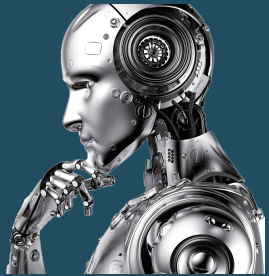
야, 인공지능!
너가 감히 내가 쓴 손글씨를 맞출 수 있을까?



201904010 박범찬
201904022 이기현
202104003 김나영

목차

table of contents

1

프로그램 필요성

Part 1

프로그램 필요성

스마트폰이나 태블릿 등 전자기기에서는 손글씨를 인식하는 기능이 지원되는 경우가 많으나, 글씨가 불규칙적일 경우 인식에 실패하는 경우가 종종 발생한다.

인식을 실패하는 경우가 자주 발생하게 되면, 사용자의 불편함을 야기하기 때문에 개선이 필요한 부분이라고 판단 되어서 해당 프로젝트를 시작하게 되었다.

이 프로젝트는 이미지 처리에 있어 좋은 성능을 보이는 CNN 알고리즘을 채택하였다.

2

프로그램 소개

프로그램 소개

제목 : 야,인공지능!너가 감히 내가 쓴 손글씨를 맞출 수 있을까?

목적

MNIST 데이터셋과 CNN 알고리즘을 활용하여 높은 확률로 해당 손글씨를 인식하는 것이 주된 목적이며,
나아가 형상화된 동작이나 점선 등을 인식하는 것까지 도전과제로 삼았다.

요약

사용자로부터 손글씨 이미지 파일 혹은 손글씨를 형상화 한 손동작 이미지 파일을 데이터로 입력받고, 해당 이미지가 무슨 글씨인지 결과값을 출력해준다.

기대효과

본 프로그램을 통하여 손글씨를 인식하는 데에 있어서 정확도 향상을 기대 할 수 있으며, 전자기기에서의 손글씨 인식의 쓰임새와 편리성이 더 확장되는 효과를 기대 할 수 있고, 더 나아가 손글씨 뿐만 아니라, 사람의 동작 등을 인식하는 프로그램으로의 발전 또한 기대 할 수 있다.

3

데이터 분석

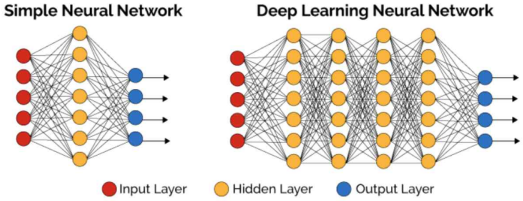
데이터 분석

인덱스	이름	데이터 타입	값	설명
0	5	범주형	5	훈련 데이터 0번째 손글씨
~	~			
59999	8	범주형	8	훈련 데이터 59999번째 손글씨
60000	7	범주형	7	테스트 데이터 0번째 손글씨
~	~			
69999	6	범주형	6	테스트 데이터 9999번째 손글씨.

4

사용 알고리즘

신경망



신경망은 인간의 두뇌에서 영감을 얻은 방식으로 데이터를 처리하도록 컴퓨터를 가르치는 인공지능 방식이다.
인간의 두뇌와 비슷한 계층 구조로 상호 연결된 노드 또는 뉴런을 사용하는 딥러닝이라고 불리는 기계 학습 과정의 유형이다.
신경망은 컴퓨터가 실수에서 배우고 지속적으로 개선하는 데 사용하는 적응형 시스템을 생성한다.
따라서 인공신경망은 문서 요약 또는 얼굴 인식과 같은 복잡한 문제를 더 정확하게 해결하려고 한다.

Part 4

신경망의 장단점

장점

1. 복잡한 문제 해결
2. 자동 특징 추출
3. 대용량 데이터 처리

단점

1. 계산 비용과 시간이 많이 소모됨
2. 데이터 요구사항이 많음
3. 해석 가능성의 어려움

Part 4

활성화함수

신경망에서 Hidden layer의 활성화 함수로 ReLu 함수를 사용하여 이진 분류를 한 뒤, Output layer에 Softmax 함수를 통해 확률 나타낸다.

Q. 이진 분류라면, Sigmoid 함수가 아닌 왜 ReLu 함수를 사용했을까?

1. 기울기 소실 방지 : Sigmoid 입력값이 극단적일 경우, 기울기가 0에 가까워진다 (= 학습이 매우 느리고, 신경망이 수렴하기 어려움)
So, ReLU는 양수 입력 값에 대해 일정한 기울기를 갖고 있으므로 이 문제를 방지할 수 있다.

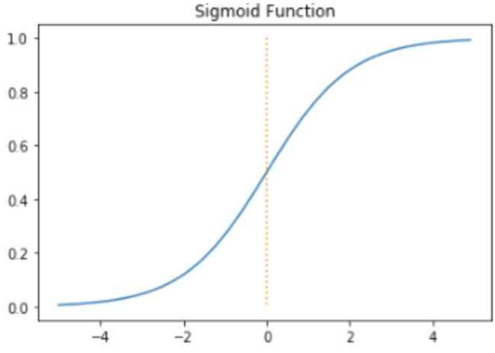
2. 계산의 효율성 : Sigmoid는 지수 함수가 포함되기 때문에 나눗셈 같은 복잡한 수학적 연산이 필요하다.
But, ReLU는 간단하게 계산할 수 있기 때문에 계산 효율성을 높일 수 있고 이는 훈련 프로세스를 더 빠르게 만들고 최적의 값으로 수렴하는 속도도 높여줄 수 있다.

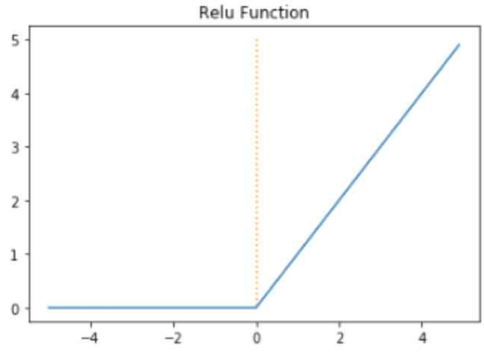
MNIST 데이터셋은 이미지 분류 작업에 사용되기 때문에, 일반적으로 CNN과 같은 이미지 처리에 특화된 알고리즘이 더 효과적인 결과를 보여주는 경향이 있다.
그러나, 이는 여전히 데이터셋의 특성, 문제 정의, 그리고 학습 프로세스의 특정 세부 사항에 따라 달라질 수 있다.

활성화함수

신경망에서 Hidden layer의 활성화 함수로 ReLu 함수를 사용하여 이진 분류를 한 뒤, Output layer에 Softmax 함수를 통해 확률 나타낸다.

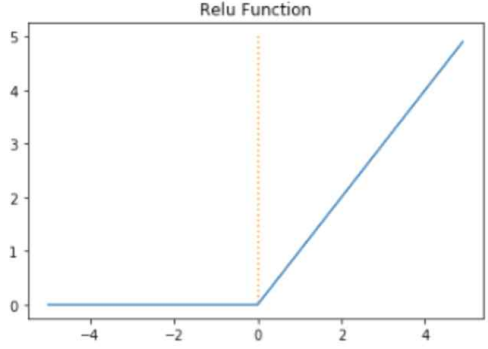
Q. 이진 분류라면, Sigmoid 함수가 아닌 왜 ReLu 함수를 사용했을까?

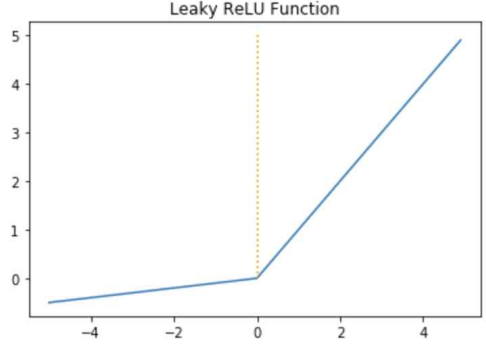




활성화함수

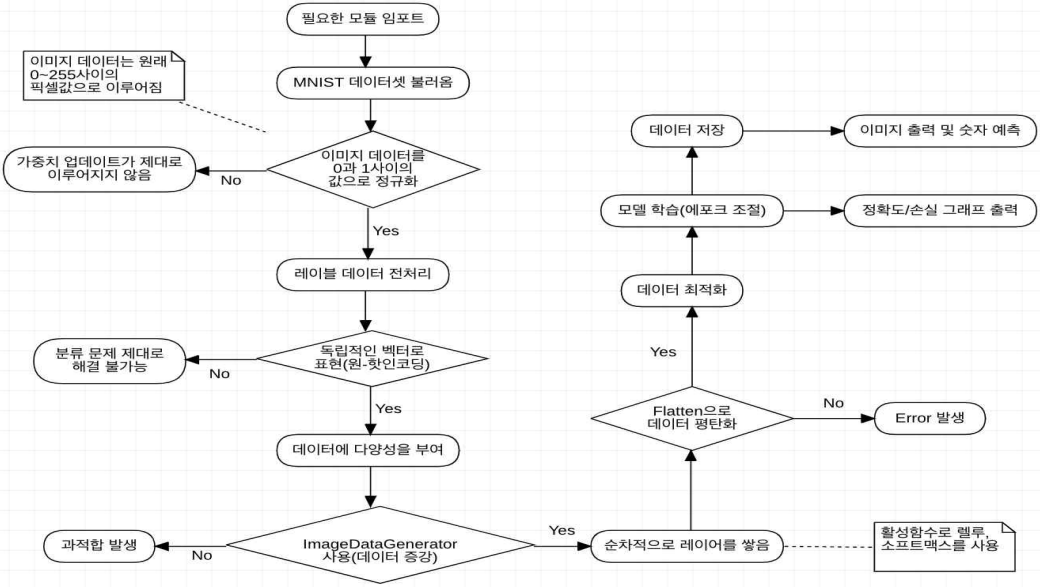
여러 장점이 있지만 여전히 ReLU 함수 또한, 단점은 있다.
입력값이 음수인 경우 기울기가 0이 되어 가중치 업데이트가 안될 수 있음(죽은 렐루 현상)
So, 이를 해결하기 위해 Leaky ReLU 함수가 등장하기도 했다.





5

순서도


필요한 모듈 임포트
이미지 데이터는 원래 0~255사이의 픽셀값으로 이루어짐
MNIST 데이터셋 불러옴
이미지 데이터를 0과 1사이의 값으로 정규화
No
가중치 업데이트가 제대로 이루어지지 않음
Yes
레이블 데이터 전처리
독립적인 벡터로 표현(원-핫인코딩)
No
분류 문제 제대로 해결 불가능
Yes
데이터에 다양성을 부여
ImageDataGenerator 사용(데이터 증강)
No
과적합 발생
Yes
순차적으로 레이어를 쌓음
활성함수로 렐루, 소프트맥스를 사용
Flatten으로 데이터 평탄화
No
Error 발생
Yes
데이터 최적화
모델 학습(에포크 조절)
정확도/손실 그래프 출력
데이터 저장
이미지 출력 및 숫자 예측

6

실제구현

In [1] :

```
# import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, BatchNormalization
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
from keras.preprocessing.image import ImageDataGenerator

img_rows, img_cols = 28, 28
(x_train, y_train), (x_test, y_test) = mnist.load_data()

if K.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

num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

datagen = ImageDataGenerator(
        rotation_range=10,
        zoom_range = 0.1,
        width_shift_range=0.1,
        height_shift_range=0.1)

datagen.fit(x_train)

model = Sequential()
model.add(Conv2D(32, kernel_size=3, activation='relu', input_shape=input_shape))
model.add(BatchNormalization())
model.add(Conv2D(32, kernel_size=3, activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(32, kernel_size=5, strides=2, padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.4))

model.add(Conv2D(64, kernel_size=3, activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size=3, activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size=5, strides=2, padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.4))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.4))
model.add(Dense(num_classes, activation='softmax'))

model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])

history = model.fit(datagen.flow(x_train,y_train, batch_size=64),
                    epochs = 50, validation_data = (x_test,y_test),
                    verbose = 1, steps_per_epoch=x_train.shape[0] // 64)

model.save('mnist_model.h5')
```

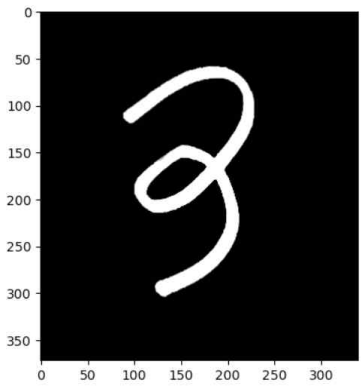
Out [1] :

```
Epoch 1/50
2023-06-19 14:46:34.788319: W tensorflow/tsl/platform/profile_utils/cpu_utils.cc:128] Failed to get CPU frequency: 0 Hz
937/937 [==============================] - 42s 44ms/step - loss: 0.3745 - accuracy: 0.8854 - val_loss: 0.0393 - val_accuracy: 0.9871
Epoch 2/50
937/937 [==============================] - 44s 47ms/step - loss: 0.1120 - accuracy: 0.9666 - val_loss: 0.0311 - val_accuracy: 0.9897
Epoch 3/50
937/937 [==============================] - 47s 50ms/step - loss: 0.0845 - accuracy: 0.9739 - val_loss: 0.0337 - val_accuracy: 0.9891
Epoch 4/50
937/937 [==============================] - 49s 52ms/step - loss: 0.0734 - accuracy: 0.9777 - val_loss: 0.0285 - val_accuracy: 0.9916
Epoch 5/50
937/937 [==============================] - 48s 51ms/step - loss: 0.0629 - accuracy: 0.9811 - val_loss: 0.0247 - val_accuracy: 0.9921
Epoch 6/50
937/937 [==============================] - 45s 48ms/step - loss: 0.0593 - accuracy: 0.9828 - val_loss: 0.0321 - val_accuracy: 0.9902
.
.
.
.
.
.
.
.
.
Epoch 45/50
937/937 [==============================] - 61s 65ms/step - loss: 0.0185 - accuracy: 0.9947 - val_loss: 0.0111 - val_accuracy: 0.9965
Epoch 46/50
937/937 [==============================] - 61s 65ms/step - loss: 0.0185 - accuracy: 0.9943 - val_loss: 0.0120 - val_accuracy: 0.9963
Epoch 47/50
937/937 [==============================] - 58s 62ms/step - loss: 0.0193 - accuracy: 0.9941 - val_loss: 0.0143 - val_accuracy: 0.9954
Epoch 48/50
937/937 [==============================] - 60s 64ms/step - loss: 0.0158 - accuracy: 0.9951 - val_loss: 0.0130 - val_accuracy: 0.9959
Epoch 49/50
937/937 [==============================] - 59s 63ms/step - loss: 0.0185 - accuracy: 0.9943 - val_loss: 0.0129 - val_accuracy: 0.9958
Epoch 50/50
937/937 [==============================] - 58s 61ms/step - loss: 0.0175 - accuracy: 0.9950 - val_loss: 0.0132 - val_accuracy: 0.9961
```

In [2] :

```
import cv2
import numpy as np
from keras.models import load_model
import matplotlib.pyplot as plt

model = load_model('mnist_model.h5')

# Load the image file
img = cv2.imread('3_Bad.png', cv2.IMREAD_GRAYSCALE)

plt.imshow(img, cmap='gray')
plt.show()

img_resized = cv2.resize(img, (28, 28))
img_ready = img_resized.reshape(1, 28, 28, 1)

img_ready = img_ready.astype('float32')
img_ready /= 255

proba = model.predict(img_ready)
digit = np.argmax(proba)

print("삐-빅, 이 숫자는", digit,"입니다. 휴-먼")

for i in range(10):
    print(f"Class {i}: {proba[0][i] * 100:.2f}%")
```

Out [2] :




```
1/1 [==============================] - 0s 61ms/step
삐-빅, 이 숫자는 3 입니다. 휴-먼
```

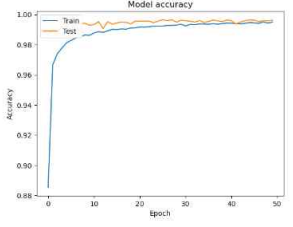
7

성능 평가

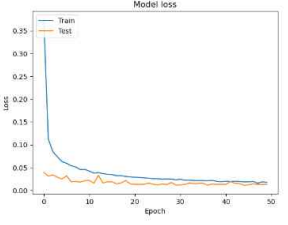
In [3] :

```
import matplotlib.pyplot as plt

def plot_curves(history):

    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()

    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()

plot_curves(history)
```

Out [3] :



→ 정확도



→ 손실률

In [4] :

```
for i in range(10):
    print(f"Class {i}: {proba[0][i] * 100:.2f}%")
```

Out [4] :

```
Class 0: 0.00%
Class 1: 0.00%
Class 2: 0.01%
Class 3: 99.98%
Class 4: 0.00%
Class 5: 0.01%
Class 6: 0.00%
Class 7: 0.00%
Class 8: 0.00%
Class 9: 0.00%
```

8

결론

Part 7

결론

요약

이 프로젝트는 손글씨 인식에 매우 우수한 성능을 보이고 있으며, 동작과 점선 인식까지 발전시킨 프로그램이다.

향후 발전사항

데이터 셋을 기본 MNIST데이터로 사용하고 있어, 배경 색을 흑백으로 맞춰줘야 한다는 한계가 있다.
또한 아직 점선 인식까지는 완벽하다고 평가 할 수는 없는 사항이다.
이를 발전시켜, 배경에 제약을 받지 않고 동작이나 글씨 부분을 추출하여 인식을 하거나
더 큰 픽셀 단위의 이미지로 발전시키는 등의 향후 발전사항이 있다.

9

코드실행


감사합니다.