[7조 머신러닝 프로젝트 보고서]

"Hand-made MNIST dataset을 이용한 머신러닝 모델 최적화 및 분석"

팀장: 전자정보공학부(IT융합전공) 20201610 정하연

전자정보공학부(IT융합전공) 20201590 송준규

차세대반도체학과 20236079 김희균

컴퓨터학부 20192447 정승연

[Index]

I. 서론

1-1. 결과 보고서 요약

Hand-made dataset의 성능이 저하된 원인을 확인해 인식 성능을 개선 시킨 Machine learning 모델을 학습하여 최적화 시켰으며, Noised Data와 Shifted Data에도 내성이 있는 모델을 완성했습니다. 본 보고서는 해당 과정을 상세히 담고 있습니다.

1-2. 프로젝트 목표

프로젝트의 첫번째 목표는 Hand-made MNIST dataset과 original MNIST dataset을 이용하여 inference 결과를 비교하고 Hand-made dataset의 성능이 저하된 원인을 밝히고, Hand-made dataset을 포함한 학습데이터를 이용하여 인식 성능을 개선시킨 machine learning model을 학습시키고 최적화 하는 것입니다.

프로젝트의 두번째 목표는 최종 모델은 Noised Data, Shifted Data 등 학습이나 평가에 방해가 되는 데이터가 존재하더라도 학습과 검증이 잘 되는 내성이 있는 모델을 제작하는 것 입니다.

II. 본론

2-1. 데이터 구하기

2-1-1. 프로젝트에 필요한 데이터 정의

프로젝트에 필요한 dataset을 총 3개로 정의했습니다. 1. original MNIST dataset, 2. hand-made MNIST dataset, original MNIST와 hand-made MNIST를 결합한 3. combined dataset 이렇게 3가지로 정의했습니다. 위와 같은 dataset을 준비하기 위해서는 데이터를 구할 필요가 있었습니다.

2-1-2. Original / Hand-made 두가지 MNIST 데이터 구하기

Original MNIST의 경우, Scikit learn의 library를 호출 함으로 구할 수 있었고, hand-made의 경우, 프로젝트를 진행하는 모든 팀의 팀원이 손수 작성하여 제공해 준 데이터를 활용하였습니다. Combined의 경우, 앞선 두가지 데이터만 있으면 조합할 수 있다고 판단하여 다음 단계로 넘어갔습니다.

2-2. 데이터 분석

Original MNIST와 hand-made data를 사용하여 dataset을 구성하기에 앞서서 각각의 데이터는 어떤 특성을 가지고 있는지 확인했습니다. 우선 각각의 data들에 대해서 3가지 사항들을 확인했습니다. 1. 데이터 shape은 어떠한가? 2. 각 데이터가 가진 값은 어떤 값인가? 3. 잘못된 라벨이나 학습이 애매해 보이는 데이터는 없는가? 위의 3가지 기준을 통해 두 데이터를 분석했습니다.

2-2-1. Original data에 대한 분석

(1) 우선, original MNIST의 경우, image를 784 개의 list로 담고 있다는 것과, 이미지 및 라벨의

수가 70_000개라는 것을 확인 할 수 있었습니다. 아래 이미지는 Origianl MNIST에 대한 shape를 print한 결과 입니다.

```
key: dict_keys(['data', 'target', 'frame', 'categories', 'feature_names', 'target_names', 'DESCR', 'details', 'url'])
X_original.shape: (70000, 784)
y_original.shape: (70000,)
```

(2) 모든 값을 확인 해 볼 수는 없었지만, 특정 N개의 값을 모두 print 해보았을 때, list에 담겨 있는 값이 0.0~255.0 사이의 값을 저장하고 있음을 확인했습니다. 아래 이미지는 Original MNIST의 특정 index에 대한 plot과 저장된 값 중 최솟값과 최댓값을 print한 결과 입니다.

min: 0.0 max: 255.0 (라벨이 5인 경우)

(3) 데이터의 라벨링이 잘못 된 것은 없는가에 대한 분석은 "Official Library인 만큼 라벨링의 오차가 있더라고 그 수는 미비할 것이다" 라는 생각과 70_000개 라는 숫자를 모두 검증 할 수 있는 시간적 여유가 없었기 때문에 생략했습니다.

2-2-2. Hand-made data에 대한 분석

(1) Hand-made MNIST의 경우, npz 파일로 부터 load 했을 때 img, label 총 두가지 key를 가지고 있음을 확인했습니다. 각각 해당하는 변수를 선언해 대입한 후, shape를 print 해봤을 때, image를 1차원 list가 아니라 28*28 로 저장되어 있음을 확인했습니다.

```
digital_data_trval.keys():  KeysView(NpzFile 'Data/digit_data_TrVal-1.npz' with keys: img, label)
digit_data_test.keys() KeysView(NpzFile 'Data/digit_data_10_Test_1.npz' with keys: img, label)
op_data_trval.keys() KeysView(NpzFile 'Data/op_data_TrVal-1.npz' with keys: img, label)
op_data_test.keys() KeysView(NpzFile 'Data/op_data_10_Test_1.npz' with keys: img, label)

digit_trval_images.shape: (15119, 28, 28)
digit_trval_labels.shape: (15119,)
op_trval_images.shape: (15329, 28, 28)
op_trval_labels.shape: (15329,)
op_test_images.shape: (2190, 28, 28)
op_test_labels .shape: (2190,)
digit_test_images.shape: (2160, 28, 28)
digit_test_labels.shape: (2160,)
```

(2) 마찬가지로 모든 값을 확인 해 볼 수는 없었지만, 특정 N개의 값을 모두 print 해보았을 때, list에 담겨있는 값이 0.0~1.0 사이의 값을 저장하고 있음을 확인했습니다. 아래 이미지는 Hand-made MNIST의 특정 index에 대한 plot과 저장된 값 중 최솟값과 최댓값을 print한 결과 입니다.

min: 0.0 max: 1.0 (라벨이 7인 경우)

Original MNIST와 다르게 잘못된 라벨이 존재할 가능성이 있어, 모든 데이터를 plot해본 결과 상당 수의 라벨링이 잘못된 데이터와, 식별이 잘 안되는 이미지가 들어 있었습니다. 이에 육

안으로 라벨에 해당하는 문자로 보이지 않으면 해당 index를 삭제하는 Data Cleaning 작업이 필요했습니다.

2-2-3. 두 데이터 분석을 통해 얻은 사실에 의한 통찰

(1) 15개(0~9, +, -, x, /, =)클래스 이외의 데이터

각 데이터의 라벨 차이가 있었습니다. 아래는 Original MNIST와 Hand-made MNIST의 label을 중복이 되지 않게 출력한 결과 입니다. 이를 통해 알 수 있는 사실은 Original과 Hand-made를 결합할 때, Hand-made에 있는 기호가 Original에는 것과, 원래 의도 했던 0~9, +, -, x, /, =를 제외 하고 더 많은 잘못된 라벨이 들어 있다는 사실입니다.

```
y_original / np.unique : [0 1 2 3 4 5 6 7 8 9]

digit_trval_labels / np.unique:
 ['+' '-' '/' '0' '1' '2' '3' '4' '5' '6' '7' '8' '9' '=' 'x']
op_trval_labels / np.unique:
 ['%' '+' '+' '-' '/' '0' '1' '10' '11' '12' '13' '14' '2' '3' '4' '5' '6'
 '7' '8' '9' '=' 'X' 'x']
op_test_labels / np.unique:
 ['%' '+' '+' '-' '/' '0' '1' '10' '11' '12' '13' '14' '2' '3' '4' '5' '6'
 '7' '8' '9' '=' 'X' 'x']
digit_test_labels/ np.unique:
 ['+' '-' '/' '0' '1' '2' '3' '4' '5' '6' '7' '8' '9' '=' 'x']
```

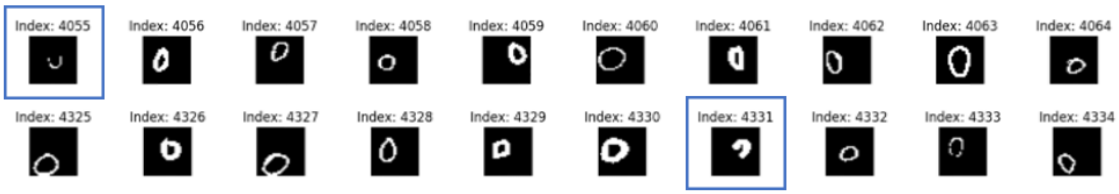
해당 사실들을 바탕으로 추후 Prepare Dataset 단계에서 의도 되지 않은 라벨은 제거할 계획을 수립하였습니다.

(2) 잘못된 라벨링

클래스 별 데이터의 특징을 알아보기 위해 15개의 클래스 순서대로 데이터를 나열했습니다. 0으로 라벨링된 데이터, 1로 라벨링된 데이터, 2로 라벨링된 데이터, ... 와 같은 순서대로 나열한 것입니다.

위는 0으로 라벨링된 데이터 10개를 임의로 plot한 결과입니다. 3216, 3220, 3222 데이터는 육안으로 보면 각각 1, 5, 3에 해당하는 데이터인데, 0으로 잘못 라벨링 되었습니다.

(3) 학습에 애매한 데이터



위는 0으로 라벨링된 또 다른 10개의 데이터를 plot한 결과입니다. 4055 데이터는 0의 위쪽 부분이 잘려 숫자 0의 형태라고 보기 어렵습니다. 4331 데이터는, 4330 데이터와 같은 굵기와

크기를 가져서 0의 왼쪽 아래부분이 잘렸다고 볼 수 있지만, 클래스 7로도 분류될 수 있는 이미지입니다. 이와 같이 라벨링은 제대로 되었지만 심하게 잘린 데이터, 혹은 다른 클래스와 유사하여 혼동을 유발할 수 있는 데이터가 존재함을 알게 되었습니다.

(4) Handmade dataset 데이터 수

약 60_000개의 Original MNIST dataset과 달리, Handmade MNIST dataset은 약 30_000개만 존재함을 알게되었습니다. 추후 handmade dataset 전처리 과정을 거친 후, original MNIST dataset를 일부 추가하여 데이터 개수에 보완이 필요함을 알게 되었습니다.

(5) 픽셀값(feature 값)의 차이

Original MNIST dataset의 픽셀값은 0에서 255 사이의 값이고, Handmade MNIST dataset의 픽셀 값은 0과 1 사이의 값이었습니다. Handmade 와 Original MNIST dataset을 통합한 dataset을 학습하려면, 픽셀 값을 통일하는 (스케일링) 과정이 필요함을 알게 되었습니다.

(6) Noise가 있는 데이터, Shift된 데이터, 테두리가 남은 데이터

현재 가지고 있는 테스트 데이터셋 이외의 어떠한 데이터가 들어와도 강력한 모델을 만들기 위해서는, 노이즈가 있는 학습 데이터셋을 구축하는 작업이 필요함을 인지하게 되었습니다. 이를 위해서는 노이즈로 분류되는 threshold를 정의해야 하고, 모델이나 함수 중 어떤 방식으로 노이즈를 처리할 지 결정하는 과정이 필요했습니다. 또한, Index 519와 같이 handmade dataset을 작성할 때 썼던 템플릿의 테두리가 포함된 데이터가 존재함을 알게 되었고, 학습에 방해가 될 수 있다고 판단했습니다.

index 520, 522와 같이 숫자가 중앙에서 벗어나 상하좌우로 치우친 데이터도 존재했습니다. 데이터의 일관성이 떨어지므로 중앙화를 하거나 상하좌우로 치우친 데이터를 추가로 제작하는 과정이 필요하겠다는 생각을 하게 되었습니다. 그리고 최종적으로 test dataset에도 관련 데이터를 구축하여 검증할 필요가 있다고 판단하였습니다.

(7) 비슷한 숫자 및 기호가 존재

Index 3888과 같이 0이지만 자칫 6으로 판별될 수도 있는 애매한 데이터가 존재하기에, confusion matrix 분석을 통해 가장 많이 혼동되는 라벨을 파악할 필요가 있다고 생각했습니다. Index 3866과 같이 기울어진 이미지도 변수로 두어 기울어진 정도를 달리한 데이터셋을 구축하는 아이디어도 생각해 보았지만, 기울기라는 변수는 1과 /를 식별하는 중요한 feature를 방해할 수 있다고 판단해 이후 전처리에서는 다루지 않기로 결정했습니다.

(8) 데이터의 크기

현재 Original, Handmade MNIST dataset 모두 28*28 크기인데, 다른 크기를 입력으로 받아도 처리가능한 모델을 만들 수 있겠다는 아이디어 또한 도출되었습니다.

2-3. Dataset 구성 및 테스트 모델 학습

앞서 프로젝트에 필요한 데이터 셋을 1. original MNIST dataset, 2. hand-made MNIST dataset, original MNIST와 hand-made MNIST를 결합한 3. combined dataset 이렇게 3가지로 정의했습니다. 이를 위해 모은 Data를 결합하여 총 4개의 Dataset을 구성했습니다.

1. Original MNIST dataset은 학습 데이터를 80%, 검증 데이터를 20%로 나누었습니다.

```
print(o_X_train.shape, o_X_test.shape, o_y_train.shape, o_y_test.shape)
```

```
(56000, 784) (14000, 784) (56000,) (14000,)
```

2. Handmade MNIST dataset은 아래 그림과 같이 구축했습니다. 우선 29290, 4212개의 TrVal set과 Test set을 제공 받았습니다. 하지만, 이전에 발견했던 **문제점 2-2(2), 2-2(3)** 과 같이 육안으로 확인하여 라벨링이 잘못된 데이터나 학습이 애매한 데이터를 삭제한 결과는 각각 27389, 4003개 였습니다. 해당 데이터를 가지고 중간단계의 1. Handmade MNIST(10 Classes), 2. Handmade MNIST(15 Classes) 총 두가지 dataset을 제작했습니다. 먼저 class 15개 먼저 구성을 하고, 기호를 제거한 class 10 dataset을 제작했습니다. 그 이유는 라벨링을 제거하면서 삭제를 진행했는데 class 10을 먼저 진행하게 되면 기호에 해당하는 index 만큼이 밀려서 결국 또 잘못된 라벨의 index를 검사해야 했기 때문입니다.

```
삭제 후 Train 데이터, 최종 Test 데이터 15 classes:
perfect_X_train shape: (27389, 784)
perfect_y_train shape: (27389,)
perfect_X_test shape: (4003, 784)
perfect_y_test shape: (4003,)

Validation Set으로 만든 Train/Test 데이터 15 classes:
X_train_15 shape: (21911, 784)
y_train_15 shape: (21911,)
X_val_15 shape: (5478, 784)
y_val_15 shape: (5478,)
```

```
class 15 -> class 10:
train set 지워진 데이터 개수: 11110
val set 지워진 데이터 개수: 2723

Validation Set으로 만든 Train/Test 데이터 10 classes:
X_train_10 shape: (10801, 784)
y_train_10 shape: (10801,)
X_val_10 shape: (2755, 784)
y_val_10 shape: (2755,)
```

위의 작업을 거쳐 최종적으로 10 class handmade(X_10), 15 class handmade(X_15) 데이터를 얻을 수 있었습니다. 하지만 15 class는 최종 모델에 Combined MNIST를 위한 준비 과정이며, 데이터 클리닝을 제거한 Test 또한 동일 합니다. Handmade MNIST dataset을 활용될 dataset은 (X_10) 에 해당하는 dataset 입니다.

3. Combined MNIST dataset(class 10)과 4. Combined MNIST(class 15)는 Handmade MNIST에서 라벨별로 모자란 개수를 세어, 총 60000, 10000개 인 Original MNIST와 비율을 맞추기 위해, 모자란 개수 만큼을 빼서 부족한 만큼을 MNIST에서 받아와 채워넣었습니다.
handmade(X_10)과 handmade(X_15)에 대해서 각각 라벨 수가 10개 15개 이기에, Train set은 라벨당 (10: 6000개, 15: 4000개), Validation set은 각각 (10: 1000개, 15: 660개) 만큼을 Original MNIST에서 받아와 구성했습니다.

```
60_000, 10_000개로 Combine된 c_X_train_10, c_y_val_10:
c_X_train_10 shape: (60000, 784)
c_y_train_10 shape: (60000,)
c_X_val_10 shape: (10000, 784)
c_y_val_10 shape: (10000,)
```

```
60_000, 10_000개로 Combine된 c_X_train_15, c_y_val_15:
c_X_train_15 shape: (59612, 784)
c_y_train_15 shape: (59612,)
c_X_val_15 shape: (9900, 784)
c_y_val_15 shape: (9900,)
```

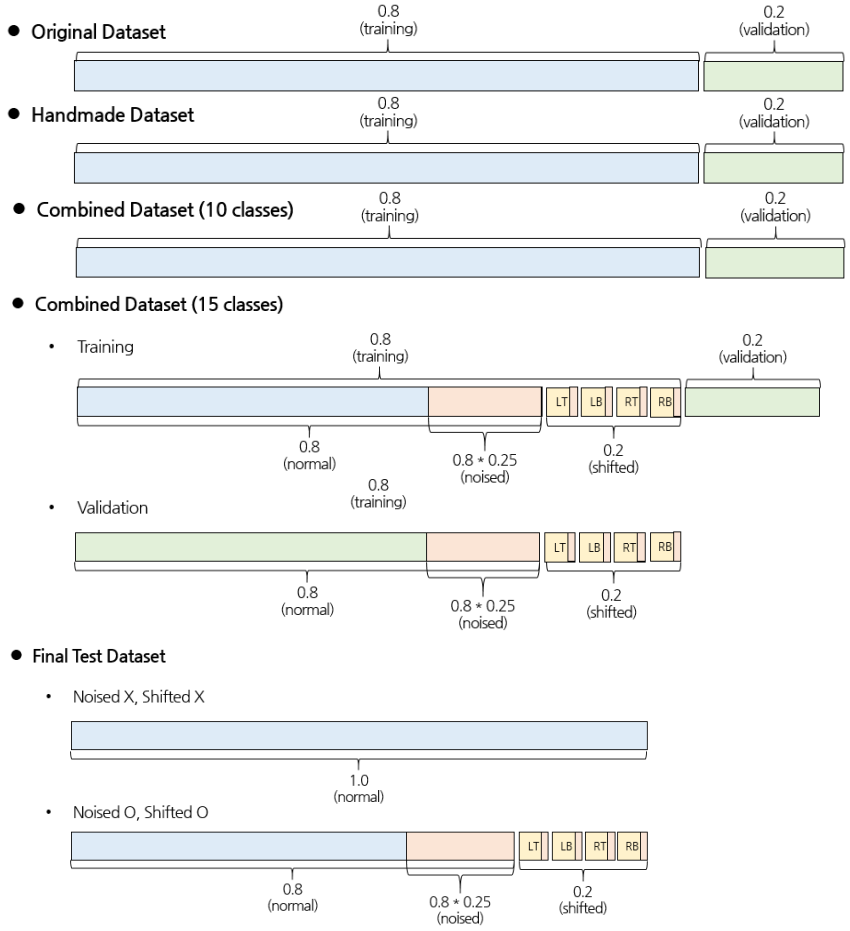
c_X_train_15의 개수가 딱 60000이 아닌 이유는, 'x' 기호의 데이터가 모자르기 때문입니다.

```
Label x: 1806 instances
Label 0: 1091 instances
Label =: 2289 instances
Label +: 2403 instances
Label -: 2353 instances
```

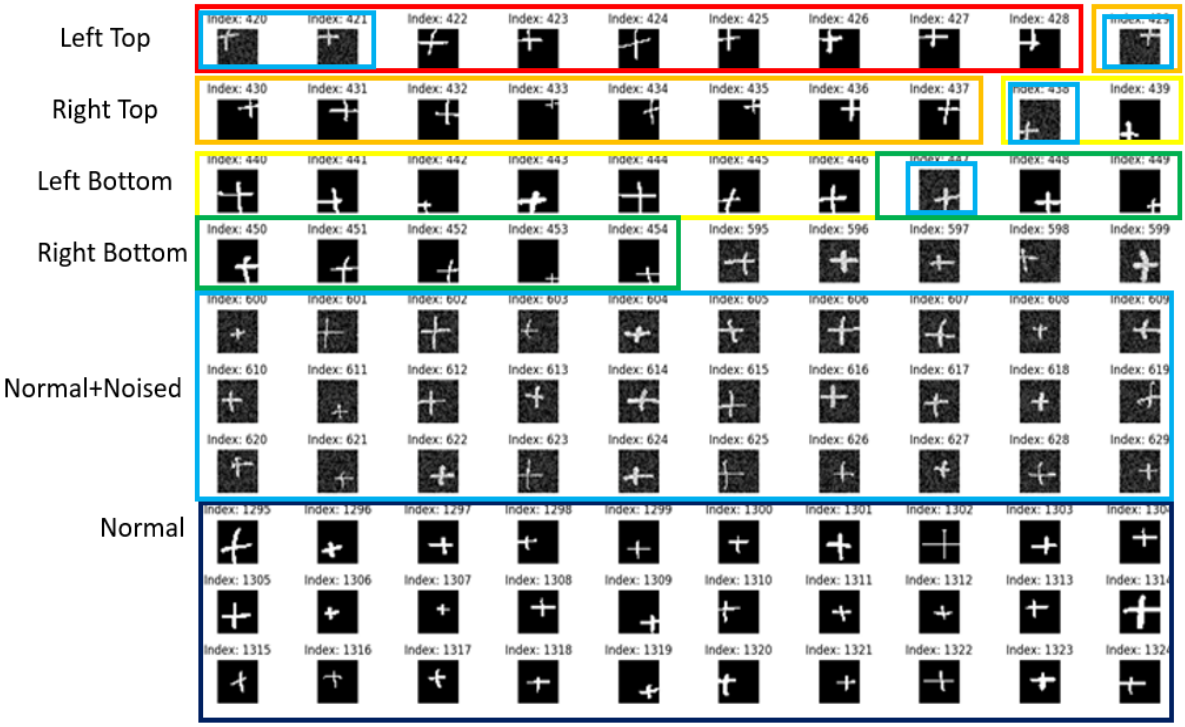
Original MNIST에는 기호가 존재하지 않아, 기호의 경우 이미 있는 데이터를 2배를 하되, 변형을 줄 생각을 하고 추가하였습니다. 나머지의 경우 2배를 하면 4000을 넘기지만 x의 경우 그렇지 않기 때문에, c_X_train_15의 경우 60000이 되지 않습니다. 위와 같이 총 4가지 데이터 셋을 구성했습니다. 데이터를 구성할 때 생긴 문제는 아니지만, 추후에 Noise와 Shift 된 데이터를 pipeline으로 해결하는 과정이 있는데, 이를 Test하기 위해 Combined MNIST(class 15)와 최종 Test의 복사본에 대해서 비율을 정하고, Noised와 Shifted를 적용한 Dataset을 구성했습니다.

중앙화 및 최대화 작업이 제대로 수행되는지 수치적으로 명확하게 확인하기 위해서, 모든 데이터를 중앙화 시킨 후 전체 데이터 중 20%는 shift 시켰습니다. Left Top, Left Bottom, Right Top, Right Bottom으로 숫자나 기호를 이동시켰습니다. 그리고 training과 validation dataset 각각의 25%에 해당하는 데이터에 노이즈를 추가했습니다.

언급했던 Combinded MNIST(class 15)와 최종 Test의 복사본의 경우 아래 그림과 같은 비율로 구성되어 있습니다. 아래는 최종 데이터셋을 시각화 한 그림 입니다.



모든 데이터 셋을 보고서에 담을 수는 없어, Noised, Shifted Final Test dataset의 '+' 라벨의 데이터 일부를 발췌해 이미지를 아래에 첨부했습니다. 아래와 같이 Normal, Normal+Noised, Shifted, Shifted+Noise의 모든 경우가 잘 저장된 것을 확인 할 수 있습니다. 아래 그림에서 하늘색으로 감싸진 부분은 Noised 데이터 입니다.



노이즈를 추가는 다음과 같이 진행했습니다. 우선 노이즈에 대한 정의가 필요했는데, 표기를 했냐 안했냐의 기준을 0.5로 두고, 0.5 이하의 값은 표기가 아니다. 그럼 0.5 이하는 노이즈다 이런식의 사고로 노이즈에 대한 정의를 나눌 수 있었습니다. 하지만, 현재 작업하는 과정 중에는 모든 픽셀이 0~1 사이의 값을 가진다고 장담 할 수 없었습니다.

```
max_vals = np.max(X, axis=1, keepdims=True)
noise = np.random.uniform(0, max_vals * (1/2), size=X.shape)
```

따라서 위와 같이, 1. 각 이미지(한개의 행) 안에서 최대 값을 찾습니다. 그리고 모든 값을 최대 값의 1/2에 해당하는 만큼을 노이즈로 정의 했습니다. 예를 들어 픽셀이 0에서 1 사이의 값을 가진다고 하면, 0과 0.5 사이의 랜덤 값을 784개 픽셀이 각각 더했습니다. 0에서 255사이인 경우는 128까지의 랜덤 값을 사용했습니다. 그래서 향후 파이프라인의 denoising 과정에서는 픽셀 값이 나올 수 있는 값의 범위는 (표기상 최댓값)으로 나눈 값 의 최댓값: 0.66, 최솟값 1.0 따라서 정리하자면, "한 이미지(한 개의 행)안에서 최댓 값으로 나눈 값이 0.66 이상인 값만을 1로 나머지는 0으로 설정한다." 가 우리의 denoising의 아이디어 였습니다. 이는 정규화의 효과가 있을 뿐만 아니라, 표기 값이 0~1 이던, 0~255던, 0~1024던 값에 구애 받지 않고 사용할 수 있는 pipeline을 구성할 핵심이 되었습니다. 0.66인 경우가 가장 strict한 경우 여서 treshold로 사용했으나, test를 하다보니 0.5를 사용할 때가 score가 가장 높음을 확인했습니다. 이는 주변 noise를 흡수하여 feature를 더 강화 했기 때문이라고 판단했습니다.

2-4. 데이터 전처리 및 파이프라인 구축

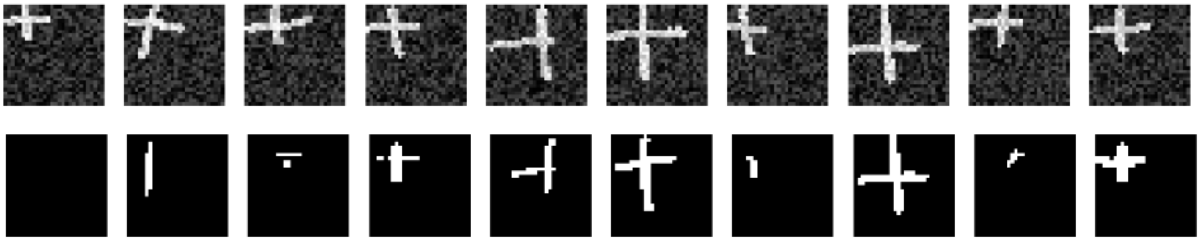
앞서 서술한 데이터에 대한 통찰의 결과로, 파이프라인을 크게 5단계로 구축했습니다.

첫 번째는 Denoise 과정입니다. 모델의 테스트 결과에서 노이즈가 심한 데이터가 들어올 수 있으며, 저희는 이에 강건한 모델을 만들기 위하여 노이즈 제거 함수를 만들어 파이프라인 첫 번째에 넣었습니다.

Denoise는 시행착오를 겪게 되었는데 해당 부분은 "노이즈를 추가한 이미지와 노이지를 추가하지 않은 이미지를 각각 학습 시켜 모델로 Denoise를 진행하자"는 아이디어를 수행하는 부분에서 발생했습니다. 우선 60000개의 모든 이미지에 노이즈를 추가하여 1:1로 KNN 모델을 학습시켰습니다. 그 후, 예측으로 이미지를 출력하게 끔 모델을 구성했는데, 모두 메모리에 올려 작업을 진행하는 특성상 out of memory 문제가 발생했었습니다.

이름	상태	26% CPU	99% 메모리	5% 디스크	0% 네트워크	1% GPU	GPU 엔진	전력 사용량	전력 사용량 ...
Python		11.7%	6,197.8MB	0MB/s	0Mbps	0%		매우 높음	낮음
Google Chrome(14)		0.1%	115.3MB	2.2MB/s	0Mbps	0%	GPU 1 - ...	매우 낮음	매우 낮음

또한 해당 모델로 학습을 시킨 모델에 Test 데이터를 넣어 보았을 때, 아래 와 같이 원하는 의도대로 노이즈가 제거가 잘 되지 않았습니다. 당연하게도 Score도 0.2로 매우 낮은 점수를 냅니다. 아래 그림은 순서대로 노이즈를 추가한 Test 데이터와 예측이 끝난 이미지 입니다.



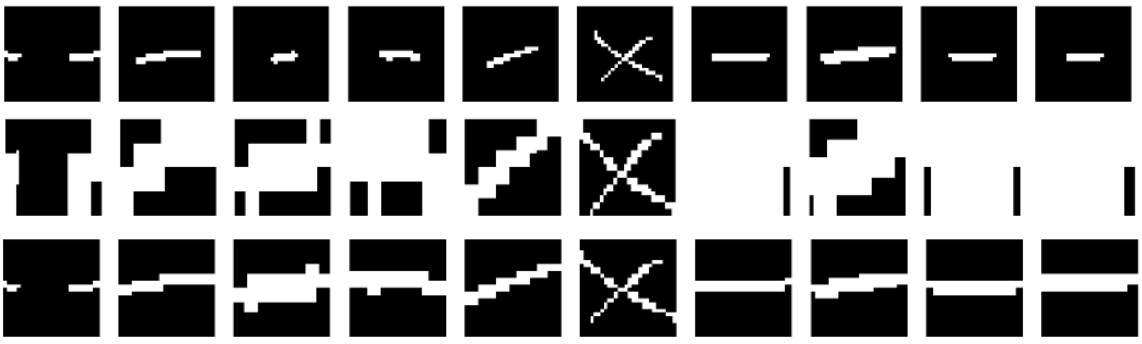
이를 통해 우리는 모델 보다는 함수를 통해서 파이프 라인을 구성해야 한다는 결론을 내릴 수 있었고, 나머지 Shited 및 테두리가 존재하는 데이터에도 모델을 사용하는 것 보다는 최대한 함수를 통해서 해결하자는 결론을 내릴 수 있었습니다.

이에 함수로 Denoise를 구성했습니다. 앞서 언급 되었던 임계값을 0.5를 사용하여 픽셀 값이 0.5~1.0 이상의 값만 의미있는 값으로 판별되도록 denoise_with_max 함수를 정의했습니다. 해당 파이프 라인을 통과 하게 되면 어떠한 값으로 표기가 되어 있었던 간에 0 or 1의 값으로 feature가 저장되는 정규화 효과 또한 제공합니다. 아래 사진은 순서대로 before Denoise, after Denoise에 대한 이미지 입니다.

두 번째는 Image Centering (중앙화) 과정입니다. 처음에는 shift된 데이터를 다루기 위해 상하좌우로 치우쳐진 데이터를 일정 비율 생성하고 이 데이터를 넣어 학습시키려는 계획이었습니다. 교수님께서 상하좌우로 치우친 데이터는 큰 의미가 없다는 조언을 해 주셨고, 360 도로 100 개

정도 다른 위치에 존재하는 데이터를 구축하지 않는 이상 파이프라인에 중앙화 과정을 넣는 것이 효과적임을 알게 되었습니다. 그래서 아래 어떤 데이터가 들어오든 숫자나 기호를 중앙에 위치시킬 수 있는 과정을 모델이 아닌 함수를 통해 구축하였습니다. 하지만 중앙화의 경우 값을 boundary box로 읽어 가운데 픽셀에 위치하게 끔 코드를 작성했는데, 대부분의 경우는 잘 작동을 하지만, 테두리가 남아있는 데이터나 굵은 노이즈가 있는 경우, 테두리나 노이즈 또한 boundary box안에 포함시켜 가운대로 이동해 오히려 shift 되는 경우도 발생했습니다. 이에 테두리와 강한 노이즈를 제거하려는 노력을 했으나, 이는 영상처리와 관련된 분야이기에, 프로젝트의 진행 방향과는 거리가 있다고 판단해 더 이상 진행하지 않았습니다. 아래는 정상인 경우와 테두리/노이즈가 포함된 경우의 이미지 입니다. 순서대로 정상 / 비정상 before centering, after centering에 대한 이미지 입니다.

세 번째는 Image Enlarging (최대화) 과정입니다. 기호를 중앙에 두어도 데이터 작성자에 따라 숫자나 기호의 크기가 매우 상이했습니다. 그래서 저희는 조금 더 명확하게 숫자나 기호를 인식하기 위해 이미지를 최대화하는 과정을 추가했습니다. 최대화의 경우 1. Tight boundary box, 2. Square boundary box 총 두가지 아이디어를 사용했습니다. Tight boundary box는 값이 적혀있는 부분을 최대한 Tight한 직사각형으로 잘라 확대하는 것입니다. Square boundary box는 해당 값들을 포함 할 수 있는 가장 작은 Square를 만들어 확대하는 것 입니다. 두가지 확대의 경우, score 향상이 있어 어떤 것을 사용하던 비슷했습니다. 추후 결론에 자세히 다루지만, 모델 별로 score가 높은 최대화가 달랐습니다. Extra trees의 경우 Tight boundary가 더 좋았고, MLP를 포함한 대다수의 모델들이 Square boundary가 더 좋았습니다. 두 확대 과정은 '-'에서 가장 큰 차이를 보이는데, 아래 이미지를 보면 확인이 가능합니다. 순서대로 원본, Tight, Square 순입니다.



네 번째는 Normalize (정규화) 과정입니다. 앞서 언급했듯 Original Dataset의 픽셀 값은 0~255 사이이고, Handmade Dataset의 픽셀 값은 0 또는 1입니다. 두 데이터를 합쳐 Combined Dataset을 만드는 과정에서 어떤 픽셀 범위가 섞여서 들어오든 이에 강건한 모델을 만들어야겠다는 생각이 들어 정규화 과정을 추가하게 되었습니다. Denoise 과정에서 1차 적으로 정규화가 살짝은 이뤄지지만, Denoise 작업을 하고 나서 정규화를 진행을 하면 더 score가 좋아지는지 안좋아 지는지

확인하기 위해 어떤 정규화 방식을 사용할 지 결정하기 위해 KNN, Extra-trees, MLP 모델에 다양한 정규화 방식을 적용하여 Accuracy를 측정해보았습니다.

```
--- No scaling ---
Training time: 0.0281 seconds
Testing time: 0.8710 seconds
Accuracy: 0.8394
----------------------------------------
--- StandardScaler ---
Training time: 0.2734 seconds
Testing time: 1.0164 seconds
Accuracy: 0.8026
----------------------------------------
--- RobustScaler ---
Training time: 0.5483 seconds
Testing time: 0.9632 seconds
Accuracy: 0.8394
----------------------------------------
--- MinMaxScaler ---
Training time: 0.1712 seconds
Testing time: 0.9132 seconds
Accuracy: 0.8394
----------------------------------------
--- Normalizer ---
Training time: 0.1604 seconds
Testing time: 1.0511 seconds
Accuracy: 0.8878
----------------------------------------
```

```
--- StandardScaler ---
Training time: 33.1587 // Accuracy: 0.856869
--- RobustScaler ---
Training time: 33.3872 // Accuracy: 0.857273
--- MinMaxScaler ---
Training time: 32.6020 // Accuracy: 0.852828
--- Normalizer ---
Training time: 32.3047 // Accuracy: 0.853434
```

```
--- StandardScaler ---
Training time: 52.8641 // Accuracy: 0.822828
--- RobustScaler ---
Training time: 74.4527 // Accuracy: 0.845253
--- MinMaxScaler ---
Training time: 74.9011 // Accuracy: 0.843333
--- Normalizer ---
Training time: 125.7301 // Accuracy: 0.848788
```

KNN, Extra Trees, MLP 순서 입니다. KNN과 MLP의 경우 Normalizer()를 사용할 때 가장 정확도가 0.05 이상 높아졌기에 Normilizer(l2)의 사용이 명백했지만, Extra-trees의 경우, 값이 크게 변하지 않아 Pipeline의 통일 성을 위해 Normalizer(l2)를 모든 Pipeline에 추가하게 되었습니다.

추후에 최종 모델을 어떻게 선정했는지에 대한 자세히 서술합니다. 최종 선정된 MLP와 Extra trees에 어떤 Pipeline을 적용하면 좋은지, Default hyper paremeter 상황에서 어떤 Combination이 가장 좋은지 확인해 보았습니다.

아래 그림은 순서대로 Extra-trees. MLP에 대한 pipeline combination 별 score를 출력한 결과입니다. Extra trees의 경우, Image_centering과 tight_image_enlargning을 했을 때 가장 score가 높았고, MLP의 경우, Image_centering과 square_image_enlargning을 했을 때 가장 score가 높았습니다. 이에, 추후 grid search에서 각 모델은 선정된 transformer로 이루어진 pipeline을 가지고 학습을 하게 됩니다.

```
Transformers: image_centering // Training time: 37.5780 // Accuracy: 0.916566
Transformers: tight_image_enlarging // Training time: 44.2627 // Accuracy: 0.930505
Transformers: square_image_enlarging // Training time: 44.3579 // Accuracy: 0.923838
Transformers: image_centering, tight_image_enlarging // Training time: 46.5714 // Accuracy: 0.930606
Transformers: image_centering, square_image_enlarging // Training time: 46.1861 // Accuracy: 0.923737
```

```
Transformers: image_centering // Training time: 162.0554 // Accuracy: 0.914040
Transformers: tight_image_enlarging // Training time: 161.9201 // Accuracy: 0.916465
Transformers: square_image_enlarging // Training time: 146.7966 // Accuracy: 0.923535
Transformers: image_centering, tight_image_enlarging // Training time: 158.4643 // Accuracy: 0.921010
Transformers: image_centering, square_image_enlarging // Training time: 130.9702 // Accuracy: 0.925960
```

위의 결과를 가지고 아래의 Pipeline 구성 코드를 모델별로 다르게 채웠습니다.

```
Pipeline([
    ('denoise', FunctionTransformer(denoise_with_max)),
    ('image_centering', ImageCenteringTransformer()),
    ('image_enlarging', ImageEnlargingTransformer()),
    ('normalizer', Normalizer()),
    ('학습할 모델', 학습할 모델())
])
```

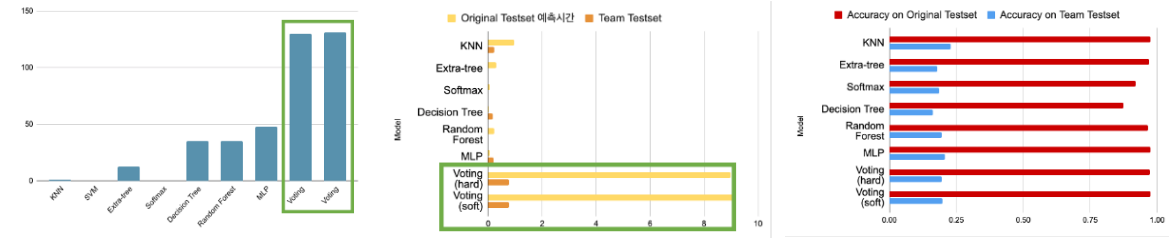
2-5. 최종 모델 선정 및 모델 학습 진행

초기에 시도한 모델은 총 8개입니다. KNN, SVM, Extra-Tree, Softmax, Decision Tree, Random Forest, MLP, 그리고 이 모델들을 합한 VotingClassifier(soft, hard)입니다.

2-5-1. Original 데이터 결과

앞서 구축한 Original Training Set을 각 모델에 학습시켰습니다. 모델을 선정하는 과정이기 때문에 파라미터 조절 없이 기본 조건으로 진행하였으며 결과는 다음과 같습니다.

Model	KNN	SVM	Extra-tree	Softmax	Decision Tree	Random Forest	MLP	Voting (hard)	Voting (soft)
Training Time (sec)	0.9169		12.76156	0.05638	35.14291	35.14291	47.88176	~~129.84410~~	~~131.20013~~
Original Testset 예측시간	0.97564		0.29157	0.05638	0.02232	0.22734	0.05838	~~8.99141~~	~~9.08884~~
Accuracy on Original Testset	0.97564		0.97000	0.91979	0.87364	0.96636	0.97400	0.97336	0.97479
Team Testset 예측시간	0.23000		0.01019	0.00128	0.16250	0.00857	0.20750	0.77231	0.76504
Accuracy on Team Testset	0.23000		0.18000	0.18500	0.16250	0.19500	0.20750	0.19500	0.19750



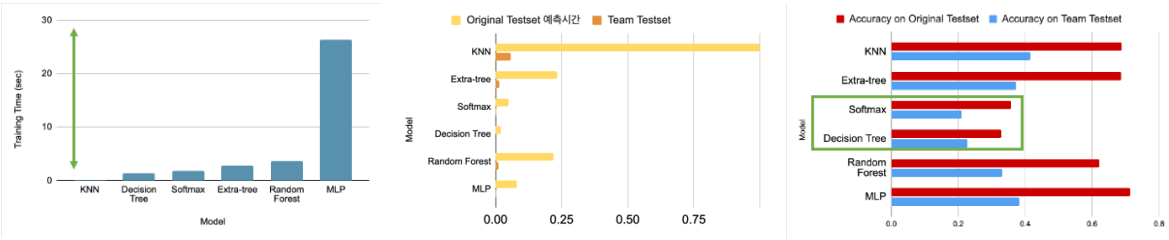
왼쪽 그래프는 Training Time, 중간은 Testset 별 Inference Time, 마지막은 Testset 별 Accuracy 입니다. (SVM을 제외한) 6개 모델을 조합한 Voting Classifier는 Training Time과 Inference Time 이 타 모델보다 확연히 오래 걸리기 때문에, 앞으로 사용하지 않기로 했습니다. 혹여나 정확도 개선을 위해 사용하더라도 많은 모델을 삽입하지 않기로 결정했습니다.

Original Training Dataset으로 학습시켰기 때문에 각 모델에 Original Testset를 넣었을 때 정확도는 90% 이상으로 높았으며, 저희 팀이 작성한 400개의 Team Testset을 넣었을 때는 정확도가 20%로 거의 예측이 잘 안되는 결과를 확인할 수 있었습니다. Team Testset은 치우친 데이터도 많고, 숫자의 크기를 다양하게 설정한 것을 주요 원인을 추론했습니다. 또한 픽셀 값 (feature 값)의 차이가 큰 영향을 미쳤을 것으로 분석됩니다.

2-5-2. Handmade 데이터 학습 결과

조금 더 다양한 데이터에 강건한 모델을 만들기 위해 어떤 모델이 좋을까 판별하고자 Handmade Training Set을 SVM과 VotingClassifier을 제외한 모델들에 학습시켜보았습니다.

Model	KNN	SVM	Extra-tree	Softmax	Decision Tree	Random Forest	MLP	Voting (hard)	Voting (soft)
Training Time (sec)	0.09063		2.82671	1.77935	1.44450	3.69965	26.31937		
Original Testset 예측시간	3.95527		0.23225	0.04901	0.02034	0.21984	0.08072		
Accuracy on Original Testset	0.68907		0.68571	~~0.35757~~	~~0.32886~~	0.62143	0.71271		
Team Testset 예측시간	0.05734		0.01308	0.00144	0.00124	0.01171	0.00197		
Accuracy on Team Testset	0.41500		0.37250	~~0.21000~~	~~0.22750~~	0.33000	0.38250		



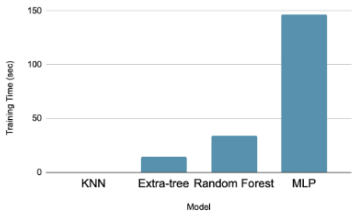
Training Time 그래프를 보면 MLP가 오래 걸린다는 사실을 알 수 있는데, 그래프 y축의 스케일이 이전 그래프보다 크지 않고 예측시간이 크지 않으며, 가장 좋은 정확도를 가지므로 계속 사용하기로 했습니다. 반면 Softmax와 Extra-Tree는 정확도가 현저히 낮아 앞으로 사용하지 않기로 결정했습니다.

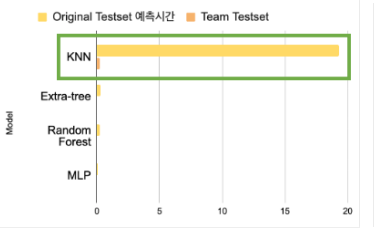
Original Testset의 정확도는 약 40~70%로 예측이 잘 되지 않았습니다. 모델이 학습한 데이터와 shift된 분포도, 노이즈 여부, 픽셀값 등에 의해 다른 점이 많아 이전보다 정확도가 훨씬 떨어졌을 것으로 분석됩니다. Team Testset에 대해서는 전반적으로 Original Training Dataset으로 학습한 모델들보다 정확도가 높았습니다. Team Testset과 Handmade Training Dataset의 작성자가 일부 겹치고, 데이터 제작 환경이 유사하기에 조금 더 높은 정확도가 나왔으리라 추측됩니다.

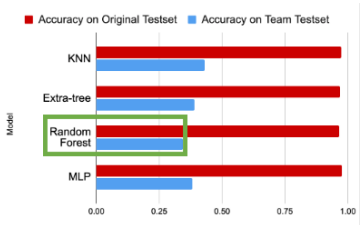
2-5-3. Combined (class 10) 데이터 학습 결과

최종적으로 Combined Dataset으로 학습한 모델을 만드는 것이 목표이기 때문에, Combined 10 Classes Training Dataset으로 모델들을 학습시켜 성능을 확인한 후, 최종 모델을 결정하는 과정입니다. 학습 결과 및 성능은 아래와 같습니다.

Model	KNN	SVM	Extra-tree	Softmax	Decision Tree	Random Forest	MLP	Voting (hard)	Voting (soft)
Training Time (sec)	0.29248		14.58727			33.62962	146.48267		
Original Testset 예측시간	~~19.26764~~		0.31900			0.24504	0.0561		
Accuracy on Original Testset	0.97514		0.96914			0.96543	0.9764		
Team Testset 예측시간	0.24254		0.01468			0.01233	0.00418		
Accuracy on Team Testset	0.43000		0.39000			~~0.34500~~	0.38250		







KNN의 예측시간이 이전 테스트 결과와 달리 타 모델과 몇 십 배 이상 차이를 보여 때문에 KNN 모델은 더이상 사용하지 않기로 했습니다. 그리고 Random Forest는 Training Time도 비교적 높은데 정확도도 낮았기 때문에 더 이상 사용하지 않기로 결정했습니다. 최종적으로 Extra-Tree와 MLP를 선택했고, 두 모델을 파인 튜닝하기로 결정하였습니다.

2-6. 최종 모델 파인 튜닝

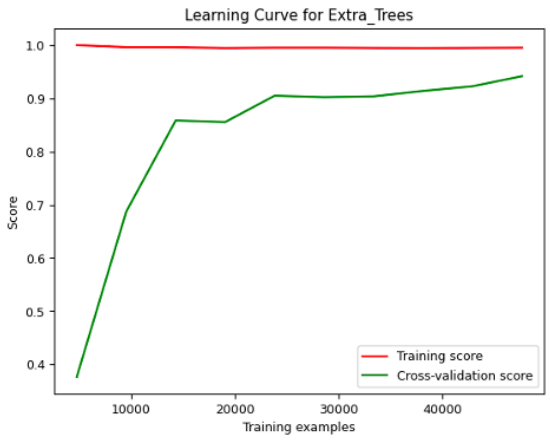
Combined 15 Classes Training Dataset을 활용하여 GridSearch를 진행했습니다.

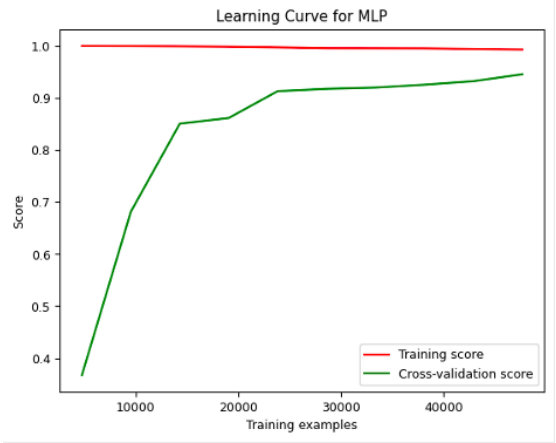
Extra-Tree는 n_estimators, max_depth를 변화시키며 n_estimators=300일 때, max_depth=20일 때 가장 높은 성능을 보임을 확인했습니다. 이 파라미터들을 찾는 데 걸린 Search Time은 415sec이며, Combined 15 Classes Validation Dataset으로 구한 정확도는 0.9282, Inference Time은 0.87sec이었습니다.

Extra-Tree의 경우 max_iter, alpha를 변화시키며 max_iter=2000일 때, alpha=0.01일 때 가장 높은 성능을 보임을 확인했습니다. 이 파라미터들을 찾는 데 걸린 Search Time은 8017sec이며, Combined 15 Classes Validation Dataset으로 구한 정확도는 0.9387, Inference Time은 0.47sec이었습니다.

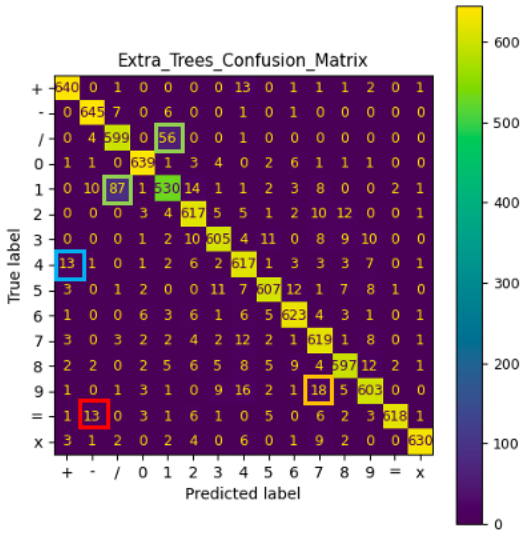
Search Time은 현저하게 차이 났지만, 모델을 배포한 후 사용 과정에서는 Accuracy와 Inference가 더 중요하다고 생각했고, 정확도가 더 높고 Inference Time이 2배로 빠른 MLP를 최적의 모델로 선정하였습니다.

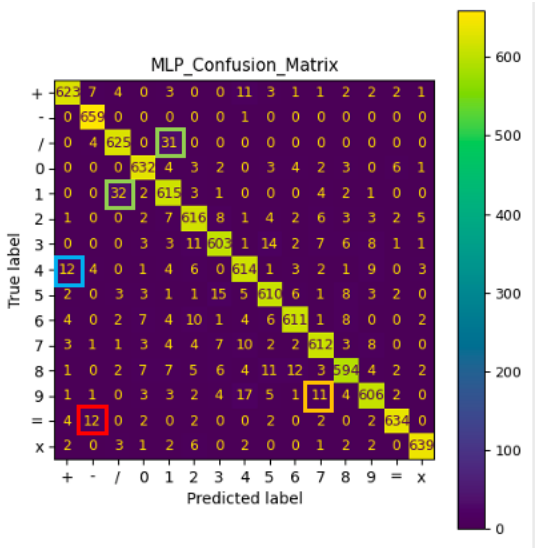
Iteration에 따른 Learning Curve를 그려 본 결과, MLP의 Learning Curve가 미세하게 더 많이 상승함을 확인할 수 있었습니다. 그래프는 아래와 같습니다.





Combined 15 Classes Validation Dataset으로 그려본 Confusion Matrix는 아래와 같습니다. 초록색 박스는 1과 '/'이 혼동됨이 나타나는 부분입니다. Extra Tree보다 MLP에서 혼동된 데이터 수가 더 적었습니다. 파란색 박스는 4와 '+'가 혼동되는 부분입니다. MLP에서 1개의 데이터가 덜 혼동되었습니다. 주황색 박스는 9와 7이 혼동된 부분입니다. MLP에서 확실히 덜 혼동되었습니다. 빨간색 부분은 '='과 '-'가 혼동되는 부분이며, MLP에서 1개의 데이터가 덜 혼동되었습니다. 1개의 데이터만 차이나는 부분에서 MLP가 좋다고 결론지을 수는 없지만, 가장 많이 혼동된 1과 '/'이 MLP에서 확실히 덜 혼동되었기 때문에, MLP가 애매한 데이터를 판별하는 데 있어서 더 좋은 성능을 보인다고 결론지었습니다.





추가적으로 더 좋은 성능 가진 모델을 만들어보고자 Voting Classifier에 두개의 모델을 넣어 학습을 진행했습니다. 이 때 사용된 학습 데이터는 역시 Combined 15 Classes Training Dataset이며, Combined 15 Classes Validation Dataset로 평가되었습니다.

Voting Classifier (soft)의 Training Time은 357sec, Voting Classifier (hard)의 Training Time은 393sec로 soft일 때 더 적게 걸렸습니다. Validation Dataset에 대한 Accuracy는 soft일 때 0.9422, hard일 때 0.9305로 soft일 때 더 좋았습니다. 그리고 Inference Time은 soft일 때 1.62sec, hard일 때 1.66sec가 걸렸습니다. 결론적으로 성능을 측정하는 모든 측면에서 soft가 더 좋은 결과를 보였습니다.

2-7. 최종 테스트

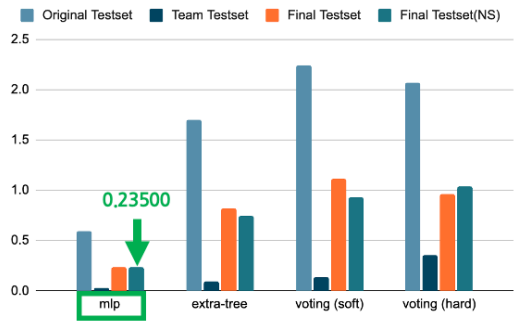
최종 테스트 데이터는 15개 클래스에 대한 데이터이며, Noise가 있고 shift된 데이터입니다. 총 개수는 5,478개 입니다. 아래 노란색으로 표시한 부분이 파인 튜닝된 MLP 모델이며, Inference는 0.23500sec, 0.92150의 정확도를 가집니다.

최종 테스트 데이터 이외에도 참고용으로 이전에 구축했던 다양한 데이터에 대해 테스트를 진행해보았습니다.

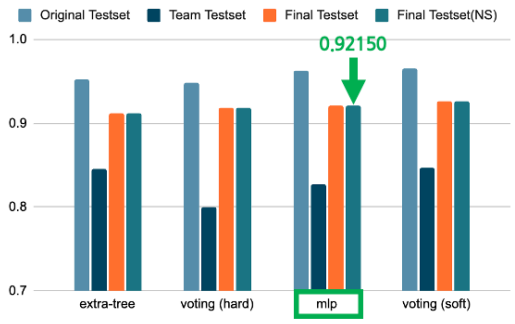
		Extra-Tree	MLP	Voting (hard)	Voting (soft)
Original Testset (10classes, 14,000)	Inference Time	1.69720	0.59328	2.06655	2.25069
	Accuracy	0.95264	0.96271	0.94907	0.96500
Team Testset (10classes, 400)	Inference Time	0.08964	0.02795	0.35179	0.13811
	Accuracy	0.84500	0.82750	0.80000	0.84750
Final Testset (15classes, 5,478)	Inference Time	0.82431	0.23394	0.96035	1.12011
	Accuracy	0.91201	0.92150	0.91822	0.92680
Final Testset (15classes, **Noised + shifted**, 5,478)	Inference Time	0.74779	**0.23500**	1.04238	0.92722
	Accuracy	0.91201	**0.92150**	0.91822	0.92680

아래 그래프는 위 표를 그래프로 변환한 이미지입니다. 두 그래프 모두 Final Testset (Noised, Shifted Version)을 기준으로 정렬되었으며, MLP는 표시된 4개 모델 중 Inference time이 0.23500으로 가장 짧고, Accuracy가 0.92150으로 두 번째로 좋았습니다. 가장 높은 정확도를 가진 모델은 VotingClassifier(soft)인데, 0.92680으로 MLP와 0.00530밖에 차이나지 않았습니다.

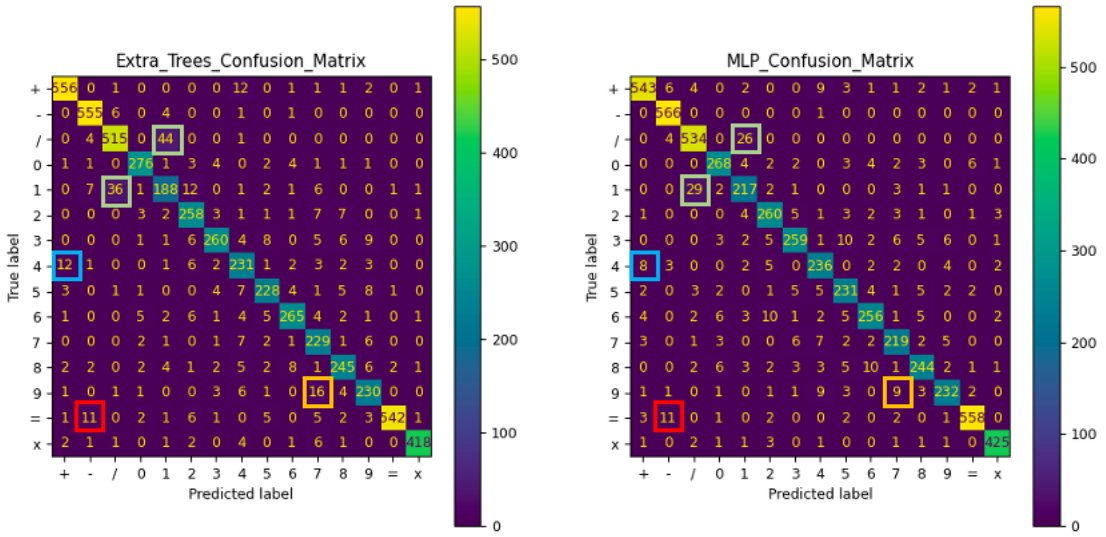
- **Inference Time** (Final Testset(NS) 기준 정렬)



- **Accuracy** (Final Testset(NS) 기준 정렬)



Final Testset (15 classes, Noised + Shifted, 5478)로 Confusion Matrix를 그려본 결과는 다음과 같습니다.



Validation 과정에서 확인한 바와 비슷하게, 혼동되는 숫자나 글자의 종류는 비슷했습니다. Extra Tree보다 MLP가 1과 '/'를 더 잘 구분했으며, 4와 '+'도 MLP에서 더 잘 구분되었습니다. 9와 7 또한 MLP에서 더 잘 구분되었습니다. 하지만 '='과 '-'는 두 모델에서 구분하는 정도가 같았습니다.

'='과 '-'에 해당하는 이미지를 출력하여 분석한 결과, '=' 이미지에서 두 줄이 붙어있는 경우도 있었기에 데이터 특성 상 해결되지 못한 한계점이라고 판단하였습니다. 결론적으로 Confusion Matrix를 살펴보며, MLP가 헷갈리는 숫자나 기호를 더 잘 구분하는 모델이라고 판단하게 되었습니다.

III. 결론

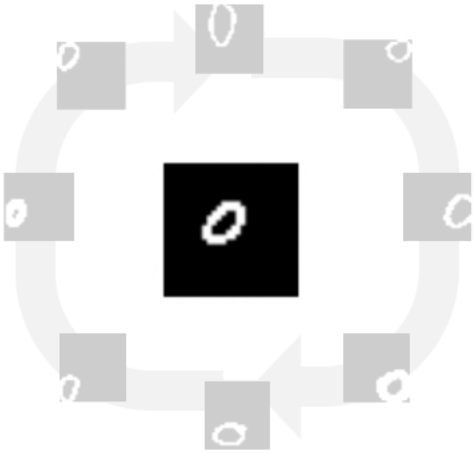
3-1. 최종 결론

프로젝트 첫 번째 목표는 Original MNIST Dataset과 Handmade MNIST Dataset을 비교한 후, Handmade MNIST Dataset의 성능을 개선시킨 Machine Learning Model을 학습시키고 최적화하는 것이었습니다. 저희는 초기에 Handmade MINST Dataset의 성능을 측정했을 때 Original MNIST Dataset보다 현저히 낮은 성능을 보임을 확인하였으며, Original Dataset과 Handmade Dataset을 결합하여 숫자 및 기호 클래스 별로 동일한 데이터 수를 가진 Combined Dataset을 구축하였습니다. 그리고 총 8개의 모델을 선정하여 단계별로 학습 및 테스트 과정을 거치며 MLP 모델이 가장 성능이 우수하다는 결론을 내렸습니다.

프로젝트의 두 번째 목표는 최종 모델은 Noised, Shifted Data에 강건해야 한다는 점이었습니다. 그래서 저희는 Final Testset에 Noise를 추가하고 Shifted된 데이터를 각각 20%, 25% 만들었으며, 이 데이터로 실험한 결과 MLP 파이프라인에 있는 denoising, centering, enlarging 과정이 잘 수행되어 Noise, Shifted Data와 거의 동일한 정확도를 가짐을 확인했습니다. 결론적으로 어떠한 데이터가 들어와도 데이터 전처리가 잘 진행되어 분류되는 모델을 구축하였습니다.

3-2. 피드백

< Shifted data 처리방식 결정 과정에서의 어려움 >

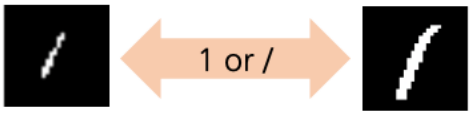


데이터를 확인하는 과정에서 위 사진과 같이 다양한 방향으로 치우친 데이터들이 많았고, 학습 시에 다양한 방향으로 치우친 데이터를 많이 생성한다면 이에 강건한 모델이 될 수 있겠다는 생각을 하게 되었습니다. 모델의 파이프라인에서 이 문제를 해결하지 않아도, 다양한 방향으로 치우친 데이터가 있으면 괜찮겠다는 판단을 내린 상태였던 것입니다.

하지만 교수님께 해당 내용을 질문한 후, 모델 파이프라인에서 이러한 데이터를 처리하는 방법이 좋다는 사실을 인지하게 되었습니다. 이미지 내 숫자나 이미지는 상하좌우 4방향으로만 치우쳐 있지 않으며, 360도 방향으로 미세하게 치우친 데이터를 모두 고려해야 비로소 해당 문제가 해결된다는 것이었습니다. 그래서 저희는 중앙화, 최대화 되는 함수를 만들어 어떠한 숫자나 기호가 들어오더라도 잘 인식되는 모델을 구현하였습니다.

파이프라인에 들어간 데이터 중앙화 과정이 정확하게 수행되는지 확인하기 위해, Final Testset의 Shifted version을 만들었습니다.

< 기울어진 1, 그리고 '/' >



왼쪽 이미지는 1, 오른쪽 이미지는 '/'입니다. 하지만 기울어진 각도에는 차이가 없습니다. 모델 파인튜닝 과정에서 Confusion Matrix를 살펴보며 구분하기 애매한 클래스들을 다뤄보려고 했으나, 위와 같이 1과 '/'의 경우, 사람마다 글씨를 쓰는 방식이 다르기에 임의로 데이터를 전처리하고 정의하기에는 애매하다고 판단 했습니다. 또한 똑바로 서있는 1이 존재하기 때문에, 기울기를 변수로 설정하여 학습한다면 -와 /, 1이 서로 혼동되는 경우도 존재할 것입니다. 이에 기울기라는 변수는 고려하지 않도록 결론 지었습니다.

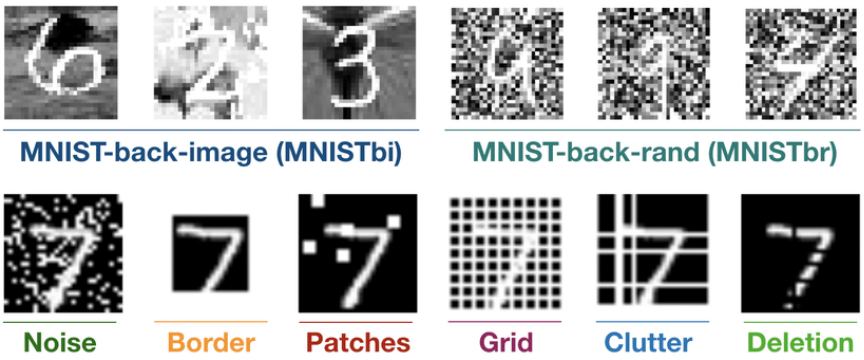
< 테두리가 남아있는 데이터 >

Handmade Dataset 제작 과정에서 사용한 10*10 표의 테두리가 잘리지 않아 해당 테두리가 포함된 이미지가 종종 존재했습니다. 최대화 과정에서 테두리가 포함되어 숫자나 기호가 최대화 되지 않는 경우가 있었습니다. 이러한 문제가 저희 프로젝트에서 아쉬운 점 중 하나입니다.

< 다양한 Noise >

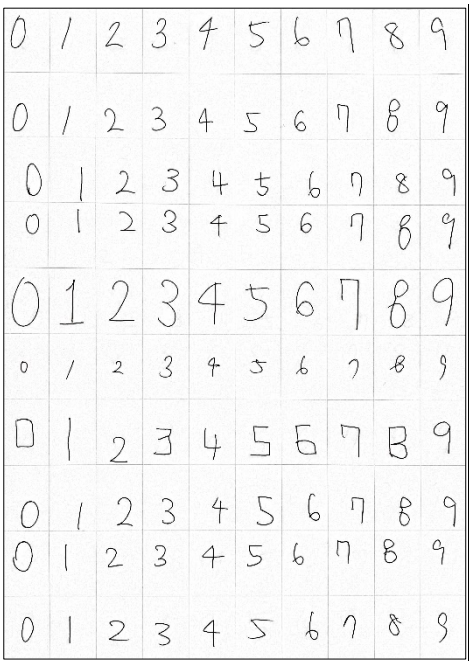
어떤 데이터가 들어와도 좋은 성능을 보이는 모델을 완성하기 위해서는 노이즈를 제거하는 과정이 필수적이라고 생각했습니다. 보통 모델이 노이즈를 학습하면 훈련 데이터에 과적합 되기 쉽고 새로운 데이터에 대한 일반화 성능이 저하될 수 있기 때문입니다.

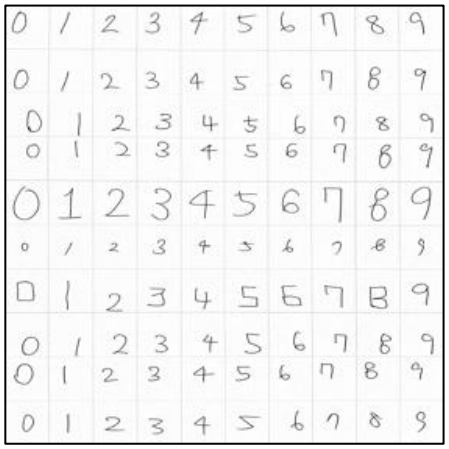
MNIST 데이터 셋에 발생하는 다양한 노이즈를 찾아보았는데, 모든 노이즈를 완벽하게 전부 제거할 수 있다면 모델의 성능은 좋아지겠지만, 세세한 노이즈 제거 과정은 영상처리의 영역이기 때문이라는 사실을 프로젝트 말미에 알게 되었습니다. 결론적으로 아래 그림에서 MNIST-back-rand와 같은 노이즈 정도만 해결되는 모델 구현으로 프로젝트를 마무리하게 되었습니다.

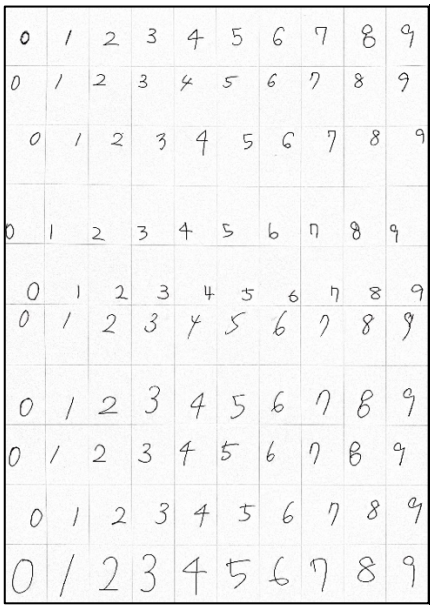


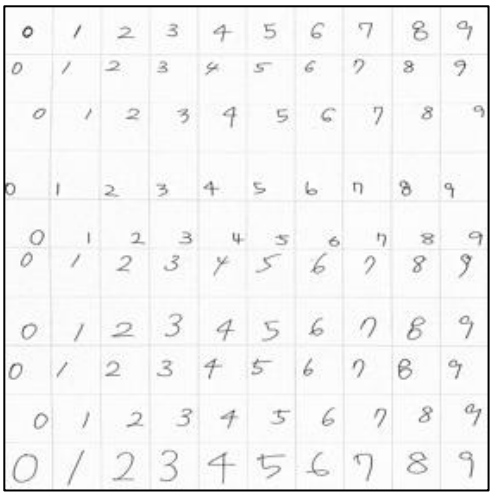
Appendix-1. Team Data set

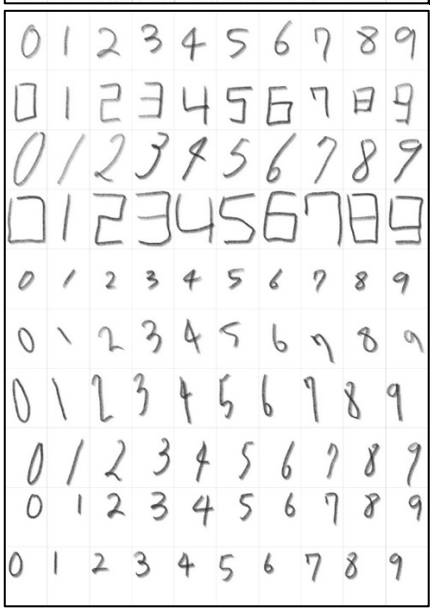
// 원본, 280*280 Resize 순서 배열 // 테두리 없이 진행

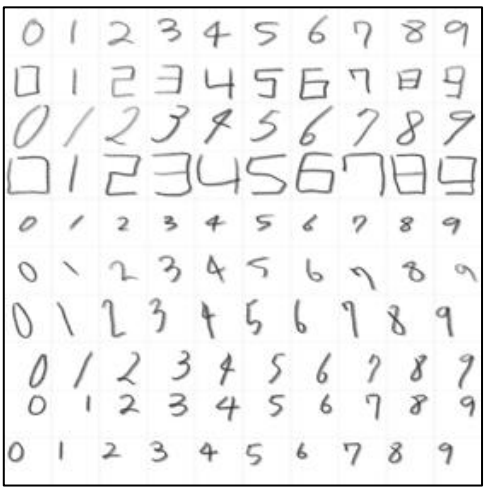


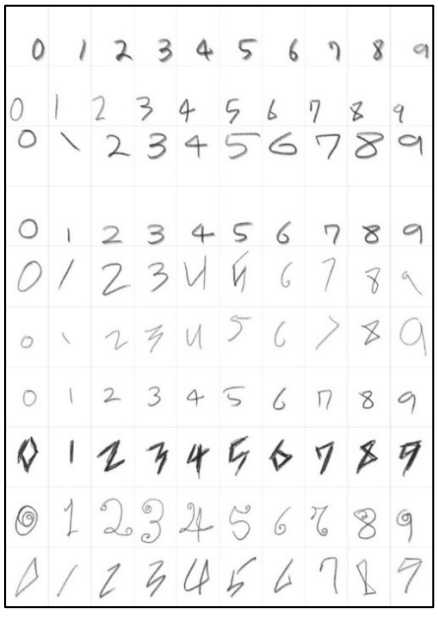


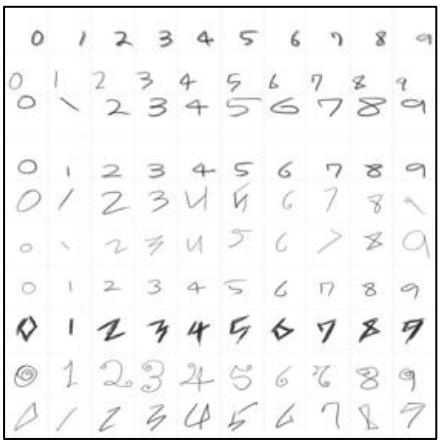












Appendix-2. 노트북 파일별 설명

"최종1. Original, Hand-made 분석.ipynb"

Original MNIST에 대한 분석과 Hand-made MNIST에 대한 분석 과정이 담겨 있습니다.

"최종2. Prepare Dataset & Deafult Train.ipynb "

Original, Handmade, Combined(class 10/ class 15) 총 4가지 Dataset을 만들고, class 15를 제외한 각 dataset을 다양한 모델에 대해 Deafult 학습을 통해 모델을 선정하는 과정이 담겨 있습니다.

"최종3. Data 전처리 및 Pipeline 구성.ipynb "

Class 15 Combined Dataset을 구성하고, Default hypter parameter로 이전에 선정된 모델을 학습시키며 pipeline에 대한 통찰을 얻고, 정규화 함수, Centering 함수, Zoom 함수, Denoise 함수 등 다양한 pipeline을 구성하는 함수를 작성하고, 작성한 함수를 Combination하여 최적의 Pipeline을 Select하는 과정과 Noised, Shifted Dataset를 제작하는 과정이 담겨 있습니다.

"최종4. 모델을 이용해서 Noise 제거 (시행착오).ipynb "

노이즈를 제거하기 위해 머신러닝 모델을 학습 시켜 해당 모델을 바탕으로 Denoise를 진행 하려 했으나, 해당 과정이 잘 안되었고, 결론적으로 함수를 사용해 모델을 구성해야 한다는 결론을 내리는 과정이 담겨 있습니다.

"최종5. Fine Tuning, Evaluation.ipynb "

최종 선정된 모델인 MLP와 Extra tree에 대해서 Grid Search를 진행하고, 각 모델을 학습 시키면서 Learning Curve와 Confusion matrix를 출력해 분석하고, 최종 Test를 진행하는 과정이 담겨 있습니다.

"Appendix. Handmade Team Test set 준비.ipynb"

Origianl dataset과 Handmade dataset을 비교하기 위한 Team Test set 제작 과정이 담겨 있습니다.

Appendix-3. 프로젝트 수행일지 모음

(1) 1차 회의지

작성자: 송준규
날짜: 2023-10-26 (목)
회의 장소: Google Meet

회의 내용: 프로젝트 설명 pdf 함께 읽고 파악하기

추진 계획: 계획서 작성 outline 잡기

결정 사항: 어떤 모델이 사용될 수 있는지 모델 공부해보기 (복습)

(2) 2차 회의지

작성자: 송준규
날짜: 2023-11-09
회의 참석자: 20236079 김희균, 20201590 송준규, 20192447 정승연, 20201610 정하연
회의 장소: Google Meet

회의 내용:

1. original data vs hand-made data
2. model 선정
    a. 선정 가능한 모델:
        random forest // ensemble (o)
        extra tree classifier (o)
        knn (o)
        mlp //
        voting classifier //
        svm (o)
        softmax (o)
        decision tree (o)
3. Data cleaning // home-made data 를 받고 나서 진행할 예정
4. 모델 최적화
5. 분석
6. 최종 선정된 모델 및 파라미터

추진 계획: 각자 모델 학습 및 분석

결정 사항:
각자 정해진 모델을 Original MNIST data set을 활용하여 학습시키고, 결과를 분석한다.
일요일 까지 카카오톡에 실행 가능한 코드와 함께 설명을 남긴다.

(3) 3차 회의지

작성자: 정하연
날짜: 2023-11-16
회의 참석자: 20236079 김희균, 20201590 송준규, 20192447 정승연, 20201610 정하연
회의 장소: Google Meet

회의 내용:

1. original data vs handmade data 자체 비교 분석
2. original dataset 으로 학습한 다양한 model 의 성능 비교 및 논의
   random forest // ensemble
   extra tree classifier
   knn
   mlp
   voting classifier
   svm
   softmax
   decision tree
3. handmade dataset - cleaning
   a. label 잘못된 것까지 확인
   b. 숫자인데 기호라고 파일명을 잘못 지정해서, 숫자데이터에서 기호 데이터를 빼는 작업이 필요함.
4. handmade dataset 으로 다양한 모델 학습
5. 모델 최적화
6. 분석
7. 최종 선정된 모델 및 파라미터

추진 계획:

1. 금요일까지 *data cleaning 방안 모색 및 공유하기*

2. 토요일까지 handmade dataset 으로 학습 및 비교 분석, 최적화
3. 일요일까지 발표자료 제출, 발표 대본 작성 및 리허설

결정 사항:

- 진행 일정 위 참고
- 중간 발표 내용 목차 아래에 정리함.

[중간 발표에 들어가야 할 것 같은 내용 정리]

1. 사용할 데이터 결정 기준 [1] 1~9,+,-,=,/,x 제외 버리기 or 흡수하기
   [있는 거 없는 거 비교]
2. 결정 기준 [2] 학습을 따로 따로 할지 데이터 [합쳐서 할지]
3. original data 로 학습한 결과, handmade(raw), handmade(cleaning) 비교
4. ~~학습시간, 예측시간 (inference time), Accuracy~~
5. 최종 선정 모델 및 데이터
6. 추후 계획

[중간발표 자료 - 목차]

1. original data (어떤 모델 사용, 어떻게 학습, 학습 결과)
   - 학습결과 분석 시: 학습 시간, 예측 시간, 정확도
2. handmade dataset - cleaning 방식
3. 학습
   a. validation (fold = 5. 보통 5-10개 하는데 적당히 5개 골라보았다.)
   b. Grid Search로 찾자
4. accuracy 비교: original VS. handmade(raw) VS. handmade(cleaning)
5. 추후 계획 (data cleaning 다른 아이디어 적용해볼 것. / 학습시간, 예측 시간, 정확도 모두를 고려하여 최적화 진행 예정)

[data cleaning - idea]

1. ***15 개 클래스에 속하지 않는 라벨은 다 버림 (11/17 까지 게시판 답변 보고 논의 후 진행).*** 0~9, 기호 순서대로 정렬 후 육안으로 제거?
2. 틀린 정답으로 나온 데이터 다시 확인 필요(1,-./) (3,5)

[최종 ipynb에 들어갈 내용 참고 (02_end_to_end_machine_learning_project)]

1. Get the Data
   a. download the data
   b. take a quick look at the data structure
   c. create a test set
2. Discover and visualize the Data to Gain Insights
   a. Visualizing Geographical Data
   b. Looking for Correlations
   c. Experimenting with Attribute combinations
3. Prepare the data for machine learning algorithms

a. data cleaning
b. handling text and categorical attributes
c. feature scaling
d. Transformation pipelines

4. Select and Train a Model
    a. Training and Evaluating on the Training set
    b. better evaluation using cross-validation
5. fine-tune your model
    a. grid search
    b. randomized search
    c. analyze the best models and their errors
    d. evaluate your system on the test set
    e. model persistence using joblib

(4) 4차 회의지

작성자: 정하연
날짜: 2023-11-27 (월)
회의 참석자: 20236079 김희균, 20201590 송준규, 20192447 정승연, 20201610 정하연
회의 장소: Google Meet

회의 내용:

- 진행상황 공유
- 계획 세우기

진행 상황:

1. 중앙화 함수 구현 완료
2. zoom 함수 구현중
3. 데이터 클리닝 진행한 데이터로 GridSearch 학습 완료

추진 계획:

0. zoom 함수 구현
1. 보고서 작성 시작
2. 발표 파트 나누기 (목)
3. pipeline에 중앙화, zoom하는 과정을 넣기 (학습 할 때 상하좌우 shift된 데이터를 임의로 생성하려면, 우리가 지금 생각한 상하좌우보다는 360도로 이동된 데이터가 필요하여 몇 백만개에 달하는 데이터를 학습해야해서 현실적으로 어려움.)

4. 최종적으로 모든 과정이 포함된 ipynb 파일 만들기 (best model 포함하여 학습 시간 단축)

(5) 5차 회의지

작성자: 정하연

날짜: 2023-11-30 (목)

회의 참석자: 20236079 김희균, 20201590 송준규, 20192447 정승연, 20201610 정하연

회의 장소: Google Meet

회의 내용:

[현재 진행상황 공유]

- 최종 데이터 및 모델 구축 완료 및 학습 중
- 최종 보고서 작성 중

[오늘 결정해야 할 사항]

- PPT 아웃라인 잡기
- PPT 파트 나누기 (각자 2분씩 총 8분 발표해야함)
- 발표 리허설 일정 잡기 (주말 중)

진행 상황:

- 최종 데이터 및 모델 구축 완료 및 학습 중
- 최종 보고서 작성 중

추진 계획:

[개인 별 해야할 일]

- 하연: 금요일 오전까지 PPT 제작 완료, 보고서 6 번 표+그래프 만들기
- 준규: 보고서 1~6 번 내용 보완
- 희균: 보고서 내용 정리 및 피드백 작성(future work 으로 할만한 사례 조사 등)
- 승연: ppt 및 보고서 참고하여 대본 작성

[중요 일정]

- 금요일까지 보고서 및 ppt 대강 완성
- 토요일 오전까지 대본 작성
- 토~일 중 리허설 진행

[발표 파트]

- 승연: 프로젝트 목표 + 데이터 전처리 과정 전반부
- 준규: 데이터 통찰 및 전처리 과정 후반부
- 하연: 모델 선정 및 최적화 과정
- 희균: 피드백

Appendix-4. 기여도

이름	기여도	역할	구체적인 아이디어 제시 및 팀 기여
정하연	33%	팀장	[아이디어 제시] Pipeline을 Transformer library를 통해 구현 모든 모델을 Grid Search 돌리는 것 지양 Tight boundary box enlarging 제시 Normalizer(l2), Normalizer(Max) 비교
			[팀기여] Softmax, Decision Tree Default 모델 학습 회의지 작성 3회 "Select Train Models, Fine-Tune Model" part 발표 "데이터 분석 일부" part 보고서 작성 "최종 모델 선정 및 모델 학습 진행" part 보고서 작성 "최종 모델 파인 튜닝" part 보고서 작성 "최종 테스트" part 보고서 작성
송준규	31%	팀원	Noised, Shifted 데이터 비율 별 생성 Noised, Shifted 데이터 활용해 Pipeline의 작동 여부 테스트 KNN 머신러닝 모델로 노이즈 제거 방안 노이즈 제거 함수 정의 (strict threshold 0.66) Square boundary box enlarging 제시
			KNN, SVM Default 모델 학습 회의지 작성 2회 최종 모델 Grid Search 진행 Data set 준비 Noised, Shifted Data set 준비 Confusion Matrix, Learning Curve 출력 "전처리 및 Pipeline" part 발표 "데이터 구하기" part 보고서 작성 "데이터 분석 일부" part 보고서 작성 "데이터 전처리 및 파이프라인 구축" part 보고서 작성
김희균	19%	팀원	Extra trees, Voting Default 모델 학습 "Final Test, Feedback" part 발표 "최종 결론, 피드백" part 보고서 작성
정승연	17%	팀원	MLP, Random Forest Default 모델 학습 "데이터 분석, 및 전처리 일부" part 발표