머신러닝

HW04

Convolution Neural Network Using Tensorflow

당교수: 박철수

과목명: 머신러닝

학과: 컴퓨터정보공학부

학번: 2020202066

이름: 장지호

1. 과제 개요 및 목적

해당 과제는 TensorFlow를 이용하여 이미지 분류 모델을 Convolution Neural Network로 구현한다. Caltech-101 dataset[i]을 이용하여 classification model을 학습한다. 이를 통해 CNN의 구조를 이해하고 Convolution/Rectified linear unit layer, Pooling layer, Fully connected layer 등 각 레이어의 기능과 작동 원리를 분석한다.

모델 성능에서 Regularization과 activation function이 어떤 영향을 미치는지 분석하고 Hyperparameter 튜닝을 통해 최적의 학습 조건을 찾는다. 그리고 모델의 accuracy와 F1-score를 계산하고 ROC curve나 learning curve를 시각화 하여 성능을 비교한다.

2. 데이터 셋 분석

Caltech-101 dataset은 총 9146장의 이미지(jpg)로 이루어져 있다. 각 이미지의 크기는 300~200 pixel이며 101이름에 맞게 101가지 class로 분류되어 있다.[ii] Class는 동식물부터 사물, 음양 문양 같은 여러가지 카테고리로 구성되어 있다. Class 별로 약 50장의 이미지가 있지만 face class의 이미지는 약 500장 있어 데이터셋마다 이미지 수의 불균형이 다소 강하다. 이미지는 대부분 컬러 이미지이고 단순한 배경으로 객체가 명확히 식별할 수 있는 편이다.

데이터셋의 장점은 다양한 카테고리가 있어 이미지 분류에 있어 일반적인 성능 평가에 적합하다. 이미지의 노이즈가 적고 배경이 깔끔해 CNN 학습에 별도의 전처리를 거치지 않고 이용할 수 있다. (차의 옆면이나 비행기의 경우 방향도 정렬되어 있음)

단점은 클래스별 이미지의 개수에 불균형이 있다. 특히 face 클래스는 약 500장인 반면, 대부분의 다른 클래스들은 50장 내외로 구성되어 있어 모델 학습 시 클래스 간 편향이 발생할 수 있다. 그리고 장점에서 언급한 깔끔한 이미지로 인해 실제 촬영한 이미지에는 분류 성능이 떨어질 수 있다.

3. Convolution Neural Network

CNN은 크게 이미지에서 유의미한 특징을 추출하는 '특징 추출(Feature Extraction)' 부분과 추출된 특징을 기반으로 최종 분류 또는 예측을 수행하는 '분류(Classification)' 부분으로 나뉜다.

Feature extraction에서는 convolution layer와 pooling layer가 반복적으로 쌓여 구성된다. 초기 레이어는 edge나 line과 같은 단순한 특징을 감지하고, 깊은 레이어로 갈수록 눈, 코, 얼굴과 복잡한 특징을 학습한다. 이때 Layer가 깊어질수록 이미지의 사이즈는 점점 줄어든다. Fully connected layer에서는 이전 layer에서 추출한 feature값을 이용해 이미지를 분류하며 이때 feature 값은 1차원 vector의 형태다.

1. Convolution layer

이미지에서 feature를 추출하는 layer. 일반적으로 이미지보다 작은 사이즈의 filter를 이용한다. Filter를 정사각행렬로 정의하여 입력된 이미지를 slide(stride)하면서 이미지와 filter가 겹치는 부분의 내적(dot product)를 계산하여 feature map(activation map)을 계산한다. 이때 padding을 통해 convolution 계산전에 이미지 외곽에 특정 값(0 또는 샘플링한 값)을 넣어 feature map의 사이즈를 조절할 수 있다.

Convolution 계산 후에는 비선형 활성화 함수인 ReLU(Rectified Linear Unit)를 이용해 음수 값을 0으로 만들고 양수는 그대로 통과시킨다.

n*n 이미지에 f*f 필터, stride k, padding p, feature map의 사이즈 m*m

$m = \lfloor \frac{n+2p-f}{k} \rfloor + 1$ ***이미지와 필터의 depth와 상관없이 feature map에서 scalar 값 1개***

2. **Pooling layer**

   Convolution layer에서 계산한 feature map을 sampling 한다. 이때 Down sampling을 통해 model의 복잡성을 제어하고 과적합(overfitting)을 방지할 수 있다.[iii]

   Max pooling은 feature map의 지정된 영역에서 최대값을 sampling 한다. 뚜렷한 feature를 보존할 수 있지만 다른 정보의 손실이 발생할 수 있다.

   Average pooling은 feature map의 지정된 영역에서 평균값을 sampling 한다. 노이즈를 감소시킨다.

   Global한 pooling으로 feature map 전체에서 Max 값 또는 Avg값을 sampling할 수도 있다.

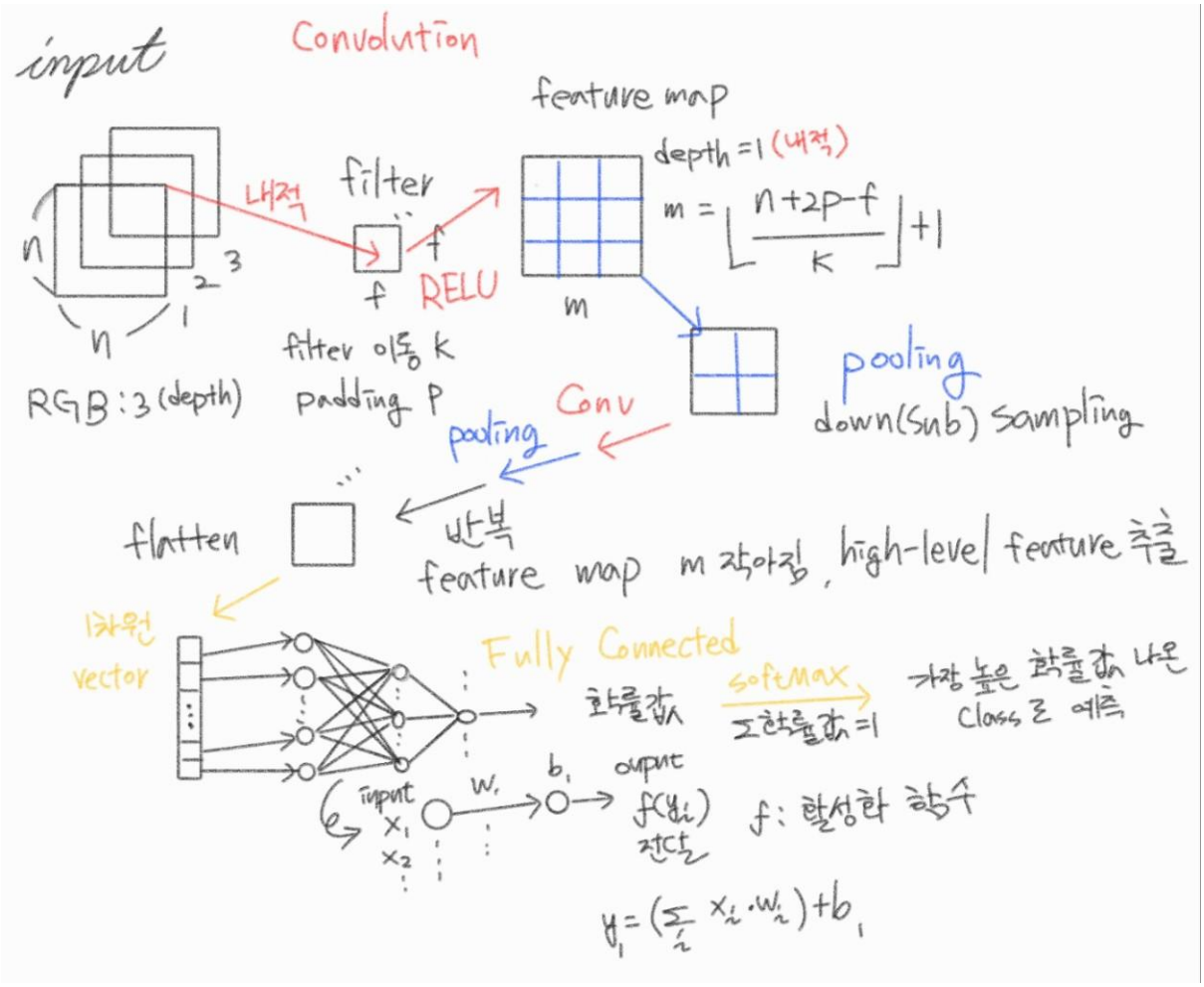
3. **Fully Connected Layer**
   앞선 layer들에서 추출된 high-level의 feature를 조합해 classification이나 regression을 수행(이번 과제에서는 classification)한다. Fully connected lay에 들어가기 앞서 feature map은 flatten연산을 통해 1차원 vector로 변환한다.
   이후 FC layer는 한 층의 모든 neuron이 다음 층의 모든 neuron과 연결되어 있다. 각 neuron은 이전 층의 입력에 weight를 곱하고 bias를 더한 후 Activision함수 (ReLU, sigmoid, tanh, etc..)를 이용해 출력 값을 다음 층에 전달한다.

   마지막 FC layer에서 최소 분류해야 할 class의 수와 같은 neuron이 있다. softmax[iv]를 이용하여 모든 class의 확률의 합이 1이 되도록 정규화 한다. 이 때 가장 높은 확률이 나온 class를 최종 예측 결과로 출력한다.

4. **활성화 함수**

   활성화 함수는 신경망의 각 노드에서 입력 신호를 출력 신호로 변환하여 다음 레이어로 전달한다. 이 때 신경망에 비선형성을 추가하게 되고 모델이 복잡한 패턴을 학습할 수 있게 된다. 또한 역전파(Backpropagation) 과정에서 가중치와 편향을 업데이트하는 데 필수적인 기울기(Gradient)를 제공한다.
   ReLU (Rectified Linear Unit): $f(x) = max(0, x)$는 Sigmoid나 Tanh 함수에 비해 학습이 훨씬 빠르고 단순한 연산으로 인해 계산 비용이 적고 구현이 간단하다. 하지만 입력값이 음수인 경우 항상 0을 반환하므로, 해당 뉴런의 기울기가 00이 되어 가중치 업데이트가 이루어지지 않을 수 있다.
   Sigmoid (시그모이드) 함수: $f(x) = \frac{1}{(1+e^{-x})}$는 입력값을 0과 1 사이의 값으로 압축하여 출력한다. 출력값이 0~1 사이로 정규화되어 확률 해석에 유리하며, 기울기 폭주 (Gradient Exploding)가 발생하지 않는다. 하지만 력값이 매우 크거나 작아지면 함수의 기울기가 0에 가까워져 역전파 시 가중치 업데이트가 매우 느려지거나 멈출 수 있다.
   Tanh (하이퍼볼릭탄젠트) 함수: $f(x) = \frac{e^{x}-e^{-1}}{e^{x}+e^{-x}}$는 입력값을 -1과 1 사이의 값으로 압축하여 출력한다. 출력의 중심이 0(Zero-centered)이므로, Sigmoid 함수보다 경사 하강법 사용 시 편향 이동이 적어 학습이 더 잘 되는 경향이 있다. Sigmoid보다 기울기 소실 증상이 덜하다. 여전히 기울기 소실 문제가 발생할 수 있다.

5. CNN 구조 요약



4. Tensorflow

Google이 개발한 오픈소스 소프트웨어 라이브러리이자 머신러닝 및 딥러닝 프로그램을 쉽게 구현할 수 있도록 다양한 기능을 제공하는 프레임워크이다. TensorFlow는 기본적으로 C++로 구현되었지만, 파이썬을 최우선으로 지원하며 자바, 고(Go) 등 다양한 언어를 지원한다.

텐서 (Tensor): 딥러닝에서 데이터를 표현하는 방식이자 다차원 배열을 의미한다. 이미지 데이터의 경우, 흑백 이미지는 2차원 배열로, RGB 컬러 이미지는 3개의 채널마다 2차원 배열로 표현되며, 이 모든 것이 텐서로 표현 가능하다.

+ GPU 사용
CUDA와 cuDNN을 이용하여 tensor계산에 GPU를 이용할 수 있다. 이 때 GPU의 병렬 처리 기능으로 CPU로 학습할 때 보다 parameter 개수가 많은 모델을 학습할 수 있고 같은 parameter개수를 가진 모델이라도 적은 시간으로 학습을 완료할 수 있다.

```
gpu_available = tf.config.list_physical_devices('GPU')
```

로 이용할 수 있는 GPU 확인 가능, Window에서 TensorFlow에서 이용 가능한 cuda, cuDNN에 있어 WSL2로 빌드ᵛ하였다.

5. Code

```
6.  # GPU 설정 및 확인
7.  def setup_gpu():
8.      """
9.      GPU 사용 설정 및 확인
10.     TensorFlow가 GPU를 사용할 수 있도록 메모리 증가를 허용
11.     """
12.     # GPU 장치 확인
13.     gpus = tf.config.experimental.list_physical_devices('GPU')
14.     if gpus:
15.         try:
16.             # GPU 메모리 증가 허용 (필요에 따라 메모리 할당)
17.             for gpu in gpus:
18.                 tf.config.experimental.set_memory_growth(gpu, True)
19.             print(f"GPU 사용 가능: {len(gpus)}개의 GPU 감지됨")
20.             print(f"GPU 장치: {[gpu.name for gpu in gpus]}")
21.         except RuntimeError as e:
22.             print(f"GPU 설정 오류: {e}")
23.     else:
24.         print("GPU를 사용할 수 없습니다. CPU로 실행됩니다.")
25.
26.     # 현재 사용 중인 장치 확인
27.     print(f"TensorFlow 버전: {tf.__version__}")
28.     print(f"사용 가능한 GPU: {tf.config.list_physical_devices('GPU')}")
```

TensorFlow 계산에서 GPU를 이용할 수 있게 WSL2 환경에서 CUDA 설치, cuDNN을 설치하고 13 line에서 확인한다. GPU 사용 불가하면 CPU로 연산.

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2021 NVIDIA Corporation
Built on Thu_Nov_18_09:45:30_PST_2021
Cuda compilation tools, release 11.5, V11.5.119
Build cuda_11.5.r11.5/compiler.30672275_0
```

```
class Config:
    """
    하이퍼파라미터 및 설정 클래스
    실험을 위한 다양한 설정값들을 관리
    """
    # 데이터셋 경로 설정
    BASE_DATASET_PATH = './caltech-101'  # Caltech-101 데이터셋 경로
    PROCESSED_DATA_PATH = './caltech_101_processed'
```

```python
# 이미지 전처리 파라미터
IMG_WIDTH = 256    # 이미지 너비
IMG_HEIGHT = 256   # 이미지 높이
CHANNELS = 3       # RGB 채널
INPUT_SHAPE = (IMG_HEIGHT, IMG_WIDTH, CHANNELS)

# 훈련 하이퍼파라미터
BATCH_SIZE = 32    # 미니배치 크기: GPU 메모리와 수렴 안정성 고려
EPOCHS = 75        # 최대 에포크 수 (Early Stopping 사용)
LEARNING_RATE = 0.001  # 학습률: Adam optimizer의 기본값

# 데이터 분할 비율
TRAIN_RATIO = 0.8  # 훈련 데이터 80%
VAL_RATIO = 0.1    # 검증 데이터 10%
TEST_RATIO = 0.1   # 테스트 데이터 10%

# 실험 설정
NUM_TRIALS = 1     # 1회만 실행으로 변경
```

IMG 사이즈는 이미지 평균 사이즈를 고려해서 초기에는 128로 설정했으나 사용가능한 하드웨어 자원 확보 후 256으로 증가했다. Bach size는 모델의 훈련 단계에서 한 번에 처리하는 데이터 샘플의 개수이다. 메모리 용량상 32보다 크게 설정하기 어려웠고 가능하더라도 모델 복잡도가 너무 상승하였다. Epoch는 학습 데이터셋을 몇 번 반복할지 나타낸다. 초기에 50으로 설정하였으나 accuracy가 60% 중반에서 벗어나지 못하여서 75로 증가하고 대신 early stopping을 사용하였다.

조기 종료(Early Stopping)는 딥러닝 모델 훈련 시 검증 데이터의 성능(예: 검증 손실 또는 검증 정확도)이 일정 횟수 동안 개선되지 않을 경우 훈련을 중단하여 과적합을 방지하고 최적의 모델을 얻는 기법이다.

Learning rate는 딥러닝 모델의 가중치(weight)를 업데이트할 때, 한 번의 업데이트에서 가중치가 얼마나 크게 변화할지를 결정하는 hyperparameter이다. 너무 큰 러닝 레이트는 학습이 불안정하고 minimum에 도달하지 못할 수 있다. 최악의 경우 Loss 값이 발산할 수 있다. 너무 작은 러닝 레이트는 학습이 매우 느리다. 그리고 local minimum에 갇힐 가능성이 있음.

Adam (moment) optimizer는 경사 하강법 기반 최적화 알고리즘으로 이전 기울기(momentum)을 사용하여 현재 기울기(gradient)를 업데이트 한다.

1차 모멘텀(Momentum)은 이전 단계에서 계산된 경사(gradient)들의 지수 이동 평균(exponentially moving average)을 계산하여 현재 가중치 업데이트에 반영한다. 이는 경사 하강이 불안정한(noisy) 경우에도 일관된 방향으로 업데이트를 진행하도록 돕고, 지역 최솟값(local minima)에서 빠져나오기 쉽게 한다.**

$$m_t = \beta_1 \cdot n_{t-1} + (1-\beta_t) \cdot g_t$$ ***여기서*** g_t ***는 현재 기울기***

2차 모멘텀은

$$[v_t = \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2]$$ ***여기서*** (v_t)***는*** 2***차모멘트(기울기제곱의평균)***

바이어스 보정에서 m_t**과** v_t**이 0에 가까워지는 문제를 방지한다.**

$$\hat{m_t} = m_t/(1-\beta_1^t), \hat{v}_t = v_t/(1-\beta_2^t)$$

파라미터 업데이트

$$\theta_t = \theta_{(t-1)} - \eta / \left(\sqrt{(\hat{v}_t)} + \varepsilon\right) \cdot \hat{m}_t$$

```
optimizer = tf.keras.optimizers.Adam(learning_rate=Config.LEARNING_RATE)
```

```
# 기존 처리된 데이터 제거 (새로운 분할을 위해)
    if os.path.exists(config.PROCESSED_DATA_PATH):
        shutil.rmtree(config.PROCESSED_DATA_PATH)
        print(f"기존 처리된 데이터 제거: {config.PROCESSED_DATA_PATH}")

    # 새 디렉토리 생성
    train_dir = os.path.join(config.PROCESSED_DATA_PATH, 'train')
    val_dir = os.path.join(config.PROCESSED_DATA_PATH, 'validation')
    test_dir = os.path.join(config.PROCESSED_DATA_PATH, 'test')
```

매번 학습할 때 새로운 학습 데이터, 테스트 데이터를 이용하기 위해 추가한 코드

```
model = Sequential([
        # Convolutional Block 1 (큰 필터로 초기 특성 추출)
        Conv2D(64, (5, 5), activation='relu', input_shape=input_shape,
padding='same'),
        BatchNormalization(),
        MaxPooling2D((2, 2)),
        Dropout(0.25),

        # Convolutional Block 2
        Conv2D(128, (3, 3), activation='relu', padding='same'),
        BatchNormalization(),
        MaxPooling2D((2, 2)),
        Dropout(0.25),

        # Convolutional Block 3
        Conv2D(256, (3, 3), activation='relu', padding='same'),
        BatchNormalization(),
        MaxPooling2D((2, 2)),
        Dropout(0.3),

        # Convolutional Block 4
        Conv2D(512, (3, 3), activation='relu', padding='same'),
        BatchNormalization(),
        MaxPooling2D((2, 2)),
```

```
        Dropout(0.3),

        # Flatten -> Fully Connected
        Flatten(),

        # Dense Layer 1
        Dense(512, activation='relu'),
        BatchNormalization(),
        Dropout(0.5),
        # Dense Layer 2
        Dense(256, activation='relu'),
        BatchNormalization(),
        Dropout(0.5),

        # Output Layer
        Dense(num_classes, activation='softmax')
    ])
```

첫 번째 블록에서는 64개의 5×5 크기 필터를 사용하여 입력 이미지의 초기 특성을 추출한다. 큰 크기의 필터를 사용함으로써 이미지의 전반적인 구조와 큰 패턴을 효과적으로 포착할 수 있도록 설계하였다. ReLU 활성화 함수를 적용하여 비선형성을 도입하고, 'same' 패딩을 통해 입력 이미지의 크기를 유지하였다. 두 번째 블록에서는 128개의 3×3 필터를 사용하여 더 세밀한 특성을 추출한다. 필터 개수를 2배로 증가시켜 모델의 표현력을 향상시키고, 작은 크기의 필터를 통해 지역적인 패턴과 세부 특성을 학습할 수 있도록 하였다. 세 번째와 네 번째 블록에서는 각각 256개와 512개의 3×3 필터를 사용하여 점진적으로 필터 수를 증가시켰다. 이러한 구조는 네트워크가 깊어질수록 더 복잡하고 추상적인 특성을 학습할 수 있도록 하며, 고수준의 의미론적 정보를 효과적으로 추출한다.

배치 정규화(Batch Normalization)

각 합성곱층과 완전연결층 뒤에 배치 정규화를 적용하여 내부 공변량 이동(Internal Covariate Shift) 문제를 해결하고 학습 안정성을 향상시켰다. 이를 통해 더 높은 학습률을 사용할 수 있으며, 초기 가중치에 대한 민감도를 줄여 안정적인 학습이 가능하다.

최대 풀링(Max Pooling)

각 합성곱 블록 후에 2×2 크기의 최대 풀링을 적용하여 특성맵의 공간적 차원을 절반으로 축소하였다. 이는 계산 복잡도를 줄이고 위치 불변성을 제공하며, 과적합을 방지하는 효과를 가져온다.

드롭아웃(Dropout)

합성곱 블록에서는 0.25~0.3의 드롭아웃 비율을, 완전연결층에서는 0.5의 높은 드롭아웃 비율을 적용하여 과적합을 효과적으로 방지하였다. 합성곱층에서는 상대적으로 낮은 비율을, 완전연결층에서는 높은 비율을 적용하여 각 층의 특성에 맞는 정규화를 수행하였다.

FC layer에서는 Flatten()으로 평탄화, 첫 번째 완전연결층에는 512개의 뉴런을, 두 번째 완전연결층에는 256개의 뉴런을 배치하고 각 층에 ReLU 활성화 함수를 적용했다. 최종 출력층에서는 클래스 수에 해당하는 뉴런을 배치하고 소프트맥스(Softmax) 활성화 함수를 적용하여 각 클래스에 대한 확률 분포를 출력하도록 설계했다.

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 256, 256, 64)	4,864
batch_normalization (BatchNormalization)	(None, 256, 256, 64)	256
max_pooling2d (MaxPooling2D)	(None, 128, 128, 64)	0
dropout (Dropout)	(None, 128, 128, 64)	0
conv2d_1 (Conv2D)	(None, 128, 128, 128)	73,856
batch_normalization_1 (BatchNormalization)	(None, 128, 128, 128)	512
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 128)	0
dropout_1 (Dropout)	(None, 64, 64, 128)	0
conv2d_2 (Conv2D)	(None, 64, 64, 256)	295,168
batch_normalization_2 (BatchNormalization)	(None, 64, 64, 256)	1,024
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 256)	0
dropout_2 (Dropout)	(None, 32, 32, 256)	0

Model: "sequential"

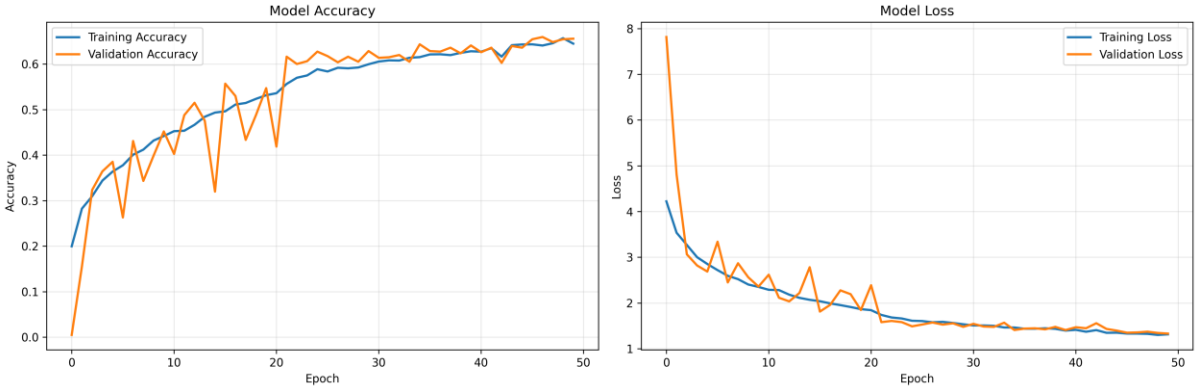
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 256, 256, 64)	4,864
batch_normalization (BatchNormalization)	(None, 256, 256, 64)	256
max_pooling2d (MaxPooling2D)	(None, 128, 128, 64)	0
dropout (Dropout)	(None, 128, 128, 64)	0
conv2d_1 (Conv2D)	(None, 128, 128, 128)	73,856
batch_normalization_1 (BatchNormalization)	(None, 128, 128, 128)	512
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 128)	0
dropout_1 (Dropout)	(None, 64, 64, 128)	0
conv2d_2 (Conv2D)	(None, 64, 64, 256)	295,168
batch_normalization_2 (BatchNormalization)	(None, 64, 64, 256)	1,024
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 256)	0
dropout_2 (Dropout)	(None, 32, 32, 256)	0

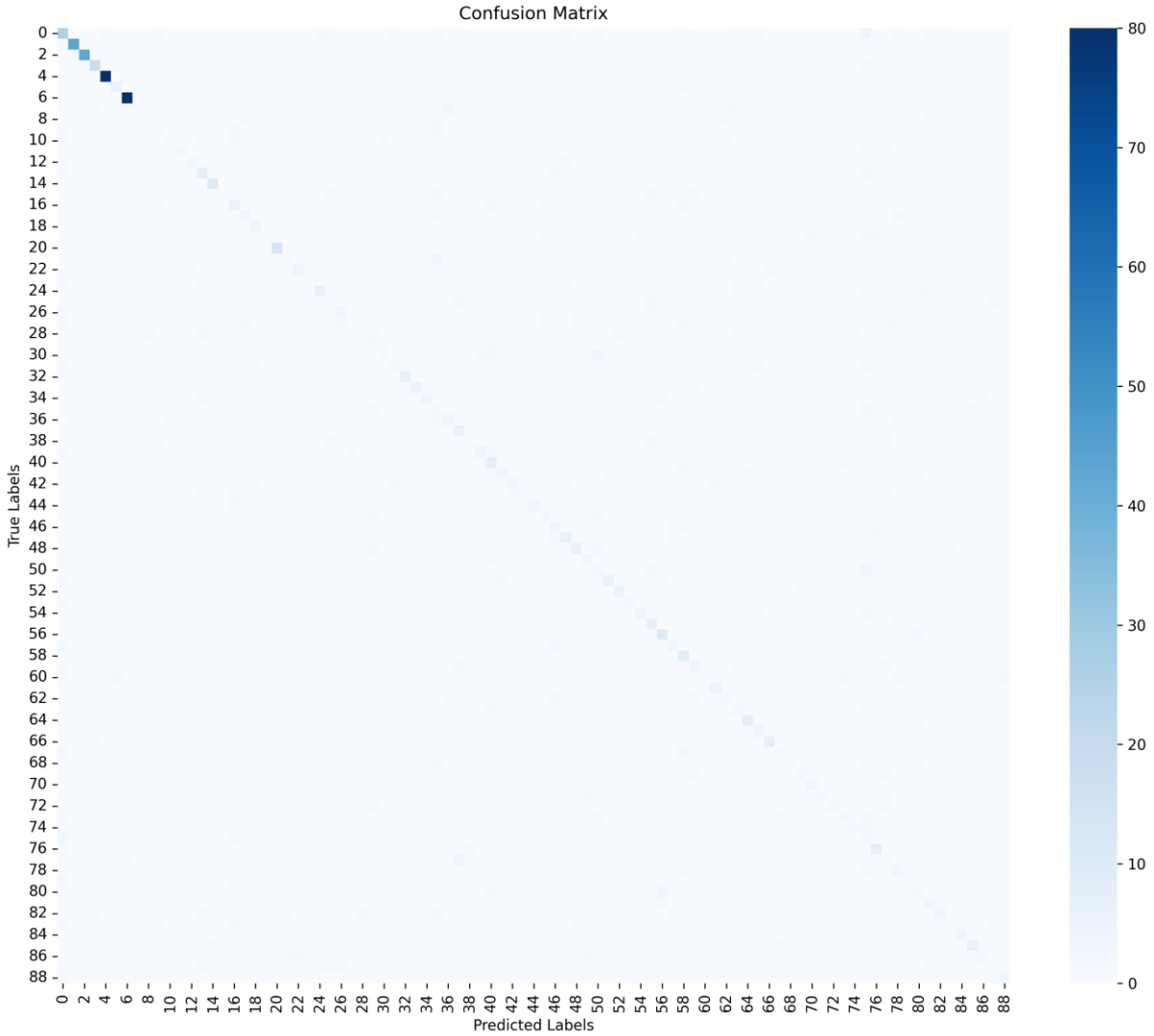
Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 256, 256, 64)	4,864
batch_normalization (BatchNormalization)	(None, 256, 256, 64)	256
max_pooling2d (MaxPooling2D)	(None, 128, 128, 64)	0
dropout (Dropout)	(None, 128, 128, 64)	0
conv2d_1 (Conv2D)	(None, 128, 128, 128)	73,856
batch_normalization_1 (BatchNormalization)	(None, 128, 128, 128)	512
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 128)	0
dropout_1 (Dropout)	(None, 64, 64, 128)	0
conv2d_2 (Conv2D)	(None, 64, 64, 256)	295,168
batch_normalization_2 (BatchNormalization)	(None, 64, 64, 256)	1,024
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 256)	0
dropout_2 (Dropout)	(None, 32, 32, 256)	0

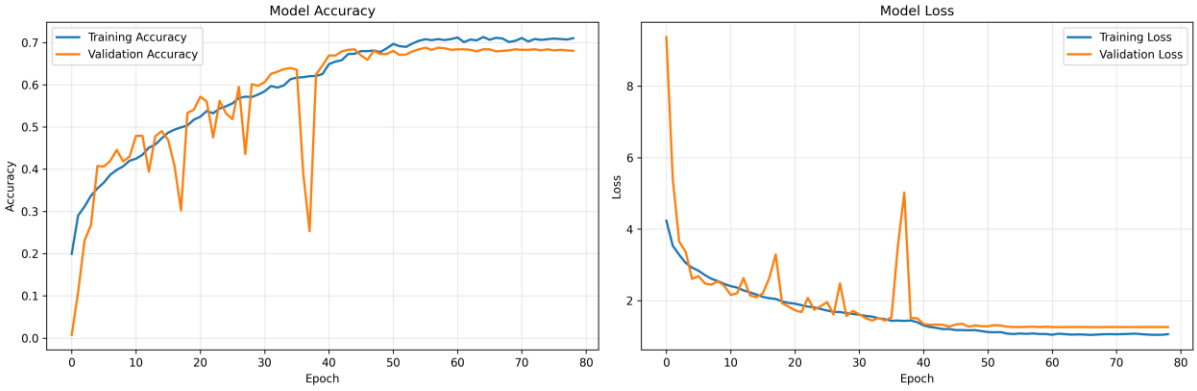
```
Total params: 68,824,537 (262.54 MB)
Trainable params: 68,821,081 (262.53 MB)
Non-trainable params: 3,456 (13.50 KB)
```

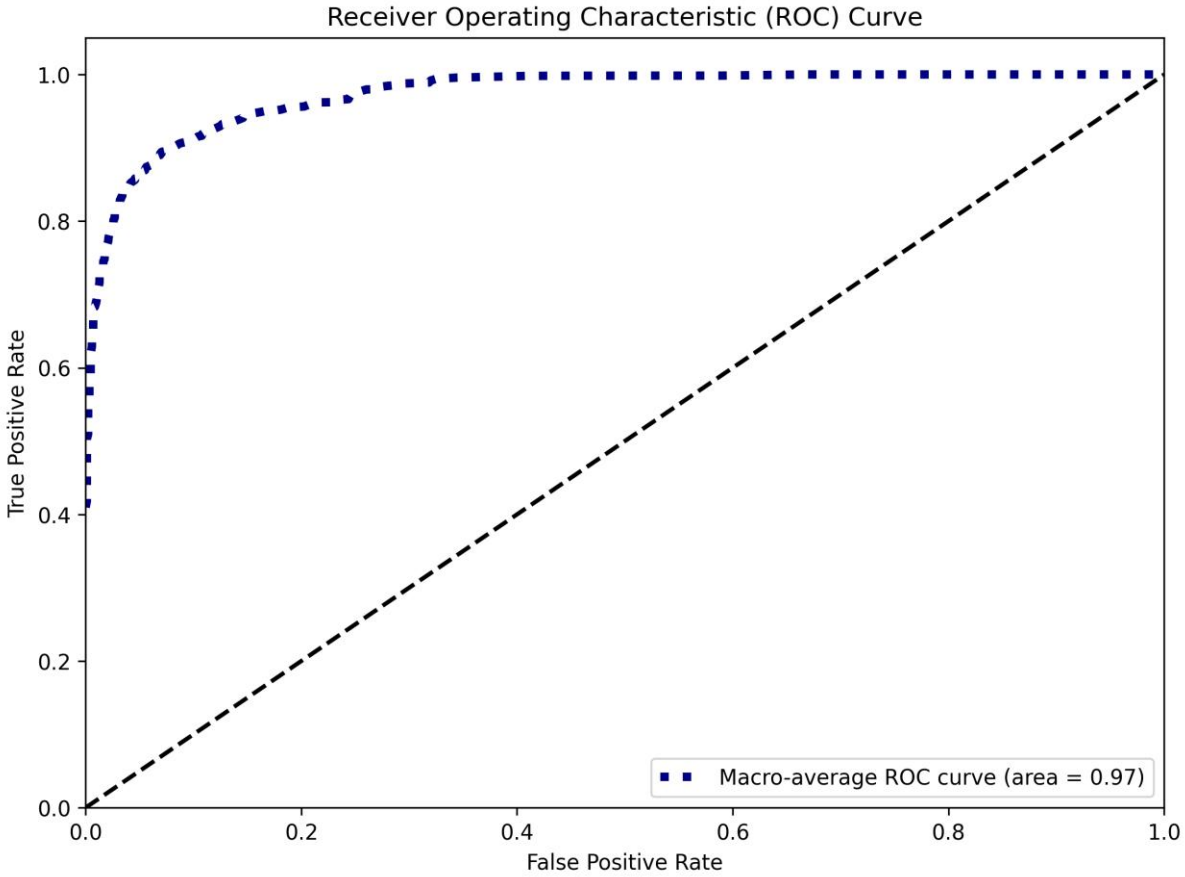
6. Plot, 하이퍼파라미터 변경 과정





128*128에 Epoch 50에서 512*512에 Epoch100으로 증가했을 때 accuracy는 대략 10%p 증가하였으나 학습시간이 너무 크게 증가하였음.





레이어 구조를 바꾸는 것을 실험.

현재 레이어 구조 요약

- **4개의 Convolutional Block (32→64→128→256 필터)**
- **각 블록 뒤에 BatchNormalization, MaxPooling, Dropout**
- **2개의 Fully Connected Layer (512→256 뉴런)**

Softmax 출력층

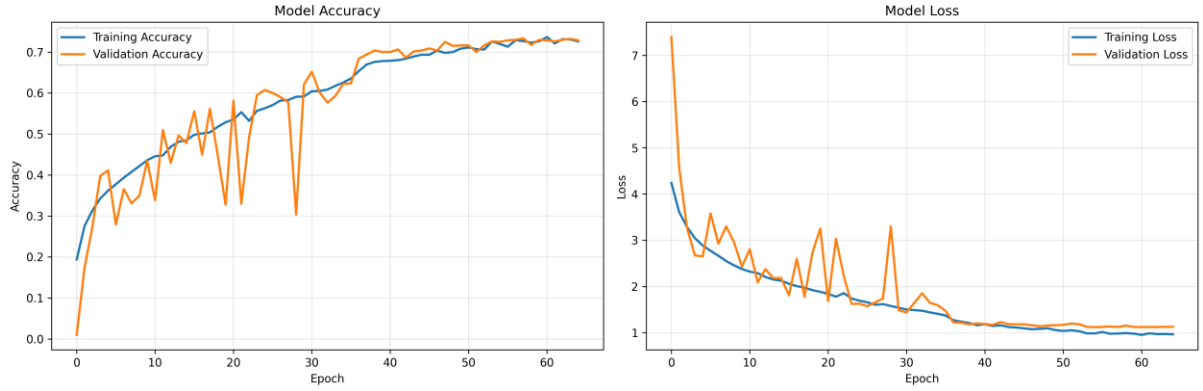
을 모든 필터 사이즈를 3x3으로 유지하는 대신 필터의 개수를 64->128->256->512 로 변경

Epoch는 75로 학습

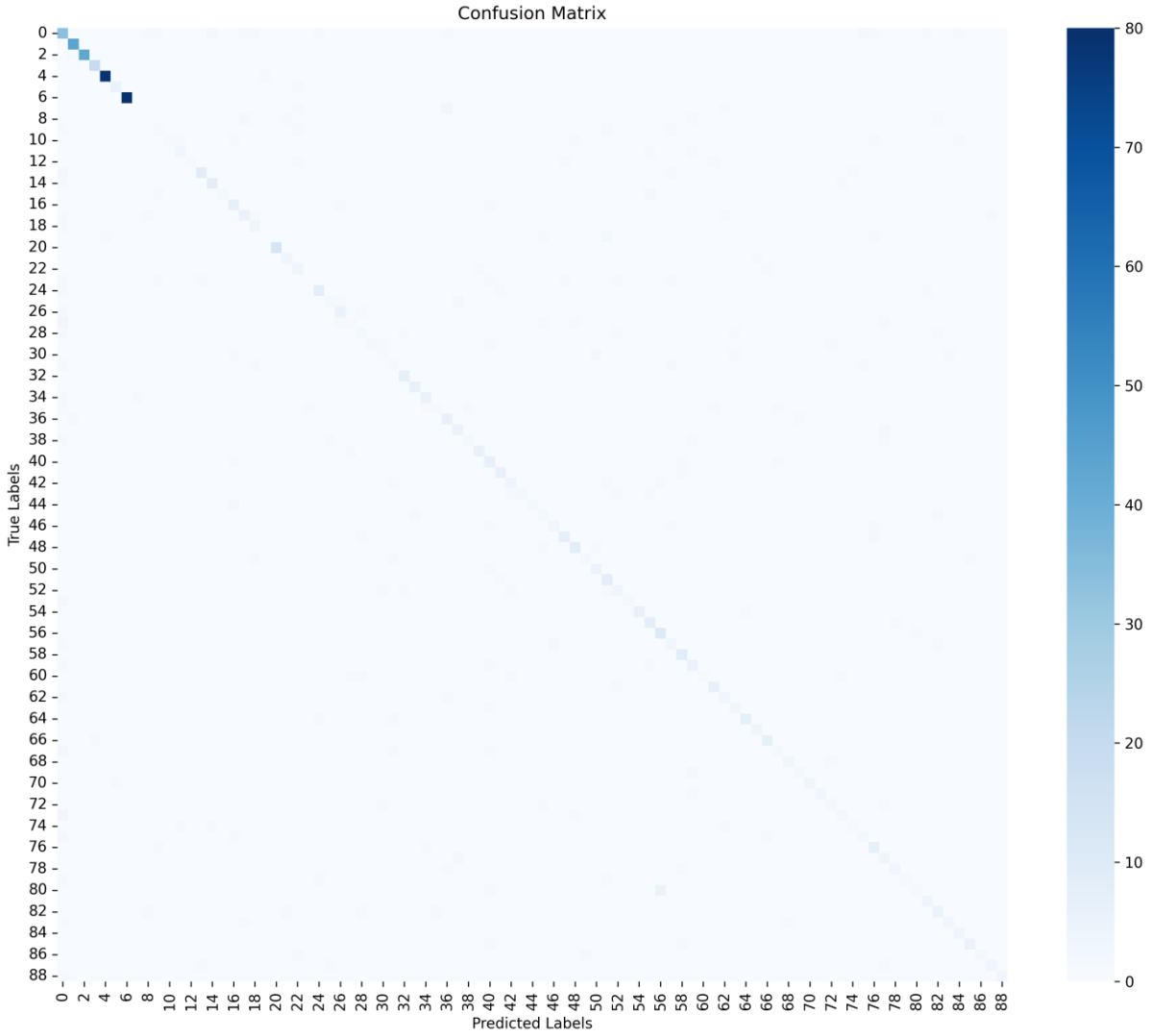
개선 후 plot

experiment_results.txt

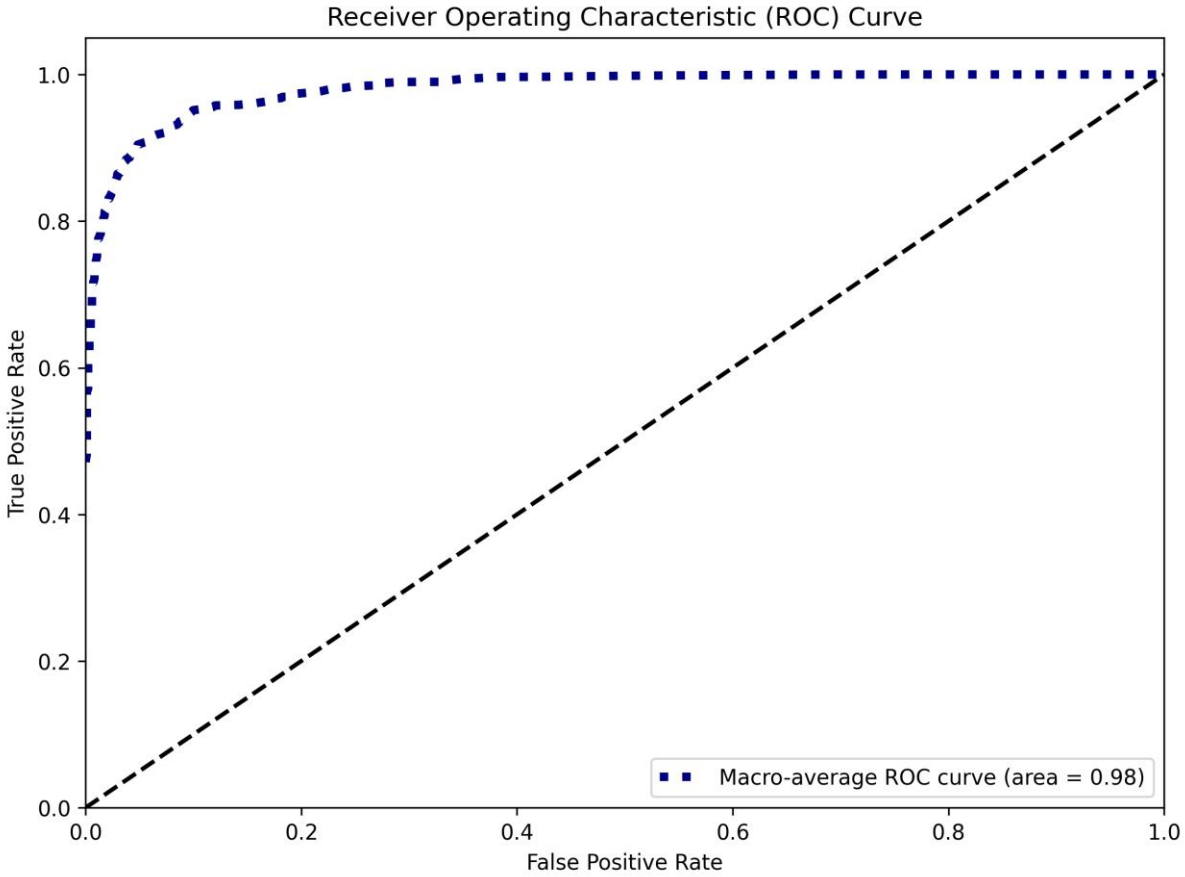
```
=== CNN 실험 결과 ===
데이터셋: Caltech-101
클래스 수: 89
실험 횟수: 1
테스트 정확도: 0.7316
F1-score: 0.5963
훈련 시간: 2444.00초
```



20 epoch와 30 epoch 사이에서 급격하게 떨어지는 구간이 있지만 이후 40 epoch 이후부터는 훈련 정확도와 비슷한 수준으로 수렴하는 모습을 보인다.



특정 클래스에서 confusion이 발생하고 있지만 다른 클래스에서는 일정한 분류 성능 보여줌



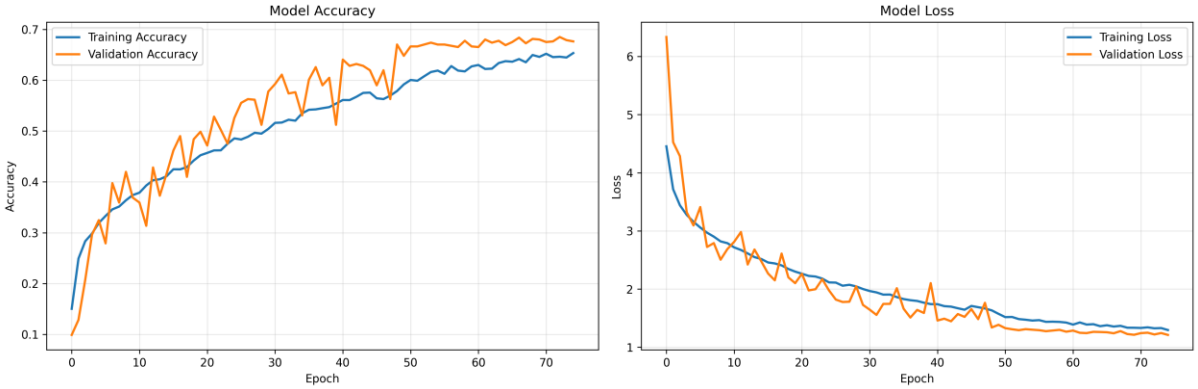
"Macro-average ROC curve (area = 0.97)"로 명시되어 있듯이, AUC 값이 0.98로 높다. AUC는 ROC 곡선 아래 영역의 넓이를 나타내는 지표로, 1에 가까울수록 모델의 성능이 좋다. ROC 곡선이 좌상단에 매우 가깝게 위치하고 있다. 이는 모델이 높은 TPR과 낮은 FPR을 동시에 달성하고 있음을 의미한다.

FC 레이어를 3단으로 실험. 학습시간에 상승이 불가피 하지만 대부분의 연산 시간이 conv 레이어에서 이루어지고 있다. 모델 복잡도가 상승할 것으로 예상된다. 뉴런 수는 512->256->128로 점진적 감소

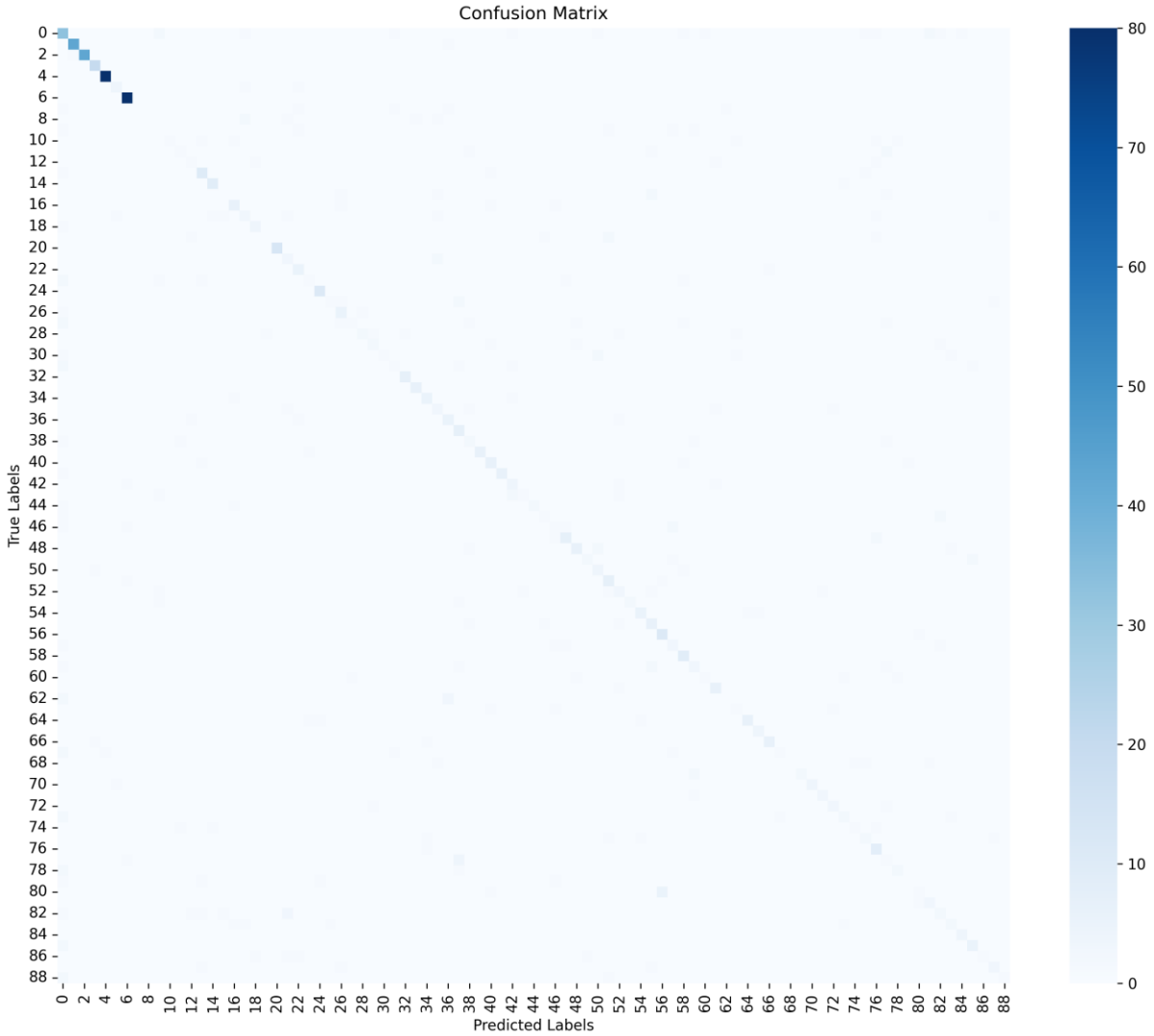
```
Total params: 68,846,553 (262.63 MB)
Trainable params: 68,842,841 (262.61 MB)
Non-trainable params: 3,712 (14.50 KB)
```

```
=== CNN 실험 결과 ===
데이터셋: Caltech-101
클래스 수: 89
실험 횟수: 1
테스트 정확도: 0.6873
F1-score: 0.5098
훈련 시간: 2741.30초
```

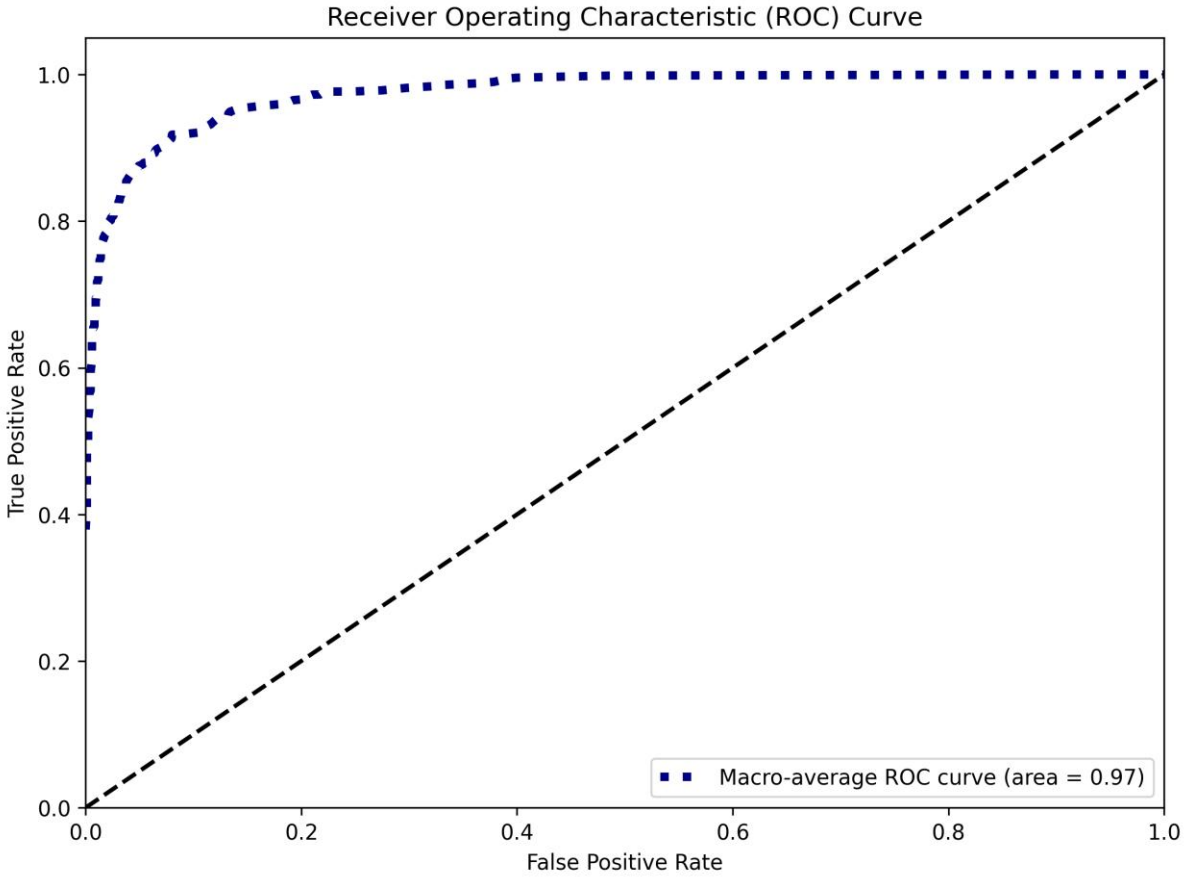
오히려 성능 감소



과적합은 완화되었지만, 전반적인 성능(정확도 약 0.7)은 하락했다.



여전히 일부 클래스에 대한 분류 정확도가 낮을 수 있습니다. 하지만 이전 Confusion Matrix와 비교했을 때, 클래스 불균형으로 인한 영향이 다소 완화



AUC값은 0.01하락, 상단 공간은 여전히 해결되지 않았음. Epoch를 더 크게 했으면 FC layer를 한 층 추가한 것에 따른 성능 효과를 검증할 수 있으나. 하드웨어 리소스와 학습 시간의 제한상 확인하지 못했다.

결론: 모델을 여러 번 평가해서 평균 성능을 비교해야 한다. 하이퍼파라미터간의 연계성과 성능에 미치는 영향력을 몇 번의 튜닝으로 파악하기에는 어려웠다.

각 파라미터가 독립적으로 작용하는 것이 아니라 상호 복합적으로 영향을 미치는 것을 확인할 수 있었다. 특히 FC 레이어를 3개로 증가시켰을 때 정확도가 오히려 감소한 현상은 단순히 레이어 수의 문제가 아니라 드롭아웃 비율, 배치 정규화의 위치, 학습률 등이 복합적으로 작용한 결과로 판단된다.

7.참고문헌

박철수. (2025). 머신러닝. Ch8-3 Convolution Neural Network. 광운대학교 인공지능융합대학. BCML lab.

박철수. (2025). 머신러닝. Ch8-2 Neural Network_RBM. 광운대학교 인공지능융합대학. BCML lab.

박철수. (2025). 머신러닝. Ch8-1 Neural Network. 광운대학교 인공지능융합대학. BCML lab.

및 각주 참고 자료

---

[i] https://www.kaggle.com/datasets/imbikramsaha/caltech-101/data

[ii] https://en.wikipedia.org/wiki/Caltech_101

[iii] https://medium.com/@abhishekjainindore24/pooling-and-their-types-in-cnn-4a4b8a7a4611

[iv] https://en.wikipedia.org/wiki/Softmax_function

[v] https://www.tensorflow.org/install/source_windows?hl=ko, https://www.tensorflow.org/install/source?hl=ko