [==============================] - 3s 2ms/step - loss: 0.0530 - accuracy: 0.9826
      Epoch 4/5
      1875/1875 [==============================] - 3s 2ms/step - loss: 0.0472 - accuracy: 0.9841
      Epoch 5/5
      1875/1875 [==============================] - 3s 2ms/step - loss: 0.0446 - accuracy: 0.9854
      <tensorflow.python.keras.callbacks.History at 0x7f4313952630>
```

Training CIFAR-10 data

MNIST 학습에 사용했던 모델을 그대로 사용해본다

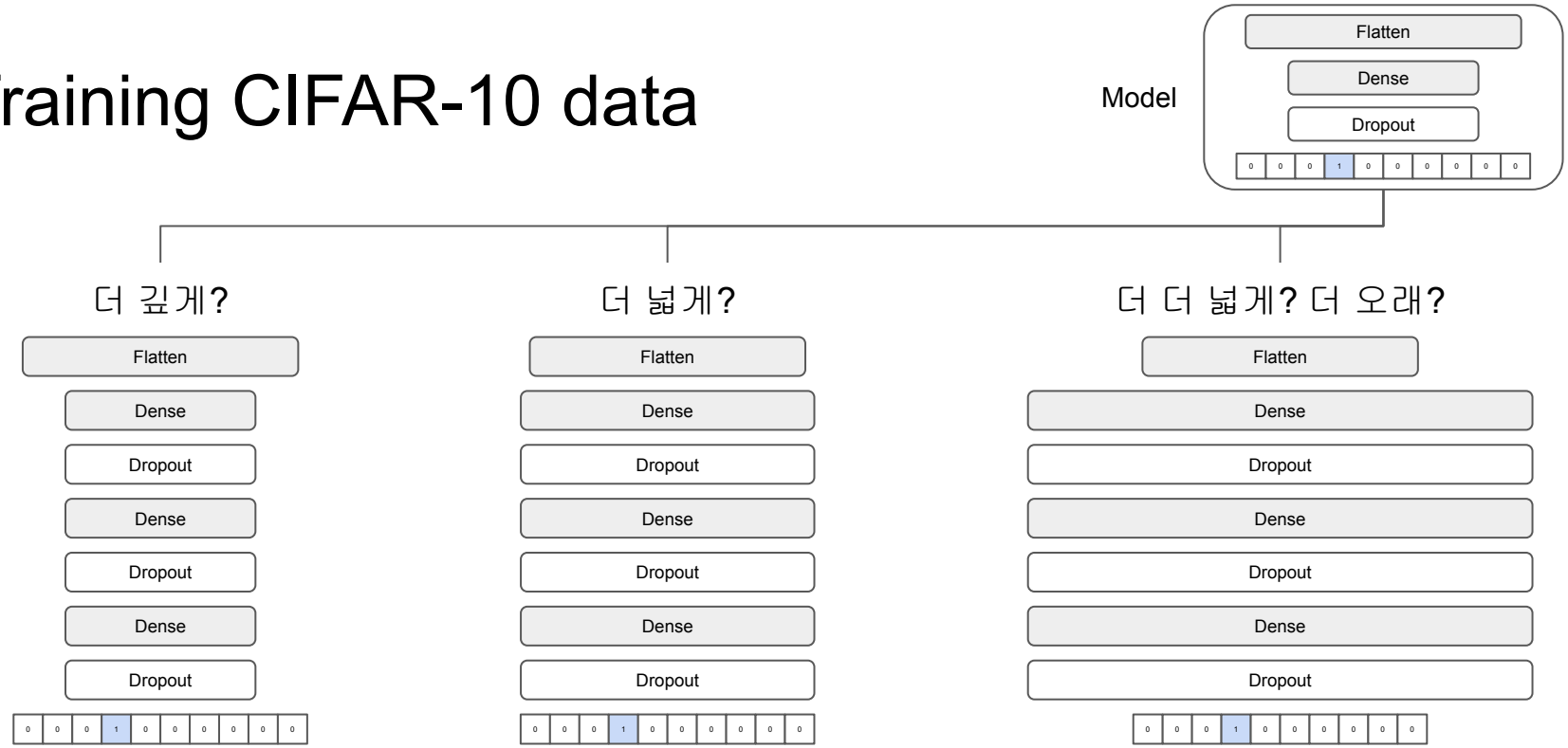
```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=x_shape),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.summary()

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

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_4 (Flatten)          (None, 3072)              0
_________________________________________________________________
dense_13 (Dense)             (None, 128)               393344
_________________________________________________________________
dropout_9 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_14 (Dense)             (None, 10)                1290
=================================================================
Total params: 394,634
Trainable params: 394,634
Non-trainable params: 0
_________________________________________________________________
```

```
Epoch 1/5
1563/1563 [==============================] - 3s 2ms/step - loss: 2.0212 - accuracy: 0.2469
Epoch 2/5
1563/1563 [==============================] - 3s 2ms/step - loss: 1.9198 - accuracy: 0.2915
Epoch 3/5
1563/1563 [==============================] - 4s 2ms/step - loss: 1.8924 - accuracy: 0.3037
Epoch 4/5
1563/1563 [==============================] - 3s 2ms/step - loss: 1.8731 - accuracy: 0.3095
Epoch 5/5
1563/1563 [==============================] - 3s 2ms/step - loss: 1.8685 - accuracy: 0.3129
313/313 [==============================] - 1s 2ms/step - loss: 1.7859 - accuracy: 0.3615
[1.7859476804733276, 0.36149999499320984]
```

```
1563/1563 [==============================] - 3s 2ms/step - loss: 1.8540 - accuracy: 0.3254
Epoch 16/20
1563/1563 [==============================] - 3s 2ms/step - loss: 1.8519 - accuracy: 0.3254
Epoch 17/20
1563/1563 [==============================] - 4s 2ms/step - loss: 1.8470 - accuracy: 0.3269
Epoch 18/20
1563/1563 [==============================] - 3s 2ms/step - loss: 1.8527 - accuracy: 0.3254
Epoch 19/20
1563/1563 [==============================] - 3s 2ms/step - loss: 1.8473 - accuracy: 0.3274
Epoch 20/20
1563/1563 [==============================] - 4s 2ms/step - loss: 1.8416 - accuracy: 0.3273
313/313 [==============================] - 1s 2ms/step - loss: 1.7571 - accuracy: 0.3696
[1.7570570707321167, 0.36959999799728394]
```

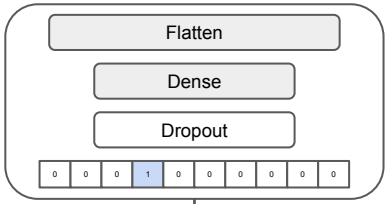
Training CIFAR-10 data



Colab에서 확인하기

Training CIFAR-10 data

Model



더 깊게?

Layer (type)	Output Shape	Param #
flatten_10 (Flatten)	(None, 3072)	0
dense_34 (Dense)	(None, 128)	393344
dropout_21 (Dropout)	(None, 128)	0
dense_35 (Dense)	(None, 128)	16512
dropout_22 (Dropout)	(None, 128)	0
dense_36 (Dense)	(None, 128)	16512
dropout_23 (Dropout)	(None, 128)	0
dense_37 (Dense)	(None, 10)	1290

Total params: 427,658
Trainable params: 427,658
Non-trainable params: 0

427,658

accuracy: 0.**3549**

더 넓게?

Layer (type)	Output Shape	Param #
flatten_11 (Flatten)	(None, 3072)	0
dense_38 (Dense)	(None, 256)	786688
dropout_24 (Dropout)	(None, 256)	0
dense_39 (Dense)	(None, 256)	65792
dropout_25 (Dropout)	(None, 256)	0
dense_40 (Dense)	(None, 256)	65792
dropout_26 (Dropout)	(None, 256)	0
dense_41 (Dense)	(None, 256)	65792
dropout_27 (Dropout)	(None, 256)	0
dense_42 (Dense)	(None, 10)	2570

Total params: 986,634
Trainable params: 986,634
Non-trainable params: 0

986,634

accuracy: 0.**3814**

더 더 넓게? & 더 오래?

Layer (type)	Output Shape	Param #
flatten_12 (Flatten)	(None, 3072)	0
dense_43 (Dense)	(None, 1024)	3146752
dropout_28 (Dropout)	(None, 1024)	0
dense_44 (Dense)	(None, 1024)	1049600
dropout_29 (Dropout)	(None, 1024)	0
dense_45 (Dense)	(None, 1024)	1049600
dropout_30 (Dropout)	(None, 1024)	0
dense_46 (Dense)	(None, 1024)	1049600
dropout_31 (Dropout)	(None, 1024)	0
dense_47 (Dense)	(None, 10)	10250

Total params: 6,305,802
Trainable params: 6,305,802
Non-trainable params: 0

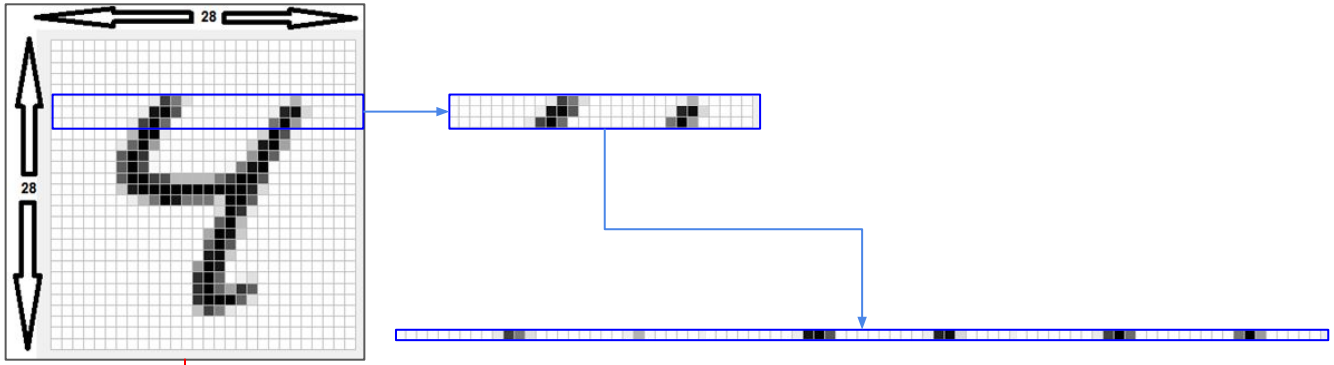
6,305,802

accuracy: 0.**4127** / accuracy: 0.**4251**

20 epoch 100 epoch

성능이 개선되지 않는 이유?

Fully connected layer (Dense layer)의 한계



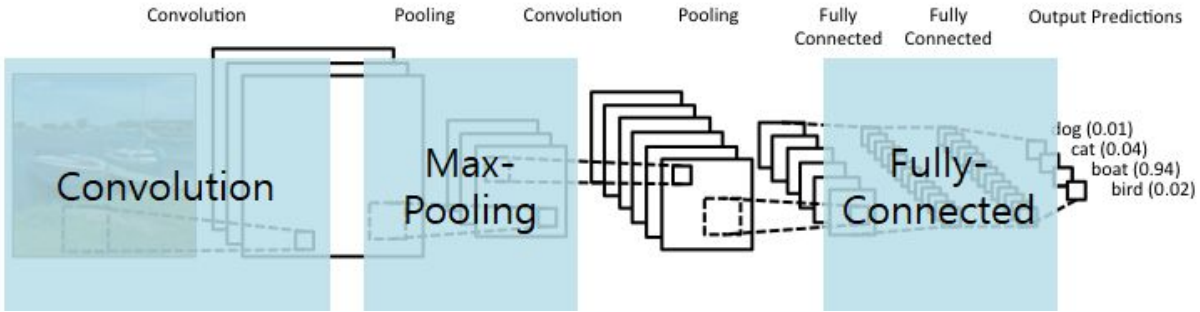

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])
```

Flatten data

데이터의 형상이 무시되는 문제점이 있음

Convolutional Neural Network (합성곱 신경망)

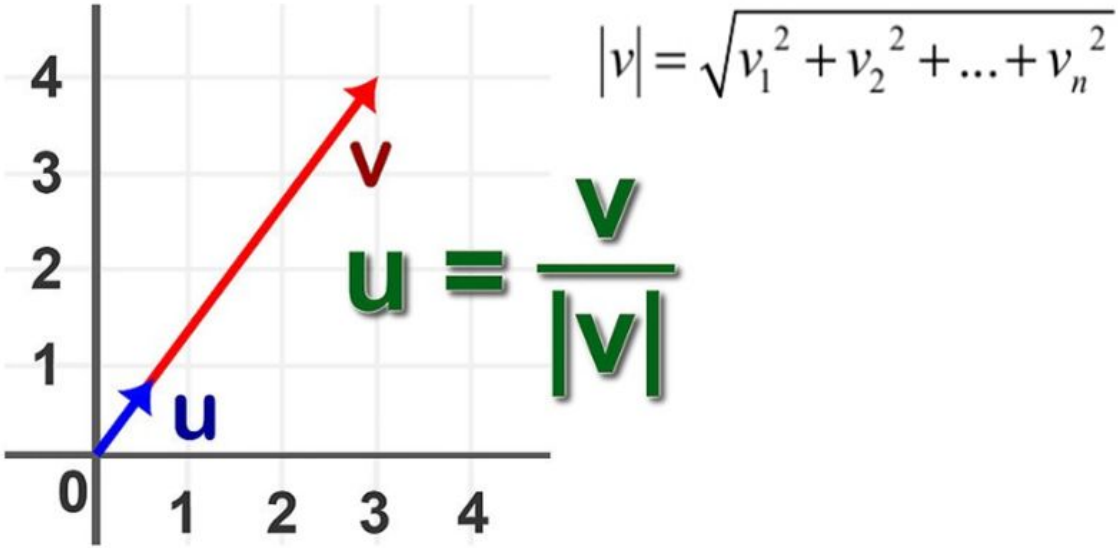
- **Layer의 구성**
  - Convolution Layer – Feature 추출
  - Max-Pooling Layer – Local Search
  - Fully-Connected Layer – 최종 분류



Convolution Layer

- 정규화 (Normalization)

- 벡터를 자신의 길이로 나누어 주면 길이가 1이고 방향만 남은 단위벡터가 됨
- 벡터의 길이는 벡터의 norm 이라고도 함

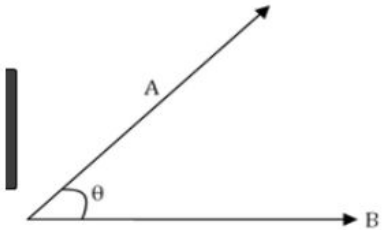


$$|v| = \sqrt{v_1^2 + v_2^2 + \ldots + v_n^2}$$

Convolution Layer

- 벡터의 내적 (Inner Product)

- 내적의 기하학적 의미는 A를 B로 사영시킨 선분의 길이와 B의 길이의 곱



$$\vec{A} \cdot \vec{B} = \overrightarrow{\|A\|}\,\overrightarrow{\|B\|}\cos\theta$$

$$\cos\theta = \frac{\vec{A} \cdot \vec{B}}{\overrightarrow{\|A\|}\,\overrightarrow{\|B\|}}$$

$$\theta = \cos^{-1}\left(\frac{\vec{A} \cdot \vec{B}}{\overrightarrow{\|A\|}\,\overrightarrow{\|B\|}}\right)$$

- 식으로 나타내면 다음과 같다

$$= x_1 y_1 + x_2 y_2 + \cdots + x_n y_n = \sum_{i=1}^{n} x_i y_i$$

$$= \mathbf{x}^T \mathbf{y} = \mathbf{y}^T \mathbf{x}$$

Convolution Layer

- **정규화 + 내적 = cos(θ)**

- 앞선 내적 공식에서 cos(θ)값은 다음과 같이 구할 수 있다

$$\cos\theta = \frac{\overline{A} \cdot \overline{B}}{\overline{\|A\|}\,\overline{\|B\|}}$$

- 정규화를 시키면 분모의 A,B벡터의 길이가 1이므로 A,B의 내적이 cos(θ) 값이 됨

- 두 벡터의 cos(θ)값은 두 벡터의 유사성을 비교하는 척도로 사용 가능 (높을수록 유사, 낮을수록 이질)

Convolution Layer

- **Concatenate**
- 2개의 열벡터 A,B가 있다면

A	B
1	2
2	4
3	6

- concat(A,B) by x 축

A	B
1	2
2	4
3	6

- concat(A,B) by y축

A,B
1
2
3
2
4
6

- concat(A,B) by z축

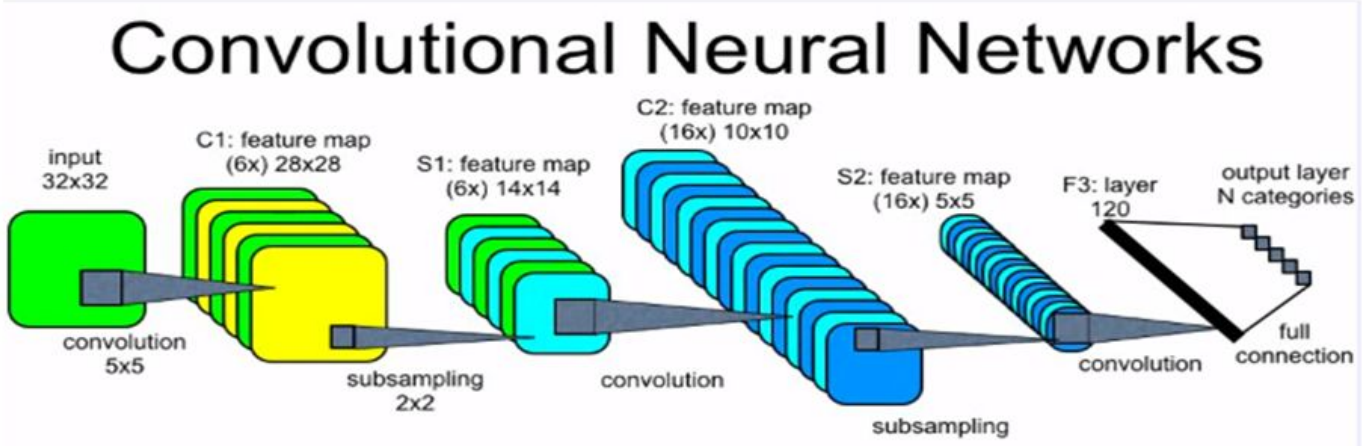
A
1
2
3

벡터가 아닌 행렬 또는 n차원 tensor에도 동일하게 적용

Convolutional Neural Network (합성곱 신경망)

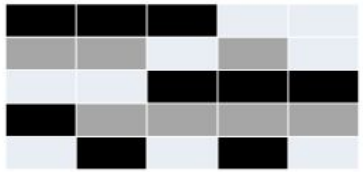
음성인식, 이미지 인식과 같은 2차원 데이터를 처리하기 좋은 딥 러닝 알고리즘

1. Feature 추출 (Convolution Stage, Cn)
2. Subsampling, Sn
3. 위 두 stage를 1차원 벡터 (그림에서는 F3)가 나올 때까지 반복
4. Classification



Convolution Layer

흰색을 0, 회색을 0.5, 검정을 1이라고 했을 때 아래 이미지에서 오른쪽과 같은 특징을 찾고 싶다고 한다면



원래 이미지

=

1	1	1
0.5	0.5	0.5
0	0	0

필터A

필터A

=

0	0	0	0	0	0	0
0	1	1	1	0	0.5	0
0	0.5	0.5	0	0.5	0	0
0	0	0	1	1	1	0
0	1	0.5	0.5	0.5	0.5	0
0	0	1	0	1	0	0
0	0	0	0	0	0	0

중앙 파란색은 원래 이미지,
주변의 옅은 색은 차원을 유지하고 경계를 구분하는 역할 (선택)

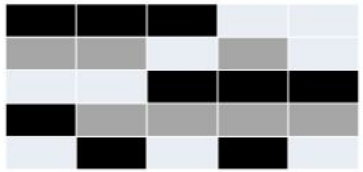
정규화 + Convolution!! (합성곱)

$$\begin{matrix}1 & 2\\ 3 & 4\end{matrix} * \begin{matrix}5 & 6\\ 7 & 8\end{matrix} = 1x5+2x6+3x7+4x8 = 70$$

Convolution은 지정된 범위에서의 벡터 내적이며 정규화가 되어 있으므로 코사인유사도와 의미가 정확히 같음

Convolution Layer

흰색을 0, 회색을 0.5, 검정을 1이라고 했을 때 아래 이미지에서 오른쪽과 같은 특징을 찾고 싶다고 한다면



원래 이미지

=

1	1	1
0.5	0.5	0.5
0	0	0

필터A

필터A

0	0	0	0	0	0	0
0	1	1	1	0	0.5	0
0	0.5	0.5	0	0.5	0	0
0	0	0	1	1	1	0
0	1	0.5	0.5	0.5	0.5	0
0	0	1	0	1	0	0
0	0	0	0	0	0	0

중앙 파란색은 원래 이미지,
주변의 옅은 색은 차원을 유지하고 경계를 구분하는 역할 (선택)

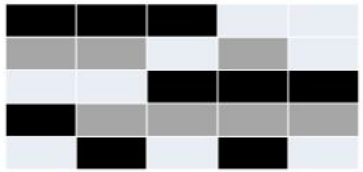
정규화 + Convolution!! (합성곱)

$$\begin{matrix}1 & 2\\ 3 & 4\end{matrix} * \begin{matrix}5 & 6\\ 7 & 8\end{matrix} = 1x5+2x6+3x7+4x8 = 70$$

Convolution은 지정된 범위에서의 벡터 내적이며 정규화가 되어 있으므로 코사인유사도와 의미가 정확히 같음

Convolution Layer

흰색을 0, 회색을 0.5, 검정을 1이라고 했을 때 아래 이미지에서 오른쪽과 같은 특징을 찾고 싶다고 한다면



원래 이미지

=

1	1	1
0.5	0.5	0.5
0	0	0

필터A

=

필터A

0	0	0	0	0	0	0
0	1	1	1	0	0.5	0
0	0.5	0.5	0	0.5	0	0
0	0	0	1	1	1	0
0	1	0.5	0.5	0.5	0.5	0
0	0	1	0	1	0	0
0	0	0	0	0	0	0

중앙 파란색은 원래 이미지,
주변의 옅은 색은 차원을 유지하고 경계를 구분하는 역할 (선택)

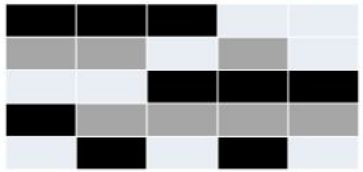
정규화 + Convolution!! (합성곱)

$$\begin{matrix}1 & 2\\ 3 & 4\end{matrix} * \begin{matrix}5 & 6\\ 7 & 8\end{matrix} = 1x5+2x6+3x7+4x8 = 70$$

Convolution은 지정된 범위에서의 벡터 내적이며 정규화가 되어 있으므로 코사인유사도와 의미가 정확히 같음

Convolution Layer

흰색을 0, 회색을 0.5, 검정을 1이라고 했을 때 아래 이미지에서 오른쪽과 같은 특징을 찾고 싶다고 한다면



원래 이미지

=

1	1	1
0.5	0.5	0.5
0	0	0

필터A

=

0	0	0	0	0	0	0
0	1	1	1	0	0.5	0
0	0.5	0.5	0	0.5	0	0
0	0	0	1	1	1	0
0	1	0.5	0.5	0.5	0.5	0
0	0	1	0	1	0	0
0	0	0	0	0	0	0

필터A

중앙 파란색은 원래 이미지,
주변의 옅은 색은 차원을 유지하고 경계를 구분하는 역할 (선택)

정규화 + Convolution!! (합성곱)

$$\begin{matrix}1 & 2\\ 3 & 4\end{matrix} * \begin{matrix}5 & 6\\ 7 & 8\end{matrix} = 1x5+2x6+3x7+4x8 = 70$$

Convolution은 지정된 범위에서의 벡터 내적이며 정규화가 되어 있으므로 코사인유사도와 의미가 정확히 같음

Convolution Layer

- Filter의 연산을 직접 해 봅시다!

0	0	0	0	0	0	0
0	1	1	1	0	0.5	0
0	0.5	0.5	0	0.5	0	0
0	0	0	1	1	1	0
0	1	0.5	0.5	0.5	0.5	0
0	0	1	0	1	0	0
0	0	0	0	0	0	0

*

1	1	1
0.5	0.5	0.5
0	0	0

필터A

- **1. 정규화를 거쳐**

0.471	0.471	0.471
0.236	0.236	0
0	0	0.471

*

0.516	0.516	0.516
0.258	0.258	0.258
0	0	0

=

2. Convolution을 하면

0.471*0.516+0.471*0.516+
...+0.471*0=**0.852**

- 수학적 의미를 엄밀하게 따지지 않는다면 해당 영역이 필터와 85.2% 유사하다고 생각하여도 무방

Convolution Layer

- 이 과정을 모든 영역에 적용하면?

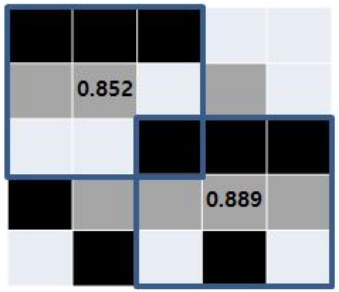
0	0	0	0	0	0	0
0	1	1	1	0	0.5	0
0	0.5	0.5	0	0.5	0	0
0	0	0	1	1	1	0
0	1	0.5	0.5	0.5	0.5	0
0	0	1	0	1	0	0
0	0	0	0	0	0	0

*

1	1	1
0.5	0.5	0.5
0	0	0

=

0.327	0.414	0.327	0.316	0.183
0.816	0.852	0.609	0.426	0.245
0.390	0.447	0.573	0.516	0.467
0.258	0.552	0.652	0.889	0.690
0.689	0.816	0.778	0.781	0.632



원래 이미지

필터A

원래 이미지에서 필터A와 가장 유사한 부분은 파란 테두리 부분!

이런 필터를 여러개 사용하여 Concatenate

Convolution Layer

- Convolution filter의 예시

- Sharpen

0	0	0	0	0
0	0	-1	0	0
0	-1	5	-1	0
0	0	-1	0	0
0	0	0	0	0

- Edge Detect

	0	1	0	
	1	-4	1	
	0	1	0	

- Blur

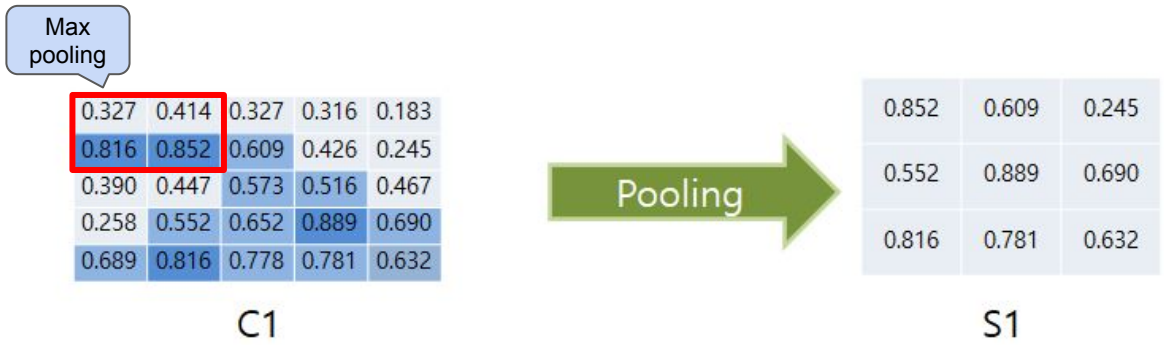
0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

- Emboss

	-2	-1	0	
	-1	1	1	
	0	1	2	

Pooling Layer

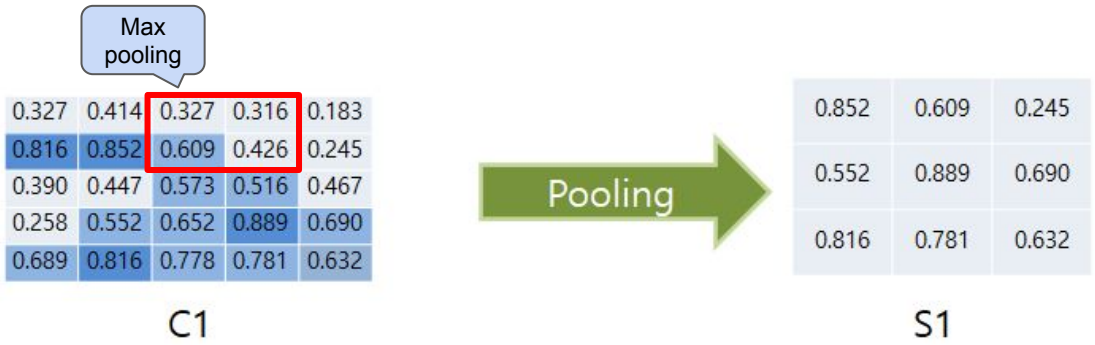
- Convolution을 실행한 C1에서 특징 (통상적으로 2x2 4칸 중 가장 큰 값을 고르는 Max Pooling을 사용)을 추려 냄



- S1을 새로운 Input처럼 생각하며 Convolution과 Subsampling을 반복

Pooling Layer

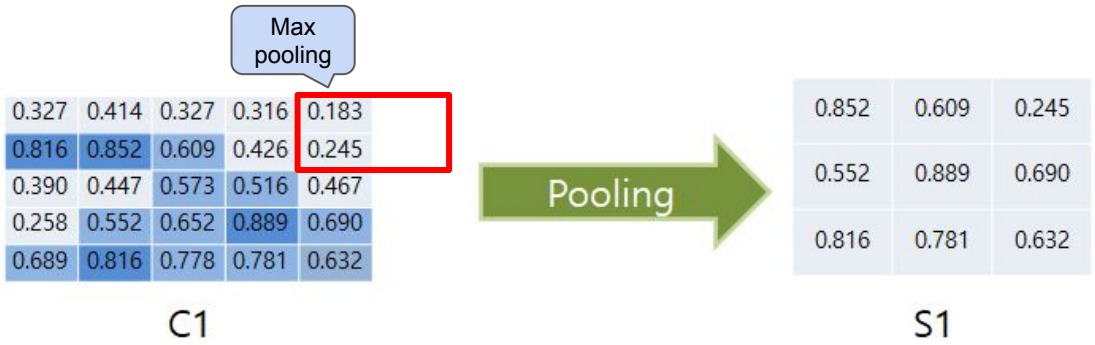
- Convolution을 실행한 C1에서 특징 (통상적으로 2x2 4칸 중 가장 큰 값을 고르는 Max Pooling을 사용)을 추려 냄



- S1을 새로운 Input처럼 생각하며 Convolution과 Subsampling을 반복

Pooling Layer

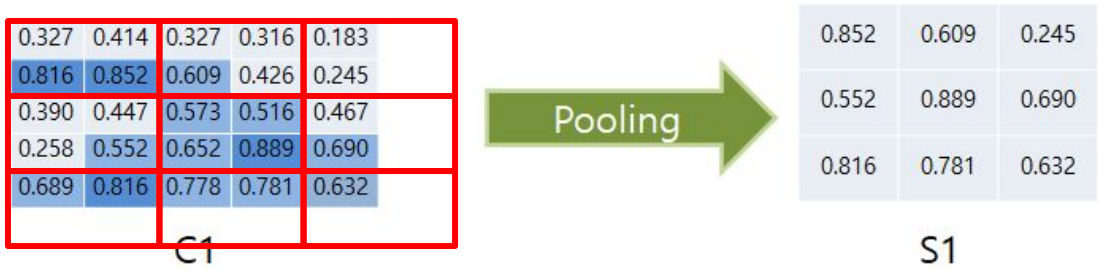
- Convolution을 실행한 C1에서 특징 (통상적으로 2x2 4칸 중 가장 큰 값을 고르는 Max Pooling을 사용)을 추려 냄



- S1을 새로운 Input처럼 생각하며 Convolution과 Subsampling을 반복

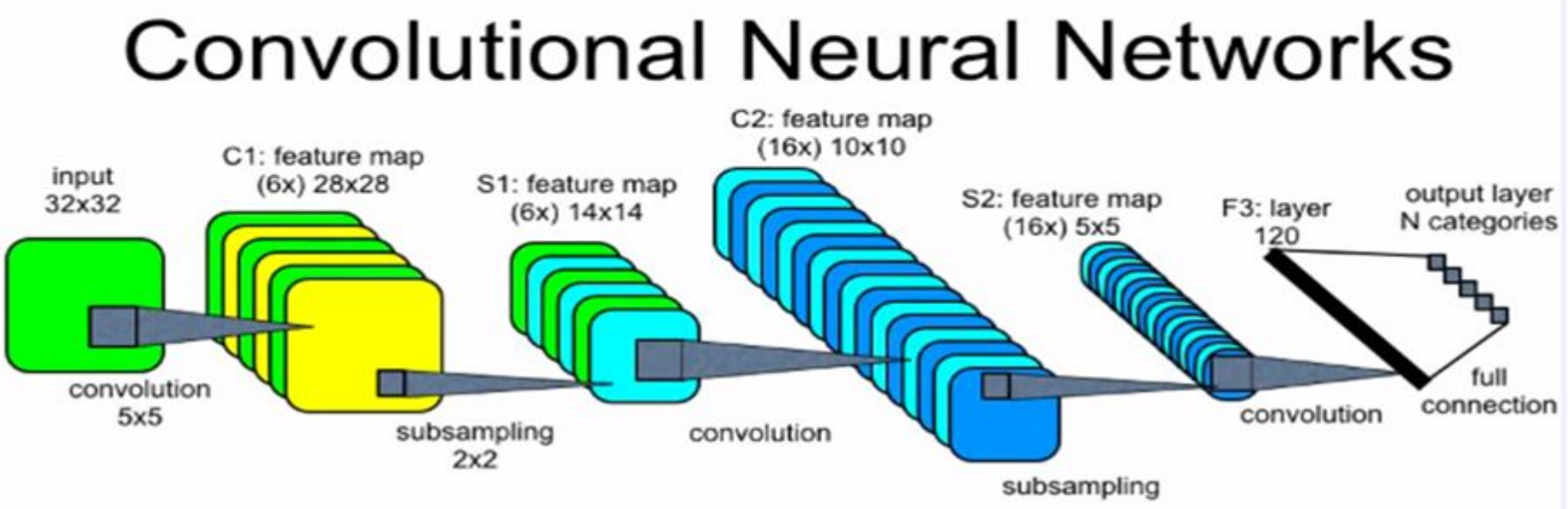
Pooling Layer

- Convolution을 실행한 C1에서 특징 (통상적으로 2x2 4칸 중 가장 큰 값을 고르는 Max Pooling을 사용)을 추려 냄



- S1을 새로운 Input처럼 생각하며 Convolution과 Subsampling을 반복

Fully Connected Layer



- MLP (Multi Layer Perceptron)에서 사용된 구조와 동일함
- 일반적으로 여러 필터를 통과한 특징들을 **flatten** 시켜
  하나의 벡터로 만든 후 최종 **output** 예측 직전단계에 활용함

Some Techniques

Data Augmentation

- Random cropping to 224 X 224 images
- Altering RGB/Grayscale
- Flipping
- ...

Some Techniques

- Input 데이터에 이미지 변형 과정을 거쳐 데이터 자체의 특징은 유지하면서 데이터의 양은 늘려 학습의 효율을 크게 증가시킬 수 있는 기법. 데이터의 수가 적을 때 효과가 좋음.

Rotation

Zoom

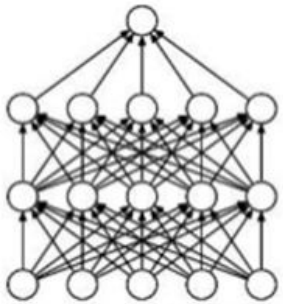
Flip (Horizontal, Vertical)

Shift (Width, Height)

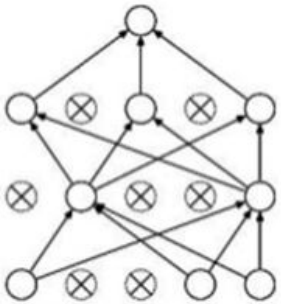
Some Techniques

Dropout – Avoiding overfitting

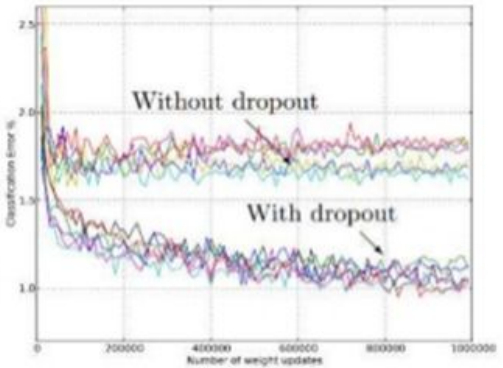
- Fully-connected layers Node의 0.0~0.5를 데이터에 따라 off함
- Mini-batch마다 사용하는 neuron의 종류가 달라짐
- 여러 개의 architecture를 ensemble하는 효과가 생김
- Better generalization



(a) Standard Neural Net



(b) After applying dropout.



Training CIFAR-10 data

합성곱 신경망 (Convolutional Neural Network)

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 4, 4, 64)          36928
_________________________________________________________________
flatten_8 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_30 (Dense)             (None, 64)                65600
_________________________________________________________________
dense_31 (Dense)             (None, 10)                650
=================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
_________________________________________________________________
```

```
Epoch 1/5
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4835 - accuracy: 0.4626
Epoch 2/5
1563/1563 [==============================] - 4s 3ms/step - loss: 1.1241 - accuracy: 0.6044
Epoch 3/5
1563/1563 [==============================] - 4s 3ms/step - loss: 0.9768 - accuracy: 0.6571
Epoch 4/5
1563/1563 [==============================] - 4s 3ms/step - loss: 0.8794 - accuracy: 0.6931
Epoch 5/5
1563/1563 [==============================] - 4s 3ms/step - loss: 0.8053 - accuracy: 0.7190
313/313 [==============================] - 1s 2ms/step - loss: 0.8871 - accuracy: 0.6957
[0.887057900428772, 0.6956999897956848]
```

훨씬 적은 파라미터 수로 좋은 결과를 얻었음

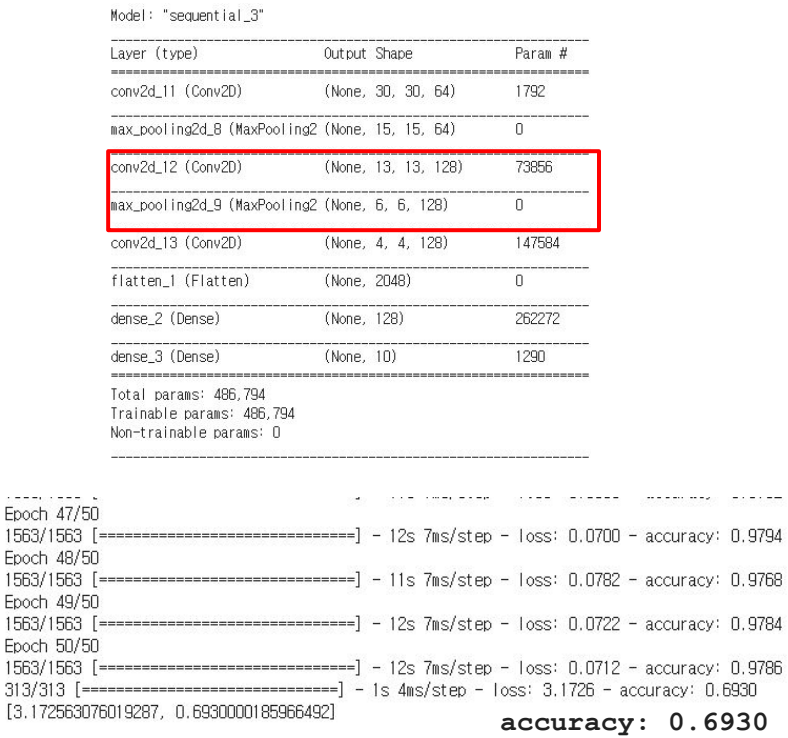
Training CIFAR-10 data

epoch 5 -> 50

```
Epoch 45/50
1563/1563 [==============================] - 4s 3ms/step - loss: 0.1037 - accuracy: 0.9643
Epoch 46/50
1563/1563 [==============================] - 5s 3ms/step - loss: 0.1198 - accuracy: 0.9587
Epoch 47/50
1563/1563 [==============================] - 4s 3ms/step - loss: 0.1102 - accuracy: 0.9617
Epoch 48/50
1563/1563 [==============================] - 4s 3ms/step - loss: 0.1116 - accuracy: 0.9626
Epoch 49/50
1563/1563 [==============================] - 4s 3ms/step - loss: 0.1026 - accuracy: 0.9651
Epoch 50/50
1563/1563 [==============================] - 5s 3ms/step - loss: 0.1105 - accuracy: 0.9628
313/313 [==============================] - 1s 2ms/step - loss: 2.5728 - accuracy: 0.6903
[2.5728492736816406, 0.6902999877929688]
```

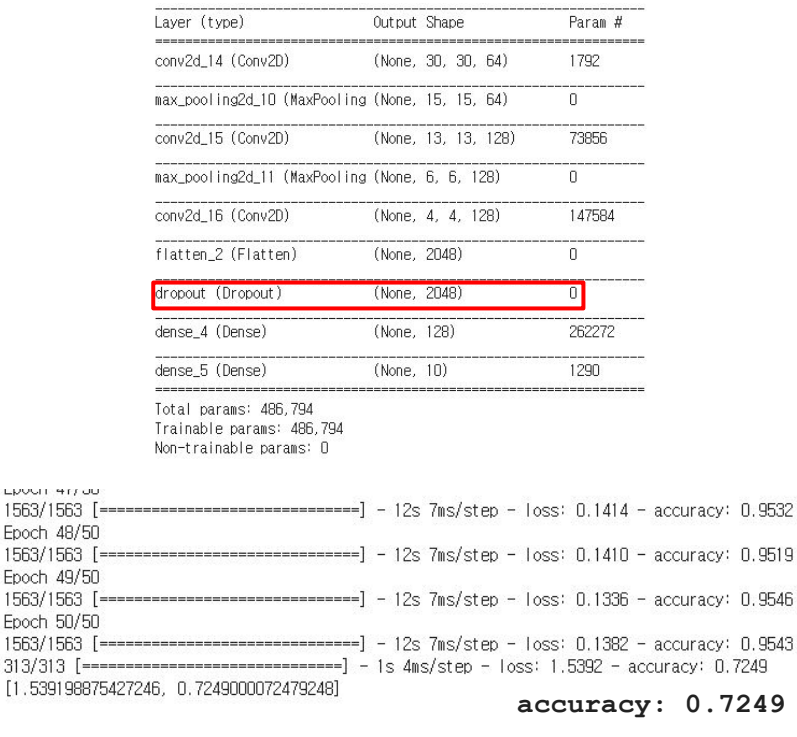
학습 epoch를 50으로 늘렸지만 test accuracy에서 개선이 없었음

Training CIFAR-10 data



```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_11 (Conv2D)           (None, 30, 30, 64)        1792
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 15, 15, 64)        0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 13, 13, 128)       73856
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 6, 6, 128)         0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 4, 4, 128)         147584
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               262272
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
=================================================================
Total params: 486,794
Trainable params: 486,794
Non-trainable params: 0
_________________________________________________________________
Epoch 47/50
1563/1563 [==============================] - 12s 7ms/step - loss: 0.0700 - accuracy: 0.9794
Epoch 48/50
1563/1563 [==============================] - 11s 7ms/step - loss: 0.0782 - accuracy: 0.9768
Epoch 49/50
1563/1563 [==============================] - 12s 7ms/step - loss: 0.0722 - accuracy: 0.9784
Epoch 50/50
1563/1563 [==============================] - 12s 7ms/step - loss: 0.0712 - accuracy: 0.9786
313/313 [==============================] - 1s 4ms/step - loss: 3.1726 - accuracy: 0.6930
[3.172563076019287, 0.6930000185966492]
accuracy: 0.6930
```

layer를 추가했으나 비슷한 성능



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_14 (Conv2D)           (None, 30, 30, 64)        1792
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 15, 15, 64)        0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 13, 13, 128)       73856
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 6, 6, 128)         0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 4, 4, 128)         147584
_________________________________________________________________
flatten_2 (Flatten)          (None, 2048)              0
_________________________________________________________________
dropout (Dropout)            (None, 2048)              0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               262272
_________________________________________________________________
dense_5 (Dense)              (None, 10)                1290
=================================================================
Total params: 486,794
Trainable params: 486,794
Non-trainable params: 0
_________________________________________________________________
1563/1563 [==============================] - 12s 7ms/step - loss: 0.1414 - accuracy: 0.9532
Epoch 48/50
1563/1563 [==============================] - 12s 7ms/step - loss: 0.1410 - accuracy: 0.9519
Epoch 49/50
1563/1563 [==============================] - 12s 7ms/step - loss: 0.1336 - accuracy: 0.9546
Epoch 50/50
1563/1563 [==============================] - 12s 7ms/step - loss: 0.1382 - accuracy: 0.9543
313/313 [==============================] - 1s 4ms/step - loss: 1.5392 - accuracy: 0.7249
[1.539198875427246, 0.7249000072479248]
accuracy: 0.7249
```

Dropout을 적용한 결과 약간의 개선

Fashion-MNIST data

레이블	클래스
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Fashion-MNIST data

데이터 불러오기 & 정규화

```
import tensorflow as tf
fashion_mnist = tf.keras.datasets.fashion_mnist
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

x_train = x_train.reshape((60000, 28, 28, 1))
x_test = x_test.reshape((10000, 28, 28, 1))

# 픽셀 값을 0~1 사이로 정규화합니다.
x_train, x_test = x_train / 255.0, x_test / 255.0
```

Fashion-MNIST data

모델 정의하기

CNN?

```
from tensorflow.keras import datasets, layers, models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

```

MLP?

```
from tensorflow import keras
# layer를 하나씩 추가하는 방식
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28,28]))
model.add(keras.layers.Dense(300, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(10, activation="softmax"))

# 층 리스트로 전달하는 방식
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28,28]),
    keras.layers.Dense(300, activation="relu"),
    keras.layers.Dense(100, activation="relu"),
    keras.layers.Dense(10, activation="softmax"),
])
```

Fashion-MNIST data

모델 컴파일 & 모델 학습 및 평가

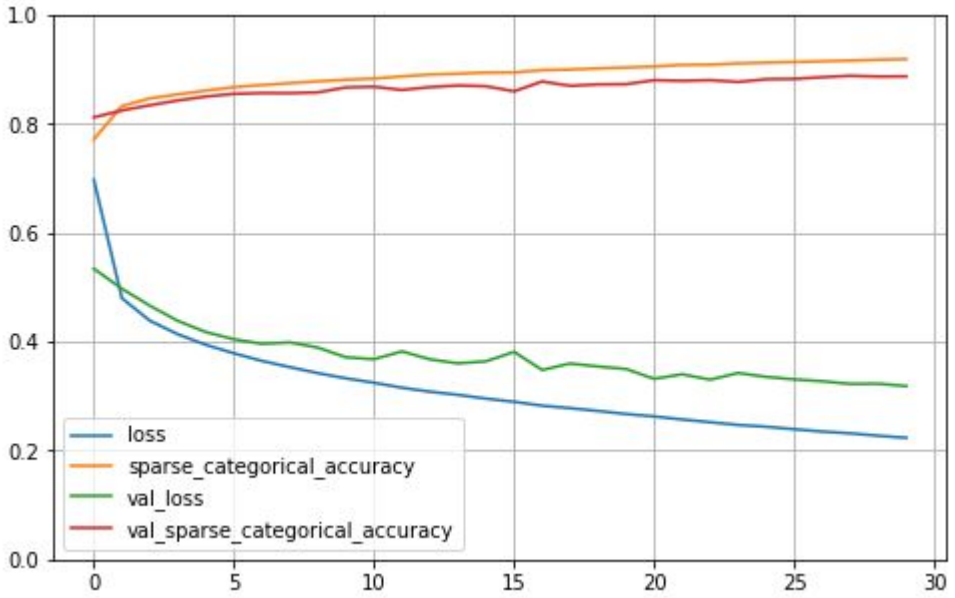
```
model.compile(loss="sparse_categorical_crossentropy", optimizer="sgd", metrics=["accuracy"])
history = model.fit(x_train, y_train, epochs=30, validation_data=(x_test, y_test))
```

```
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2521 - sparse_categorical_accuracy: 0.9088 - val_loss: 0.3303 - val_sparse_categorical_accuracy: 0.8804
Epoch 24/30
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2471 - sparse_categorical_accuracy: 0.9111 - val_loss: 0.3425 - val_sparse_categorical_accuracy: 0.8771
Epoch 25/30
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2438 - sparse_categorical_accuracy: 0.9126 - val_loss: 0.3354 - val_sparse_categorical_accuracy: 0.8821
Epoch 26/30
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2392 - sparse_categorical_accuracy: 0.9137 - val_loss: 0.3308 - val_sparse_categorical_accuracy: 0.8827
Epoch 27/30
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2350 - sparse_categorical_accuracy: 0.9150 - val_loss: 0.3274 - val_sparse_categorical_accuracy: 0.8858
Epoch 28/30
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2317 - sparse_categorical_accuracy: 0.9164 - val_loss: 0.3225 - val_sparse_categorical_accuracy: 0.8885
Epoch 29/30
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2273 - sparse_categorical_accuracy: 0.9179 - val_loss: 0.3228 - val_sparse_categorical_accuracy: 0.8870
Epoch 30/30
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2234 - sparse_categorical_accuracy: 0.9190 - val_loss: 0.3186 - val_sparse_categorical_accuracy: 0.8873
```

Fashion-MNIST data

학습 곡선 확인하기

```
import pandas as pd
import matplotlib.pyplot as plt

pd.DataFrame(history.history).plot(figsize=(8,5))
plt.grid(True)
plt.gca().set_ylim(0, 1)
plt.show()
```



Fashion-MNIST data

모델 평가

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

```
313/313 [==============================] - 1s 2ms/step - loss: 0.3247 - sparse_categorical_accuracy: 0.8860
[0.32465073466300964, 0.8859999775886536]
```

데이터 예측하기

```
1 X_new = x_test[:5]
2 y_proba = model.predict(X_new)
3 y_proba.round(2)
```

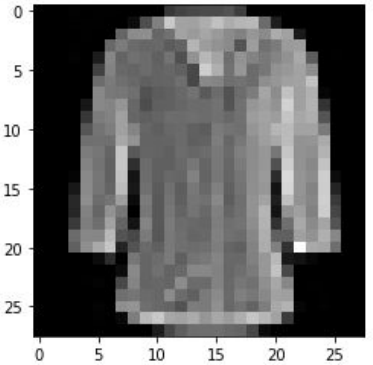
```
array([[0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.01, 0.  , 0.98],
       [0.  , 0.  , 0.98, 0.  , 0.02, 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.  , 1.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.  , 1.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.49, 0.  , 0.02, 0.  , 0.01, 0.  , 0.48, 0.  , 0.  , 0.  ]],
```

T-shirt/top

Shirt

```
1 plt.imshow(X_new[4].reshape(28,28), cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f31d03fea90>
```



Fashion-MNIST data

모델 저장 & 복원

```
1 model.save('fashion_mnist_mlp_model.h5')
```

```
1 model_loaded = keras.models.load_model('fashion_mnist_mlp_model.h5')
```

```
1 y_proba = model_loaded.predict(X_new)
2 y_proba.round(2)
```

```
array([[0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.01, 0.  , 0.98],
       [0.  , 0.  , 0.98, 0.  , 0.02, 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.  , 1.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.  , 1.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.49, 0.  , 0.02, 0.  , 0.01, 0.  , 0.48, 0.  , 0.  , 0.  ]],
```

복원된 모델의 예측결과가
저장한 모델과 같은 것을 확인함