모델 개발 매뉴얼

팀명 : 아덱폭주족

1. 학습 배포를 위한 SW 및 HW

- **SW :** Windows 사양 : Windows10 Pro
  시스템 종류 : 64비트 운영 체제
  응용프로그램 : Visual Studio Code, JupyterNotebook , stable diffusion web ui
- **HW :** GPU : NVDIA GeForce RTX 4070 Ti
  프로세서 : 13th Gen Intel (R) Core(TM) i9-13900K 3.00 GHz
  RAM: 32.0GB
  시스템 종류 : x64 기반 프로세서

2. 파일 저장 구조

- 전처리한 데이터셋 종류와 이름

1. 모델 학습용 train data : 제공 데이터 + 상하&좌우 반전 + 회전 + noise(diffusion) + padding (증강)
 코드에서 저장위치
 : '/content/drive/MyDrive/공모전/CJ 미래기술챌린지가 아니고 KISTI/신승희/train test/split/train'
 backend.AI 기준 저장 위치 : seoul1:L3.train

1.1 중복 처리 알고리즘 검증 및 학습용 validation data
 코드에서 저장 위치
 : '/content/drive/MyDrive/공모전/CJ 미래기술챌린지가 아니고 KISTI/신승희/train test/split/validation'
 backend.AI 기준 저장 위치 seoul1:L3.validation

2. 실제 모델 검증용 test data : 기존 제공 데이터를 사용
 코드상: 저장 위치 : C://Users/User/Desktop/min/DATA/delamination
 backend.AI 기준 저장 위치 seoul1:L3.test

- 데이터 분류 학습 모델 종류와 모델 이름

학습에 사용한 모델 종류: Tensorflow.keras에 있는 이미지 분류 모델 중 efficientNet B0 사용
최종적으로 학습시킨 모델 : C:/Users/User/Desktop/min/model/231003effi.h5

3. 모델 실행 방법

- EfficientNet 모델 & 모델 정확도 확인

EfficientNet 모델

1. 환경 설정 및 데이터 로드
   필요한 라이브러리 및 모듈을 가져오고, GPU를 사용할 수 있는지 확인
   Train 및 Test 데이터 디렉토리 경로를 지정하고 이미지 크기를 설정
   Train 및 Test 데이터를 불러와서 이미지와 레이블을 준비
2. 데이터 전처리
   'train_dir'와 'test_dir' 경로를 설정하여 훈련 데이터 및 테스트 데이터의 디렉토리 경로를 지정
   'load_img' 및 'img_to_array' 함수를 사용하여 이미지 파일을 로드하고 Numpy 배열로 변환

LabelEncoder를 사용하여 클래스 레이블을 정수로 변환하고 훈련 데이터를 섞기
Multi-label 분류를 위해 레이블을 One-Hot 인코딩
Train 데이터를 학습과 검증 세트로 나눠줍니다.

3. 모델 생성
EfficientNetB0를 기본 모델로 불러오기
모델의 출력 레이어를 수정하여 멀티 레이블 분류를 수행하도록 설정
모델 컴파일: 옵티마이저는 Adam을 사용하고, 손실 함수는 binary cross-entropy를 사용
4. 모델 학습
모델을 학습시키며, LossCallback과 EarlyStopping을 사용하여 학습 진행 상황을 모니터링하고 조기 종료
5. 테스트 데이터 예측
테스트 데이터에 대한 예측을 수행
6. 특정 클래스 및 임계값 기반 예측 수정
'classes_to_check' 변수에 특정 클래스 이름을 설정, 이 클래스들에 대한 예측이 임계값을 초과하면 해당 예측을 유지
각 이미지에 대해 다음을 수행
특정 클래스들 중 하나라도 임계값을 초과하는지 확인
만약 어떤 특정 클래스들의 예측이 임계값을 초과하면, 원래 예측을 유지
특정 클래스들의 예측이 임계값 이하인 경우, 'goodpcb' 클래스로 예측
일부 클래스는 임계값을 초과하고 일부 클래스는 미만인 경우, 필요에 따라 로직을 조정
7. 최종 예측 변환
최종 예측을 NumPy 배열로 변환

- 모델 정확도 확인

1. 클래스별 정확도 계산
'test_labels_encoded'와 'final_predictions'를 사용하여 각 클래스(레이블)별 정확도를 계산
클래스별로 실제 레이블 ('class_y_true')과 예측 레이블 ('class_y_pred')을 가져와, 임계값을 기준으로 예측과 실제 레이블을 비교하여 정확도를 계산
계산된 정확도를 'class_accuracies' 리스트에 추가
2. 클래스별 정확도 출력
각 클래스의 정확도를 출력, 클래스 이름과 해당 클래스의 예측 정확도를 표시
3. 랜덤 샘플로 예측 결과 출력
'random.sample' 함수를 사용하여 10개의 랜덤한 테스트 이미지를 선택
각 이미지에 대해 다음을 수행
실제 레이블('actual_label')을 가져오고, 임계값을 초과하는 예측 레이블('predicted_label') 찾기
예측 확률값('prediction_probs')을 가져오기
이 정보를 출력
4. Loss 및 Validation Loss 그래프 그리기
학습 중에 발생한 손실('loss')과 검증 중에 발생한 손실('val_loss')을 에폭('epoch')별로 저장한 정보를 사용하여 Loss 그래프를 그리기
이를 위해 Matplotlib을 사용하여 Loss와 Validation Loss를 시각화

- 중복 처리 알고리즘 & 예측 결과 확인 및 저장

중복 처리 알고리즘

1. 'predicted_classes' 리스트 초기화
   예측된 클래스 레이블을 저장할 빈 리스트인 'predicted_classes'를 초기화
2. 테스트 이미지에 대한 예측 수행
   모델을 사용하여 테스트 이미지에 대한 예측을 수행하고 결과를 'predictions_raw'에 저장
3. 임계값(threshold) 설정
   임계값을 'threshold' 변수에 설정, 임계값을 클래스 레이블을 업데이트할 때 사용
4. 정상 클래스 및 불량 클래스 목록 설정
   'classes_to_check' 변수에는 업데이트할 클래스의 목록이 포함 ('delamination','popcorn','scratch')
5. 최종 예측을 위한 새로운 배열 ('final_predictions') 생성
   빈 리스트인 'final_predictions'를 초기화하여 최종 예측을 저장할 배열을 생성
6. 각 테스트 이미지에 대한 처리
   'predictions_raw'에서 각 이미지의 예측을 가져와서 처리
   먼저 특정 클래스들 중 하나라도 임계값을 초과하면 해당 예측을 업데이트
      'updated_prediction' 변수를 현재 예측 ('predictions_raw[i]')의 복사본을 생성
      'goodpcb' 클래스의 확률을 0으로 하여 'goodpcb'가 아닌 것으로 설정
      업데이트된 예측 ('updated_prediction')을 'final_prediction'리스트에 추가
   만약 모든 특정 클래스의 예측이 임계값 이하인 경우
      'updated_prediction' 변수는 모든 요소가 0인 배열로 초기화
      'goodpcb' 클래스의 확률을 1로 하여 'goodpcb'인 것으로 설정
      업데이트된 예측 ('updated_prediction')을 'final_prediction'리스트에 추가
   위의 두 조건을 만족하지 않으면 현재 예측을 그대로 'final_predictions' 리스트에 추가
7. 최종 예측을 Numpy 배열로 변환
   모든 테스트 이미지에 대한 업데이트된 예측('final_predictions')을 Numpy 배열로 변환

예측 결과 확인 및 저장

1. 'sklearn.metrics'에서 'accuracy_score'를 가져오기
   모델의 예측 정확도를 계산하기 위해 'sklearn.metrics' module에서 'accuracy_score' 함수 불러오기
2. 모델의 예측 가져오기
   'model.predict(test_images)'를 사용하여 모델이 테스트 이미지에 대한 예측을 수행하고
   결과를 'prediction'에 저장
3. 임계값(threshold) 설정
   예측값을 이진 분류로 변환하는데 사용하고 일반적으로 0.5를 사용
4. 예측값을 이진 분류로 변환
   이진 분류로 변환은 각 예측이 임계값보다 큰지 여부를 확인하여 이루어지고 결과적으로 0 또는 1로 변환
   'binary_predictions' 변수에 이진 분류로 변환된 예측값을 저장
5. 각 클래스 별로 정확도 계산
   각 클래스의 정확도를 계산하기 위해 class_accuracies 빈 리스트를 초기화
   'enumerate(le.classes_)'를 사용하여 클래스 이름과 해당 인덱스를 가져오기
   클래스별로 다음을 수행

해당 클래스에 대한 예측과 실제 레이블을 가져오기
'accuracy_score' 함수를 사용하여 클래스별 정확도를 계산
클래스 이름과 정확도를 class_accuracies 리스트에 추가

6. 결과 출력
각 클래스에 대한 정확도를 출력, 이는 클래스명과 해당 클래스의 예측 정확도를 표시
7. 모델 저장

- Test

1. 새로운 테스트 이미지 폴더 경로 정의
'new_test_dir' 변수에 검증에 사용하고자 하는 기본 제공 데이터가 저장된 디렉토리 경로를 지정
2. 새로운 테스트 데이터 로드
'new_test_files' : 새로운 테스트 이미지 파일의 경로를 저장하는 빈 리스트를 초기화
'new_test_labels' : 각 이미지의 클래스 레이블을 저장하는 빈 리스트를 초기화
'os.walk()'를 사용하여 'new_test_dir'에서 모든 이미지 파일을 찾고, .png 확장자를 가진 파일만 선택
이미지 파일의 경로를 'new_test_files'에 추가하고, 파일의 경로에서 레이블을 폴더이름을 통해 추출하여 'new_test_labels'에 추가
'load_img' 및 'img_to_array' 함수를 사용하여 이미지를 로드하고, 'new_test_images' 배열에 추가
3. 모델을 사용한 예측
모델을 사용하여 'new_test_images'에 대한 예측을 수행하고, 이 결과를 'predictions_raw' 변수에 저장
4. 클래스 예측 임계값(threshold) 정의
'threshold' 변수에 클래스 예측에서 예측 확률을 분류하는 데 사용할 임계값을 설정
5. 새로운 테스트 이미지 루프
'for' 루프를 사용하여 각 새로운 테스트 이미지에 대한 예측을 수행
6. 원본 클래스 레이블 가져오기
현재 처리 중인 테스트 이미지에 대한 원본 클래스 레이블을 'original_class_label' 변수에 저장
7. 예측된 확률 가져오기
'predicted_probabilities' 변수에 현재 이미지에 대한 각 클래스에 대한 예측 확률을 저장
8. 클래스 이름 가져오기
'le.classes_'를 사용하여 클래스 이름을 가져와 'class_names' 변수에 저장
9. 클래스 확률 저장 딕셔너리 생성
'class_probabilities' 변수는 클래스 이름과 해당 예측 확률을 묶아 저장하는 딕셔너리를 생성
10. 임계값을 초과한 클래스 찾기
'predicted_classes_above_threshold' 변수에 임계값을 초과하는 클래스를 찾아서 저장
11. 결과 출력
현재 테스트 이미지에 대한 결과를 출력
원본 클래스 레이블을 출력하고, 각 클래스의 예측 확률을 출력
임계값을 초과한 클래스와 해당 클래스의 확률을 출력
가독성을 높이기 위해 다음 테스트 이미지로 넘어가기 전에 빈 줄을 출력

- GradCam

1. 모델과 이미지 준비

저장된 모델 파일 ('C:/Users/User/Desktop/min/model/231003effi.h5')을 로드
Grad-CAM 결과를 저장할 경로를 정의합니다 ('save_path')
Grad-CAM을 생성할 이미지의 경로를 정의합니다 ('image_path')
입력 이미지를 로드하고 크기를 (224, 224)로 조정

2. 모델 부분 나누기
모델을 입력부터 'top_activation' 레이어까지의 부분('model_1')과 'global_average_pooling2d'와 'dense' 레이어로 구성된 부분('model_2')으로 나누기

3. Gradient 계산
TensorFlow의 GradientTape을 사용하여 Grad-CAM을 생성할 이미지의 특정 클래스의 확률을 최대화하기 위한 그라디언트를 계산

4. Grad-CAM 생성
각 특성 맵의 그라디언트를 이용하여 클래스별로 Grad-CAM 히트맵을 계산
그레디언트를 정규화하고 컬러맵을 적용
히트맵을 원본 이미지 크기로 다시 조정
히트맵을 저장하고 이미지에 겹쳐서 표시

5. 저장 및 출력
히트맵을 이미지로 저장
원본 이미지에 히트맵을 적용하여 겹쳐진 이미지를 생성
결과 이미지는 'C:/Users/User/Desktop/min/gradcam/del/00000-2029740302_heatmap.png' 및 'C:/Users/User/Desktop/min/gradcam/del/00000-jjjjjjjjujjjjjangggggggggg.png' 경로에 저장