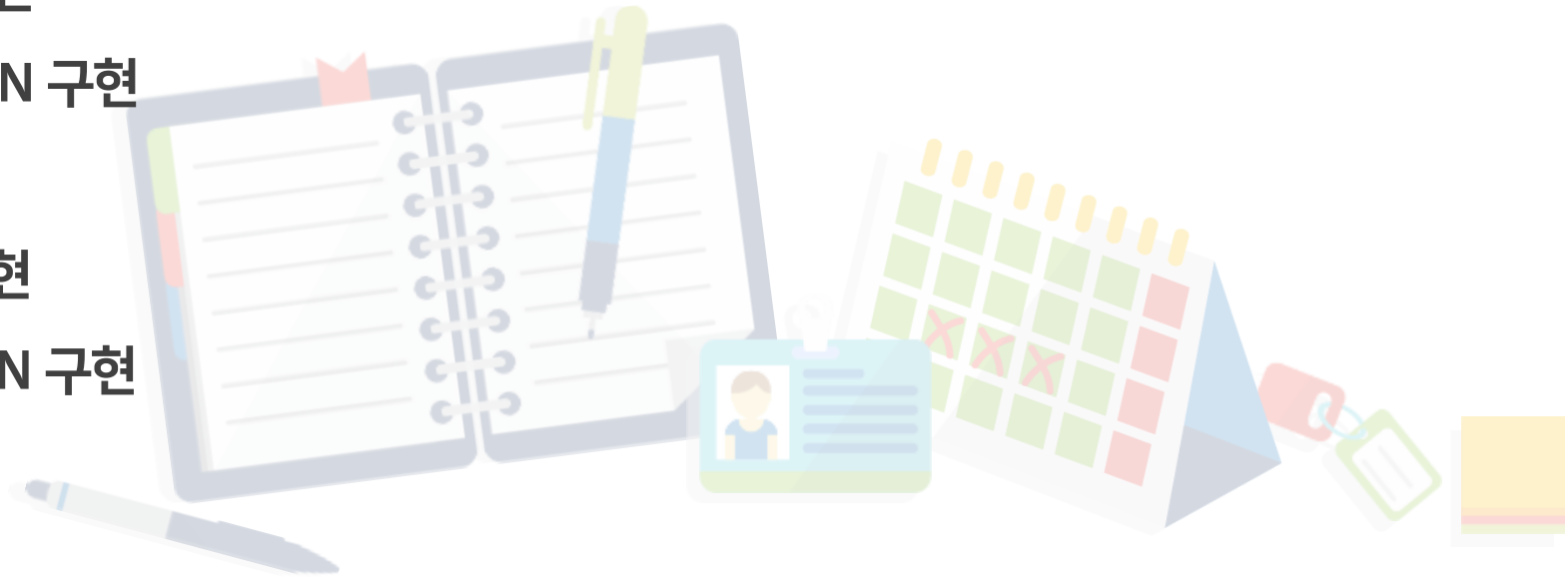
케라스로 구현하는 딥러닝

ANN / DNN / CNN

강사 윤영주
yj.youn1103@gmail.com

INDEX

1. KERAS 설치
2. ANN
    1) 필기체를 구분하는 ANN 구현
    2) 결과 데이터를 예측하는 회귀 ANN 구현
3. DNN
    1) 필기체를 구분하는 DNN 구현
    2) 컬러 이미지를 분류하는 DNN 구현
4. CNN
    1) 필기체를 구분하는 CNN 구현
    2) 컬러 이미지를 분류하는 CNN 구현



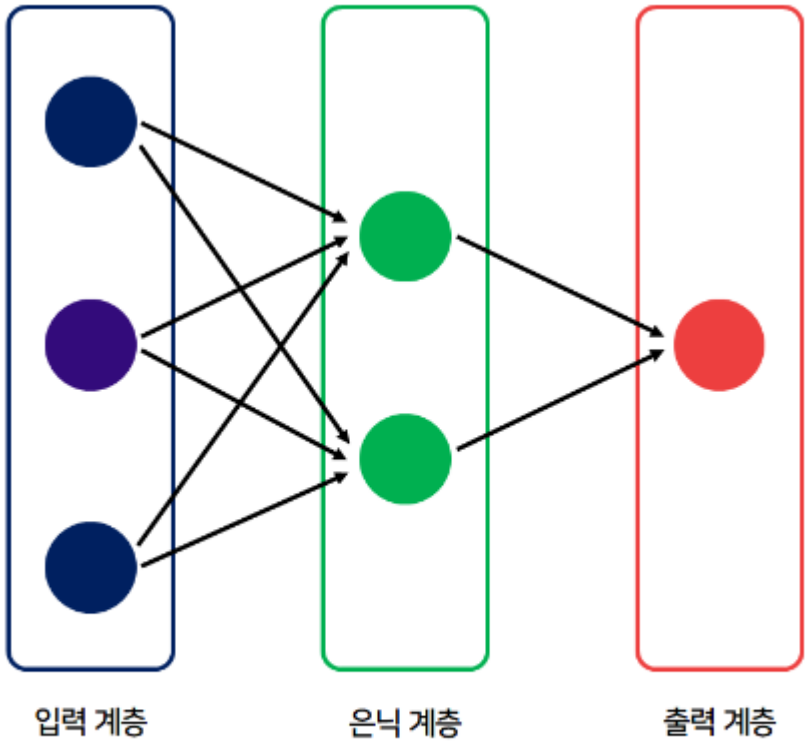
1. KERAS 설치

01. KERAS 설치

- 텐서플로우 설치
- 케라스 설치
  - ✓ Pip install keras
  - ✓ Conda install keras

2. ANN

02. ANN Artificial Neural Network



ANN의 구조

02. ANN Artificial Neural Network

- 인공신경망(ANN)은 생체의 신경망을 흉내 낸 인공지능
- 모든 비선형 함수를 학습 가능
- 입력 / 은닉 / 출력 계층으로 구성. 각 계층은 여러 노드로 구성
- 은닉 계층은 한 개 이상 포함 가능
- ANN의 활용
  1. 분류 : 입력 정보를 클래스별로 분류하는 방식
  2. 회귀 : 입력 정보로 다른 값을 예측

02. ANN Artificial Neural Network

- 분류 ANN
  - 입력 정보를 바탕으로 해당 입력이 어느 클래스에 속하는지 결정
  - 입력 계층은 필기체 숫자 그림을 받아들이고, 출력 계층은 분류한 결과를 출력
  - 분류할 클래스 수만큼 출력 노드를 만드는 방법이 효과적
  - 판별은 분류 클래스 노드 값을 비교하여 가장 큰 쪽을 선택하도록 구현

02. ANN Artificial Neural Network

- 회귀 ANN
  - 입력값으로부터 추력값을 직접 예측하는 방법
  - 실제 데이터의 규칙을 잘 표현하는 함수를 찾는 것이 목표
  - 직전으로 된 회귀함수 외에 곡선, 다차원 회귀함수를 찾기도 가능
  - 학습에 사용하지 않은 정보로도 예측이 가능

    집값을 예측한다 하면 수많은 집 정보를 이용해 학습한 후 임의의 집 정보로 시세 예측 가능

02. ANN Artificial Neural Network

- ANN 구현 방법 및 단계

1. 인공지능 구현용 패키지 가져오기
2. 인공지능에 필요한 매개변수 설정
3. 인공지능 모델 구현
4. 학습과 성능 평가용 데이터 가져오기
5. 인공지능 학습 및 성능 평가
6. 인공지능 학습 결과 분석

02. 1) 필기체를 구분하는 분류 ANN 구현

- 분류 ANN은 클래스가 둘 이상인 데이터를 분류하는 인공지능 방법
- 분류 ANN을 위한 인공지능 모델 구현
- 구현 방법 및 단계

  ① 필요 모듈 불러오기

```
from keras import layers, models
import tensorflow as tf
```

02. 1) 필기체를 구분하는 분류 ANN 구현

- 구현 방법 및 단계

② 분산 방식 모델링을 포함하는 ANN 모델

```
def ANN_models_func(Nin, Nh, Nout):
    x = layers.Input(shape=(Nin,))
    h = layers.Activation('relu')(layers.Dense(Nh)(x))
    y = layers.Activation('softmax')(layers.Dense(Nout)(h))
    model = models.Model(x, y)
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

02. 1) 필기체를 구분하는 분류 ANN 구현

- 구현 방법 및 단계

② 분산 방식 모델링을 포함하는 ANN 모델

```
def ANN_models_func(Nin, Nh, Nout):
    x = layers.Input(shape=(Nin,))
    h = layers.Activation('relu')(layers.Dense(Nh)(x))
    y = layers.Activation('softmax')(layers.Dense(Nout)(h))
    model = models.Model(x, y)
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

✓ 순서대로 입력 계층의 노드 수, 은닉 계층의 노드 수, 출력값이 가질 클래스 수, 출력 노드 수

02. 1) 필기체를 구분하는 분류 ANN 구현

- 구현 방법 및 단계

② 분산 방식 모델링을 포함하는 ANN 모델

```
def ANN_models_func(Nin, Nh, Nout):
    x = layers.Input(shape=(Nin,))
    h = layers.Activation('relu')(layers.Dense(Nh)(x))
    y = layers.Activation('softmax')(layers.Dense(Nout)(h))
    model = models.Model(x, y)
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

✓ 입력 계층은 지정 노드 수 지정

02. 1) 필기체를 구분하는 분류 ANN 구현

- 구현 방법 및 단계

② 분산 방식 모델링을 포함하는 ANN 모델

```
def ANN_models_func(Nin, Nh, Nout):
    x = layers.Input(shape=(Nin,))
    h = layers.Activation('relu')(layers.Dense(Nh)(x))
    y = layers.Activation('softmax')(layers.Dense(Nout)(h))
    model = models.Model(x, y)
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

✓ 은닉 계층 지정

✓ 은닉 계층의 입력은 입력 노드이므로 입력 노드 지정

✓ 활성 함수 'relu'지정 / f(x) = max(x, 0)

02. 1) 필기체를 구분하는 분류 ANN 구현

- 구현 방법 및 단계

② 분산 방식 모델링을 포함하는 ANN 모델

```
def ANN_models_func(Nin, Nh, Nout):
    x = layers.Input(shape=(Nin,))
    h = layers.Activation('relu')(layers.Dense(Nh)(x))
    y = layers.Activation('softmax')(layers.Dense(Nout)(h))
    model = models.Model(x, y)
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

✓ 출력 계층의 입력은 은닉 계층이므로 은닉 노드 지정

✓ 분류의 경우 출력 노드 활성화 함수로 소프트맥스 연산 사용

02. 1) 필기체를 구분하는 분류 ANN 구현

- 구현 방법 및 단계

② 분산 방식 모델링을 포함하는 ANN 모델

```
def ANN_models_func(Nin, Nh, Nout):
    x = layers.Input(shape=(Nin,))
    h = layers.Activation('relu')(layers.Dense(Nh)(x))
    y = layers.Activation('softmax')(layers.Dense(Nout)(h))
    model = models.Model(x, y)
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

✓ 케라스는 컴파일을 수행해 타깃 플랫폼에 맞게 딥러닝 코드를 구성

✓ Loss는 손실함수 지정, optimizer은 최적화 함수 지정,
metrics는 성능 검증을 위해 정확도 측정

02. 1) 필기체를 구분하는 분류 ANN 구현

- 구현 방법 및 단계

② 분산 방식 모델링을 포함하는 ANN 모델

```
def ANN_models_func(Nin, Nh, Nout):
    x = layers.Input(shape=(Nin,))
    h = layers.Activation('relu')(layers.Dense(Nh)(x))
    y = layers.Activation('softmax')(layers.Dense(Nout)(h))
    model = models.Model(x, y)
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

✓ 모델은 입력과 출력을 지정

02. 1) 필기체를 구분하는 분류 ANN 구현

- 구현 방법 및 단계

③ 연쇄 방식 모델링을 포함하는 ANN 모델

```
def ANN_seq_func(Nin, Nh, Nout):
    model = models.Sequential()
    model.add(layers.Dense(Nh, activation='relu', input_shape=(Nin,)))
    model.add(layers.Dense(Nout, activation='softmax'))
    model.compile(loss='categorical_crossentropy',
                  optimizer='adam', metrics=['accuracy'])
    return model
```

✓ 함수형 프로그래밍 방식 활용

✓ 분산 방식과 모델을 지정하는 부분만 다르고 상수 정의 및 설정 모델 컴파일 코드는 같음

✓ 분산 방식과 다르게 모델 먼저 설정

02. 1) 필기체를 구분하는 분류 ANN 구현

- 구현 방법 및 단계

③ 연쇄 방식 모델링을 포함하는 ANN 모델

```
def ANN_seq_func(Nin, Nh, Nout):
    model = models.Sequential()
    model.add(layers.Dense(Nh, activation='relu', input_shape=(Nin,)))
    model.add(layers.Dense(Nout, activation='softmax'))
    model.compile(loss='categorical_crossentropy',
                  optimizer='adam', metrics=['accuracy'])
    return model
```

✓ 모델 구조 설정

✓ 첫번째 add 단계에서 입력과 은닉 계층 형태가 동시에 정해짐

✓ 연쇄 방식은 추가되는 계층을 간편하게 기술 가능

02. 1) 필기체를 구분하는 분류 ANN 구현

- 구현 방법 및 단계

④ 분산 방식 모델링을 포함하는 객체지향형 ANN 모델

```
class ANN_models_class(models.Model):
    def __init__(self, Nin, Nh, Nout):
        # Prepare network layers and activate functions
        hidden = layers.Dense(Nh)
        output = layers.Dense(Nout)
        relu = layers.Activation('relu')
        softmax = layers.Activation('softmax')

        # Connect network elements
        x = layers.Input(shape=(Nin,))
        h = relu(hidden(x))
        y = softmax(output(h))

        super().__init__(x, y)
        self.compile(loss='categorical_crossentropy',
                     optimizer='adam', metrics=['accuracy'])
```

- ✓ 코드의 재사용성을 높이기 위해 객체지향 방식으로 구현 가능
- ✓ 전문가가 만든 인공지능 모델을 객체로 불러 쉽게 활용할 수 있음
- ✓ 클래스를 만들고 models.Model로 특성 상속
- ✓ models.Model은 신경망에서 사용하는 학습, 예측, 평가와 같은 함수 제공
- ✓ 클래스 초기화

02. 1) 필기체를 구분하는 분류 ANN 구현

- 구현 방법 및 단계

④ 분산 방식 모델링을 포함하는 객체지향형 ANN 모델

```
class ANN_models_class(models.Model):
    def __init__(self, Nin, Nh, Nout):
        # Prepare network layers and activate functions
        hidden = layers.Dense(Nh)
        output = layers.Dense(Nout)
        relu = layers.Activation('relu')
        softmax = layers.Activation('softmax')

        # Connect network elements
        x = layers.Input(shape=(Nin,))
        h = relu(hidden(x))
        y = softmax(output(h))

        super().__init__(x, y)
        self.compile(loss='categorical_crossentropy',
                     optimizer='adam', metrics=['accuracy'])
```

✓ 은닉 계층과 출력 계층 정의

✓ 비선형성 Activation함수 정의

✓ 상속받은 상위 클래스 초기화 진행

02. 1) 필기체를 구분하는 분류 ANN 구현

- 구현 방법 및 단계

⑤ 연쇄 방식 모델링을 포함하는 객체지향형 ANN 모델

```
class ANN_seq_class(models.Sequential):
    def __init__(self, Nin, Nh, Nout):
        super().__init__()
        self.add(layers.Dense(Nh, activation='relu', input_shape=(Nin,)))
        self.add(layers.Dense(Nout, activation='softmax'))
        self.compile(loss='categorical_crossentropy',
                     optimizer='adam', metrics=['accuracy'])
```

✓ 연쇄 방식은 객체지향 또는 클래스 없이 기본 형태로 구성 가능

✓ 앞의 계층에 새로운 계층을 계속 추가하는 형태

✓ 입력 계층을 별도로 정의하지 않고 은닉 계층부터 주가

02. 1) 필기체를 구분하는 분류 ANN 구현

- 구현 방법 및 단계

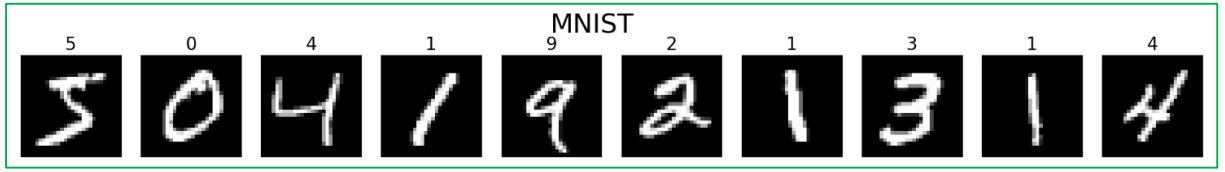
⑥ 분류 ANN에 사용할 데이터 가져오기

✓ MNIST는 6만 건의 필기체 숫자를 모은 공개 데이터

```
import numpy as np
from keras import datasets  # mnist
from keras.utils import np_utils  # to_categorical
```

✓ 라이브러리 가져오기

02. 1) 필기체를 구분하는 분류 ANN 구현



- 구현 방법 및 단계

⑥ 분류 ANN에 사용할 데이터 가져오기

```
def Data_func():
    (X_train, y_train), (X_test, y_test) = tf.keras.datasets.mnist.load_data()

    Y_train = np_utils.to_categorical(y_train)
    Y_test = np_utils.to_categorical(y_test)

    L, H, W = X_train.shape
    X_train = X_train.reshape(-1, W * H)
    X_test = X_test.reshape(-1, W * H)

    X_train = X_train / 255.0
    X_test = X_test / 255.0

    return (X_train, Y_train), (X_test, Y_test)
```

✓ 변수 저장

✓ 0부터 9까지 숫자로 구성된 출력값은 0과 1로 표현되는 이진법 벡터로 변환

02. 1) 필기체를 구분하는 분류 ANN 구현

- 구현 방법 및 단계

⑥ 분류 ANN에 사용할 데이터 가져오기

```
def Data_func():
    (X_train, y_train), (X_test, y_test) = tf.keras.datasets.mnist.load_data()

    Y_train = np_utils.to_categorical(y_train)
    Y_test = np_utils.to_categorical(y_test)

    L, H, W = X_train.shape
    X_train = X_train.reshape(-1, W * H)
    X_test = X_test.reshape(-1, W * H)

    X_train = X_train / 255.0
    X_test = X_test / 255.0

    return (X_train, Y_train), (X_test, Y_test)
```

✓ ANN 학습을 위해 벡터 이미지 형채로 바꿈

✓ -1은 행렬의 행을 자동 설정

✓ 0~225사이의 정수로 구성된 입력값을 255로 나눔

02. 1) 필기체를 구분하는 분류 ANN 구현

- 구현 방법 및 단계

⑦ 분류 ANN 학습 결과의 그래프 구현

```
import matplotlib.pyplot as plt
```

✓ 패키지 불러오기

✓ 손실 함수 및 정확도 함수 설정

```
# from keraspp.skeras import plot_loss
def plot_loss(history, title=None):
    if not isinstance(history, dict):
        history = history.history
    plt.plot(history['loss'])
    plt.plot(history['val_loss'])
    if title is not None:
        plt.title(title)
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend(['Training', 'Validation'], loc=0)
```

```
# from keraspp.skeras import plot_acc
def plot_acc(history, title=None):
    if not isinstance(history, dict):
        history = history.history
    plt.plot(history['accuracy'])
    plt.plot(history['val_accuracy'])
    if title is not None:
        plt.title(title)
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend(['Training', 'Validation'], loc=0)
```

02. 1) 필기체를 구분하는 분류 ANN 구현

- 구현 방법 및 단계

⑧ 분류 ANN 학습 및 성능 분석

✓ 파라미터 정의

```
Nin = 784
Nh = 100
number_of_class = 10
Nout = number_of_class
```

✓ 생성해 두었던 모델 사용. 데이터 불러오기

```
# model = ANN_models_func(Nin, Nh, Nout)
# model = ANN_seq_func(Nin, Nh, Nout)
# model = ANN_models_class(Nin, Nh, Nout)
model = ANN_seq_class(Nin, Nh, Nout)
(X_train, Y_train), (X_test, Y_test) = Data_func()
```

02. 1) 필기체를 구분하는 분류 ANN 구현

- 구현 방법 및 단계

⑧ 분류 ANN 학습 및 성능 분석

✓ Fit함수를 이용해 모델 학습 진행

```
history = model.fit(X_train, Y_train, epochs=5, batch_size=100, validation_split=0.2)
```

✓ 5회 학습 / 100개씩 나눠서 넣기 / 20% 성능 검증 활용

```
Epoch 1/5
480/480 [==============================] - 2s 3ms/step - loss: 0.3895 - accuracy: 0.8934 - val_loss: 0.2178 - val_accuracy: 0.9408
Epoch 2/5
480/480 [==============================] - 1s 3ms/step - loss: 0.1871 - accuracy: 0.9469 - val_loss: 0.1647 - val_accuracy: 0.9545
Epoch 3/5
480/480 [==============================] - 1s 2ms/step - loss: 0.1398 - accuracy: 0.9597 - val_loss: 0.1380 - val_accuracy: 0.9617
Epoch 4/5
480/480 [==============================] - 1s 3ms/step - loss: 0.1115 - accuracy: 0.9679 - val_loss: 0.1264 - val_accuracy: 0.9633
Epoch 5/5
480/480 [==============================] - 1s 2ms/step - loss: 0.0926 - accuracy: 0.9731 - val_loss: 0.1132 - val_accuracy: 0.9674
```

02. 1) 필기체를 구분하는 분류 ANN 구현

- 구현 방법 및 단계

⑧ 분류 ANN 학습 및 성능 분석

✓ 학습에 사용되지 않은 test데이터로 성능 최종 평가

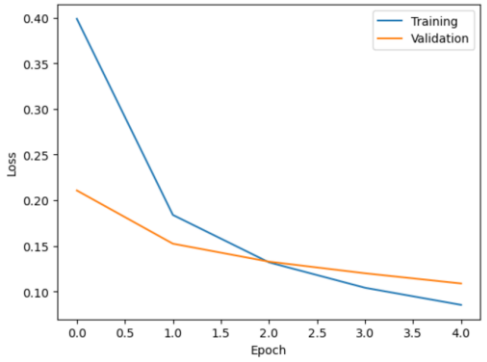
```
performace_test = model.evaluate(X_test, Y_test, batch_size=100)
print('Test Loss and Accuracy ->', performace_test)
```

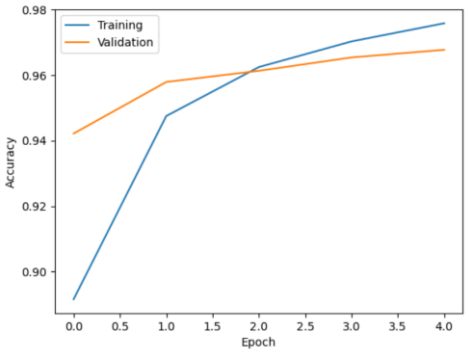
```
100/100 [==============================] - 0s 2ms/step - loss: 0.1091 - accuracy: 0.9676
Test Loss and Accuracy -> [0.109053835272789, 0.9675999879837036]
```

✓ 손실과 정확도 그래프 확인

```
import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'
```

```
plot_loss(history)
plt.show()
plot_acc(history)
plt.show()
```





02. 2) 결과 데이터를 예측하는 회귀 ANN 구현

- 보스턴 집값 예측하는 회귀 ANN 구현
- 구현 방법 및 단계

① 회귀 ANN 모델 구현

```
from keras import layers, models
import tensorflow as tf
```

✓ 케라스 서브패키지인 layers와 models는 각각 계층을 구성하는 툴과 계층을 합쳐 하나의 모델로 만들어 줌

02. 2) 결과 데이터를 예측하는 회귀 ANN 구현

- 구현 방법 및 단계

① 회귀 ANN 모델 구현

```
class ANN(models.Model):
    def __init__(self, Nin, Nh, Nout):
        # Prepare network layers and activate functions
        hidden = layers.Dense(Nh)
        output = layers.Dense(Nout)
        relu = layers.Activation('relu')

        # Connect network elements
        x = layers.Input(shape=(Nin,))
        h = relu(hidden(x))
        y = output(h)

        super().__init__(x, y)
        self.compile(loss='mse', optimizer='sgd')
```

✓ 신경망 계층 정의

✓ ANN 각 계층의 신호 연결 상황 정의

✓ 입력계층은 Nin 길이를 가지는 1차원 열 벡터

✓ 손실함수 mse는 평균자승오류 mean squared error

✓ 최적화는 sgd 사용

02. 2) 결과 데이터를 예측하는 회귀 ANN 구현

- 구현 방법 및 단계

② 학습 평가용 데이터 가져오기

```
from keras import datasets
from sklearn import preprocessing

def Data_func():
    (X_train, y_train), (X_test, y_test) = tf.keras.datasets.boston_housing.load_data()
    scaler = preprocessing.MinMaxScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)
    return (X_train, y_train), (X_test, y_test)
```

✓ 패키지 불러오기

✓ 데이터 불러오기

✓ 최소값과 최대값을 0과 1로 정규화

✓ X_train으로 학습과 변환을 한 뒤 X_test 변환데이터 불러오기

02. 2) 결과 데이터를 예측하는 회귀 ANN 구현

- 구현 방법 및 단계

③ 회귀 ANN 학습 결과 그래프

```
from keraspp.skeras import plot_loss
import matplotlib.pyplot as plt
```

✓ 앞서 02. 1)의 그래프 그대로 사용

✓ 해당 함수를 keraspp폴더 아래 keras.py 파일에 넣어두고 필요시 사용

02. 2) 결과 데이터를 예측하는 회귀 ANN 구현

- 구현 방법 및 단계

④ 회귀 ANN 학습 및 성능 분석

```
Nin = 13
Nh = 5
Nout = 1
```

✓ 출력층은 회귀를 통해 결과값을 직접 예측하기 때문에 1로 지정

```
model = ANN(Nin, Nh, Nout)
(X_train, y_train), (X_test, y_test) = Data_func()
history = model.fit(X_train, y_train, epochs=100,
                    batch_size=100, validation_split=0.2, verbose=2)
```

✓ ANN 구현과 유사

```
performace_test = model.evaluate(X_test, y_test, batch_size=100)
print('₩nTest Loss -> {:.2f}'.format(performace_test))
```

✓ 성능 평가

```
plot_loss(history)
plt.show()
```

02. 2) 결과 데이터를 예측하는 회귀 ANN 구현

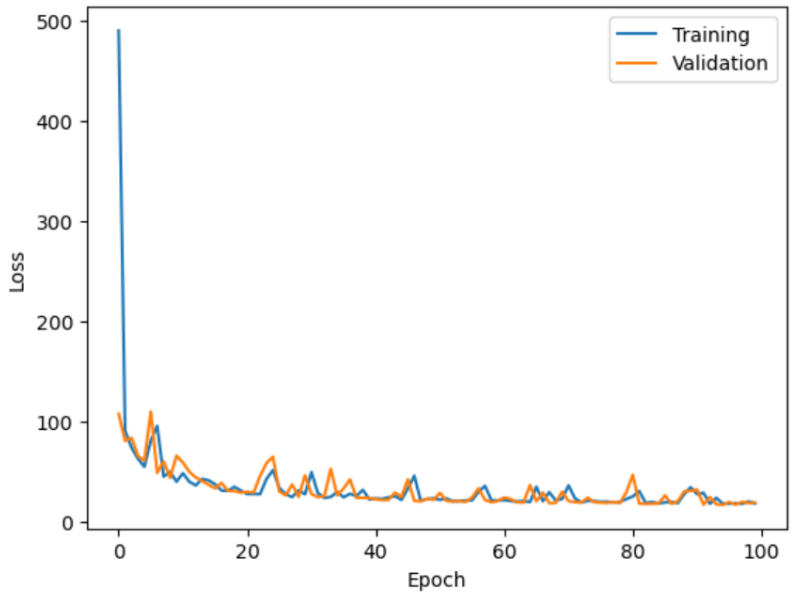
- 구현 방법 및 단계

④ 회귀 ANN 학습 및 성능 분석

```
Epoch 1/100
4/4 - 1s - loss: 489.8331 - val_loss: 107.5058 - 664ms/epoch - 166ms/step
Epoch 2/100
4/4 - 0s - loss: 90.5829 - val_loss: 80.5713 - 42ms/epoch - 10ms/step
Epoch 3/100
4/4 - 0s - loss: 73.8577 - val_loss: 83.5623 - 40ms/epoch - 10ms/step
Epoch 4/100
4/4 - 0s - loss: 62.7855 - val_loss: 65.5345 - 42ms/epoch - 10ms/step
Epoch 5/100
4/4 - 0s - loss: 54.8131 - val_loss: 60.7337 - 42ms/epoch - 10ms/step
```

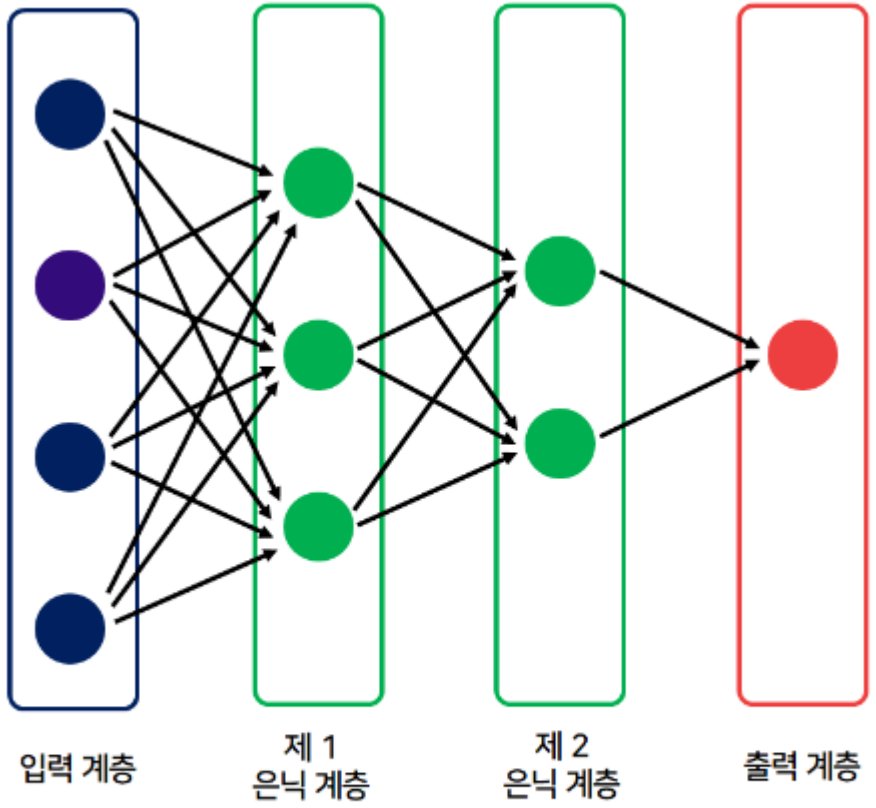
```
Epoch 98/100
4/4 - 0s - loss: 17.7832 - val_loss: 19.7744 - 37ms/epoch - 9ms/step
Epoch 99/100
4/4 - 0s - loss: 20.1491 - val_loss: 18.6146 - 38ms/epoch - 9ms/step
Epoch 100/100
4/4 - 0s - loss: 18.1204 - val_loss: 19.0067 - 39ms/epoch - 10ms/step
2/2 [==============================] - 0s 3ms/step - loss: 21.2105

Test Loss -> 21.21
```



3. DNN

03. DNN Deep Neural Network



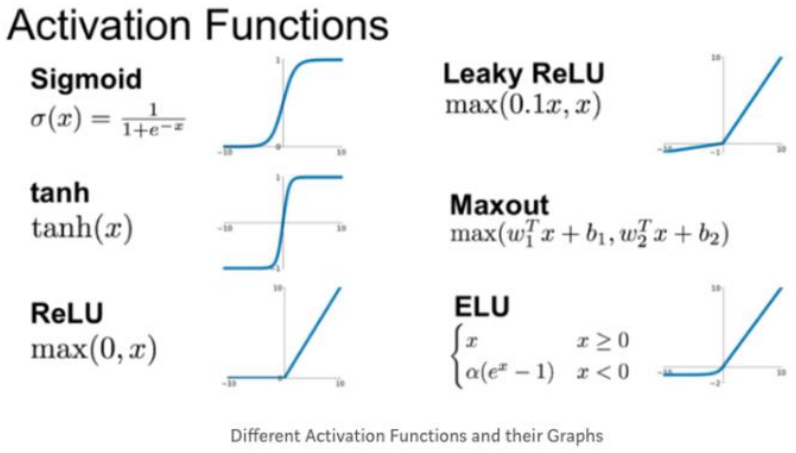
DNN의 구조

03. DNN Deep Neural Network

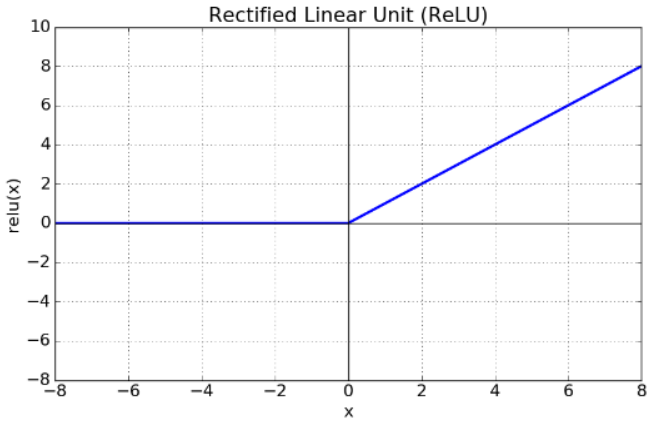
- 심층신경망(DNN)은 은닉 계층을 많이 쌓아서 만든 인공지능 기술
- 주로 하나의 은닉 계층을 포함하는 ANN과 달리 수십, 수백개의 은닉 계층으로 구성
- 계층이 많아져 우수한 성능을 낼 수 있고 적용 분야도 다양
- 다수의 은닉 계층은 입력 신호를 더 정교하게 처리
- 과적합을 얼마나 방지하느냐에 따라 좋은 모델이 될 수 있음

03. DNN Deep Neural Network

- 경사도 소실과 ReLU 활성화 함수
  - DNN은 여러 은닉 계층으로 구성되어 신경망의 최적화 과정 시 경사도 소실 발생 가능
  - 경사도 소실 문제를 극복하기 위해 ReLU함수 사용
  - ReLU는 입력이 0보다 큰 구간에서 직선 함수이기 때문에 값이 커져도 경사도 구하기 가능



Different Activation Functions and their Graphs



03. DNN Deep Neural Network

- DNN 구현 방법 및 단계

1. 기본 매개변수 설정
2. 분류 DNN 모델 구현
3. 데이터 준비
4. DNN의 학습 및 성능 평가

03. 1) 필기체를 구분하는 분류 DNN 구현

- 구현 방법 및 단계

① 기본 매개변수 설정

```
Nin = 784
Nh_l = [100, 50]
number_of_class = 10
Nout = number_of_class
```

03. 1) 필기체를 구분하는 분류 DNN 구현

- 구현 방법 및 단계

② DNN 모델 구현

```
from keras import layers, models
import tensorflow as tf

class DNN(models.Sequential):
    def __init__(self, Nin, Nh_l, Nout):
        super().__init__()
        self.add(layers.Dense(Nh_l[0], activation='relu', input_shape=(Nin,), name='Hidden-1'))
        self.add(layers.Dense(Nh_l[1], activation='relu', name='Hidden-2'))
        self.add(layers.Dense(Nout, activation='softmax'))
        self.compile(loss='categorical_crossentropy',
                     optimizer='adam',
                     metrics=['accuracy'])
```

✓ 연쇄방식으로 계층 기술

✓ 2개의 히든 계층 생성

03. 1) 필기체를 구분하는 분류 DNN 구현

- 구현 방법 및 단계

② DNN 모델 구현

```
from keras import layers, models
import tensorflow as tf

class DNN(models.Sequential):
    def __init__(self, Nin, Nh_l, Nout):
        super().__init__()
        self.add(layers.Dense(Nh_l[0], activation='relu', input_shape=(Nin,), name='Hidden-1'))
        self.add(layers.Dense(Nh_l[1], activation='relu', name='Hidden-2'))
        self.add(layers.Dense(Nout, activation='softmax'))
        self.compile(loss='categorical_crossentropy',
                     optimizer='adam',
                     metrics=['accuracy'])
```

✓ 연쇄방식으로 계층 기술

✓ 2개의 히든 계층 생성

③ 데이터 준비는 ANN에서 사용할 데이터 가져오기와 같음

03. 1) 필기체를 구분하는 분류 DNN 구현

- 구현 방법 및 단계

③ 데이터 준비

```
import numpy as np
from keras import datasets
from keras.utils import np_utils

def Data_func():
    (X_train, y_train), (X_test, y_test) = tf.keras.datasets.mnist.load_data()

    Y_train = np_utils.to_categorical(y_train)
    Y_test = np_utils.to_categorical(y_test)

    L, W, H = X_train.shape
    X_train = X_train.reshape(-1, W * H)
    X_test = X_test.reshape(-1, W * H)

    X_train = X_train / 255.0
    X_test = X_test / 255.0

    return (X_train, Y_train), (X_test, Y_test)
```

✓ ANN과 같음

03. 1) 필기체를 구분하는 분류 DNN 구현

- 구현 방법 및 단계

④ 학습 및 성능 평가

```
import matplotlib.pyplot as plt
from keraspp.skeras import plot_loss, plot_acc

model = DNN(Nin, Nh_l, Nout)
(X_train, Y_train), (X_test, Y_test) = Data_func()

history = model.fit(X_train, Y_train, epochs=5, batch_size=100, validation_split=0.2)
performace_test = model.evaluate(X_test, Y_test, batch_size=100)
print('Test Loss and Accuracy ->', performace_test)

plot_loss(history)
plt.show()
plot_acc(history)
plt.show()
```

✓ ANN과 같음

03. 1) 필기체를 구분하는 분류 DNN 구현

- 구현 방법 및 단계

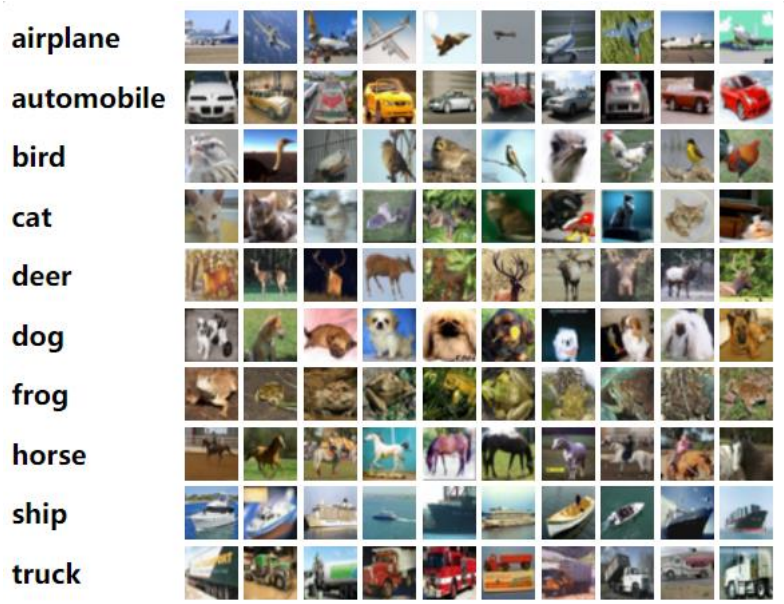
④ 학습 및 성능 평가

```
Epoch 1/5
480/480 [==============================] - 2s 3ms/step - loss: 0.3785 - accuracy: 0.8919 - val_loss: 0.1934 - val_accuracy: 0.9457
Epoch 2/5
480/480 [==============================] - 1s 3ms/step - loss: 0.1645 - accuracy: 0.9515 - val_loss: 0.1436 - val_accuracy: 0.9592
Epoch 3/5
480/480 [==============================] - 1s 3ms/step - loss: 0.1164 - accuracy: 0.9660 - val_loss: 0.1222 - val_accuracy: 0.9653
Epoch 4/5
480/480 [==============================] - 1s 3ms/step - loss: 0.0884 - accuracy: 0.9740 - val_loss: 0.1103 - val_accuracy: 0.9681
Epoch 5/5
480/480 [==============================] - 1s 3ms/step - loss: 0.0704 - accuracy: 0.9789 - val_loss: 0.0975 - val_accuracy: 0.9720
100/100 [==============================] - 0s 2ms/step - loss: 0.0924 - accuracy: 0.9715
Test Loss and Accuracy -> [0.09240478277206421, 0.9714999794960022]
```

- ✓ ANN과 거의 유사
- ✓ MNIST가 비교적 이미지가 간단해 성능 차이가 없으나 데이터가 많거나 복잡한 이미지에서는 일반적으로 DNN이 더 우수한 성능을 보임

03. 2) 컬러 이미지를 분류하는 DNN 구현

- 필기체보다 복잡도가 높은 컬러 이미지를 DNN으로 분류
- 사용할 데이터셋은 CIFAR-10
  10가지 사물이 담긴 컬러 이미지로 총 6만장, 32*32*3 픽셀로 이루어져 있음



03. 2) 컬러 이미지를 분류하는 DNN 구현

- 구현 방법 및 단계

① 데이터 가져오기

```
import numpy as np
from keras import datasets
from keras.utils import np_utils
import tensorflow as tf
```

✓ 패키지 가져오기

```
def Data_func():
    (X_train, y_train), (X_test, y_test) = tf.keras.datasets.cifar10.load_data()

    Y_train = np_utils.to_categorical(y_train)
    Y_test = np_utils.to_categorical(y_test)

    L, W, H, C = X_train.shape
    X_train = X_train.reshape(-1, W * H * C)
    X_test = X_test.reshape(-1, W * H * C)

    X_train = X_train / 255.0
    X_test = X_test / 255.0

    return (X_train, Y_train), (X_test, Y_test)
```

✓ 10가지 클래스로 구분된 2차원 배열로 변환

✓ 컬러 이미지 포함 4차원

03. 2) 컬러 이미지를 분류하는 DNN 구현

- 구현 방법 및 단계

② DNN 모델링

```
from keras import layers, models

class DNN(models.Sequential):
    def __init__(self, Nin, Nh_l, Pd_l, Nout):
        super().__init__()

        self.add(layers.Dense(Nh_l[0], activation='relu',
                              input_shape=(Nin,), name='Hidden-1'))
        self.add(layers.Dropout(Pd_l[0]))

        self.add(layers.Dense(Nh_l[1], activation='relu',
                              name='Hidden-2'))
        self.add(layers.Dropout(Pd_l[1]))

        self.add(layers.Dense(Nout, activation='softmax'))

        self.compile(loss='categorical_crossentropy',
                     optimizer='adam',
                     metrics=['accuracy'])
```

✓ Dropout은 지정된 확률만큼 노드의 신호가 단절

03. 2) 컬러 이미지를 분류하는 DNN 구현

- 구현 방법 및 단계

③ 학습 효과 분석

```
from keraspp.skeras import plot_loss, plot_acc
import matplotlib.pyplot as plt
```

✓ 효과 분석하는 코드 가져오기

03. 2) 컬러 이미지를 분류하는 DNN 구현

- 구현 방법 및 단계

③ 학습 효과 분석

```
from keraspp.skeras import plot_loss, plot_acc
import matplotlib.pyplot as plt
```

✓ 효과 분석하는 코드 가져오기

03. 2) 컬러 이미지를 분류하는 DNN 구현

- 구현 방법 및 단계

④ 학습 및 성능 평가

```
Pd_l=[0.0, 0.0]
Nh_l = [100, 50]
number_of_class = 10
Nout = number_of_class

(X_train, Y_train), (X_test, Y_test) = Data_func()
model = DNN(X_train.shape[1], Nh_l, Pd_l, Nout)
history = model.fit(X_train, Y_train, epochs=100, batch_size=100, validation_split=0.2)

performace_test = model.evaluate(X_test, Y_test, batch_size=100)
print('Test Loss and Accuracy ->', performace_test)

plot_acc(history)
plt.show()
plot_loss(history)
plt.show()
```

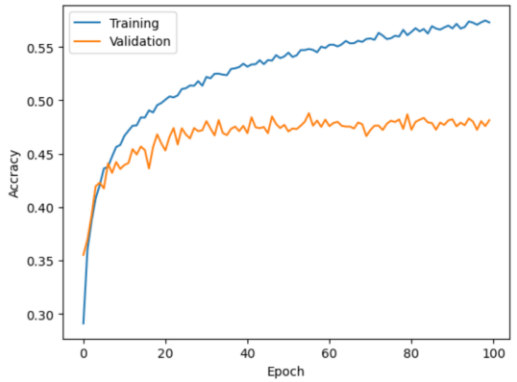
✓ 학습 진행 경과는 history 변수에 저장

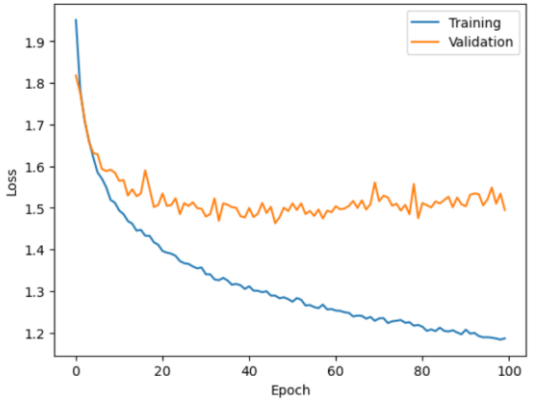
03. 2) 컬러 이미지를 분류하는 DNN 구현

- 구현 방법 및 단계

④ 학습 및 성능 평가

```
Epoch 97/100
400/400 [==============================] - 2s 4ms/step - loss: 1.1878 - accuracy: 0.5708 - val_loss: 1.5486 - val_accuracy: 0.4724
Epoch 98/100
400/400 [==============================] - 2s 4ms/step - loss: 1.1857 - accuracy: 0.5730 - val_loss: 1.5094 - val_accuracy: 0.4805
Epoch 99/100
400/400 [==============================] - 2s 4ms/step - loss: 1.1832 - accuracy: 0.5748 - val_loss: 1.5338 - val_accuracy: 0.4758
Epoch 100/100
400/400 [==============================] - 2s 4ms/step - loss: 1.1860 - accuracy: 0.5729 - val_loss: 1.4946 - val_accuracy: 0.4814
100/100 [==============================] - 0s 2ms/step - loss: 1.4676 - accuracy: 0.4901
Test Loss and Accuracy -> [1.4675611257553 1, 0.4900999963283539]
```

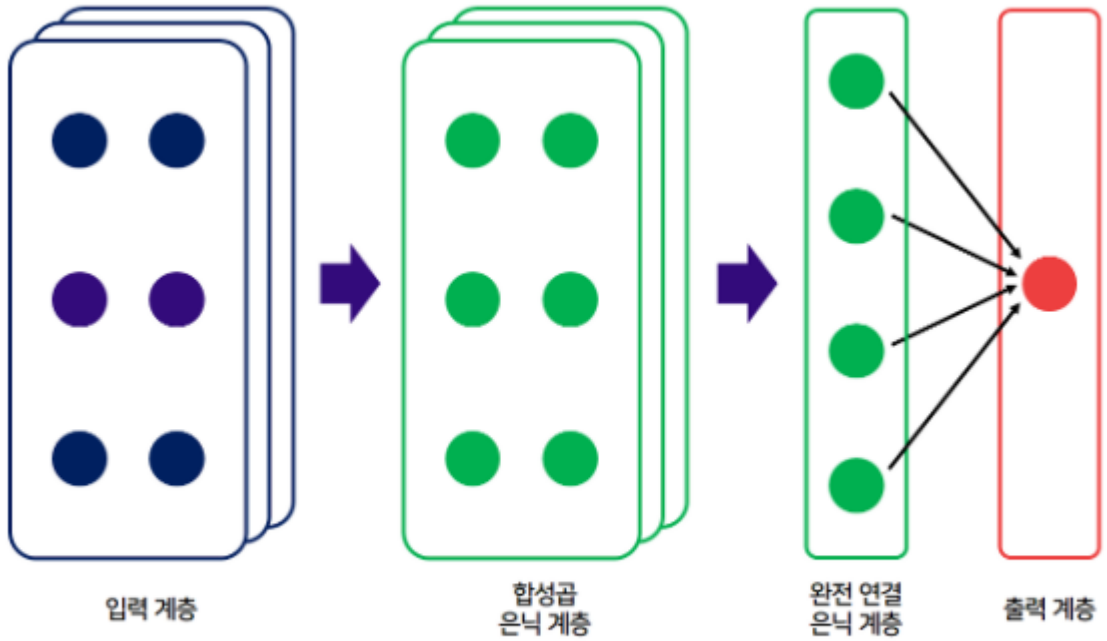




- ✓ 10가지 사물을 인공지능이 약 49%까지 분류할 수 있다는 의미
- ✓ 드롭아웃을 하지 않아서 학습데이터와 검증 데이터 간에 성능 차이가 큼
- ✓ 드롭아웃 설정값 조정하여 과적합 줄이기 가능

4. CNN

04. CNN Convolutional Neural Network



CNN의 구조

04. CNN Convolutional Neural Network

- 합성곱신경망(CNN)은 영상 처리에 많이 활용되는 합성곱을 이용하는 인공신경망 기술
- 합성곱을 이용해 가중치 수를 줄여 연산량을 줄이면서 이미지 처리를 효과적으로 함
- 이미지의 특징점을 효율적으로 찾을 수 있어 인공신경망 효능을 높일 수 있음
- CNN 원리
  - ✓ CNN은 합성곱 필터를 이용해 신경망 동작을 수행
  - ✓ 여러 작은 필터가 이미지 위를 돌아다니며 특징점을 찾아 합성곱 결과를 다음 계층으로 보냄
  - ✓ DNN 대비 적은 수의 가중치로 이미지 처리 효율적으로 가능

04. 1) 필기체를 분류하는 CNN 구현

- 구현 방법 및 단계

① 분류 CNN 모델링

```
import keras
from keras import models, layers
from keras import backend
import tensorflow as tf
```

✓ 서브 패키지 불러오기

✓ Models : 연쇄 방식 모델링 객체인 squentia 사용

✓ Layers : Conv2D, Maxpooling2D, Flatten

04. 1) 필기체를 분류하는 CNN 구현

- 구현 방법 및 단계

① 분류 CNN 모델링

```
import keras
from keras import models, layers
from keras import backend
import tensorflow as tf
```

✓ 서브 패키지 불러오기

✓ Models : 연쇄 방식 모델링 객체인 squentia 사용

✓ Layers : Conv2D, Maxpooling2D, Flatten

✓ Backend : 딥러닝 엔진 직접 제어

04. 1) 필기체를 분류하는 CNN 구현

- 구현 방법 및 단계

① 분류 CNN 모델링

```
class CNN(models.Sequential):
    def __init__(self, input_shape, num_classes):
        super().__init__()

        self.add(layers.Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
        self.add(layers.Conv2D(64, (3, 3), activation='relu'))
        self.add(layers.MaxPooling2D(pool_size=(2, 2)))
        self.add(layers.Dropout(0.25))
        self.add(layers.Flatten())
        self.add(layers.Dense(128, activation='relu'))
        self.add(layers.Dropout(0.5))
        self.add(layers.Dense(num_classes, activation='softmax'))

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

- ✓ 첫 번째 은닉 계층 정의
- ✓ 3X3 커널 32개로 구성
- ✓ 최대 풀링을 수행해 인접한 2X2 셀을 묶어 가장 큰 값만 내보내는 부속 계층
- ✓ 드롭아웃 진행
- ✓ Flatten은 2차원 이미지를 1차원 벡터로 변환

04. 1) 필기체를 분류하는 CNN 구현

- 구현 방법 및 단계

② 분류 CNN을 위한 데이터 준비

```
from keras import datasets

class DATA():
    def __init__(self):
        num_classes = 10

        (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
        img_rows, img_cols = x_train.shape[1:]

        if backend.image_data_format() == 'channels_first':
            x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
            x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
            input_shape = (1, img_rows, img_cols)
        else:
            x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
            x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
            input_shape = (img_rows, img_cols, 1)

        x_train = x_train.astype('float32')
        x_test = x_test.astype('float32')
        x_train /= 255
        x_test /= 255

        y_train = keras.utils.to_categorical(y_train, num_classes)
        y_test = keras.utils.to_categorical(y_test, num_classes)

        self.input_shape = input_shape
        self.num_classes = num_classes
        self.x_train, self.y_train = x_train, y_train
        self.x_test, self.y_test = x_test, y_test
```

04. 1) 필기체를 분류하는 CNN 구현

- 구현 방법 및 단계

② 분류 CNN을 위한 데이터 준비

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

✓ DNN과 다른 점 1) 이미지를 벡터화 하지 않고 그대로 사용 2) 흑백 이미지의 채널 정보를 처리하려면 추가적인 차원을 이미지 데이터에 포함해야 함

```
if backend.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)
```

✓ 이미지 앞단에 채널 추가

✓ 맨 앞에 1로 채널 수 표시

✓ 맨 뒤에 1로 채널 수 표시

04. 1) 필기체를 분류하는 CNN 구현

- 구현 방법 및 단계

② 분류 CNN을 위한 데이터 준비

```
if backend.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)
```

✓ 샘플 수, 채널 수, 이미지의 가로 길이, 이미지의 세로 길이로 구성

✓ DNN과 달리 이미지의 가로와 세로가 구분되어 다뤄짐

04. 1) 필기체를 분류하는 CNN 구현

- 구현 방법 및 단계

  ③ 분류 CNN 학습 효과 분석

```
from keraspp.skeras import plot_loss, plot_acc
import matplotlib.pyplot as plt
```

  ✓ 그래프 그리는 기능 가져오기

04. 1) 필기체를 분류하는 CNN 구현

- 구현 방법 및 단계

  ③ 분류 CNN 학습 및 성능 평가

```
batch_size = 128
epochs = 10

data = DATA()
model = CNN(data.input_shape, data.num_classes)

history = model.fit(data.x_train, data.y_train,
          batch_size=batch_size,
          epochs=epochs,
          validation_split=0.2)

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

plot_loss(history)
plt.show()
plot_acc(history)
plt.show()
```

✓ 준비된 데이터를 data 인스턴스에 넣어두고 모델을 model 인스턴스에 저장

✓ 20% 분리하여 검증 데이터로 사용

✓ 검증 데이터는 학습 데이터의 일부분이므로 평가 데이터와는 다름

✓ 데이터 평가

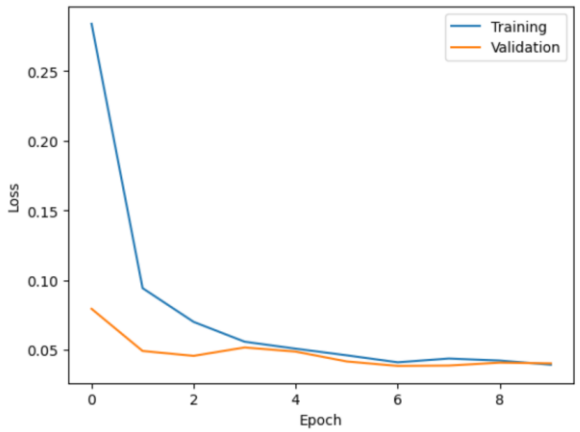
04. 1) 필기체를 분류하는 CNN 구현

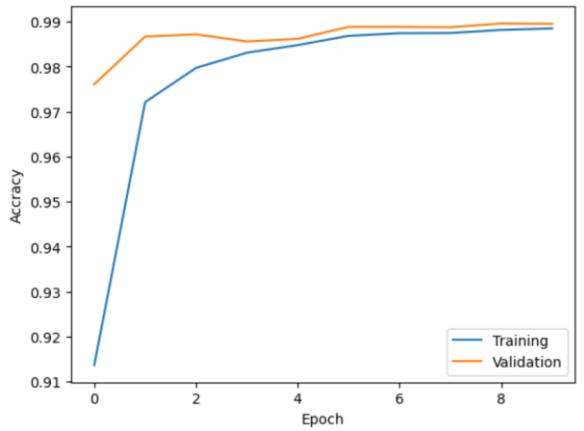
- 구현 방법 및 단계

  ③ 분류 CNN 학습 및 성능 평가

```
Epoch 9/10
375/375 [==============================] - 25s 68ms/step - loss: 0.0422 - accuracy: 0.9881 - val_loss: 0.0407 - val_accuracy: 0.9896
Epoch 10/10
375/375 [==============================] - 27s 73ms/step - loss: 0.0392 - accuracy: 0.9885 - val_loss: 0.0402 - val_accuracy: 0.9895
313/313 [==============================] - 3s 10ms/step - loss: 0.0327 - accuracy: 0.9893

Test loss: 0.03266153112053871
Test accuracy: 0.989300012588501
```





04. 2) 컬러 이미지를 분류하는 CNN 구현

- CIFAR-10 컬러 이미지 분류하는 CNN 구현 방법 및 단계

1. 분류 CNN 패키지 가져오기
2. 분류 CNN 모델링
3. 분류 CNN을 위한 데이터 준비
4. 분류 CNN의 학습 및 성능 평가를 위한 머신 클래스
5. 분류 CNN의 수행

04. 2) 컬러 이미지를 분류하는 CNN 구현

- 구현 방법 및 단계

① 분류 CNN 패키지 가져오기

```
from sklearn import model_selection, metrics
from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf
```

```
import numpy as np
import matplotlib.pyplot as plt
import os
```

```
from keras import backend as K
from keras.utils import np_utils
from keras.models import Model
from keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense, Dropout
```

```
from keraspp import skeras
from keraspp import sfile
```

04. 2) 컬러 이미지를 분류하는 CNN 구현

- 구현 방법 및 단계

① 분류 CNN 패키지 가져오기

```python
import datetime
import uuid
import os


def unique_filename(type='uuid'):
    if type == 'datetime':
        filename = datetime.datetime.now().strftime("%y%m%d_%H%M%S")
    else:  # type == "uuid"
        filename = str(uuid.uuid4())
    return filename


def makenewfold(prefix='output_', type='datetime'):
    suffix = unique_filename('datetime')
    foldname = 'output_' + suffix
    os.makedirs(foldname)
    return foldname

```

✓ Sfile.py

04. 2) 컬러 이미지를 분류하는 CNN 구현

- 구현 방법 및 단계

  ② 분류 CNN 모델링

  ✓ 사용할 인공신경망 모델은 LeNet

LeNet의 구조

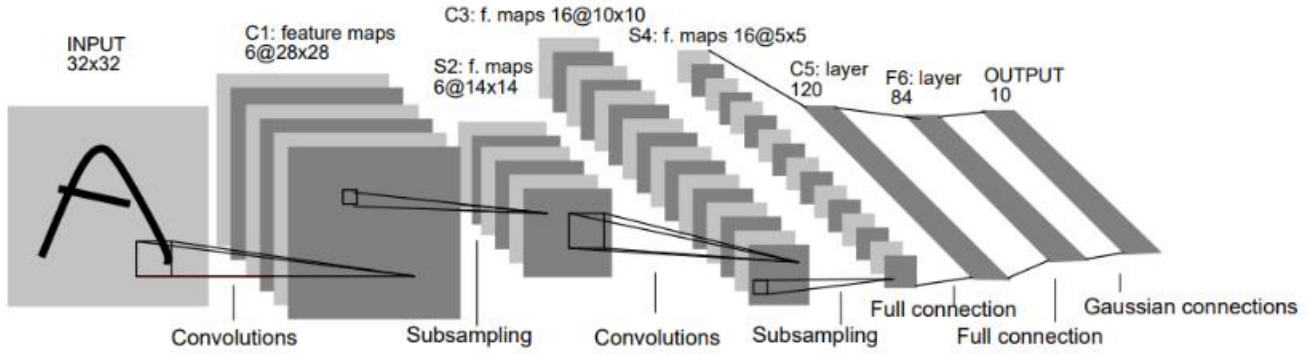


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

04. 2) 컬러 이미지를 분류하는 CNN 구현

- 구현 방법 및 단계

② 분류 CNN 모델링

```
# 2. 분류 CNN 모델링
class CNN(Model):
    def __init__(self, nb_classes):
        super(CNN,self).__init__()
        self.nb_classes = nb_classes

        self.conv2D_A = Conv2D(32, kernel_size=(3, 3), activation='relu')
        self.conv2D_B = Conv2D(64, (3, 3), activation='relu')
        self.maxPooling2D_A = MaxPooling2D(pool_size=(2, 2))
        self.dropout_A = Dropout(0.25)
        self.flatten = Flatten()

        self.dense_A = Dense(128, activation='relu')
        self.dropout_B = Dropout(0.5)
        self.dense_B = Dense(nb_classes, activation='softmax', name='preds')

    def call(self, x):
        h = self.conv2D_A(x)
        h = self.conv2D_B(h)
        h = self.maxPooling2D_A(h)
        h = self.dropout_A(h)
        h = self.flatten(h)

        h = self.dense_A(h)
        h = self.dropout_B(h)

        y = self.dense_B(h)

        return y
```

✓ 모델에 사용될 구성 요소

✓ 모델 호출 시 사용되는 함수

04. 2) 컬러 이미지를 분류하는 CNN 구현

- 구현 방법 및 단계

② 분류 CNN 모델링

```
nb_classes = 10
model = CNN(nb_classes=nb_classes)
model.compile(loss='categorical_crossentropy',
                      optimizer='adadelta', metrics=['accuracy'])
```

✓ 모델 사용 준비

04. 2) 컬러 이미지를 분류하는 CNN 구현

- 구현 방법 및 단계

③ 분류 CNN을 위한 데이터 준비

```
# 3. 분류 CNN을 위한 데이터 준비
class DataSet:
    def __init__(self, X, y, nb_classes, scaling=True,
                 test_size=0.2, random_state=0):
        self.X = X
        self.add_channels()
        X = self.X
```

✓ 주어진 데이터를 머신러닝에 사용하기 적합하도록 조정하는 기능인 DataSet 클래스를 만듬

✓ 데이터 선언하고 초기화 진행

✓ 입력값인 X를 멤버 변수로 지정한 후 채널 정보 추가

04. 2) 컬러 이미지를 분류하는 CNN 구현

- 구현 방법 및 단계

③ 분류 CNN을 위한 데이터 준비

```
# the data, shuffled and split between train and test sets
X_train, X_test, y_train, y_test = model_selection.train_test_split(
    X, y, test_size=0.2, random_state=random_state)

print(X_train.shape, y_train.shape)

X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
```

✓ 데이터 세팅

✓ 80% 학습 데이터 지정

✓ 이미지 데이터가 정수인 경우가 있으니 32비트 규격의 실수로 바꾸어 줌

04. 2) 컬러 이미지를 분류하는 CNN 구현

- 구현 방법 및 단계

③ 분류 CNN을 위한 데이터 준비

```
# 3. 분류 CNN을 위한 데이터 준비
class DataSet:
    def __init__(self, X, y, nb_classes, scaling=True,
                 test_size=0.2, random_state=0):
        self.X = X
        self.add_channels()
        X = self.X
```

✓ 주어진 데이터를 머신러닝에 사용하기 적합하도록 조정하는 기능인 DataSet 클래스를 만듬

✓ 데이터 선언하고 초기화 진행

✓ 입력값인 X를 멤버 변수로 지정한 후 채널 정보 추가

04. 2) 컬러 이미지를 분류하는 CNN 구현

- 구현 방법 및 단계

③ 분류 CNN을 위한 데이터 준비

```
    if scaling:
        # scaling to have (0, 1) for each feature (each pixel)
        scaler = MinMaxScaler()
        n = X_train.shape[0]
        X_train = scaler.fit_transform(
            X_train.reshape(n, -1)).reshape(X_train.shape)
        n = X_test.shape[0]
        X_test = scaler.transform(
            X_test.reshape(n, -1)).reshape(X_test.shape)
        self.scaler = scaler

    print('X_train shape:', X_train.shape)
    print(X_train.shape[0], 'train samples')
    print(X_test.shape[0], 'test samples')
```

✓ 스케일링 실시

✓ 스케일링 기준은 학습 데이터인 X_train이어야만 함

✓ 출력값은 원핫 인코딩을 이용해 정수값을 이진 벡터로 바꾸어 줌

04. 2) 컬러 이미지를 분류하는 CNN 구현

- 구현 방법 및 단계

③ 분류 CNN을 위한 데이터 준비

```
        # convert class vectors to binary class matrices
        Y_train = np_utils.to_categorical(y_train, nb_classes)
        Y_test = np_utils.to_categorical(y_test, nb_classes)

        self.X_train, self.X_test = X_train, X_test
        self.Y_train, self.Y_test = Y_train, Y_test
        self.y_train, self.y_test = y_train, y_test
```

✓ Nb_classes는 클래스 수. 클래스 수만큼 이진 원소를 가진 벡터로 바꾸어 줌

✓ 학습과 검증에 사용할 데이터를 멤버 변수로 등록

04. 2) 컬러 이미지를 분류하는 CNN 구현

- 구현 방법 및 단계

③ 분류 CNN을 위한 데이터 준비

```
def add_channels(self):
    X = self.X

    if len(X.shape) == 3:
        N, img_rows, img_cols = X.shape

        if K.image_dim_ordering() == 'th':
            X = X.reshape(X.shape[0], 1, img_rows, img_cols)
            input_shape = (1, img_rows, img_cols)
        else:
            X = X.reshape(X.shape[0], img_rows, img_cols, 1)
            input_shape = (img_rows, img_cols, 1)
    else:
        input_shape = X.shape[1:]  # channel is already included.

    self.X = X
    self.input_shape = input_shape
```

✓ 흑백 이미지인지 검사

✓ 'th' 시애노 방식의 데이터 포멧을 사용한다면 채널 정보를 길이정보 바로 다음인 두 번째 차원에 삽입하고 텐서플로 방식의 포멧인 경우 맨 마지막에 넣어줌

04. 2) 컬러 이미지를 분류하는 CNN 구현

- 구현 방법 및 단계

③ 분류 CNN을 위한 데이터 준비

```
from keras import datasets
(X, y), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
nb_classes = 10
data = DataSet(X, y, nb_classes)
print('data.input_shape', data.input_shape)
```

```
(40000, 32, 32, 3) (40000, 1)
X_train shape: (40000, 32, 32, 3)
40000 train samples
10000 test samples
data.input_shape (32, 32, 3)
```

✓ DataSet클래스는 다음과 같이 사용 가능

04. 2) 컬러 이미지를 분류하는 CNN 구현

- 구현 방법 및 단계

③ 분류 CNN을 위한 데이터 준비

```
epochs=2
batch_size=128
verbose=1
history = model.fit(data.X_train, data.Y_train,
                    batch_size=batch_size, epochs=epochs,
                    verbose=verbose,
                    validation_data=(data.X_test, data.Y_test))
```

✓ 만들어진 데이터를 이용해 앞서 정의한 CNN 모델로 간단하게 학습하는 방법

04. 2) 컬러 이미지를 분류하는 CNN 구현

- 구현 방법 및 단계

④ 분류 CNN의 학습 및 성능 평가를 위한 머신 클래스

```
# 4. 분류 CNN의 학습 및 성능 평가를 위한 머신 클래스
class Machine():
    def __init__(self, X, y, nb_classes=2, fig=True):
        self.nb_classes = nb_classes
        self.set_data(X, y)
        self.set_model()
        self.fig = fig

    def set_data(self, X, y):
        nb_classes = self.nb_classes
        self.data = DataSet(X, y, nb_classes)
        print('data.input_shape', self.data.input_shape)
```

✓ 학습 및 성능 평가를 쉽게 수행할 수 있는 상위 개념 클래스

✓ 클래스 초기화

✓ 데이터 설정 함수

04. 2) 컬러 이미지를 분류하는 CNN 구현

- 구현 방법 및 단계

④ 분류 CNN의 학습 및 성능 평가를 위한 머신 클래스

```
    def set_model(self):
        nb_classes = self.nb_classes
        data = self.data
        self.model = CNN(nb_classes=nb_classes)
        self.model.compile(loss='categorical_crossentropy',
                           optimizer='adadelta', metrics=['accuracy'])

    def fit(self, epochs=10, batch_size=128, verbose=1):
        data = self.data
        model = self.model

        history = model.fit(data.X_train, data.Y_train,
                            batch_size=batch_size, epochs=epochs,
                            verbose=verbose,
                            validation_data=(data.X_test, data.Y_test))
        return history
```

✓ 모델 설정 함수

✓ 학습 진행 멤버 함수

04. 2) 컬러 이미지를 분류하는 CNN 구현

- 구현 방법 및 단계

④ 분류 CNN의 학습 및 성능 평가를 위한 머신 클래스

```
    def run(self, epochs=100, batch_size=128, verbose=1):
        data = self.data
        model = self.model
        fig = self.fig

        history = self.fit(epochs=epochs,
                           batch_size=batch_size, verbose=verbose)

        score = model.evaluate(data.X_test, data.Y_test, verbose=0)

        print('Confusion matrix')
        Y_test_pred = model.predict(data.X_test, verbose=0)
        y_test_pred = np.argmax(Y_test_pred, axis=1)
        print(metrics.confusion_matrix(data.y_test, y_test_pred))

        print('Test score:', score[0])
        print('Test accuracy:', score[1])

        # Save results
        suffix = sfile.unique_filename('datatime')
        foldname = 'output_' + suffix
        os.makedirs(foldname)
        skeras.save_history_history(
            'history_history.npy', history.history, fold=foldname)
        model.save_weights(os.path.join(foldname, 'dl_model.h5'))
        print('Output results are saved in', foldname)

        if fig:
            plt.figure(figsize=(12, 4))
            plt.subplot(1, 2, 1)
            skeras.plot_acc(history)
            plt.subplot(1, 2, 2)
            skeras.plot_loss(history)
            plt.show()

        self.history = history

        return foldname
```

04. 2) 컬러 이미지를 분류하는 CNN 구현

- 구현 방법 및 단계

④ 분류 CNN의 학습 및 성능 평가를 위한 머신 클래스

```
def run(self, epochs=100, batch_size=128, verbose=1):
    data = self.data
    model = self.model
    fig = self.fig

    history = self.fit(epochs=epochs,
                       batch_size=batch_size, verbose=verbose)

    score = model.evaluate(data.X_test, data.Y_test, verbose=0)

    print('Confusion matrix')
    Y_test_pred = model.predict(data.X_test, verbose=0)
    y_test_pred = np.argmax(Y_test_pred, axis=1)
    print(metrics.confusion_matrix(data.y_test, y_test_pred))

    print('Test score:', score[0])
    print('Test accuracy:', score[1])
```

✓ 성능 평가 전체를 진행하는 run()함수 구현

✓ 함수 내에서 학습과 성능 평가를 담당하는 history

✓ 오류 메트릭스

04. 2) 컬러 이미지를 분류하는 CNN 구현

- 구현 방법 및 단계

④ 분류 CNN의 학습 및 성능 평가를 위한 머신 클래스

```
        # Save results
        suffix = sfile.unique_filename('datatime')
        foldname = 'output_' + suffix
        os.makedirs(foldname)
        skeras.save_history_history(
            'history_history.npy', history.history, fold=foldname)
        model.save_weights(os.path.join(foldname, 'dl_model.h5'))
        print('Output results are saved in', foldname)

        if fig:
            plt.figure(figsize=(12, 4))
            plt.subplot(1, 2, 1)
            skeras.plot_acc(history)
            plt.subplot(1, 2, 2)
            skeras.plot_loss(history)
            plt.show()

        self.history = history

        return foldname
```

- ✓ 현재 시각을 초 단위로 구해 새로운 이름 생성
- ✓ 새로운 저장용 폴더 생성
- ✓ History_history.npy에 저장
- ✓ 가중치는 dl_model.h5에 저장
- ✓ 저장 폴더가 겹치지 않기 위해 생성한 문자열 앞에 임의로 'output_' 붙임
- ✓ 학습 곡선 그리기

04. 2) 컬러 이미지를 분류하는 CNN 구현

- 구현 방법 및 단계

⑤ 분류 CNN의 학습 및 성능 평가 수행

```
# 5. 분류 CNN의 학습 및 성능 평가 수행
from keras import datasets
import keras
assert keras.backend.image_data_format() == 'channels_last'

# from keraspp import aicnn
class MyMachine(Machine):
    def __init__(self):
        (X, y), (x_test, y_test) = datasets.cifar10.load_data()
        super(MyMachine,self).__init__(X, y, nb_classes=10)

def main():
    m = MyMachine()
    m.run(epochs=2)

main()
```

✓ 분류 CNN을 위한 머신에 기반하여 이미지 분류

✓ Epochs 100으로 실행

실습 시-작 ☺