딥러닝이론및응용 기말 Project

학번: 20217137

이름: 강슬기

**# 서약**

아래 보고서는 **본인의 힘만으로 작성**해야 하며, 다른 학생에게 질문과 다른 학생의 코드를 참고하는 행위는 모두 금지합니다

\* 수업에서 제공한 코드, 노트북은 모두 재활용 가능하며, 카피로 규정하지 않습니다.

\* 수업 자료 이외에 참고자료가 있다면, 출처와 사용 부분에 모두 표시하는 경우는 모두 합당한 자료로 인정하겠습니다.

\* 위에 대해서 모두 이해하고 동의했다면, 아래 `서약글`에 다음을 작성해주세요:

"본인은 위 서약글을 이해하고 동의하며, 프로젝트를 수행하는데 있어서 반칙을 할 경우 (제공자 포함) 본 프로젝트에 대한 점수가 반영되지 않는다는 것에 동의합니다."

학번: 20217137

이름: 강슬기

서약글: 본인은 위 서약글을 이해하고 동의하며, 프로젝트를 수행하는데 있어서 부정행위를 할 경우(제공자 포함)본 프로젝트에 대한 점수가 반영되지 않는다는 것에 동의합니다.

\*모든 코드에는 주석을 작성해 주세요

최종 제출시, 본 보고서와 .ipynb 노트북파일, test에 사용한 모델(.pt)파일을 압축해 제출해 주세요.

중요: 사용한 기법은 자신이 이해한 것 만을 사용하세요. 설명하지 않은 기법을 사용하면 그 부분을 제외하고 채점하겠습니다. 예를 들어서 자신의 힘으로 찾은 코드를 이용하려 하는 경우 내용을 이해하고 보고서에 이해한 내용이 충분히 설명이 되어야만 사용을 허용합니다.

Step 1: Dataset 준비하기

<코드 캡쳐 첨부>

# basecode 그대로 사용하였으며 경로만 Local path로 설정했습니다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

#basecode그대로 사용하였습니다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

Step 2: Dataset에 대한 Data Loaders 구성

<코드 캡쳐 첨부>

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 그래픽 디자인, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

데이터 전처리에 대한 설명

위 작성한 코드에 대해서 설명하세요

Dataset 부분은 기본으로 제공된 basecode를 그대로 사용하였습니다.

현재 데이터셋은 20,000장으로 test set까지 포함된 데이터셋이라 학습 데이터가 상대적으로 부족합니다. Data Augmentation은 적은 데이터셋 크기를 보완하고, 데이터 편향을 줄여주며, 다양한 데이터 패턴을 학습도록하여, 과적합을 방지할 수 있기 때문에 본 챌린지에서는 Data Augmentation이 많이 중요하다고 생각했습니다.

Albumentations 라이브러리는 처음 사용해 보는 것이기 때문에 albumentations API Reference (참고자료: <https://albumentations.ai/docs/api_reference/full_reference/> )에서 transform 기법을 하나하나 살펴보며 적용하였습니다.

적용한 주요 변환 기법은 다음과 같습니다 :

* Resize: 이미지를 128x128로 고정(데이터 크기와 모델 입력 크기로 맞추어서).
* HorizontalFlip: 50% 확률로 이미지를 좌우 반전하여 데이터 다양성 확보.
* RandomBrightnessContrast: 70% 확률로 이미지 밝기와 대비를 랜덤하게 변경.
* RandomResizedCrop: 이미지의 특정 영역을 랜덤하게 크롭적용.
* ShiftScaleRotate: 이미지를 30도 범위 내에서 회전, 이동 변환.
* CoarseDropout: 랜덤으로 이미지에 구멍을 뚫음

Transform의 확률(p)을 0.5, 0.7, 0.8 등으로 높게 설정하여 배치마다 증강이 높은 확률로 적용되도록 설정하였습니다. 데이터 증강기법을 통해서 모델이 적은 데이터셋에서도 과적합이 일어나지 않게끔 학습할 수 있도록 하였습니다.

Dataset에 대한 loaders 부분도 기본으로 제공된 basecode를 그대로 사용하여 구현하였습니다.

Train/Validation 데이터 분할은 8:2의 비율로 나누어 상대적으로 많은 데이터를 학습에 활용할 수 있도록 설정했습니다.

Batch Size는 16으로 설정하였습니다. 작은 배치 사이즈를 사용하면 모델이 더 자주 가중치 업데이트를 수행할 수 있어 빠른 수렴이 가능합니다. 하지만, 너무 작은 사이즈로 설정하면 학습 속도가 느려지거나, Overfitting의 위험성이 있습니다. 본 프로젝트를 진행할 때, 성능이 좋은 그래픽 카드를 사용하였기 때문에 배치 크기를 작게 설정하더라도 1 epoch 당 최대 3분을 넘기지 않아 학습 시간에는 큰 영향을 미치지 않았고, 데이터 증강 기법을 통해 다양한 데이터 변형을 적용하여 적은 배치 사이즈의 단점을 보완하면서도 일반화 성능을 높이는 전략을 사용했습니다.

Dataset loader에서 num\_workers 값을 0으로 설정하여 사용했습니다. Kaggle이나 Google Colab 환경에서는 num\_workers 값을 2나 최대 4까지 설정하면 데이터 로드 시간이 단축되어 효율적인 학습이 가능하였지만, local 환경에서는 num\_workers 값을 설정할 경우 train loop가 정상적으로 작동하지 않는 문제가 있었기에 num\_workers=0으로 설정하였습니다.

|  |  |  |
| --- | --- | --- |
| 텍스트, 중소형 고양이, 고양잇과, 고양이이(가) 표시된 사진  자동 생성된 설명 | 텍스트, 스크린샷, 중소형 고양이, 집고양이이(가) 표시된 사진  자동 생성된 설명 | 텍스트, 스크린샷, 중소형 고양이, 고양잇과이(가) 표시된 사진  자동 생성된 설명 |
| RandomResizedCrop  랜덤으로 이미지를 크롭하고 원래 128,128 크기로 조정합니다. | ShiftScaleRotate  이미지를 좌우/상하로 이동시키거나, 확대/축소하고, 회전을 시킬 수 있습니다. 이미지가 변형될 때 interpolation은 default값으로 선형 보간이 사용됩니다. | CoarseDropout  이미지에 랜덤으로 구멍(holes)을 뚫어 일부 픽셀을 제거합니다. |
| 텍스트, 중소형 고양이, 고양잇과, 고양이이(가) 표시된 사진  자동 생성된 설명 | 텍스트, 중소형 고양이, 고양이, 고양잇과이(가) 표시된 사진  자동 생성된 설명 | |
| HorizontalFlip  Y 축을 중심으로 수평으로 뒤집습니다. 이미지를 좌우반전시킵니다. | RandomBrightnessContrast  이미지의 밝기와 대비를 랜덤하게 변경합니다. | |

Step 3: Neural Network 생성

- Pretrained model을 허용하지 않습니다. (직접 모델을 설계해 주세요)

<코드 캡쳐 첨부>

Kaggle 챌린지에 제출한 최종 제출 모델 - 간단한 CNN 모델.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

설계한 모델을 출력 후 네트워크를 구성한 방법과 이유를 각 단계별로 설명  
최종 제출 모델 - 간단한 CNN 모델.

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

VGG, ResNet, EfficientNet, AlexNet과 같은 다양한 모델 구조를 참고하여 네트워크를 구성하고 학습을 진행했었습니다. 그 결과, 네트워크의 깊이가 더 깊어지고 무거워진다고 해서 성능이 반드시 비약적으로 증가하는 것은 아니라는 점을 확인할 수 있었습니다.

챌린지 마감일에는 단순한 CNN 구조를 기반으로 k-fold 교차 검증을 적용하여 학습을 진행했으며, Public Score에서 VGG, ResNet, EfficientNet, AlexNet보다 더 높은 점수를 기록했기에 최종 결과물로 단순 CNN 모델을 선택하게 되었습니다. 그러나 private score에서는 EfficientNet 모델이 가장 높은 score를 기록하였음을 덧붙입니다..

입력 : 이미지는 N(batch size), 3(channels, RGB), 128(height), 128(width) 로 들어가고, word 임베딩벡터는 N(batch size), 15, 300으로 들어가게 됩니다.

출력 : 포스터 이미지에 대해서 분류한 결과를 영화 장르 8개 클래스의 확률 값을 출력합니다.

**CNN 레이어 구성:**

Conv2d 레이어 5개와 maxpool 레이어 5개, 글로벌 평균 풀링 1개로 구성되어 있습니다.

3x3 커널 크기, stride=1, padding=1을 사용하여 feature map 크기를 유지하고(출력차원 유지), 2x2 크기 MaxPooling을 사용하여 다운샘플링, global avg pooling으로 특징을 1x1로 축소하여 FC 레이어에 전달하였습니다.

|  |  |  |  |
| --- | --- | --- | --- |
| **layer** | **레이어 구조** | **입출력 차원** | **설명** |
| conv1 | Conv2d(3, 32, 3, 1, 1) | 3x128x128 → 32x128x128 | 3x3 커널로 채널 32개 생성 |
| pool1 | MaxPool2d(2, 2) | 32x128x128 → 32x64x64 | 2x2 맥스풀링으로 크기 1/2 |
| conv2 | Conv2d(32, 64, 3, 1, 1) | 32x64x64 → 64x64x64 | 채널 수 64로 증가 |
| pool2 | MaxPool2d(2, 2) | 64x64x64 → 64x32x32 | 2x2 맥스풀링으로 크기 1/2 |
| conv3 | Conv2d(64, 64, 3, 1, 1) | 64x32x32 → 64x32x32 | 동일 채널 유지 |
| pool3 | MaxPool2d(2, 2) | 64x32x32 → 64x16x16 | 2x2 맥스풀링으로 크기 1/2 |
| conv4 | Conv2d(64, 128, 3, 1, 1) | 64x16x16 → 128x16x16 | 채널 수 128로 증가 |
| pool4 | MaxPool2d(2, 2) | 128x16x16 → 128x8x8 | 2x2 맥스풀링으로 크기 1/2 |
| conv5 | Conv2d(128, 128, 3, 1, 1) | 128x8x8 → 128x8x8 | 채널 수 유지 |
| pool5 | MaxPool2d(2, 2) | 128x8x8 → 128x4x4 | 2x2 맥스풀링으로 크기 1/2 |
| gap | AdaptiveAvgPool2d(1) | 128x4x4 → 128x1x1 | 글로벌 avg 풀링으로 특징 축소 |

**Word2Vec, Fully Connected 레이어 구성:**

Word2vec 텍스트 임베딩 벡터를 cnn을 통과하여 나온 특징과 결합할 수 있도록 128차원으로 변환합니다. Batch Normalization을 추가하였습니다.

손실 함수로는 sigomid와 BCELoss를 하나의 단이 클래스로 결합하는 BCEWithLogitsLoss()를 사용하였기 때문에, 네트워크의 마지막 출력 활성화 함수는 적용하지 않았습니다. (참고자료: <https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html>)

|  |  |  |  |
| --- | --- | --- | --- |
| **Layer** | **레이어 구조** | **입출력 차원** | **설명** |
| word2vec\_fc | Linear(15\*300, 128) | 15x300→ 128 | Word2Vec 임베딩 입력을 128차원으로 변환 |
|  | BatchNorm1d(128) | 128 → 128 | 배치 정규화 적용 |
|  | ReLU(inplace=True) | 128 → 128 | 비선형 활성화 함수 (ReLU) 적용 |
|  | Dropout(0.5) | 128 → 128 | 드롭아웃으로 과적합 방지 |
| fc | Linear(128+128, 128) | 256 → 128 | 이미지 특징과 Word2Vec 특징 결합 |
|  | BatchNorm1d(128) | 128 → 128 | 배치 정규화 적용 |
|  | ReLU(inplace=True) | 128 → 128 | activation func적용 (ReLU) |
|  | Dropout(0.5) | 128 → 128 | 드롭아웃으로 과적합 방지 |
| fc | Linear(128, 8) | 128 → 8 | 최종 출력: 8개의 클래스 확률 예측 |

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. CNN, pooling 레이어를 거친 image의 feature map인 x1을 FC레이어에 전달하기 위해 flatten 시켜 2차원 벡터로 만들었습니다. Dim=1을 기준으로 벡터를 펴기 때문에 batch size는 유지한채 나머지 차원들이 1차원 벡터로 변환됩니다.
2. w2v.view(w2v.size(0), -1) word2vec 또한 batch size인 w2v.size(0)를 남기고, 남은 모든 차원을 1차원으로 자동으로 계산하여 변환합니다. (참고자료: <https://pytorch.org/docs/stable/generated/torch.Tensor.view.html>)
3. 이미지 특징 벡터와 w2v 특징 벡터를 결합하여 batch size는 그대로 두고 나머지특징 벡터를 연결합니다.
4. 마지막 x = self.fc(x)를 통해 최종 출력을 반환되며 8개의 클래스에 대한 출력값이 반환됩니다. 손실함수로 BCEWithLogitsLoss를 사용하기 때문에 활성화 함수는 적용되지 않으며, 최종 출력값은 logits 형태로 나오게 됩니다.

Step 4: Cost (Loss) Function과 Optimizer 선택

<코드 캡쳐 첨부>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Optimizer와 Cost 함수를 선정한 이유와 선정하는데 중요하다고 생각하는 내용을 모두 작성합니다.

Optimizer로는 Adam을 선택했습니다. Adam은 파라미터마다 학습률을 자동으로 조정하며, 모멘텀도 자동으로 계산학습률을 조정하고, 기울기의 이동 평균과 제곱 이동 평균을 동시에 사용하는 Momentum과 RMSProp를 결합한 알고리즘으로 학습 속도가 빠르고, 사용하기 편하기 때문에 Adam을 사용하였습니다.

본 프로젝트는 영화의 포스터 이미지와 제목의 Word2Vec 데이터를 이용해 영화 장르를 예측하는 다중 레이블 분류 문제로, 각 레이블별이 독립적으로 0 또는 1의 확률을 예측해야 하기 때문에 Binary Cross-Entropy Loss 기반의 손실 함수를 사용해야 합니다.

처음엔 cost함수를 BCELoss()으로 설정하고, 네트워크의 마지막에 sigmoid 활성화 함수를 적용하지 않아 학습을 진행하여서 학습의 결과가 굉장히 이상하게 나왔었습니다. 정확률이 0.0이었고, loss는 0.68 수준으로 굉장히 높게 나와 문제가 있다는 것을 확인했습니다.

Pytorch 공식 문서의 Loss 함수들의 설명을 찾아보니, BCELoss는 네트워크에서 sigmoid를 적용하여 사용해야하고, BCEWithLogitsLoss는 sigmoid와 BCELoss를 결합한 형태로, 수치적으로 더 안정적이며 Sigmoid를 따로 적용할 필요가 없다는 점을 확인하여 BCEWithLogitsLoss를 사용하니 학습이 정상적으로 이루어졌습니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

(참고자료: <https://pytorch.org/docs/stable/nn.html#loss-functions>)

Step 5: 구성한 모델에 대한 Train and Validate 진행

<코드 캡쳐 첨부>

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

Train과 validation code에서 image와 word2vector, label들을 device(CUDA)로 전송한 후 모델을 통해 예측값을 생성합니다. 데이터 loader를 통해 데이터를 배치단위로 로딩합니다.

**순전파(forward pass)**에서는 모델에 images와 w2v를 입력해 예측값(outputs)을 생성하고, 예측값과 정답 레이블을 비교하여 loss를 계산합니다.

**역전파(backward pass)**에서는 optimizer.zero\_grad()로 이전 학습 배치의 기울기를 초기화하고, loss.backward()로 역전파를 통해 기울기를 계산하여 아담 optimizer.step()으로 가중치를 업데이트합니다. validation에서는 기울기를 with torch.no\_grad()으로 계산하지 않습니다.

모델의 예측값을 Sigmoid를 사용해 확률값으로 변환하여 예측값(all\_pred)과 실제 레이블(all\_label)을 cpu로 보내 저장합니다. 전체 loss를 배치 개수로 나누어 평균 손실을 return합니다.

Step 6: CNN model training/validation 분석

<코드 캡쳐 첨부>

최종 train loop

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

최종적으로 k-fold를 적용하여 학습을 진행하였습니다. 멀티 라벨 문제이기 때문에, MultilabelStratifiedKFold 라이브러리를 이용해서 사용하여 8개의 fold로 교차 검증을 하였습니다. iterative-stratification 는 데이터셋을 8개의 fold로 나누어 각 fold에 비슷한 레이블 비율을 유지한다는 장점도 있습니다. (참고자료: <https://cvml.tistory.com/25> , <https://github.com/trent-b/iterative-stratification> )

각 장르별로 One Hot Encoding하여 이진 벡터로 변환하였습니다. MultilabelStratifiedKFold는 입력 레이블 데이터가 이진 벡터 방식이여야 하기 때문에, 문자열 형식으로 저장되어 있는 train\_df[’genres’]를 0 1 0 과 같은 형식으로 이진벡터로 변환하였습니다..

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

Fold당 epoch를 돌도록 설정 하였기 때문에, epoch 수는 너무 크지 않게 10으로 설정 했습니다. 마지막 그래프 출력을 위한 train과 val loss, acc 변수를 선언하여 epoch마다 정확률과 손실을 저장하도록 했습니다. Kf.split으로 fold마다 학습/검증 데이터를 분할하도록 데이터 로더를 학습루프 안에서 구현했습니다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

Best\_val\_loss는 무한대로 설정을 해서 학습이 시작되고 최저 loss를 기록할때마다 저장되도록 구현하였습니다. 정확률 또한 0.0으로 설정하여 학습이 시작되고 최고 acc를 갱신하면 저장되도록 구현하였습니다. Early stopping 을 설정하여 3epoch동안 loss 갱신이 없으면 조기 종료 시킵니다. 본 학습 루프에서는 조기 종료되면 다음 fold가 실행됩니다. Lr 스케줄러 또한 설정하여 5epoch동안 손실이 개선되지 않을 경우 학습률을 0.5배씩 감소시킵니다. Train과 validaion에서 임계값은 0.5와 0.4로 테스트해본 결과.4로 설정했을 때 Kaggle score가 더 높았기에 0.4로 설정하였습니다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

val\_loss, val\_acc를 갱신할때마다 best loss 모델을 저장하도록 했고, 나중에 혹시 추가 학습을 진행하게 되는 상황을 위해서 model\_state\_dict()와 optimizer.state\_dict()를 모델도 같이 저장하였습니다. 최저 loss가 갱신되면 early stopping 카운팅이 0으로 초기화 되고, 아닐경우엔 +1씩 증가하게 됩니다. 학습이 모두 끝난 후에는 저장되었던 모든 epoch별 train, val Loss값과 Acc값을 그래프로 시각화 합니다.

위에서 수행한 training + validation 과정을 설명하세요

training loss와 validation loss 그래프를 통해서 분석

Ex) hyper-parameter, model을 변경하면서 성능 개선한 과정을 최대한 설명하세요

overfitting, underfitting 분석 등..

<서술형>

(1번째 학습시도)첫 학습 시도 네트워크 모델– 실습에서 진행했던 VGG를 참고하여 간단하게 구성한 네트워크

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명(1번째 학습시도)첫 학습 시도 네트워크 모델– 실습에서 진행했던 VGG를 참고하여 간단하게 구성한 네트워크

딥러닝 실습과제에서 사용했던 VGG 모델의 구조를 참고하여 첫 네트워크 구성은 conv3개로 간단하게 구성하여 학습이 되는지 시도해봤습니다. Loss 함수로는 BCELoss를 사용하여 하지만 첫 시도에서는 sigmoid를 사용하지 않아 학습이 제대로 이루어지지 않아 acc가 0%를 기록하여 그래프가 바닥을 찍는 수준이었지만, cost 함수로 BCEWithLogitsLoss를 사용하고 나서는 학습이 제대로 이루어지고, acc가 어느정도 상승하는 것을 확인하였습니다.

|  |  |
| --- | --- |
| **Loss 그래프: 0.68 수준**  **텍스트, 라인, 그래프, 스크린샷이(가) 표시된 사진  자동 생성된 설명** | **Accuracy 그래프: train, val 모두 정확률이 0**  **텍스트, 스크린샷, 라인, 그래프이(가) 표시된 사진  자동 생성된 설명** |
| **Loss 그래프: 0.45 대**  **라인, 그래프, 도표, 경사이(가) 표시된 사진  자동 생성된 설명** | **Accuracy그래프: 0.2이상으로 학습이 진행됨.**  **라인, 텍스트, 그래프, 도표이(가) 표시된 사진  자동 생성된 설명** |

(2 번째 학습시도) 첫 학습 네트워크 구조에 batch Normalization 추가하고 레이어를 확장한 모델

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 메뉴이(가) 표시된 사진

자동 생성된 설명(2 번째 학습시도) 첫 학습 네트워크 구조에 batch Normalization 추가하고 레이어를 확장한 모델

라인, 텍스트, 그래프, 도표이(가) 표시된 사진

자동 생성된 설명처음 학습을 시도했던 네트워크는 간소화된 VGG 구조를 기반으로 하였으며, Conv2D 레이어 3개와 MaxPooling 레이어 3개로 구성하여 구현한 모델이 정상적으로 학습되는지 확인하기 위한 목적으로 간단하게 만들었습니다. 첫번째 네트워크로 학습이 정상적으로 진행된다는 것을 확인하였고, 이후 학습 성능을 향상시키기 위해 두 번째 네트워크에서는 Batch Normalization을 추가하고, cnn 레이어를 더 추가하여 구현하였습니다. 이미지의 특징을 더 세밀하게 학습시키기 위해 Conv2D 레이어는 13개, MaxPooling 레이어는 5개로 늘렸습니다. 모든 Conv2D 레이어 뒤에 Batch Normalization을 추가하였습니다. Word2Vec의 특징을 더 많이 학습시키기 위해 FC 레이어를 512 → 최종 출력 구조에서 512 → 1024 → 최종 출력 구조로 확장하였습니다. 또한, 과적합 방지를 위해 드롭아웃 비율을 기존 0.5에서 0.6으로 조금 높였습니다.

라인, 그래프, 도표, 텍스트이(가) 표시된 사진

자동 생성된 설명

1번째 네트워크 구조로 학습을 진행 하였을때는 , **loss 0.45수준, Acc가 0.19 수준**으로 학습이 진행됨만 확인 했었습니다. 2번 째 네트워크로 **(VGG에 batch normalization추가)** 학습을 했을때는, **Loss 0.45 수준으로 정체, Acc가 0.82수준**으로 **정확률이 0.19 에서 0.82 로 크게 개선**된 것을 알 수 있습니다. 하지만 train과 validation 그래프 간의 격차가 크고 검증 손실과 정확률이 정체된 것을 보면 모델이 훈련 데이터에 과적합 된것으로 판단됩니다.

Kaggle score 비교분석:

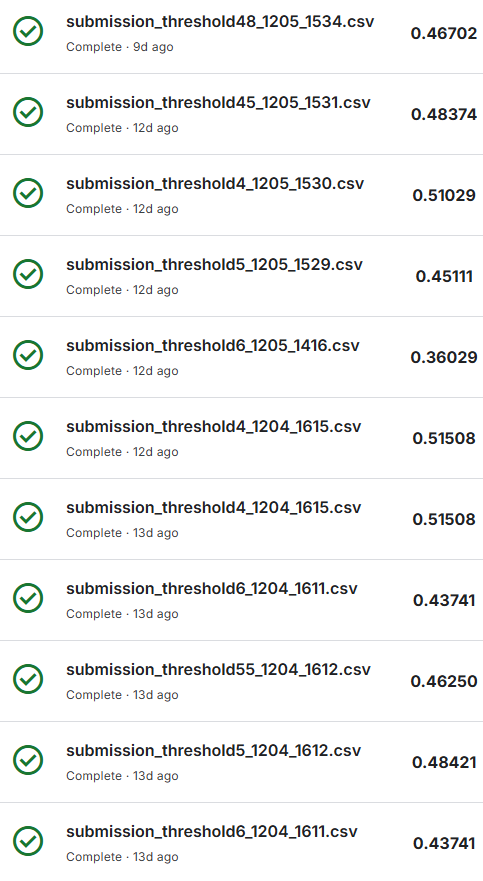
텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명첫번째 네트워크보다 batch normalization을 추가하고 레이어를 늘린 네트워크의 score가 높은 것을 보면 batch normalization이 학습을 안정화 시키고 성능 개선에 큰 역할을 한다는 것을 확인했습니다.

(3 번째 학습시도) VGG + batch normalization 네트워크의 모델을 학습할 때 임계값을 조정해가며 테스트

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

모델의 예측 결과에서 threshold(임계값)을 변경하여 테스트 해보았습니다.

VGG + batch normalization 네트워크의 결과에서 test할 때의 임계값을 0.4부터 0.6까지 다양하게 설정해가며 최적의 임계값을 찾으려고 시도해봤습니다. 0.4, 0.5, 0.55, 0.6으로 바꾸어가며 실험해본결과 test에서 0.4일 때 score가 가장 높은 것을 확인하였고, 이후 학습에서 네트워크 구조를 변경할 때마다 임계값을 0.4또는 0.5로 바꾸어가며 score를 높이려고 노력하였습니다.

(4 번째 학습시도) resnet 기반 네트워크사용 ( 참고자료: <https://roytravel.tistory.com/339>,

<https://velog.io/@euisuk-chung/%ED%8C%8C%EC%9D%B4%ED%86%A0%EC%B9%98-%ED%8C%8C%EC%9D%B4%ED%86%A0%EC%B9%98%EB%A1%9C-CNN-%EB%AA%A8%EB%8D%B8%EC%9D%84-%EA%B5%AC%ED%98%84%ED%95%B4%EB%B3%B4%EC%9E%90-ResNet%ED%8E%B8> ,

<https://phil-baek.tistory.com/entry/ResNet-Deep-Residual-Learning-for-Image-Recognition-%EB%85%BC%EB%AC%B8-%EB%A6%AC%EB%B7%B0>)

텍스트, 스크린샷, 소프트웨어, 운영 체제이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 도표이(가) 표시된 사진

자동 생성된 설명텍스트, 도표, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 문서이(가) 표시된 사진

자동 생성된 설명위의 resnet 논문을 설명한 블로그의 글을 참고하여BasicBlock 클래스를 만들어 ResNet의 Residual Learning을 구현해보았습니다. Conv+bn+relu, conv+bn, shortcut(스킵커넥션)을 구현하였습니다. 입력 데이터츨력과 더해 잔차(residual) 학습을 하도록 합니다. 만약 입력과 출력의 차원이 다르다면 1x1 conv를 통해 차원을 맞춘 뒤에 더하고 bn을 적용합니다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

ResNet 네트워크는 여러 개의 BasicBlock을 쌓아서 레이어를 만드는 구조이기 때문에 \_make\_layer 함수를 만들어 BasicBlock을 여러 개 연결하여 하나의layer를 생성하도록 하였습니다. BasicBlock 구조를 만들어서 nn.Sequential으로 묶어서 하나의 layer가 되게 반환합니다.

텍스트, 스크린샷, 폰트, 문서이(가) 표시된 사진

자동 생성된 설명VGG는 단순하게 레이어를 깊게 쌓은 구조로 인해 파라미터 수가 많고 계산량이 크지만, ResNet모델의 구조는 스킵 커넥션으로 적은 파라미터로 좋은 성능을 낼 수 있고, VGG는 네트워크가 깊어질수록 과적합이 발생하는 문제가 있지만, ResNet은 스킵 커넥션으로 기울기 소실(Gradient Vanishing)문제를 개선하였기 때문에 깊어질수록 성능이 더 좋아지고, VGG에 비해 학습시간이 줄어든다는 장점이 있기에 RenNet 모델을 참고하여 네트워크를 적용해 보았습니다.

텍스트, 라인, 그래프, 도표이(가) 표시된 사진

자동 생성된 설명

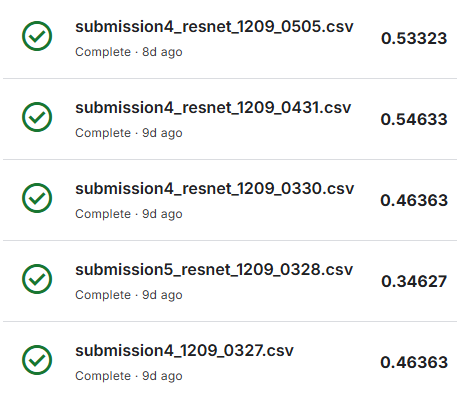
텍스트, 라인, 도표, 그래프이(가) 표시된 사진

자동 생성된 설명

ResNet 네트워크로 변경하여 학습을 진행한 결과, 검증 Loss는 약 0.38~0.39 수준에서 정체되며, Acc는 0.82 수준에서 학습 내내 정체되었습니다. 학습은 30 에포크로 설정되었지만 Early Stopping에 의해 24 에포크에서 학습이 중단되었습니다.

이전 VGG 네트워크와 비교했을 때, 검증 Loss와 Accuracy의 최종 값이 큰 차이 없이 과적합이발생한 것도 비슷하게 나타났습니다. 이는 ResNet이 더 깊은 구조를 가지고 있음에도 불구하고 성능 면에서는 VGG와 비슷한 결과를 내고 있습니다.

학습이 완료된 모델로 test를 측정할 때 임계값을 0.4와 0.5로 바꾸어보고, 네트워크에 dropout을 더 강하게 주고, 데이터 증강 기법에서 transform의 확률을 높게 설정, learning rate를 0.0001, 0.00001로 조절하여 학습을 해보았지만 loss는 0.4수준으로 정체되고, acc또한 0.81 수준에서 정체되었습니다. 결과는 대부분 비슷했습니다. Kaggle에서 score를 측정한 결과 데이터 증강을 조절한 모델의 임계값이 0.4일 때 score가 가장 높았음을 확인할 수 있었습니다.



(5 번째 학습시도) resnet 기반 네트워크에 LSTM추가 사용

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 라인, 그래프, 도표이(가) 표시된 사진

자동 생성된 설명ResNet 기반의 네트워크 구조에 LSTM을 추가하여 학습을 해보았습니다.

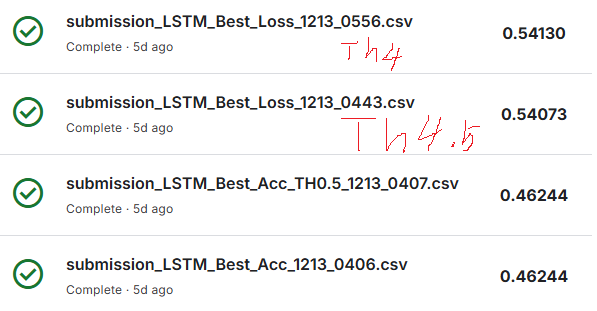
LSTM을텍스트, 라인, 그래프, 도표이(가) 표시된 사진

자동 생성된 설명 추가하여 학습한 결과 14 epoch에서 학습이 조기종료 되었고, loss는 0.42 수준, Acc는 0.81 수준으로 이전의 학습결과와 비슷하게 보였습니다.

텍스트, 라인, 그래프, 도표이(가) 표시된 사진

자동 생성된 설명

LSTM을 추가하여 학습한 모델의 결과 또한 이전 resnet 모델보다 더 못하거나 비슷한 수준의 결과를 보였습니다.



(4 번째 학습시도) EfficinetNet기반 네트워크

(참고 자료 : <https://lynnshin.tistory.com/53>, <https://velog.io/@krec7748/Pytorch-EfficientNet-%EA%B5%AC%ED%98%84> )

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 도표, 다채로움이(가) 표시된 사진

자동 생성된 설명 EfficientNet은 MBConvBlock 모듈을 사용해 네트워크를 효율적으로 설계하여 성능을 높인 모델입니다. 확장 비율(Expand Ratio)과 Squeeze-and-Excitation(SE) 블록을 통해 모델의 성능을 높이면서도 계산 비용을 줄여 효율적으로 네트워크를 확장 모델입니다.

MBConvBlock 블록을 만들어 입력 데이터를 처리하며 확장(Expand), Depthwise Convolution, SE 블록, Pointwise Convolution)을 수행하게 됩니다. 이를 통해서 효율적으로 특징을 학습하고, 스킵 커넥션을 추가하였습니다. 입력채널×expand 비율을 통해 채널을 확장하고, 각 채널에 대해 독립적으로 conv를 수행하여서 계산량을 줄입니다.(Depthwise Convolution), 채널별로 중요도를 학습해서 채널간 가중치를 조정하게 됩니다.(Squeeze-and-Excitation (SE)) self.project 에서 out\_channels로 차원을 줄여서 최종 출력을 생성하게 됩니다.(Pointwise Convolution)

텍스트, 스크린샷, 라인, 도표이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

EfficnetNet – B0의 모델 구조를 참고하여 네트워크를 구현하였습니다. 초기에 입력 이미지를 처리하는 stem단계에서 3x128x128 -> 32x64x64로 크기를 줄이고 채널수를 확장하고, 7개의 MBConvBlock의 레이어로 각 expand\_ratio 만큼 확장되게끔 구현했습니다.

텍스트, 라인, 그래프, 도표이(가) 표시된 사진

자동 생성된 설명라인, 그래프, 도표, 텍스트이(가) 표시된 사진

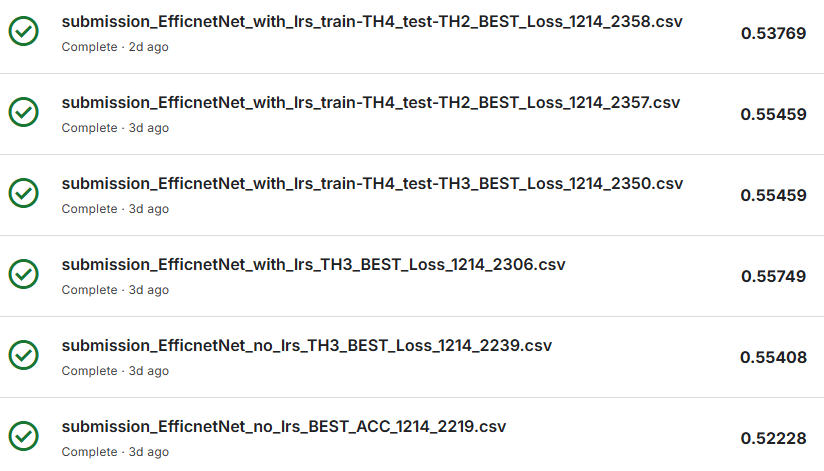
자동 생성된 설명

텍스트, 라인, 그래프, 도표이(가) 표시된 사진

자동 생성된 설명텍스트, 라인, 그래프, 도표이(가) 표시된 사진

자동 생성된 설명 Learning rate scheduler를 추가하지 않고 학습한 EfficinetNet 구조의 네트워크는 loss 0.37을 기록, Acc 0.828을 기록하고 8epoch 에서 조기 종료 되었습니다.

Learning rate scheduler를 추가하여 학습한 EfficinetNet 구조의 네트워크는 loss 0.369를 기록, Acc 0.842을 기록하고 23epoch 에서 조기 종료 되었습니다.



efficientNet 네트워크로 학습할 결과는 score가 0.52 이상을 기록하고 있어 지금까지 실험한 네트워크 중에서는 가장 높은 점수를 기록하였습니다. 또한

Learning Rate Scheduler의 적용하고 나서의 loss와 acc가 살짝 더 좋아지고, score가 높은 것을 보면 오랜 epoch동한 학습하게 되면 Learning Rate Scheduler를 적용하는 것이 성능 개선의 측면에 유의미한 역할을 한다고 생각합니다.

최종 제출 모델 - 간단한 CNN 모델.

텍스트, 라인, 그래프, 도표이(가) 표시된 사진

자동 생성된 설명텍스트, 그래프, 라인, 스크린샷이(가) 표시된 사진

자동 생성된 설명

최종으로 채택된 모델은 K-Fold 교차검증(8 fold)를 수행하였고, 각 폴드별로 10 epoch씩 수행되며 3epoch 동안 loss개선이 없을 경우 조기종료 되도록 설정하였습니다.

Optimizer은 Adam, Learning rate는 0.0001, weight\_decay는 1e-5, batch size는 50, Loss Function은 BCEWithLogitsLoss, Learning Rate Scheduler는 ReduceLROnPlateau (patience=5, factor=0.5)으로 설정하였고, Early stopping은 3 epoch로 설정했습니다.

학습 결과를 살펴보면 첫번째 fold동안은 loss가 0.684, acc 0.29 부터 시작하여 점차 학습을 진행하여 최저 loss 0.3329, 최고 acc 0.8453을 기록하여 k-fold를 적용하여 학습한 결과 손실은 빠르게 줄어들고 정확률은 loss가 줄어드는 만큼 올라가는 것을 확인할 수 있습니다. 초기 모델 학습단계에서는 loss가 빠르게 감소하면서 underfitting 문제는 나타나지 않았고, 40 epoch 이후 학습과 검증 손실값이 비슷하게 수렴하는 경향이 있었지만 early stopping 을 적용하여 과적합을 방지하였습니다.

Step 7: Predict with Test Data

<코드 캡쳐 첨부>

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

Step 8: Training Techniques

성능 개선을 위해서 사용한 기법 중에서 특별히 효과적이었던 부분이나 강조하고자 하는 내용을 작성해주세요.

텍스트, 폰트, 라인, 번호이(가) 표시된 사진

자동 생성된 설명

모델 비교하며 분석을 진행한 결과, VGG, ResNet, EfficientNet과 같은 깊은 네트워크를 사용했을 때보다, K-Fold를 적용한 단순화된 네트워크가 더 높은 성능을 보였습니다. 이는 네트워크의 레이어를 깊게 쌓아 복잡한 특징을 학습시키는 전략보다, K-Fold 교차 검증을 통해 데이터의 활용도를 극대화하며 모델을 학습시키는 방법이 더 효과적일 수 있다는 것을 보여줍니다.

결론적으로, 단순한 구조의 모델이 K-Fold 교차 검증을 통해 안정적이고 일관된 성능을 보였으며, 이런 전략이 과적합 방지와 성능 최적화에 기여하였음을 확인할 수 있었습니다.

마지막 소감: 영상처리와 딥러닝 수업에서도 기말 프로젝트로 Kaggle 챌린지를 진행했던 터라, 이번 딥러닝 이론 및 응용 수업의 챌린지에 온 힘을 다 쏟지 못한 점이 정말 아쉽습니다. 챌린지가 마감된 후에야 적용하지 못했던 기법들이 하나둘씩 떠올라서 "왜 그때 적용하지 않았을까?" 하는 후회도 많이 들었고요. 하지만 정말 재미있었습니다. Pre-trained 모델을 사용하지 않고 순수하게 모두가 같은 조건에서 경쟁했던 것이 더 큰 의미를 주었고, 덕분에 혼자 loss 값과 씨름하면서 이를 줄이기 위해 여러 방법을 시도하는 과정이 굉장히 즐거웠습니다. 수업에서 배웠던 기법들을 하나씩 적용할 때마다 눈에 띄게 loss 값이 줄어들고, Kaggle 스코어가 올라가는 순간마다 제 순위 변동이 바로 보이니 경쟁심이 더 불타올랐습니다. 제출 기회가 하루 5번으로 제한되어 있었던 것도 감질나면서도 흥미로웠습니다. 임계값 조정, 다양한 네트워크 모델 시도, 배치 정규화(Batch Normalization), K-Fold 교차 검증 등 여러 기법들을 적용하면서 직접 비교 테스트를 진행했던 것이 매우 유익했지만, 끝내 모든 아이디어를 다 시도해보지 못한 게 아쉬웠습니다.마지막에는 혼동 행렬 그래프와 F1 Score 계산을 완전히 잊고 있다가 대회 종료 직전에야 코드가 더 있다는 것을 깨달았는데, 그때는 이미 제출 기회를 소모한 상태라 허무하기도 하고 아쉬움이 많이 남았습니다.

졸업을 앞두고 있어서 다음 학기 수업에 참여할 수는 없지만, 정말 Kaggle 챌린지만 따로 참여할 수 있는 기회가 있다면 좋겠다는 생각이 듭니다. 이번 챌린지는 부족함도 많았지만, 그만큼 배움과 성장이 있었던 즐거운 경험이었습니다.