

인텔 AI개발자 1기 미니 프로젝트

인간의 패배, 기계의 승리

MNIST 프로젝트

https://github.com/desafin/mnist_test.git

목차

01	모델 생성 과정
02	모델 트레이닝과 검증
03	모델을 로드해 숫자를 분류
04	결과 화면
05	개선할점과 느낀점
06	그외 질문

모델 생성 과정

<https://wikidocs.net/63618>

빠른 학습을 위해
CUDA가속을
사용하였습니다

[2]

```
device = 'cuda' if torch.cuda.is_available() else 'cpu'  
  
# 랜덤 시드 고정  
torch.manual_seed(777)  
  
# GPU 사용 가능일 경우 랜덤 시드 고정  
if device == 'cuda':  
    torch.cuda.manual_seed_all(777)  
  
print(device)
```

cuda

모델 생성 과정

하이퍼 파라미터 설정

[3]

```
learning_rate = 0.001  
training_epochs = 15  
batch_size = 100
```

모델 생성 과정

데이터 셋은 토치비전에서
제공하는 것을
사용하였습니다

```
mnist_train = datasets.MNIST(root='MNIST_data/', # 다운로드 경로 지정
                              train=True, # True를 지정하면 훈련 데이터로 다운로드
                              transform=transforms.ToTensor(), # 텐서로 변환
                              download=True)

mnist_test = datasets.MNIST(root='MNIST_data/', # 다운로드 경로 지정
                             train=False, # False를 지정하면 테스트 데이터로 다운로드
                             transform=transforms.ToTensor(), # 텐서로 변환
                             download=True)
```


모델 생성 과정

모델을 선언합니다

```
# 1번 레이어 : 합성곱층(Convolutional layer)
합성곱(in_channel = 1, out_channel = 32, kernel_size=3, stride=1, padding=1) + 활성화 함수 ReLU
맥스풀링(kernel_size=2, stride=2))

# 2번 레이어 : 합성곱층(Convolutional layer)
합성곱(in_channel = 32, out_channel = 64, kernel_size=3, stride=1, padding=1) + 활성화 함수 ReLU
맥스풀링(kernel_size=2, stride=2))

# 3번 레이어 : 합성곱층(Convolutional layer)
합성곱(in_channel = 64, out_channel = 128, kernel_size=3, stride=1, padding=1) + 활성화 함수 ReLU
맥스풀링(kernel_size=2, stride=2, padding=1))

# 4번 레이어 : 전결합층(Fully-Connected layer)
특성맵을 펼친다. # batch_size x 4 x 4 x 128 → batch_size x 2048
전결합층(뉴런 625개) + 활성화 함수 ReLU

# 5번 레이어 : 전결합층(Fully-Connected layer)
전결합층(뉴런 10개) + 활성화 함수 Softmax
```

모델 트레이닝과 검증

모델과 데이터셋을 GPU에 올립니다

```
# CNN 모델 정의  
model = CNN().to(device)
```

비용과 옵티마이저 함수를 정의합니다

```
criterion = torch.nn.CrossEntropyLoss().to(device) # 비용 함수에 소프트맥스 함수 포함되어 있음.  
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

모델 트레이닝과 검증

트레이닝에는
1분정도밖에 걸리지않았습니다..

```
[Epoch: 1] cost = 0.225836322
[Epoch: 2] cost = 0.0630451366
[Epoch: 3] cost = 0.0464125462
[Epoch: 4] cost = 0.0374096446
[Epoch: 5] cost = 0.0314301141
[Epoch: 6] cost = 0.0262634754
[Epoch: 7] cost = 0.0218371768
[Epoch: 8] cost = 0.0185317155
[Epoch: 9] cost = 0.0162785761
[Epoch: 10] cost = 0.0133264251
[Epoch: 11] cost = 0.00993826892
[Epoch: 12] cost = 0.0105726905
[Epoch: 13] cost = 0.008105997
[Epoch: 14] cost = 0.00686452957
[Epoch: 15] cost = 0.00774975354
```

모델을 로드해 숫자를 분류

파이썬과 OPENCV라이브러리를 이용해
모델을 불러와

테스트 이미지를 119개를 분류하고

결과 이미지를 저장하도록
프로그램을만들었습니다



결과 화면

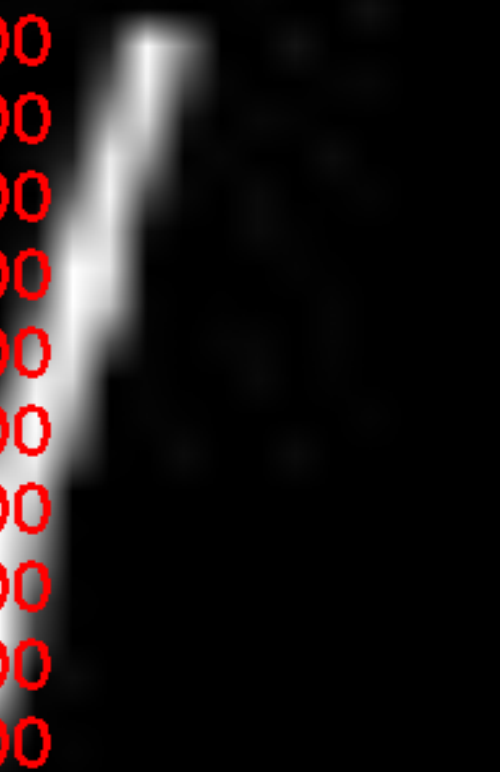
Predicted Label: 0

Class 0: 1.0000
Class 1: 0.0000
Class 2: 0.0000
Class 3: 0.0000
Class 4: 0.0000
Class 5: 0.0000
Class 6: 0.0000
Class 7: 0.0000
Class 8: 0.0000
Class 9: 0.0000



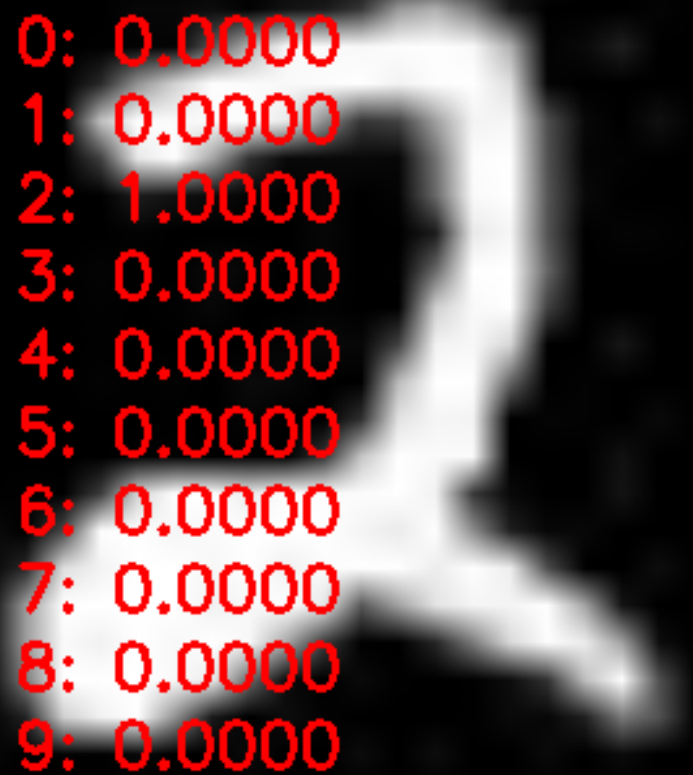
Predicted Label: 1

Class 0: 0.0000
Class 1: 1.0000
Class 2: 0.0000
Class 3: 0.0000
Class 4: 0.0000
Class 5: 0.0000
Class 6: 0.0000
Class 7: 0.0000
Class 8: 0.0000
Class 9: 0.0000



Predicted Label: 2

Class 0: 0.0000
Class 1: 0.0000
Class 2: 1.0000
Class 3: 0.0000
Class 4: 0.0000
Class 5: 0.0000
Class 6: 0.0000
Class 7: 0.0000
Class 8: 0.0000
Class 9: 0.0000



결과 화면


Predicted Label: 3

Class 0: 0.0000
Class 1: 0.0000
Class 2: 0.0000
Class 3: 1.0000
Class 4: 0.0000
Class 5: 0.0000
Class 6: 0.0000
Class 7: 0.0000
Class 8: 0.0000
Class 9: 0.0000




Predicted Label: 4

Class 0: 0.0000
Class 1: 0.0000
Class 2: 0.0000
Class 3: 0.0000
Class 4: 0.9767
Class 5: 0.0000
Class 6: 0.0000
Class 7: 0.0000
Class 8: 0.0233
Class 9: 0.0000

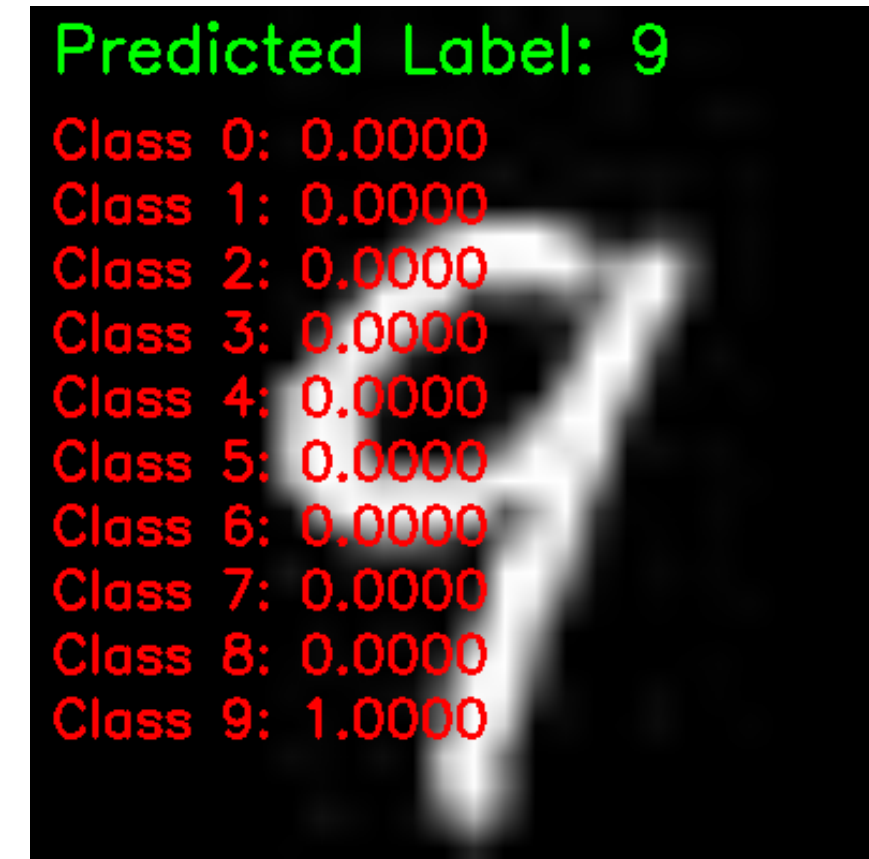
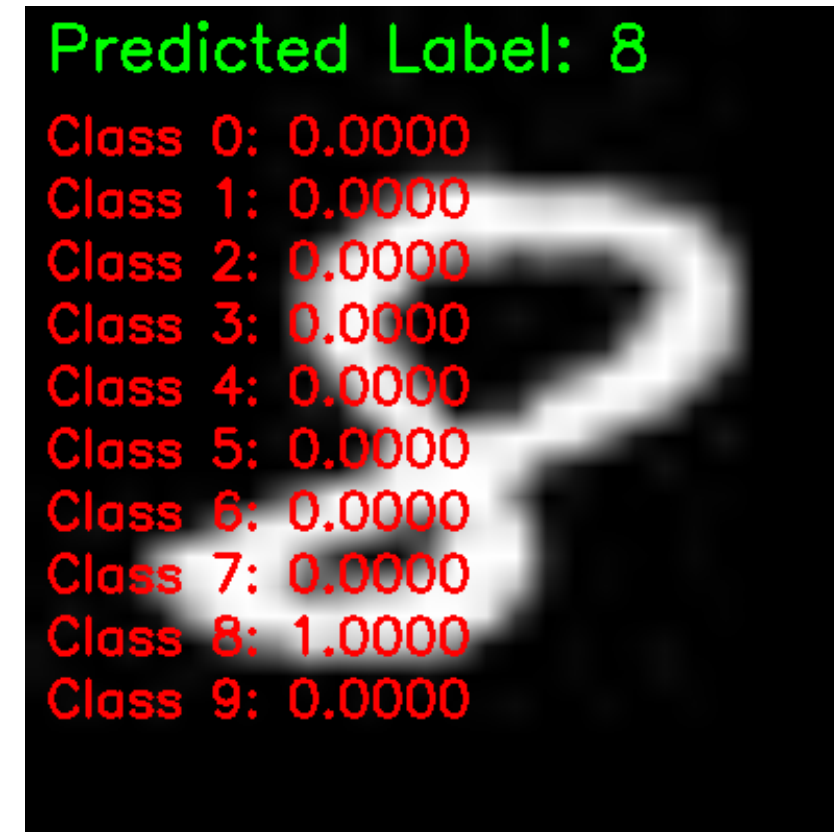
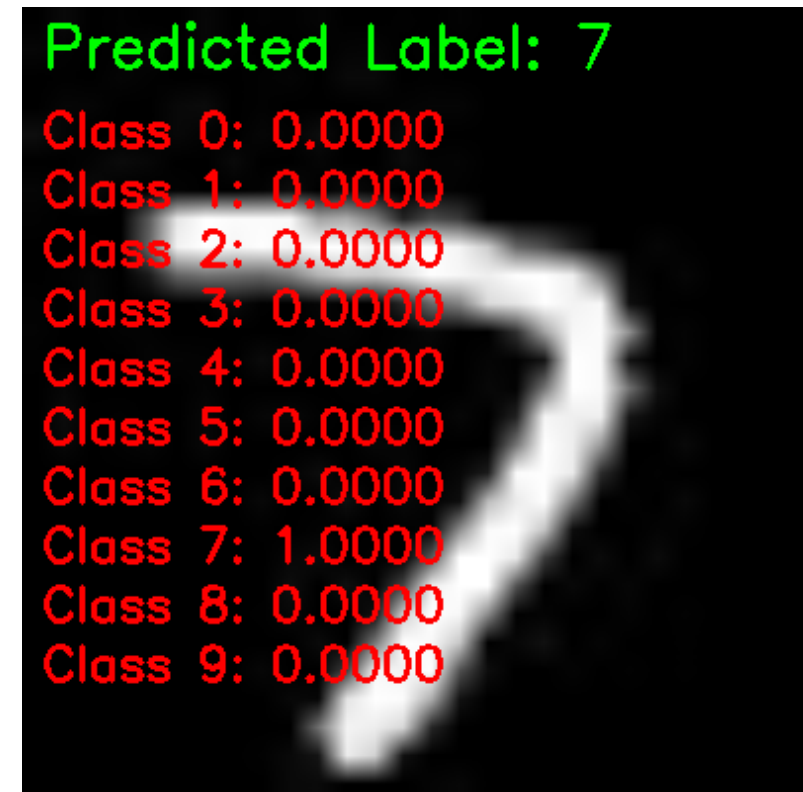
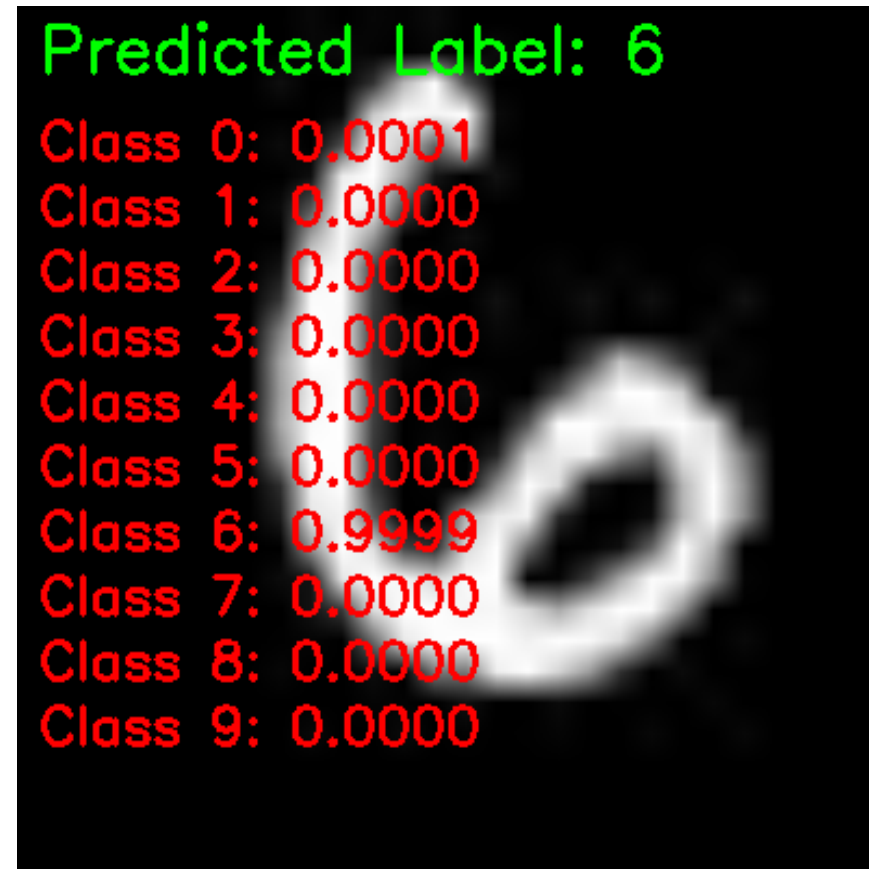


Predicted Label: 5

Class 0: 0.0000
Class 1: 0.0000
Class 2: 0.0000
Class 3: 0.0000
Class 4: 0.0000
Class 5: 1.0000
Class 6: 0.0000
Class 7: 0.0000
Class 8: 0.0000
Class 9: 0.0000



결과 화면



결과 화면

테스트 이미지 119개 중에
분류를 실패한 이미지가 하나도
없었습니다

결과 화면

테스트 이미지 119개 중에
분류를 실패한 이미지가 하나도
없었습니다

Predicted Label: 9



Class 0:	0.0000
Class 1:	0.0000
Class 2:	0.0000
Class 3:	0.0000
Class 4:	0.0100
Class 5:	0.0000
Class 6:	0.0000
Class 7:	0.0000
Class 8:	0.2069
Class 9:	0.7830

개선할점과 느낀점

숫자의 특징을 잡아
룰베이스로 구별해내는것은
거의 불가능에 가깝겠다고 구현중에 느꼈다..

이정도 정확성에 개선할점이있을까..
인지능력이 떨어진 사람보다
이 딥러닝 모델이 훨씬 더 잘 구별할것같다..

그외 질문

감사합니다

[HTTPS://GITHUB.COM/DESAFIN/MNIST_TEST.GIT](https://github.com/desafin/mnist_test.git)
