딥러닝[A] 기말 프로젝트

[1단계 - Binary Classification]

조: 20조

팀원: 2071010 정유진

2171104 정시훈

목차

1. 서론

1.1 프로젝트 목표 및 정의 ……

2. 프로젝트 구성 ……

2.1 환경 설정 및 라이브러리 Import ……

2.2 데이터 Import 및 전처리 ……

2.3 모델 구현(Model1 ~ Model5, 총 5단계) ……

2.4 모델 성능 및 최종 비교 ……

3. 결론 ……

3.1 프로젝트 요약 및 주요 결과 ……

3.2 학습과정 및 고찰 ……

3.3 향후 개선 방향 및 제언 ……

1. 서론

1.1 프로젝트 목표 및 정의

- 본 과제의 목표는 제공된 PCB 이미지 datasets을 사용하여 딥러닝 모델을 구축하고, 기판의 결함 여부(정상/불량)를 판별하는 것이다.
- 총 5가지 모델(3개의 CNN, 1개의 Data Augmentation, 1개의 Pretrained Network 모델)을 단계적으로 구현하고, 각 모델의 아키텍처, 학습 전략, 데이터 처리 방식의 변화가 성능에 미치는 영향을 분석하여 최적의 결함 검출 모델을 탐색한다.
- 이번 보고서에는 3단계의 과제 중 1단계인 결함 여부 검출에 중점을 둔다.

2. 프로젝트 구성

2.1 환경 설정 및 라이브러리 Import

- 전체적인 코드 구현을 위한 라이브러리들을 상단에 불러온다.

```python
import os
import json
import random
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras import layers, models, applications
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from sklearn.model_selection import train_test_split
from tensorflow.keras.applications.densenet import preprocess_input
from tensorflow.keras.applications import DenseNet121
```

Python

[그림 1-1] 라이브러리 import 정보

2.2 데이터 Import 및 전처리

- 학습용(trainval_label.csv) 및 테스트용(test_label.csv) CSV 파일로부터 이미지 파일 경로와 결함 검출 여부에 대한 정보가 들어있는 Defects 라벨 정보를 불러온다.
- Defects 라벨은 결함이 있는 이미지는 1, 없는 정상인 이미지는 0으로 구성되어 있으며, 이진 분류(Binary Classification)의 타겟 변수이다.
- 이미지 파일을 읽어 신경망 모델의 입력으로 사용할 수 있도록 텐서 형태로 변환하고, 모든 이미지의 크기를 일정하게 맞추는 전처리 함수 decode_img를 정의한다.
- 본 이미지는 흑백 사진이므로(grayscale) 3채널이 아닌 1채널로 로드한다.

```python
#학습용 데이터 불러오기

train_csv = './DeepPCB_split/trainval_label.csv'
df = pd.read_csv(train_csv)
train_image_dir = './DeepPCB_split/train'

#filepath -> csv 파일의 filename 열에 저장되어 있는 이미지 이름과 동일한 이미지의 실제 경로(폴더)를 filepath에 저장
#예를 들어, filename -> file1.jpg 이면서 train_image_dir이 './train_images'라면 './train_images/file1.jpg 형태로 저장된다
df['filepath'] = df['filename'].apply(lambda x: os.path.join(train_image_dir, x))
'''
| filename  | filepath                 |
| --------- | ------------------------ |
| file1.jpg | ./train\_images/file1.jpg |
| file2.jpg | ./train\_images/file2.jpg |
| file3.jpg | ./train\_images/file3.jpg |
'''
#model 학습에 사용하기 위해 각각의 열을 list화
#filepaths: 이미지 파일의 경로 리스트
#labels: 'Defect' 열을 기반으로 한 정답 라벨 리스트(정상은 0, 불량은 1로 구성)
filepaths = df['filepath'].tolist()
labels = df['Defects'].tolist()

#테스트용 데이터 불러오기
test_csv = './DeepPCB_split/test_label.csv'
test_df = pd.read_csv(test_csv)
test_dir = './DeepPCB_split/test'

test_df['filepath'] = test_df['filename'].apply(lambda x: os.path.join(test_dir, x))

test_filepaths = test_df['filepath'].tolist()
test_labels = test_df['Defects'].tolist()


IMG_SIZE = 224  # CNN에서 보통 사용하는 표준 크기
BATCH_SIZE = 32 # 한 번의 모델 가중치 업데이트에 사용되는 개수

#decode_img: 이미지 경로 -> 실제 이미지 tensor로 변환하는 전처리 함수
#위 함수를 tf.data.Dataset에 연결해서 학습에 쓸 최종 데이터를 만든다.
def decode_img(img_path, label):
    # 이미지 경로로 읽기
    img = tf.io.read_file(img_path)
    # JPEG 디코딩, 흑백 이미지이므로 channels = 1
    img = tf.image.decode_jpeg(img, channels=1)
    #명시적 형 변환(float 32)
    img = tf.image.convert_image_dtype(img, tf.float32)
    # 이미지 크기 재지정
    img = tf.image.resize(img, [IMG_SIZE, IMG_SIZE])
    return img, label
```

Python

[그림 1-2] 학습 및 테스트 데이터 import 및 전처리

- 데이터는 학습/검증 데이터 비율을 8:2로 나누며, 텐서플로우의 tf.data API를 활용하여 효율적으로 구성한다.
- map, shuffle, batch, prefetch 등의 함수들을 적용하여 자원 사용을 극대화하고 병목 현상을 최소화하도록 설계했다.

```python
#리스트 분할(80% 학습, 20% 검증)
train_files, val_files, train_labels, val_labels = train_test_split(
    filepaths, labels, test_size=0.2, random_state=42, stratify=labels
)

#Dataset 만들기
#학습용 Dataset
train_dataset = tf.data.Dataset.from_tensor_slices((train_files, train_labels))
train_dataset = train_dataset.map(decode_img, num_parallel_calls=tf.data.AUTOTUNE)
train_dataset = train_dataset.shuffle(1000).batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
#검증용 Dataset
val_dataset = tf.data.Dataset.from_tensor_slices((val_files, val_labels))
val_dataset = val_dataset.map(decode_img, num_parallel_calls=tf.data.AUTOTUNE)
val_dataset = val_dataset.batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
#테스트용 Dataset
test_dataset = tf.data.Dataset.from_tensor_slices((test_filepaths, test_labels))
test_dataset = test_dataset.map(decode_img, num_parallel_calls=tf.data.AUTOTUNE)
test_dataset = test_dataset.batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
```

Python

[그림 1-3] tf.dataset API를 활용한 데이터 로딩 및 학습 파이프라인 구축

2.3 모델 정의 및 학습

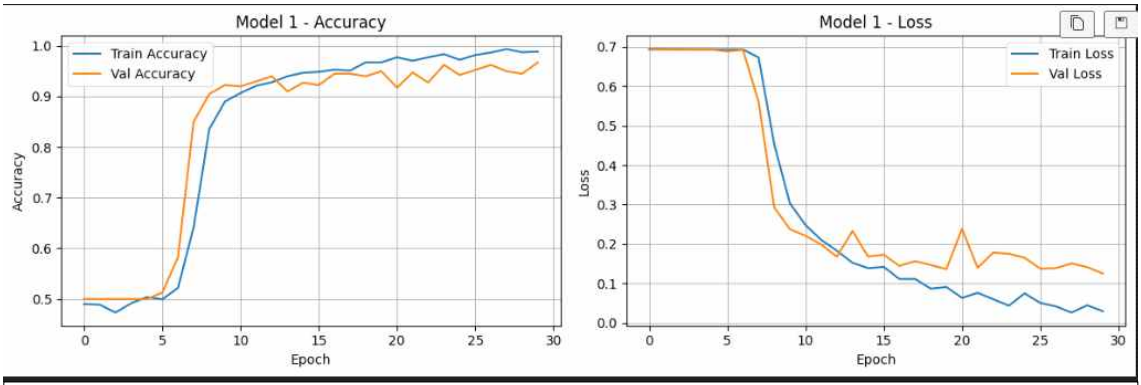
2.3.1 Model_1(CNN)

- Model_1은 가장 기본적인 CNN 아키텍처로 구성되어 있으며, PCB 결함 검출 문제에 대한 베이스라인 성능을 측정한다.
- 총 4개의 블록(Conv2D + MaxPooling2D)을 사용하여 계층적인 특징을 추출하고, GlobalAveragePooling2D()를 통해 특징맵을 벡터화한 후 Dense layer를 통해 분류를 수행한다.

```python
EPOCH = 30

#model_1 : softmax + sparse_categorical_crossentropy, 출력층 2개
model_1 = models.Sequential([
    layers.InputLayer(input_shape=(IMG_SIZE, IMG_SIZE, 1)),

    layers.Conv2D(16, (3,3), activation='relu'),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(32, (3,3), activation='relu'),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(128, (3,3), activation='relu'),
    layers.MaxPooling2D(2,2),

    layers.GlobalAveragePooling2D(), #Flatten 대신 사용
    layers.Dense(128, activation='relu'),
    layers.Dense(2, activation='softmax') # 2개 클래스 (정상/불량)
])
model_1.summary()
model_1.compile(optimizer='adam',
                loss='sparse_categorical_crossentropy',
                metrics=['accuracy'])
#모델 학습
history_1 = model_1.fit(train_dataset,
                        validation_data=val_dataset,
                        epochs = EPOCH
                        )

test_loss_1, test_acc_1 = model_1.evaluate(test_dataset, verbose = 0)
```

Python

[그림 1-4] Model_1의 코드 구성



[그림 1-4] Model_1의 학습/검증 데이터 정확도 및 손실도 시각화

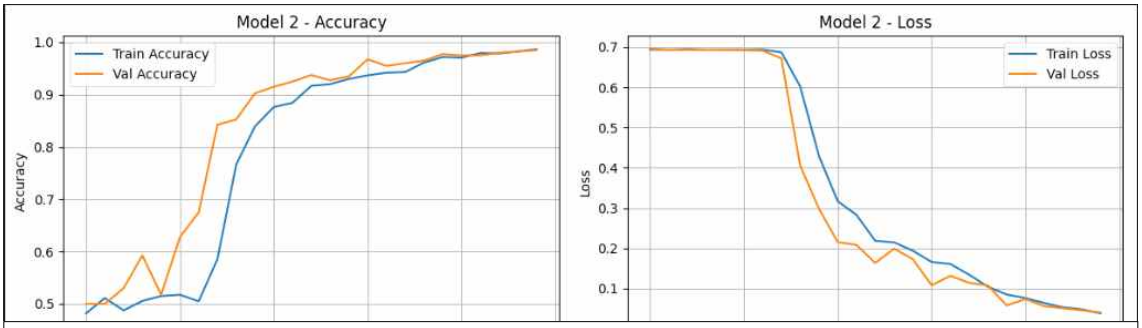
- 첫 모델임에도 불구하고 좋은 결과를 보여주었지만 여러 번 테스트 시 과적합이 발생하였다.

2.3.2 Model_2(CNN)

- Model_1은 초기 에폭 내 빠른 성능 향상을 보였지만, 과적합에 대응해야 하는 문제가 있었다.
- 따라서 이러한 문제를 방지하기 위해 EarlyStooping, ReduceLROnPlateau를 이용한 콜백 함수를 생성하여 과적합 발생 시 학습을 중단하도록 조정하도록 한다.
- Model_2는 Model_1과 거의 동일한 구조를 갖지만 이진 분류에 좀 더 적합한 sigmoid + binary_crossentropy 조합을 사용하여 구성하였다.

```
# 변경점

# EPOCH = 25 # 에폭의 수를 30 -> 25로 감소시킴
EPOCH = 25

#콜백함수 지정
early_stop = EarlyStopping(monitor='val_loss', patience=8, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr=1e-6, verbose=1)

#model_2 : sigmoid + binary_crossentropy, 출력층 1개
model_2 = models.Sequential([
    layers.InputLayer(input_shape=(IMG_SIZE, IMG_SIZE, 1)),

    layers.Conv2D(16, 3, activation='relu'),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(32, 3, activation='relu'),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(64, 3, activation='relu'),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(128, 3, activation='relu'),
    layers.MaxPooling2D(2,2),

    layers.GlobalAveragePooling2D(), #Flatten 대신 사용
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(1, activation='sigmoid')
])
model_2.summary()
model_2.compile(optimizer='adam',
                loss='binary_crossentropy',
                metrics=['accuracy'])


#모델 학습
history_2 = model_2.fit(train_dataset,
                        validation_data=val_dataset,
                        epochs = EPOCH,
                        callbacks=[early_stop, reduce_lr]
                        )
```

[그림 1-5] Model_2의 코드 구성



[그림 1-5] Model_2의 학습/검증 데이터 정확도 및 손실도 시각화

- 이전 모델에 비해 과적합 부분이 많이 개선되었고 검증 정확도는 낮아졌지만 손실 감소 및 안정적 학습 경향을 보였다.

2.3.3 Model_3(Data Augmentation)

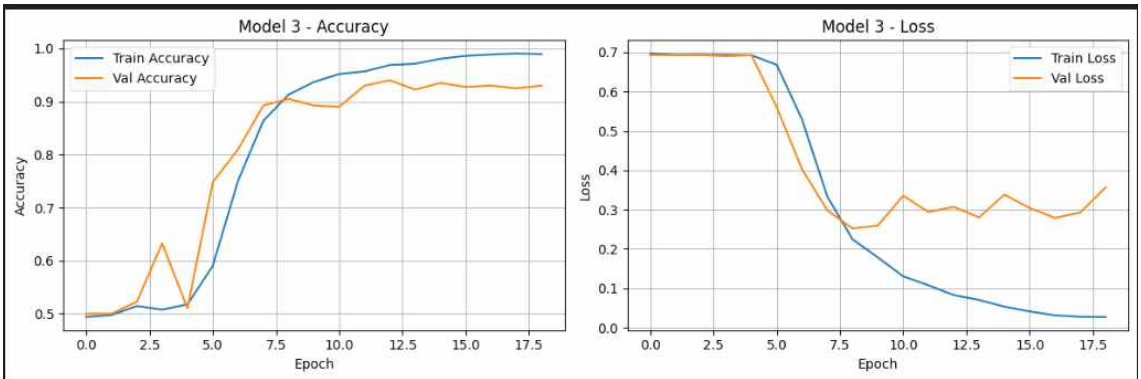
- 단순 CNN구조를 벗어나 과적합을 보다 효과적으로 완화할 수 있고, 모델의 일반화 성능을 향상시킬 수 있는 데이터 증강(Data Augmentation)기법을 적용했다.
- 구조는 model_2와 유사하되 증강된 학습 데이터를 사용하여 학습을 수행한다.
- 증강 강도는 원본의 구조를 크게 훼손하지 않는 선에서 보수적으로 설정, 주요 특징은 보존한다.
- 콜백 함수도 부분 조정하여 빠른 종료 및 최적값 탐색을 유도했다.

```
# 변경점
# 콜백 함수 조정(factor 0.5 -> 0.3 조정)

early_stop = EarlyStopping(monitor='val_loss', patience=10, min_delta=0.0005, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.3, patience=5, min_lr=1e-6, verbose=1)

# 에폭의 수를 25 -> 20으로 감소시킴
EPOCH = 20
#data augmentation layer 정의(다양성을 증가시켜 과적합 방지 및 성능 증가 기대)
data_augmentation = models.Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.02),      # ±2도 회전 (구조 뒤틀림 방지)
    layers.RandomZoom(0.03),          # ±3% 줌 (모서리 정보 손실 방지)
    layers.RandomTranslation(0.03, 0.03)  # ±3% 범위만 이동
])

def augment(image, label):
    image = data_augmentation(image)
    return image, label

augmented_train_dataset = train_dataset.map(augment)

model_3 = models.Sequential([
    layers.Input(shape=(IMG_SIZE, IMG_SIZE, 1)),

    layers.Conv2D(16, 3, activation='relu'),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(32, 3, activation='relu'),
    layers.MaxPooling2D(),

    layers.Conv2D(64, 3, activation='relu'),
    layers.MaxPooling2D(),

    layers.Conv2D(128, 3, activation='relu'),
    layers.MaxPooling2D(),

    layers.Flatten(),
    layers.Dropout(0.5),
    layers.Dense(128, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])
```

[그림 1-6] Model_3의 코드 구성

```python
model_3.summary()
model_3.compile(
    optimizer='Adam',  # 현재 1e-4였음 → 증가
    loss='binary_crossentropy',
    metrics=['accuracy']
)

history_3 = model_3.fit(
    augmented_train_dataset,
    validation_data=val_dataset,
    epochs=EPOCH,
    callbacks=[early_stop, reduce_lr]
)
```

[그림 1-6] Model_3의 코드 구성



[그림 1-7] Model_3의 학습/검증 데이터 정확도 및 손실도 시각화

- 이전 모델에 비해 더욱 향상된 모델을 기대했지만, 오히려 정확도 및 과적합 문제가 심각해지는 상황이 발생했다.
- Augmentation 과정에서 주요 특징이 왜곡되었거나 노이즈가 증가되어 이러한 문제가 발생한 것으로 추정된다.

2.3.4 Model_4(ResNet 기반 CNN)

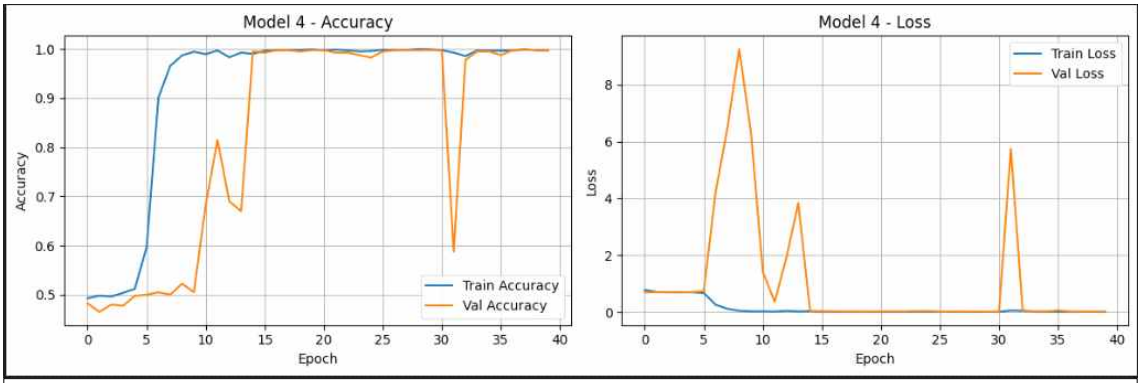
- ResNet의 잔차 블록 구조를 활용해 깊은 네트워크에서 발생하는 기울기 소실 문제를 방지하고, 정보 손실을 최소화한다.
- Model_3를 바탕으로 레이어 수를 더욱 늘리고 Dropout 비율을 조정하여 더욱 강력한 특징 추출과 과적합 억제를 기대한다.

```python
# 변경점 ResNet구조에 레이어 하나 추가 x = residual_block(x, 128)
#                                   x = layers.MaxPooling2D()(x)
# 드롭아웃 수치 (0.5 -> 0.6)

EPOCH = 40 #loss가 중간에 튀는 문제 발생, 최적 가중치에 도달할 수 있도록 에폭 수를 늘렸음음

def residual_block(x, filters):
    # shortcut 경로 : 입력 x 의 채널 수를 filters와 동일하게 맞춤
    shortcut = layers.Conv2D(filters, (1, 1), padding="same")(x)  # 크기 맞추기

    # 주 경로 : 3x3 컨볼루션 두 번
    x = layers.Conv2D(filters, (3, 3), padding='same', activation='relu')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Conv2D(filters, (3, 3), padding='same')(x)
    x = layers.BatchNormalization()(x)

    # shortcut과 주 경로를 더함
    x = layers.Add()([x, shortcut])  # 동일한 크기로 유지
    x = layers.ReLU()(x)

    return x
```

```python
# Model 4: ResNet 구조
input_layer = layers.Input(shape=(IMG_SIZE, IMG_SIZE, 1))
x = layers.Conv2D(64, (3,3), activation='relu', padding="same")(input_layer)
x = layers.BatchNormalization()(x)

x = residual_block(x, 64)
x = layers.MaxPooling2D()(x)

x = residual_block(x, 128)
x = layers.MaxPooling2D()(x)

x = layers.GlobalAveragePooling2D()(x)
x = layers.Dense(128, activation='relu')(x)
x = layers.Dropout(0.6)(x)
output_layer = layers.Dense(2, activation='softmax')(x)


early_stop = EarlyStopping(monitor='val_loss',patience=15, restore_best_weights=True, verbose=1)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=7, min_lr=1e-8, verbose=1)

model_4 = models.Model(inputs=input_layer, outputs=output_layer)
model_4.compile(optimizer='adam',
                loss='sparse_categorical_crossentropy',
                metrics=['accuracy'])

# 모델 학습
model_4.summary()
history_4 = model_4.fit(train_dataset,
                        validation_data=val_dataset,
                        epochs=EPOCH,
                        callbacks=[early_stop, reduce_lr]
                        )
test_loss_4, test_acc_4 = model_4.evaluate(test_dataset, verbose = 0)
```

Python

[그림 1-7] Model_4의 코드 구성



[그림 1-6] Model_4의 학습/검증 데이터 정확도 및 손실도 시각화

- 지금까지의 Model 중 가장 높은 테스트 정확도를 보여주었지만, 과적합을 제어하는 균형의 필요성을 알 수 있다.

2.3.5 Model_5 (DenseNet121을 활용한 Pretrained Network 모델)

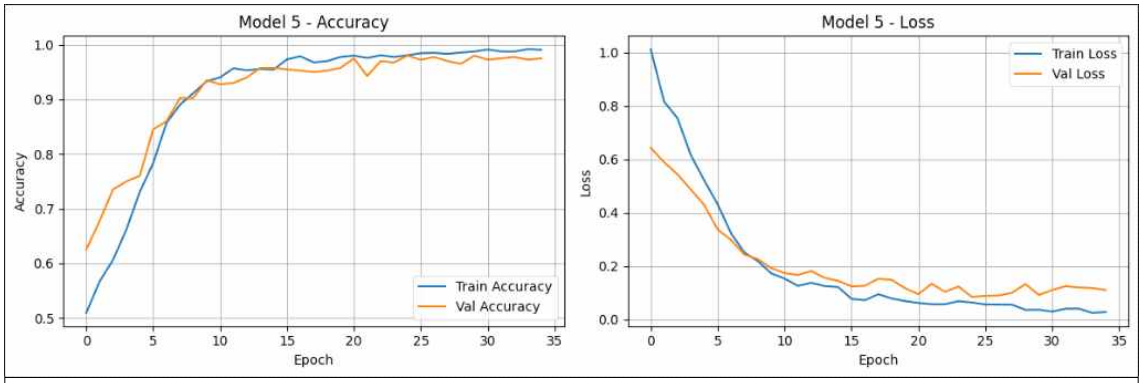
- 이번에는 ImageNet으로 이미 잘 학습되어있는 DenseNet121이라는 CNN 아키텍처를 백본으로 활용한 모델을 구축했다.

- Transfer Learning을 통해 적은 양의 PCB 데이터셋에서도 높은 수준의 특징을 추출하고, 이를 바탕으로 미세 조정 과정까지 거쳐 최고의 성능을 끌어내고자 한다.

```python
# 변경점 베이스 모델의 뉴런 수 증가 64->128, 128->256
#        학습률 1e-3 -> 1e-4 -> 1e-6 -> 1e-5 -> 5e-5
#        회전률 0.1 -> 0.13 -> 0.15 -> 0.2
#        확대/축소 0.1 -> 0.15

EPOCH = 80

# 입력 레이어 정의 (흑백 이미지이므로 1채널)
input_layer = layers.Input(shape=(IMG_SIZE, IMG_SIZE, 1))

# 데이터 증강 및 전처리 레이
x = layers.Lambda(tf.image.grayscale_to_rgb)(input_layer)

x = layers.RandomFlip("horizontal")(x)
x = layers.RandomRotation(0.2)(x)            # ±36도 정도도 회전 (구조 뒤틀림 방지)
x = layers.RandomZoom(0.15)(x)               # ±15% 확대/축소 (모서리 정보 손실 방지)
x = layers.RandomContrast(0.1)(x)            # 대비 거의 유지 (원본이 고대비)
x = layers.RandomTranslation(height_factor=0.1, width_factor=0.1)(x) # ±10% 범위만 이동

# DenseNet121 전용 전처리
# preprocess_input 함수는 보통 0-255 범위의 RGB 이미지를 기대함
# tf.image.convert_image_dtype에서 이미 0-1로 변환되었으므로, 다시 255를 곱한다
x = layers.Lambda(lambda img: img * 255.0)(x)
x = layers.Lambda(preprocess_input)(x)


# 사전 학습된 DenseNet121 모델 불러오기
base_model = applications.DenseNet121(input_shape=(IMG_SIZE, IMG_SIZE, 3), # 3채널 이미지 입력 지정
                                       include_top=False,                  # imageNet 데이터셋의 학습된 가중치 사용
                                       weights='imagenet')
base_model.trainable = True

# 베이스 모델 위에 새로운 분류기(Classification Head) 추가
x = base_model(x, training=True) # 전처리된 RGB 이미지가 베이스 모델의 입력이 됨
x = layers.GlobalAveragePooling2D()(x)

x = layers.Dense(128, activation='relu')(x)
x = layers.Dropout(0.55)(x)
x = layers.Dense(256, activation='relu')(x)
x = layers.Dropout(0.6)(x)

# 최종 출력 레이어 (2개 클래스: 정상/불량, Softmax 활성화)
output_layer = layers.Dense(2, activation='softmax')(x)

# 최종 모델 생성
model_5 = models.Model(inputs=input_layer, outputs=output_layer)

model_5.summary()

model_5.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=5e-5),
                loss='sparse_categorical_crossentropy',
                metrics=['accuracy'])

early_stop = EarlyStopping(monitor='val_loss',patience=10, restore_best_weights=True, verbose=1)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=7, min_lr=1e-8, verbose=1)

# 8. 모델 학습
history_5 = model_5.fit(
    train_dataset, # 입력은 1채널, 모델 내부에서 RGB로 변환 및 증강
    validation_data=val_dataset,
    epochs=EPOCH,
    callbacks=[early_stop, reduce_lr]
)

test_loss_5, test_acc_5 = model_5.evaluate(test_dataset, verbose = 0)
```

Python

[그림 1-8] Model_5의 코드 구성

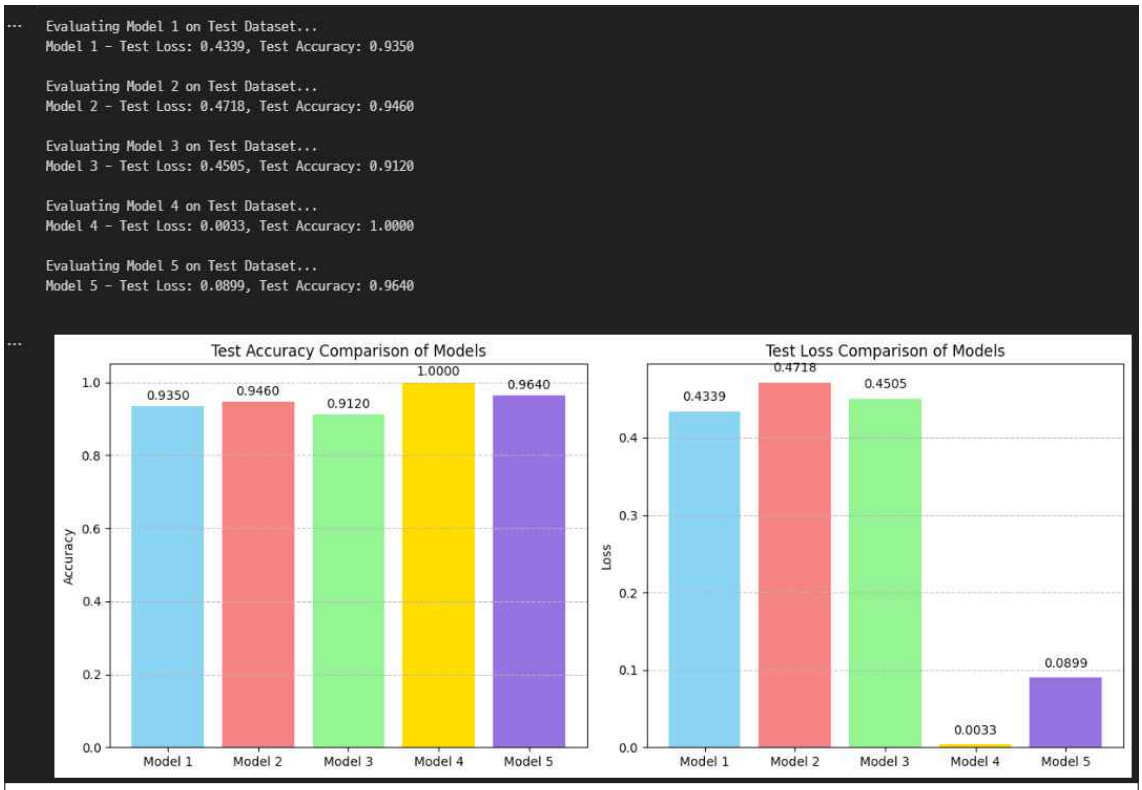


[그림 1-9] Model_5의 학습/검증 데이터 정확도 및 손실도 시각화

- DenseNet121의 강력한 특징 추출 능력 덕분에 비교적 적은 양의 PCB 이미지 데이터로도 높은 성능을 달성하였고, 특히 Fine-tuning을 통해 백본 네트워크가 도메인 특화된 정보를 학습하도록 유도한 점이 성능 향상에 크게 기여했다.

2.4 모델 성능 및 최종 비교

- 2.3의 과정을 통해 학습된 모델들을 테스트 데이터셋을 통해 평가하고, 그 결과를 서로 비교하여 얼마나 정확도가 높아졌는지, 차이는 얼마나 나는지 등을 검증했다.



[그림 2-1] Model_5의 학습/검증 데이터 정확도 및 손실도 시각화

- 결과를 통해 Model_4의 정확도가 가장 높았으며 loss 또한 가장 낮은 모습을 보여주었다.
- 그러나 Model_4의 이론 설명 부분에서도 알 수 있었듯이 loss 값이 불안정한 부분이 있어 해당 부분을 보완하면 더욱 강력한 모델을 기대해볼 수 있다.
-

3. 결론

3.1 프로젝트 요약 및 주요 결과

- 본 프로젝트에서는 PCB 기판의 결함 유무를 판별하기 위해, 기본적인 CNN 모델부터 Data Augmentation, 그리고 Pretrained Network까지 총 5가지의 모델을 단계적으로 구현하고 성능을 평가하였다.
- 실험 결과, Model_4가 가장 높은 정확도로 우수한 성능을 보여주었다.

3.2 학습 과정 및 고찰

- 이번 모델 개선 과정을 통해 다음의 사항들을 학습하고 경험할 수 있었다.

– 이진 분류 문제 정의의 중요성

◆ 출력층 활성화 함수(sigmoid)와 손실 함수(binary_crossentropy)를 문제에 맞게 설정하는 것이 초기 성능 개선에 긍정적인 영향을 미쳤다.

– 과적합 제어 기법의 효과

◆ Dropout과 데이터 증강은 제한된 데이터셋에서 모델의 일반화 성능을 높이는 데 기여하였다. 특히 데이터 증강을 통한 성능 향상이 더욱 뚜렷하게 나타났으며, Dropout과의 조합이 과적합 억제에 효과적이었다.

– 아키텍처의 영향

◆ 잔차 연결(ResNet) 구조를 도입함으로써 정보 손실을 줄이고 더 깊은 네트워크 학습이 가능해짐을 확인하였다.

– 전이학습의 강력함

◆ DenseNet121과 같은 대규모 사전 학습 모델을 기반으로 한 전이학습이 가장 큰 성능 향상을 이끌어냈으며, 올바른 전처리, 적절한 데이터 증강, 낮은 학습률을 사용한 미세 조정이 특히 중요했다.

– 하이퍼파라미터 튜닝의 필요성

학습률, 배치 크기, 옵티마이저 설정, 콜백 함수 구성 등 다양한 하이퍼파라미터가 모델 성능에 결정적 영향을 미쳤다. 결론적으로 반복적 실험과 분석을 통해 최적값을 찾아가는 과정이 필수적임을 체감했다.

- **데이터의 한계와 학습 관찰**

  모델을 학습시키면서 깊은 구조의 모델일수록 데이터 부족이 두드러졌으며, Model 4, 5에는 데이터 증강이 필수적으로 사용되어야 했다.

3.3 향후 개선 방향 및 제언

- 더 다양한 데이터 증강 전략 탐색

  예: `CutMix`, `MixUp`, `AutoAugment` 등 최신 증강기법 적용을 고려한다.

- 정교한 미세 조정 기법

  예: 사전 학습 모델의 **계층별 차등 학습률(Differential Learning Rate)** 적용,

  혹은 동결 해제 범위를 더욱 세밀하게 조정하는 실험해본다.

- 앙상블(Ensemble) 모델 구축

  여러 우수 모델의 예측을 결합하여, 보다 안정적이고 강인한 성능 확보할 수 있다.

- 다른 최신 사전 학습 모델 활용

  예: `EfficientNetV2`, `Vision Transformer (ViT)` 등을 적용하는 시도를 해본다.

- 결함 종류 세분화 (Multi-label 또는 Multi-class 분류 확장)

  현재는 결함 유무만 판별하지만, 향후 open, short, spur 등

  구체적인 결함의 종류까지 분류하는 문제로 확장 가능하다.