**Machine Learning Project**

**2024 – Fall / Final Report**

**“ Generating Colored MNIST Dataset Classifier ”**

로고, 그래픽, 폰트, 상징이(가) 표시된 사진

자동 생성된 설명

| **Class** | 머신 러닝 (2150034001, 수 09:00) |
| --- | --- |
| **Major** | 전자정보공학부 IT융합전공 |
| **Team** | JWCA |
| **Teammate** | 정재훈 (20202996)  안지수 (20221602)  위찬우 (20202982)  최연서 (20221628) |

**< Contents >**

**1. Introduction**

**2. Flow of Project**

**2-1.** 데이터 전처리

**2-2.** 데이터 증강

**2-3.** 모델 선정

**2-4.** 성능 평가

**2-5.** 오류 분석 및 개선

**3. Conclusion**

**4. Appendix**

**1. Introduction**

이번 프로젝트는 기존의 MNIST 데이터셋을 확장하여 전경색과 배경색, 색상 정보를 포함한 MNIST 데이터셋에 대해 머신 러닝을 통한 분류기를 생성하는 것이 목표이다. 데이터셋은 각 샘플에 대한 픽셀 값과 숫자, 색상 정보를 담고 있고 이를 분류 모델이 학습한 뒤, 숫자와 색상을 구분하는 작업을 수행하는 과정을 구현해보고자 한다. 본 프로젝트에서는 Homework 1에서 진행해보았던 숫자 분류 뿐만 아니라 전경(Foreground) 색상과 배경(Background) 색상 분류 작업도 수행해보고자 한다. 본 프로젝트를 통해 다중 분류 문제에서의 데이터 확장 절차와 모델의 성능 최적화 절차를 진행해보며 모델 개발 작업의 전반적인 진행 흐름을 경험해보고 역량을 향상시키고자 한다.

본격적으로 프로젝트를 진행하기 전, 프로젝트에서 사용할 데이터셋을 사전에 제공된 함수를 통해 확보 및 분석해보았다. 최초의 MNIST 데이터셋을 Pytorch를 통해 획득하고 이를 Image로 변환한 뒤, 전경색과 배경색이 겹치지 않도록 색상을 설정하였다. 색상을 설정하는 과정에서는 프로젝트 진행 간 동일한 데이터를 보장하기 위해 Random Seed를 설정함으로써 결과의 재현성을 보장하였다. 데이터셋에 색상을 추가한 Colored MNIST가 어떻게 구성될지 확인하기 위해 최초의 MNIST 데이터셋에서 임의의 샘플 한 개를 Colored MNIST 데이터셋으로의 변환 과정을 적용한 뒤 시각화해보았다. 해당 시각화 자료를 확인한 뒤 색상이 적용된 Sample이 실제로 어떻게 구성되어 있는지 확인해보았다.

텍스트, 그래픽, 그래픽 디자인, 스크린샷이(가) 표시된 사진

자동 생성된 설명

그림 1. 최초 MNIST Dataset의 첫 번째 Sample에 색상을 적용한 모습

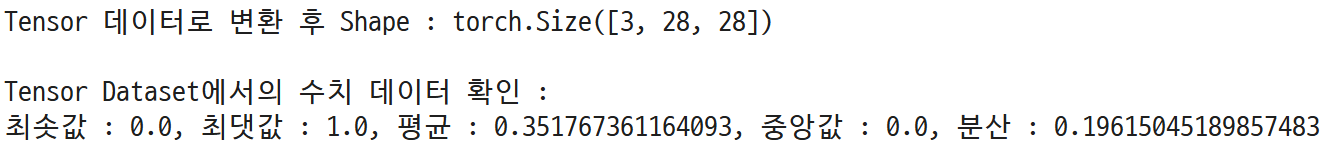


그림 2. 이미지 Sample을 Tensor로 변환했을 때의 수치형 정보

해당 정보를 통해 하나의 Sample에 대한 Pixel 값이 R, G, B로 분리되어 저장되어 있다는 것을 알게 되었고, 이를 DataFrame으로 변환하여 확인해보았다. Pixel 값을 DataFrame 형태를 통해 확인해보니 실제 시각화 이미지와 동일한 형태로 데이터가 구성되어 있음을 알 수 있었다.

텍스트, 라인, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

그림 3-1. Sample의 R Pixel에 대한 값

텍스트, 스크린샷, 번호, 문서이(가) 표시된 사진

자동 생성된 설명

그림 3-2. Sample의 G Pixel에 대한 값

텍스트, 스크린샷, 번호, 문서이(가) 표시된 사진

자동 생성된 설명

그림 3-3. Sample의 B Pixel에 대한 값

**2. Flow of Project**

**2-1.** 데이터 전처리

모델의 학습 및 프로젝트 진행에 위 데이터를 적절하게 사용하기 위해서 2차원 데이터로 구성하고자 R, G, B에 대한 Pixel 값을 하나의 Row에 위치하도록 하고자 하였다. 즉, 하나의 Sample이 하나의 Row에 R, G, B에 대한 각 784개의 Pixel 정보를 Column (Feature)으로 구성되도록 하고자 하였다. 사전에 제공된 함수들을 활용하여 PyTorch를 통한 최초의 데이터셋 획득 후 Image로 변환하여 색상을 적용한 뒤 해당 Sample을 DataFrame의 Row로 구성하는 작업을 수행하였다. 이 작업을 사전에 분리되어 획득된 Train MNIST Dataset, Test MNIST Dataset에 각각 적용하여 전처리 전 Colored MNIST Dataset을 구성하였다.

스크린샷, 텍스트, 라인, 평행이(가) 표시된 사진

자동 생성된 설명

그림 4. Colored MNIST Train Set

하나의 Row에 R, G, B Pixel 값이 모두 들어있고, 각 Column은 28 x 28 Size의 Pixel 정보를 R, G, B 로 분리된 상태로 구성되어 있다. 데이터 증강 전 Train Set은 60,000개, Test Set은 10,000개의 데이터로 구성되어 있다.

Train Set과 Test Set으로만 프로젝트를 진행하게 되면 최적화 과정에서 Test Set이 학습에 관여하게 되는 상황이 발생할 것이라고 판단했다. 후술할 모델 선정 과정에서의 최적의 하이퍼파라미터 탐색 과정에서 Test Set을 사용하게 되면 성능 평가를 위한 Test Set을 학습에 관여시킴으로써 Test Set을 사용하는 의미가 퇴색되고, Train Set에 대한 성능만 향상되고 일반화 성능은 저하되는 Overfitting (과대적합) 등의 문제가 발생한다. 따라서, Test Set은 최종 성능 평가 시에만 활용하고 최적화 과정 등에서의 성능 평가에는 Validation Set을 활용할 것이다. Train Set에서 일부를 Validation Set으로 분리하여 사용하려고 한다. 기본적으로 데이터 Sample의 수가 충분하다고 판단되어 Cross-Validation 등의 기법을 사용하지 않을 것이므로 전처리 과정에서 Train Set으로부터 분리하는 과정으로 Validation Set을 구성하고자 한다.

이 데이터셋들을 모델 학습에 그대로 사용하기보다는 Feature Scaling을 통해 학습의 효율을 올리고자 한다. Feature Scaling이란 변수의 값 범위를 일정한 수준으로 맞추는 데이터 전처리 과정 중 하나로 범위를 일정한 수준으로 맞추어 차이를 줄임으로써 학습 과정에서의 불안정성과 일부 특성의 영향이 지나치게 커지는 현상을 방지할 수 있다. 또한, 학습에 경사 하강법 등의 최적화 알고리즘 및 손실 함수 개념을 사용하는 경우 최적화 경로의 균형을 잡아 원활한 학습이 가능하도록 할 수 있다. 현재의 데이터셋의 경우 Pixel 값이 고정된 범위 (0에서 255)이므로 각 특성의 범위가 다른 것은 아니다. 하지만 범위의 간격을 줄임으로써 모델 학습의 속도를 향상시키고, 수렴의 안정성과 특정 Feature 지배 방지, 다양한 알고리즘과의 호환성 향상 등을 통해 학습 성능을 향상시킬 수 있다.

MNIST 데이터셋은 Pixel 값의 최솟값, 최댓값이 명확하고 범위의 크기를 줄이는 것이 목표이기 때문에 정규화 (Normalization) 기법 중 하나인 Min-Max Scaling을 진행하고자 한다. 이번 MNIST Dataset의 경우 최솟값이 0이므로, 모든 데이터를 최댓값인 255로 나누는 간단한 방법을 통해 Scaling을 진행하였다. 데이터의 일관성을 위해 Train Set과 Validation Set, Test Set에 동일하게 적용하였다.

폰트, 텍스트, 화이트, 라인이(가) 표시된 사진

자동 생성된 설명

그림 5. 하나의 Value에 대한 Min-Max Scaling 계산 과정

| **K-Nearest**  **Neighbors** | **Accuracy** | **F1-Score** | **Duration Time (sec.)** |
| --- | --- | --- | --- |
| **Before Min-Max** | **0.9047** | **0.9040** | **127.2927** |
| **After Min-Max** | **0.9047** | **0.9040** | **113.8248** |

표 1. K-NN에서의 Min-Max Scaling 전과 후의 성능 비교

| **Decision Tree** | **Accuracy** | **F1-Score** | **Duration Time (sec.)** |
| --- | --- | --- | --- |
| **Before Min-Max** | **0.7321** | **0.7288** | **86.5735** |
| **After Min-Max** | **0.7333** | **0.7302** | **63.4723** |

표 2. Decision Tree에서의 Min-Max Scaling 전과 후의 성능 비교

위 과정들을 바탕으로 Colored MNIST Dataset의 Train Set, Validation Set, Test Set을 구성하고 분석 및 전처리를 완료하였다. 각 데이터셋은 훈련 데이터(Sample)와 정답 데이터(Label)가 분리되어 있는 상태로 저장되어 있다.

**2-2.** 데이터 증강

데이터 증강이란 데이터셋의 크기를 늘리고 다양성을 확보하여 모델이 더 강건한 성능을 발휘하도록 돕는 과정이다. 본 프로젝트에서는 MNIST 데이터셋과 유사한 형태로 데이터를 생성하기 위해 다양한 폰트를 활용하여 숫자 데이터를 확장하였으며, 기존 데이터에 전경색과 배경색만 변화시켜 색깔 데이터를 확장하고자 한다. 아래의 몇 가지 증강 기법을 통해 학습 데이터의 다양성을 증가시키려고 한다.

**데이터 증강 방법 (다양한 폰트 활용)**

다양한 폰트를 활용하여 숫자 데이터를 생성하기 위해 사전에 제공된 폰트 외에도 구글 폰트에서 식별이 어려운 폰트를 제외한 100개 이상의 폰트를 수집하였다. 굵기(Bold, Regular 등)에 따라 다른 스타일로 간주하여 총 1,159개의 고유한 폰트를 선정하고, 각 폰트마다 숫자 0부터 9까지 각각 세 개씩 총 30개의 이미지를 생성하였다. 이를 통해 총 34,770개의 이미지를 생성하며 데이터셋을 확장하였다.

**데이터 증강 및 변환 (폰트를 활용한 데이터 생성)**

기하학적 변환은 생성된 숫자 이미지에 다양한 변화를 주어 데이터의 다양성을 확보하고 모델의 강건한 성능을 학습하도록 돕는다. 본 프로젝트에서는 다음과 같은 변환을 적용하였다.

**1. 회전 (Rotation)**

- 숫자 이미지를 -10°, 0°, +10°로 회전하여 다양한 각도로 배치된 데이터를 생성

주방용품이(가) 표시된 사진

자동 생성된 설명디자인이(가) 표시된 사진

낮은 신뢰도로 자동 생성된 설명주방용품이(가) 표시된 사진

자동 생성된 설명

그림 6. AlegreyaSans-Bold 폰트를 왼쪽부터 -10°, 0°, +10° 만큼 회전시킨 이미지

**2. 중앙화 (Translation)**

- 이미지를 중앙에 배치하기 위해 x축은 (IMAGE\_SIZE[0] – text\_width) // 2 + dx, y축은 (IMAGE\_SIZE[1] – text\_height) // 2 + dy로 설정

- 기본적으로 dx = 0, dy = -6을 사용하여 중앙화를 진행하였으며, 일부 글자는 상단 또는 하단으로 이동한 경향이 있었지만 일반화 성능을 증가시킬 것으로 판단하여 유지

그림 7-1. dx = 0, dy = 0인 경우의 데이터 생성 예시

그림 7-2. dx = 0, dy = -6인 경우의 데이터 생성 예시

**3. 노이즈 추가 (Noise) (진행하지 않음)**

- 중간 보고서 작성 때 노이즈를 추가하여 일반화 성능을 높이려는 시도를 하였으나 얇은 폰트들이 다수 존재하는 데이터 특성 때문에 노이즈가 100까지 높아지는 경우 원래의 데이터가 많이 훼손되는 경향을 발견, 오히려 좋지 않은 데이터를 넣는 상황이 발생

- 따라서, 노이즈를 0에서 5 정도만 추가하였는데, 기존 데이터와의 차이가 거의 미미하여 노이즈가 없다고 봐도 무방할 정도였기 때문에 별도의 진행이 불필요하다고 판단

**데이터 정제**

위 과정들을 통해 데이터 증강을 진행한 뒤, 오히려 성능에 방해될 데이터들을 정제하는 과정을 진행하였다.

**1. 중앙화 이후에도 과도하게 하단에 위치하여 정확한 정보를 담고 있지 않은 데이터 삭제**

- 특정 폰트들은 글자의 문장 밑줄 기준을 과도하게 아래쪽으로 분포시키는 경향 존재

- 숫자를 육안으로도 인식하는 것이 불가, 해당 데이터를 삭제



그림 8. 과도하게 하단에 위치한 삭제된 데이터 예시

**2. ‘4’와 ‘1’을 구분하기 어려운 특정 이미지 데이터 제거**

- 특정 폰트를 이용해 생성한 4의 기둥 부분을 제외한 나머지 획이 육안으로 자세히 봐야 식별할 수 있는 수준의 얇고 연한 선으로 구성되는 경향 발견, 해당 데이터의 가치가 낮다고 판단하여 데이터 삭제



그림 9. ‘4’와 ‘1’이 잘 구분되지 않는 삭제된 데이터 예시

**3. ‘0’와 ‘8’을 구분하기 어려운 특정 이미지 데이터 제거**

- 특정 폰트를 이용해 생성한 0의 가운데 부분에 줄이 그어져 θ처럼 식별되는 것 확인, 8과 혼동될 여지가 있고 실제로 0을 해당 이미지처럼 사용하는 경우가 많지 않기 때문에 해당 데이터 삭제



그림 10. ‘0’의 원 안에 줄이 그어져있는 삭제된 데이터 예시

2번과 3번의 경우, 해당 숫자만 삭제하려고 했으나 전체적인 숫자의 개수 밸런스를 맞춰주기 위해 해당 폰트에 해당하는 모든 숫자를 제거하는 방향으로 진행하였다. 해당 작업은 수작업으로 진행하였고 이를 통해 약 4,000개의 데이터를 제거하여 30,813개의 데이터를 증강을 통해 확보하였다. 이 데이터들을 추후 모델의 성능을 보다 향상시키기 위하여 필요에 따라 추가하여 사용하고자 한다.

**2-3.** 모델 선정

데이터의 준비가 끝난 뒤, 이제 모델에 데이터를 학습시켜 분류 작업 수행 성능을 평가하고 최적의 모델을 생성하여 프로젝트의 목표를 달성해보려고 한다. 모델 선정은 다음과 같은 절차로 진행하고자 한다. 우선, 각 분류 작업에 적합하다고 판단되는 모델 후보군을 선정한다. 선정한 후보군들에 대해 각각 최적의 하이퍼파라미터를 탐색하고 해당 하이퍼파라미터로 설정된 모델 간 성능 비교를 통해 최종 모델을 선택하고자 한다.

텍스트, 폰트, 도표, 스크린샷이(가) 표시된 사진

자동 생성된 설명

그림 11. 모델 선정 절차를 나타낸 그림

**모델 후보군 선정 및 각 모델 별 최적의 하이퍼파라미터 탐색**

우선 숫자 분류 및 전경색 분류, 배경색 분류 작업에 대해 각각 적합하다고 판단되는 모델들을 후보군으로 설정한다. 후보군을 설정한 뒤, 각 모델 별로 여러 하이퍼파라미터 조합에 대한 성능을 비교하여 최적의 하이퍼파라미터를 탐색한다. 가장 성능이 좋다고 판단되는 하이퍼파라미터의 조합을 모델 간 성능 비교 시 사용하려고 한다. 성능 비교의 경우 후술할 성능 평가 기법을 바탕으로 평가하려고 한다. 가장 많이 다루는 하이퍼파라미터 위주로 임의의 값을 지정해보며 성능을 비교해보았다. 하이퍼파라미터 탐색의 경우, 동일한 데이터셋을 사용하여 경향 파악이 충분하다는 점과 시간 및 자원 효율성을 고려하여 축소된 Dataset을 활용하였다.

**숫자 분류 모델**

| **Best 5 Hyperparameter of K-Nearest Neighbors (Number Classification)** | | | |
| --- | --- | --- | --- |
| **metric** | **n\_neighbors** | **weights** | **Accuracy** |
| **Manhattan** | **3** | **Distance** | **0.6124** |
| **Manhattan** | **5** | **Distance** | **0.6070** |
| **Euclidean** | **3** | **Distance** | **0.6064** |
| **Manhattan** | **3** | **Uniform** | **0.6045** |
| **Manhattan** | **7** | **Distance** | **0.5997** |

표 3. K-NN (K-근접 이웃) 모델의 숫자 분류 모델 성능 Top 5의 하이퍼파라미터

| **Best 5 Hyperparameter of Decision Tree (Number Classification)** | | | | |
| --- | --- | --- | --- | --- |
| **min\_samples\_leaf** | **min\_samples\_split** | **max\_depth** | **max\_feature** | **Accuracy** |
| **4** | **2** | **50** | **sqrt** | **0.6540** |
| **4** | **4** | **50** | **sqrt** | **0.6540** |
| **4** | **8** | **50** | **sqrt** | **0.6540** |
| **8** | **2** | **50** | **sqrt** | **0.6387** |
| **8** | **4** | **50** | **sqrt** | **0.6387** |

표 4. Decision Tree 모델의 숫자 분류 모델 성능 Top 5의 하이퍼파라미터

| **Best 5 Hyperparameter of Random Forest (Number Classification)** | | | |
| --- | --- | --- | --- |
| **max\_depth** | **min\_samples\_split** | **n\_estimators** | **Accuracy** |
| **30** | **2** | **400** | **0.9281** |
| **30** | **5** | **300** | **0.9265** |
| **30** | **5** | **350** | **0.9251** |
| **25** | **5** | **350** | **0.9246** |
| **25** | **2** | **300** | **0.9237** |

표 5. Random Forest 모델의 숫자 분류 모델 성능 Top 5의 하이퍼파라미터

| **Best 5 Hyperparameter of Gradient Boosting (Number Classification)** | | |
| --- | --- | --- |
| **learning\_rate** | **max\_depth** | **Accuracy** |
| **0.2** | **2** | **0.8334** |
| **0.1** | **5** | **0.7813** |
| **0.1** | **2** | **0.7598** |
| **0.05** | **2** | **0.6918** |
| **0.01** | **2** | **0.5106** |

표 6. Gradient Boosting 모델의 숫자 분류 모델 성능 Top 5의 하이퍼파라미터

| **Best 5 Hyperparameter of Extra Trees (Number Classification)** | | | | |
| --- | --- | --- | --- | --- |
| **estimators** | **min\_samples\_split** | **max\_depth** | **max\_feature** | **Accuracy** |
| **400** | **2** | **30** | **sqrt** | **0.9327** |
| **500** | **2** | **35** | **sqrt** | **0.9315** |
| **300** | **4** | **40** | **sqrt** | **0.9312** |
| **400** | **4** | **35** | **sqrt** | **0.9312** |
| **640** | **2** | **40** | **sqrt** | **0.9312** |

표 7. Extra Trees 모델의 숫자 분류 모델 성능 Top 5의 하이퍼파라미터

**전경색 분류 모델**

| **Best 5 Hyperparameter of K-Nearest Neighbors (Foreground Color Classification)** | | | |
| --- | --- | --- | --- |
| **metric** | **n\_neighbors** | **weights** | **Accuracy** |
| **Manhattan** | **3** | **Distance** | **0.9351** |
| **Manhattan** | **5** | **Distance** | **0.9235** |
| **Euclidean** | **3** | **Distance** | **0.9231** |
| **Manhattan** | **3** | **Uniform** | **0.9120** |
| **Manhattan** | **7** | **Distance** | **0.9042** |

표 8. K-NN (K-근접 이웃) 모델의 전경색 분류 모델 성능 Top 5의 하이퍼파라미터

| **Best 5 Hyperparameter of Decision Tree (Foreground Color Classification)** | | | | |
| --- | --- | --- | --- | --- |
| **min\_samples\_leaf** | **min\_samples\_split** | **max\_depth** | **max\_feature** | **Accuracy** |
| **4** | **4** | **50** | **sqrt** | **0.9642** |
| **4** | **8** | **50** | **sqrt** | **0.9637** |
| **8** | **2** | **50** | **sqrt** | **0.9432** |
| **8** | **4** | **50** | **sqrt** | **0.9420** |
| **8** | **8** | **50** | **sqrt** | **0.9408** |

표 9. Decision Tree 모델의 전경색 분류 모델 성능 Top 5의 하이퍼파라미터

| **Best 5 Hyperparameter of Random Forest (Foreground Color Classification)** | | | |
| --- | --- | --- | --- |
| **max\_depth** | **min\_samples\_split** | **n\_estimators** | **Accuracy** |
| **10** | **2** | **300** | **0.9992** |
| **10** | **3** | **300** | **0.9992** |
| **30** | **2** | **350** | **0.9990** |
| **30** | **2** | **400** | **0.9990** |
| **10** | **3** | **200** | **0.9990** |

표 10. Random Forest 모델의 전경색 분류 모델 성능 Top 5의 하이퍼파라미터

| **Best 5 Hyperparameter of Gradient Boosting (Foreground Color Classification)** | | |
| --- | --- | --- |
| **learning\_rate** | **max\_depth** | **Accuracy** |
| **0.2** | **2** | **0.9985** |
| **0.1** | **2** | **0.9887** |
| **0.1** | **5** | **0.9881** |
| **0.05** | **2** | **0.9881** |
| **0.01** | **2** | **0.9854** |

표 11. Gradient Boosting 모델의 전경색 분류 모델 성능 Top 5의 하이퍼파라미터

| **Best 5 Hyperparameter of Extra Trees (Foreground Color Classification)** | | | | |
| --- | --- | --- | --- | --- |
| **estimators** | **min\_samples\_split** | **max\_depth** | **max\_feature** | **Accuracy** |
| **300** | **2** | **30** | **sqrt** | **0.9999** |
| **400** | **2** | **30** | **sqrt** | **0.9999** |
| **300** | **4** | **40** | **sqrt** | **0.9999** |
| **400** | **4** | **35** | **sqrt** | **0.9999** |
| **400** | **2** | **40** | **sqrt** | **0.9999** |

표 12. Extra Trees 모델의 전경색 분류 모델 성능 Top 5의 하이퍼파라미터

**배경색 분류 모델**

| **Best 5 Hyperparameter of K-Nearest Neighbors (Background Classification)** | | | |
| --- | --- | --- | --- |
| **metric** | **n\_neighbors** | **weights** | **Accuracy** |
| **Manhattan** | **3** | **Distance** | **1.0000** |
| **Manhattan** | **5** | **Distance** | **1.0000** |
| **Manhattan** | **3** | **Uniform** | **1.0000** |
| **Manhattan** | **3** | **Uniform** | **1.0000** |
| **Manhattan** | **5** | **Uniform** | **1.0000** |

표 13. K-NN (K-근접 이웃) 모델의 배경색 분류 모델 성능 Top 5의 하이퍼파라미터

| **Best 5 Hyperparameter of Decision Tree (Background Classification)** | | | | |
| --- | --- | --- | --- | --- |
| **min\_samples\_leaf** | **min\_samples\_split** | **max\_depth** | **max\_feature** | **Accuracy** |
| **4** | **2** | **50** | **sqrt** | **0.9999** |
| **4** | **4** | **50** | **sqrt** | **0.9999** |
| **4** | **8** | **50** | **sqrt** | **0.9999** |
| **8** | **2** | **50** | **sqrt** | **0.9999** |
| **8** | **4** | **50** | **sqrt** | **0.9999** |

표 14. Decision Tree 모델의 배경색 분류 모델 성능 Top 5의 하이퍼파라미터

| **Best 5 Hyperparameter of Random Forest (Background Classification)** | | | |
| --- | --- | --- | --- |
| **max\_depth** | **min\_samples\_split** | **n\_estimators** | **Accuracy** |
| **30** | **2** | **400** | **1.0000** |
| **30** | **5** | **300** | **1.0000** |
| **30** | **5** | **350** | **1.0000** |
| **25** | **5** | **350** | **1.0000** |
| **25** | **2** | **300** | **1.0000** |

표 15. Random Forest 모델의 배경색 분류 모델 성능 Top 5의 하이퍼파라미터

| **Best 5 Hyperparameter of Gradient Boosting (Background Classification)** | | |
| --- | --- | --- |
| **learning\_rate** | **max\_depth** | **Accuracy** |
| **0.01** | **2** | **0.9998** |
| **0.01** | **3** | **0.9998** |
| **0.05** | **2** | **0.9997** |
| **01** | **2** | **0.9997** |
| **0.2** | **2** | **0.9997** |

표 16. Gradient Boosting 모델의 배경색 분류 모델 성능 Top 5의 하이퍼파라미터

| **Best 5 Hyperparameter of Extra Trees (Background Color Classification)** | | | | |
| --- | --- | --- | --- | --- |
| **estimators** | **min\_samples\_split** | **max\_depth** | **max\_feature** | **Accuracy** |
| **300** | **2** | **30** | **sqrt** | **1.0000** |
| **500** | **2** | **35** | **sqrt** | **1.0000** |
| **400** | **2** | **30** | **sqrt** | **1.0000** |
| **400** | **4** | **35** | **sqrt** | **1.0000** |
| **400** | **2** | **40** | **sqrt** | **1.0000** |

표 17. Extra Trees 모델의 배경색 분류 모델 성능 Top 5의 하이퍼파라미터

**모델 간 성능 비교**

각 모델마다 하이퍼파라미터를 결정하고 이제 모델 간 성능을 비교하고자 각 모델에게 동일한 훈련 데이터와 평가 데이터를 부여하고 성능을 비교한다. 성능 비교의 경우 마찬가지로 후술할 성능 평가 기법을 바탕으로 평가하려고 한다.

| **Number** | **Accuracy** | **F1-Score** |
| --- | --- | --- |
| **K-NN** | **0.8942** | **0.8938** |
| **Decision Tree** | **0.6773** | **0.6752** |
| **Random**  **Forest** | **0.9413** | **0.9412** |
| **Gradient**  **Boosting** | **0.8334** | **0.8317** |
| **Extra Tree** | **0.9429** | **0.9428** |

표 18. 숫자 분류 작업을 수행한 모델 간 성능 비교 예시

| **Foreground** | **Accuracy** | **F1-Score** |
| --- | --- | --- |
| **K-NN** | **0.9287** | **0.9293** |
| **Decision Tree** | **0.9639** | **0.9639** |
| **Random**  **Forest** | **0.9992** | **0.9993** |
| **Gradient**  **Boosting** | **0.9982** | **0.9982** |
| **Extra Tree** | **0.9999** | **0.9999** |

표 19. 전경색 분류 작업을 수행한 모델 간 성능 비교

| **Background** | **Accuracy** | **F1-Score** |
| --- | --- | --- |
| **K-NN** | **1.0000** | **1.0000** |
| **Decision Tree** | **0.9999** | **0.9999** |
| **Random**  **Forest** | **0.9066** | **0.9064** |
| **Gradient**  **Boosting** | **0.9998** | **0.9998** |
| **Extra Tree** | **1.0000** | **1.0000** |

표 20. 배경색 분류 작업을 수행한 모델 간 성능 비교

성능 지표와 소요 시간 등을 고려한 결과, 세 작업 모두 Extra Trees 모델이 가장 좋은 성능을 보여 해당 모델을 최종 모델로 선정하기로 하였다.

**최종 모델 선정**

앞서 비교한 성능을 바탕으로 가장 우수하다고 판단되는 모델을 최종 모델로 선정하고 최종 성능 평가 진행 및 오류 분석을 통한 추가적인 최적화, 성능 향상 작업을 진행하고자 한다. 현재까지 진행 상황에서의 최종 모델은 Extra Tree로 해당 모델에 대해 Test Set을 이용하여 최종 성능을 평가해보았다.

| **Final Score of Extra Tree Model** | | |
| --- | --- | --- |
| **Classification Task** | **Accuracy** | **F1-Score** |
| **Number** | **0.9368** | **0.9367** |
| **Foreground**  **Color** | **0.9995** | **0.9995** |
| **Background**  **Color** | **1.0000** | **1.0000** |

표 21. 최종 모델 성능 평가

최종 모델의 성능 평가 시에는 Train Set으로는 Validation Set으로 사용한 12,000개의 Sample을 제외한 48,000개의 데이터를, Test Set으로는 사전에 구성한 10,000개의 데이터를 사용하였습니다.

**2-4.** 성능 평가

앞선 모델 선정 과정에서 모델의 최적의 하이퍼파라미터를 탐색한 뒤 이를 기반으로 모델 별 성능을 비교한 뒤 최종 모델을 선정할 계획이라고 하였고, 최종 모델에 대한 최종 성능 또한 평가할 것이다. 이러한 성능 평가 과정에서 어떤 방법으로 평가를 진행할 것인지, 지표로 어떤 것을 활용할 것인지에 대해 설명해보겠다.

가장 많이 사용되는 정확도(Accuracy)는 전체 데이터 중 모델이 바르게 분류한 비율을 의미한다. 정확도는 데이터셋이 불균형하게 구성되어 있을 경우, 모델의 분류 기준이 명확하지 않음에도 단순히 불균형한 데이터로 인해 정확도가 크게 향상될 수 있다는 단점이 있다. 데이터 증강을 진행하지 않은 Colored MNIST Dataset의 경우 색깔 (전경색과 배경색)에 대해서는 비교적 균형적이지만, 숫자 레이블에 대해서는 비교적 불균형적인 데이터이다. 따라서 정확도만을 단독으로 사용하기에는 만족스러운 평가가 될 수 없을 것이다.

폰트, 텍스트, 화이트, 타이포그래피이(가) 표시된 사진

자동 생성된 설명 텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

그림 12. (좌) 정확도, Accuracy 공식 / (우) 각 숫자 Label에 해당되는 Sample의 개수

모델이 Positive로 분류한 데이터 중 실제 Positive로 분류한 비율인 정밀도(Precision)는 확실한 Positive 데이터를 제외한 모든 데이터를 Negative로 분류할 경우 비정상적인 수치를 나타낼 수 있다는 단점이 있다. 따라서 이 지표 또한 단독으로 쓰기에는 어려움이 있을 것이다.

폰트, 텍스트, 번호, 화이트이(가) 표시된 사진

자동 생성된 설명

그림 13-1. 정밀도, Precision 공식

폰트, 도표, 디자인, 타이포그래피이(가) 표시된 사진

자동 생성된 설명

그림 13-2. 정밀도가 올바르게 활용되지 못하는 경우

이러한 단점들을 방지하고자 정밀도를 비롯하여 실제 값이 Positive인 데이터 중 모델이 Positive로 분류한 비율인 재현율(Recall)과 동일한 개념의 Negative Case인 특이도(Specificity)를 이용하여 보다 안정적으로 활용할 수 있는 지표인 F1-Score와 ROC AUC(ROC Area Under the Curve)를 중점적으로 활용하고자 한다. 정밀도와 재현율을 통해 F1-Score를 계산할 수 있고, 재현율과 특이도를 통해 TPR(True Positive Rate)와 FPR(False Positive Rate)를 계산할 수 있다. TPR, FPR을 이용하여 ROC Curve 및 ROC AUC를 확인할 수 있다.

폰트, 텍스트, 화이트, 번호이(가) 표시된 사진

자동 생성된 설명 폰트, 텍스트, 화이트, 번호이(가) 표시된 사진

자동 생성된 설명폰트, 텍스트, 화이트, 번호이(가) 표시된 사진

자동 생성된 설명

그림 14. (좌) 재현율, Recall / (우) 특이도, Specificity

폰트, 텍스트, 화이트, 번호이(가) 표시된 사진

자동 생성된 설명폰트, 텍스트, 화이트, 번호이(가) 표시된 사진

자동 생성된 설명폰트, 텍스트, 화이트, 번호이(가) 표시된 사진

자동 생성된 설명

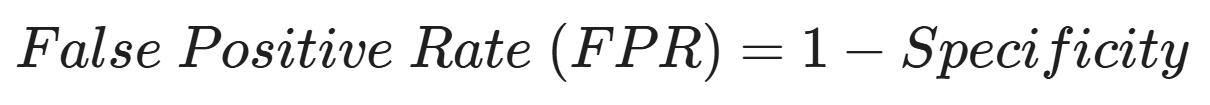
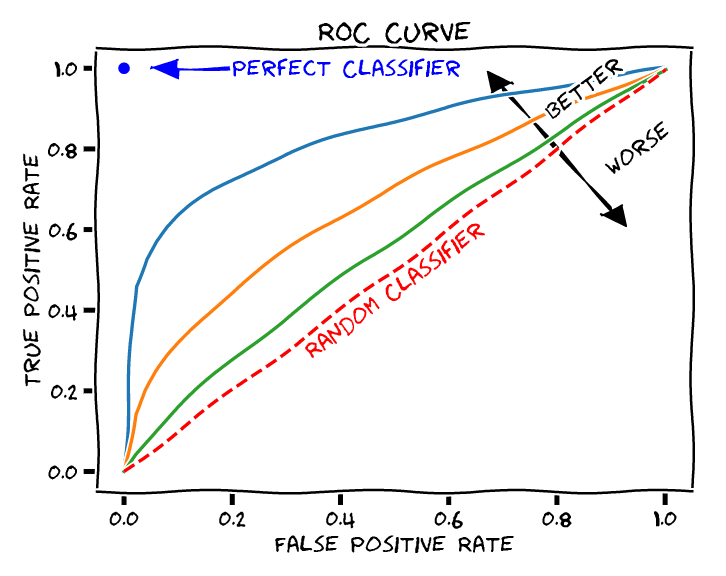


그림 15. TPR과 FPR 공식

폰트, 텍스트, 화이트, 라인이(가) 표시된 사진

자동 생성된 설명

그림 16. F1-Score 공식

 텍스트, 라인, 도표, 그래프이(가) 표시된 사진

자동 생성된 설명

그림 17. ROC Curve와 ROC AUC

F1-Score, ROC AUC는 데이터셋이 불균형적이어도 특정 데이터에 편향되지 않기 때문에 공정한 성능 평가 기법이 될 수 있기 때문에 두 지표를 중심으로 성능 평가를 진행하려고 계획했으나, F1-Score가 Classification Report를 통해 쉽게 확인할 수 있다는 장점이 있어 이 지표를 주로 사용하려고 한다. 추가적인 성능 향상 및 최적화를 위해 별도의 데이터 증강을 진행하여 각 클래스마다 데이터의 수를 동등하게 하거나, 계층적 샘플링을 통해 Train Set과 Validation Set, Test Set의 클래스 비율을 조정한다면 두 기법을 제외한 다른 평가 기법 (정확도 등) 또한 같이 활용하고자 한다.

**2-5.** 오류 분석 및 개선

최종적으로 선택한 Extra Trees 모델이 전경색, 배경색 분류 작업은 매우 잘 수행하는 것으로 확인된다. 따라서, 개선의 여지가 있는 숫자 분류 모델을 중점으로 오류 분석 및 개선 작업을 진행하고자 하였다. 숫자 분류에서 어떤 부분에서 오류가 있었는지 오류 행렬 (Confusion Matrix)를 통해 확인해보았다.

텍스트, 스크린샷, 사각형, 직사각형이(가) 표시된 사진

자동 생성된 설명

그림 18. 숫자 분류 작업 오류 행렬 I

오류 행렬을 확인한 결과, ‘4’를 ‘9’로, ‘5’를 ‘3’으로 잘못 분류한 비율이 높은 것으로 나타났다. 따라서, 4와 5에 대한 데이터를 추가로 학습시켜 분류 성능을 향상시키고자 하였다.

| **Final Score of Extra Tree Model** | | |
| --- | --- | --- |
| **Add Data** | **Accuracy** | **F1-Score** |
| **Before** | **0.9368** | **0.9367** |
| **After** | **0.9383** | **0.9382** |

표 22. 데이터 추가 학습 후 모델 성능 I

위 결과는 기존 Train Set에 데이터 증강을 통해 생성한 2,000개의 데이터를 추가로 학습한 결과이다. 잘못 분류한 비율이 높은 데이터를 추가로 학습시키니 약간의 성능 향상 (약 0.002)을 확인할 수 있었다. 하지만, 추가된 데이터로 인해 다른 곳에서 오분류 비율이 높아지는지 확인해보았다.

텍스트, 스크린샷, 사각형, 직사각형이(가) 표시된 사진

자동 생성된 설명

그림 19. 숫자 분류 작업 오류 행렬 II

오류 행렬을 확인한 결과, 데이터를 추가한 4와 5에 대한 잘 분류한 비율은 증가했지만, ‘9’의 잘못된 분류 데이터 비율이 상승하였다. 따라서, ‘9’에 대한 데이터 또한 추가하려고 했으나 이와 같은 상황이 반복될 것으로 생각하여 데이터 클래스 개수를 보다 균형적으로 맞추는 방법을 수행하고자 하였다. 클래스의 개수를 비슷하게 맞추어 데이터셋의 균형을 높여 성능을 향상시키고자 하였다. 샘플이 적은 클래스는 약 500개, 샘플이 많은 클래스는 추가하지 않거나 조금만 추가하여 학습을 진행하였다.

| **Final Score of Extra Tree Model** | | |
| --- | --- | --- |
| **Add Data** | **Accuracy** | **F1-Score** |
| **Before** | **0.9383** | **0.9382** |
| **After** | **0.9414** | **0.9413** |

표 23. 데이터 추가 학습 후 모델 성능 II

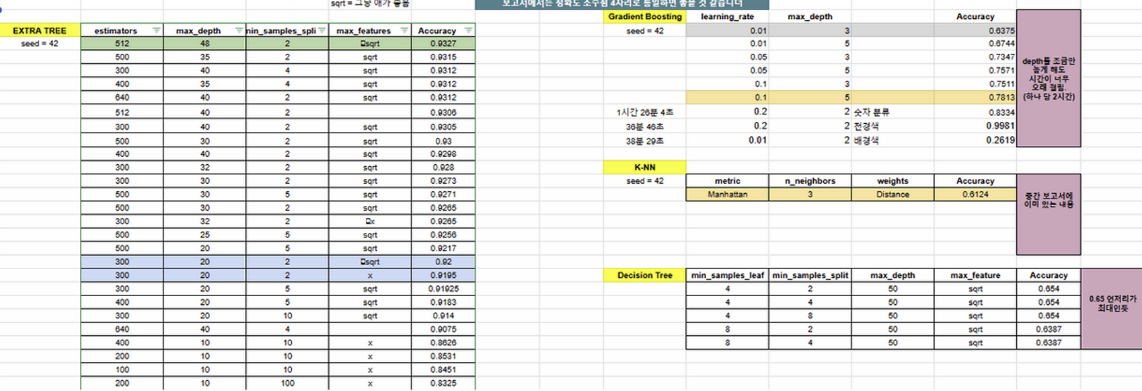
1차 오류 분석 시 사용한 Train Set에 각 클래스 별 증강 데이터 3,400개를 추가하여 학습에 사용하였다. 결과를 확인해보니 데이터셋의 균형을 맞춤으로써 성능 향상을 확인할 수 있었다. 데이터의 양이 많은 클래스에 모델이 맞춰지는 경향으로 인해 비슷하게 생긴 숫자의 오분류 비율이 높아졌던 것으로 파악할 수 있었다.

**3. Conclusion**

이번 머신 러닝 프로젝트를 통해 모델이 학습할 데이터를 전처리하고, 모델의 하이퍼파라미터를 탐색한 결과를 바탕으로 모델 간 비교 및 최종 모델 선정을 진행해보았으며, 최종 선정 모델의 오류를 분석하여 성능을 추가적으로 향상시켜 보았다. 중간 보고서에서 고려했던 사항 또한 충족시키고자 프로젝트 진행 과정에서 여러 작업을 수행하였다.

**하이퍼파라미터 탐색 과정 개선**

각자가 맡은 모델의 하이퍼파라미터를 직접 탐색해보며 최적의 성능을 찾고자 하였고, 결정한 최적의 파라미터를 바탕으로 모델 간 성능 비교를 진행하였다.

텍스트, 번호, 스크린샷, 평행이(가) 표시된 사진

자동 생성된 설명

그림 20. JWCA 팀의 하이퍼파라미터 탐색 과정

**데이터 증강, 계층적 샘플링 등을 통한 Dataset의 Balance 맞추기**

이번 오류 분석을 통해 성능을 향상시키는 과정에서 단순히 오분류된 클래스의 데이터만 추가할 경우, 다른 클래스의 오분류가 늘어나는 Trade-Off 현상이 발생하였고, 이로 인해 증강한 데이터를 이용하여 데이터셋의 균형을 맞추는 방식을 통해 성능을 향상시켰다.

**오류 분석을 통한 최종 모델 최적화 및 성능 향상**

오류 행렬을 통해 오분류된 데이터를 확인하고, 그에 맞춰 데이터 증강을 통해 획득한 데이터를 추가로 학습시켜 최종 선정 모델의 성능을 향상시켰다.

앞서 서술한 프로젝트 진행 현황 및 계획, 고려 사항 등을 통해 프로젝트의 목표인 Colored MNIST의 숫자-전경색-배경색 분류 작업 모델 생성을 성공적으로 달성하였고, 데이터 증강 및 머신 러닝 모델 개발의 전반적인 과정을 완성도 높게 체험할 수 있는 기회가 된 프로젝트였다.

**4. Appendix**

[**https://youtu.be/8TAz2YK6VOw**](https://youtu.be/8TAz2YK6VOw)

**발표 영상 YouTube Link**