CIFAR-10 이미지 분류 모델 개발 및 분석

이름**:** 김동환

소속**:** 국민대학교 자동차융합대학 자동차**IT**융합학과

학번**: 20215228**

날짜**: 2024.11.25**

목차

1. 요약

본 연구는 CIFAR-10 데이터셋을 기반으로 한 이미지 분류 모델을 설계하고 최적화하여 테스트 데이터에서 81.61%의 정확도를 달성하였다. 약 647만 개의 파라미터로 높은 효율성과 성능을 구현했으며, 드롭아웃, 배치 정규화 등 다양한 기법을 통해 안정적인 학습을 가능하게 했다. 일부 클래스에서 성능 편차가 확인되었으나, 향후 모델 구조 확장과 데이터 증강 개선을 통해 추가적인 성능 향상을 기대할 수 있다.

2. 서론

이미지 분류는 컴퓨터 비전 분야에서 기초적인 문제 중 하나이다. 이미지 분류란 주어진 이미지를 사전에 정의된 여러 클래스 중 하나로 분류하는 일이다. 이는 다양한 분야에서 활용되는데, 예를 들어 자율주행, 의료 영상 분석, 이미지 검색, 얼굴 인식, 제조 시설 내 불량품 판정 등 다양한 응용 분야에서 필수적인 기술로 자리 잡고 있다.

2.1 이미지 분류의 중요성과 실제 사례

이미지 분류의 중요성으로서는 자동화된 데이터 처리 측면에서 이미지 분류는 대규모 이미지 데이터를 자동으로 분석하고 처리하는 데 기여한다. 예를 들어, SNS에서는 이미지 분류를 사용해 부적절한 컨텐츠를 필터링하거나 자동 태깅을 수행한다. 자율주행에서 응용은 교통 상황을 분석하고 신호등 인식, 보행자 탐지, 차선 유지 등 기능을 수행할 수도 있다. 의료 진단에서도 이미지 분류는 암, 심혈관 질환 등 각종 의료 영상으로 판단 할 수 있는 질병을 조기에 발견할 수 있도록 돕는다.

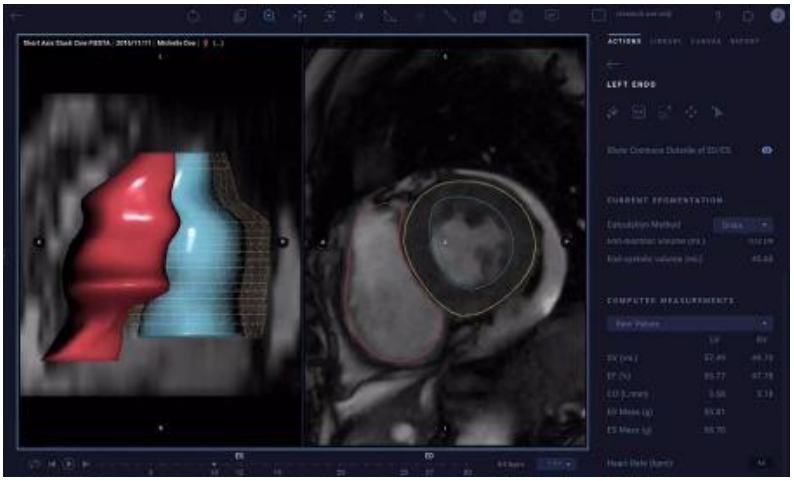


그림1. 아터리스사의 영상 판독 화면.
Fig1. Arteris' video reading screen.[1]

2.2 문제 정의

본 프로젝트에서는 CIFAR-10 데이터셋을 활용해 다중 클래스 이미지 분류 문제를 해결하고자 한다. CIFAR-10 데이터셋은 10개의 클래스로 구성된 총 60,000개의 소형 이미지로 이루어진 데이터셋이다. 본 연구의 문제 정의는 주어진 이미지를 10개의 클래스 중 하나로 분류하기 위한 딥러닝 모델을 개발하는 것이다.

[1] 출처: https://www.apple-economy.com/news/articleView.html?idxno=72326

이를 위해 모델 성능 최적화를 목표로 다양한 모델들을 실험하며, 과대적합 방지 기법(드롭아웃, 데이터 증강 등)을 적용해 모델의 일반화 성능을 확보한다. 또한, 모델의 일반화 성능을 테스트 데이터에서 평가해 실무 환경에서도 활용 가능한 모델을 설계하고자 한다.

2.3 목표

본 연구의 목표는 CIFAR-10 데이터셋에서 80% 정확도 성능을 달성하는 딥러닝 모델을 설계하고자한다. 이를 위해 데이터 증강, 드롭아웃, 배치 정규화 등 다양한 기법을 활용하여 안정적인 학습을 도모한다. 궁극적으로는 학습 시간과 자원을 효율적으로 활용하면서 높은 정확도를 제공하는 모델 설계를 목표한다. 또한, 앞으로는 모델을 더 발전시켜 CIFAR-100과 같은 대규모 데이터셋으로 확장 가능성을 탐구하며, 이 프로젝트를 통해 학문적 성과와 실무적 가치를 모두 확보하는 것을 목표한다.

3. 데이터셋 설명

본 연구에 사용하는 데이터셋인 CIFAR-10(캐나다 고급 연구소) 데이터셋은 머신러닝 및 컴퓨터 비전 알고리즘에 널리 사용하는 이미지 모음이다. CIFAR연구소의 연구원들이 개발하였으며 10가지 클래스의 32x32 컬러 이미지 60,000개로 구성되어 있다. 각 클래스는 비행기(Airplane), 자동차(Automobile), 새(bird), 고양이(Cat), 사슴(Deer), 개(Dog), 개구리(Frog), 말(Horse), 배(Ship), 트럭(Truck)이다. 유의사항으로 클래스는 완전히 상호 배타적이며, 자동차와 트럭 사이에는 중복이 없다. “자동차” 클래스에는 SUV와 세단 등이 포함된다. 또한, “트럭” 클래스에는 대형 트럭만 포함된다. 여기는 픽업트럭도 포함되지 않는다. 데이터셋은 훈련용 데이터 50,000개와 테스트용 데이터 10,000개로 나뉘어 있다. 각 클래스는 훈련 및 테스트 데이터에 균등하게 분포되어 있어 모델 성능 평가에 적합하다.[2]

CIFAR-10 데이터셋은 원본 형태로는 모델 훈련에 적합하지 않다. 이를 위해 정규화, 원-핫 인코딩, 훈련 데이터와 검증 데이터 분리와 같은 전처리 과정을 거친다. 첫 번째로 정규화는 각 픽셀 값을 0부터 1 범위로 정규화하여 모델 학습을 안정적으로 수행할 수 있도록 한다. 두 번째로 원-핫 인코딩은 클래스 라벨을 원-핫 벡터로 변환하여 분류 문제에 적합한 형태로 만든다. 세 번째로 훈련-검증 데이터 분리는 훈련 데이터의 20%를 검증 데이터로 분리하여 과적합 여부를 판단하는 것이다.

그림2. CIFAR-10에서 각 클래스의 임의의 이미지.
Fig2. Random images of each class from CIFAR-10.

CIFAR-10 데이터셋의 특징으로는 이미지의 크기가 작고 간단해 모델 설계 및 실험에 용이하고, 각 클래스가 균등하게 분포되어 있어 데이터 불균형 문제가 없다. 또한 다양한 물체가 포함되어 있어 이미지 분류 모델 개발 기초 단계에 적합하다.

[2] CIFAR-10 Datasets: https://www.cs.toronto.edu/~kriz/cifar.html

4. 모델 설계와 훈련 및 평가

CIFAR-10 데이터셋의 다중 클래스 이미지 분류 문제를 해결하기 위해 인공신경망에서부터 시작해 심층 신경망 그리고 합성곱 신경망 모델(Convolution Neural Network, CNN)을 설계하고 발전시켰다. 모델을 평가하는 주요 지표로는 정확도(Accuracy), 정밀도(Precision), 재현율(Recall), F1-Score, ROC-AUC, 혼동행렬이다. 이를 위해 단일 모델에 대한 훈련, 검증 세트에 대한 정확도와 손실을 시각화하는 함수[3]와 F1-Score, 정밀도-재현율 곡선, ROC 커브를 시각화하는 함수를 만들었다.[4] 손실함수로는 다중 클래스와 라벨을 정수화 시키는 환경에서 사용하는 `sparse_categorical_crossentropy`함수를 사용하였다.

4.1 초기 설계

모델 설계는 단층 인공 신경망(Artificial Neural Network)으로 시작하였다(모델1). 이는 입력 데이터와 출력 클래스 간의 관계를 단순하게 모델링하여 기본 성능을 확인하는데 초점을 맞추었다. 하지만, 층이 매우 얇고 이미지 특성을 살리지 못하는 매우 단순한 밀집층을 사용한 모델인 만큼, 훈련 세트에 대한 정확도도 매우 낮다. 또한, 검증 세트에 대한 정확도는 들쭉 날쭉한 모습을 보인다. 검증 세트에 대한 손실도 마찬가지로 매우 들쭉날쭉한 모습을 보이며 좋지 않은 성능을 보인다.

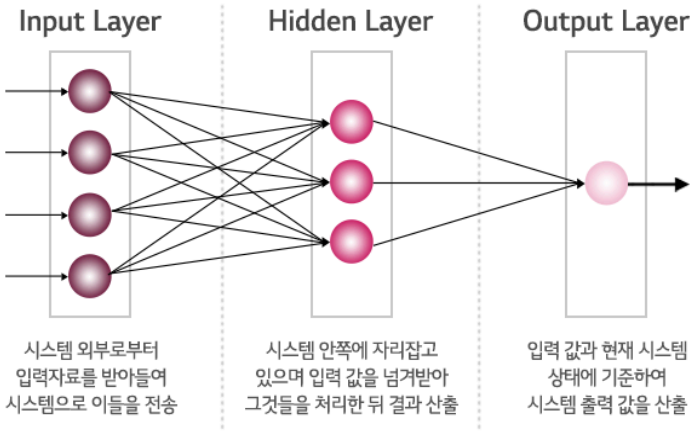


그림3. 기본적인 인공신경망 알고리즘
Fig3. Basic artificial neural network algorithm[5]

```
CIFAR10_1layer_NN = Sequential(
    [Input(shape=(train_scaled.shape[1:])),
    dense = Dense(10, activation='softmax')],
    name='CIFAR10_1layer_NN')
CIFAR10_1layer_NN.compile(loss='sparse_categorical_crossentropy',
                          metrics=['accuracy'])
CIFAR10_1layer_NN.summary()
```

[3] Kim DongHwan, hwan1111. (2024). plot_accuracy_and_loss [Source Code]. https://github.com/hwan1111/Computer-Vision/blob/835b5731a6c78fcea29ad7d10e48ee42c044f225/plot_accuracy_and_loss.py
[4] Kim DongHwan, hwan1111. (2024). visualize_model_char [Source Code]. https://github.com/hwan1111/Computer-Vision/blob/835b5731a6c78fcea29ad7d10e48ee42c044f225/visualize_model_char.py
[5] (출처: LG CNS. (2017). 인공신경망이란 무엇인가? https://www.lgcns.com/blog/cns-tech/ai-data/14558/)

```
Model: "CIFAR10_1layer_NN"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 10)	30,730

```
Total params: 30,730 (120.04 KB)
Trainable params: 30,730 (120.04 KB)
Non-trainable params: 0 (0.00 B)
```

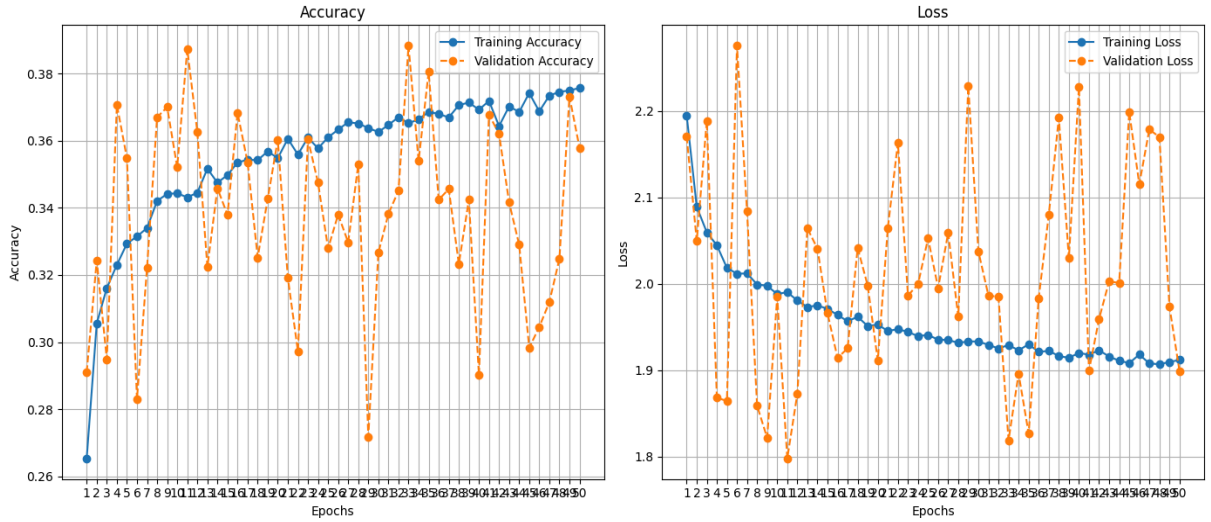


그림4. 단층 신경망 정확도, 손실 그래프
Fig4. Single layer neural network accuracy, loss graph

이를 통해 심층 신경망의 필요성을 확인하였다.

4.2 심층 신경망으로의 확장과 옵티마이저의 추가

단층 신경망에서 하나의 레이어를 추가한 심층 신경망으로 확장하여 하고 옵티마이저를 추가하여 성능을 개선시켰다(모델2). 옵티마이저란 신경망에서 가중치와 편향을 업데이트하는 알고리즘이다. 옵티마이저의 주요 역할은 손실 함수 값 최소화, 학습 속도와 안정성 조절이다. 주요한 옵티마이저의 종류로는 SGD(Stochastic Gradient Descent), Momentum, Adam, RMSProp가 있다.

```
CIFAR10_opt = Sequential(
    [Input(shape=(train_scaled.shape[1:])),
    Dense(100, activation='sigmoid', name='Dense_1'),
    Dense(10, activation='softmax', name='Dense_2')],
    name='CIFAR10_opt')

CIFAR10_opt.compile(
    optimizer='RMSProp',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])

CIFAR10_opt.summary()
```

Model: "CIFAR10_opt"

Layer (type)	Output Shape	Param #
Dense_1 (Dense)	(None, 100)	307,300
Dense_2 (Dense)	(None, 10)	1,010

Total params: 308,310 (1.18 MB)
Trainable params: 308,310 (1.18 MB)
Non-trainable params: 0 (0.00 B)

모델2에는 활성화 함수로 시그모이드를 채택했고, 옵티마이저로 RMSProp를 사용했다. 시그모이드는 초기 특징 추출과 확률적 해석에 강점이 있으며, RMSProp는 CIFAR-10처럼 복잡한 데이터셋에서 안정적인 학습을 지원한다는 강점이 있기 때문이다. 이는 모델 1보다 성능이 대략 20%정도 향상되어서 정확도가 약 45%가 되었다. 들쭉날쭉하던 곡선도 비교적 평탄해졌다. 하지만 아직 좋은 성능이라고는 못하는 수준이다.

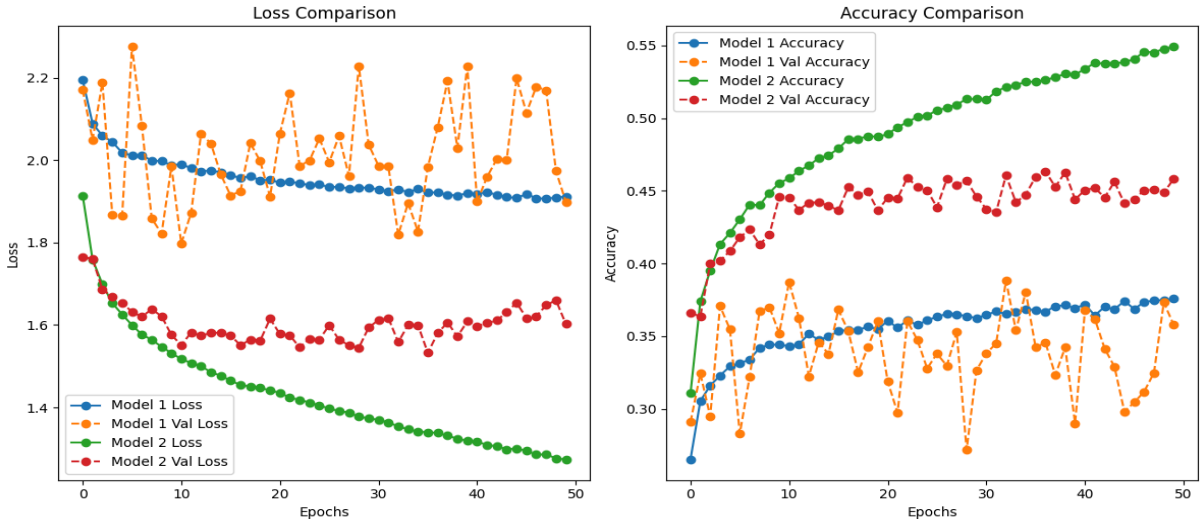


그림5. 단층 신경망과 심층 신경망 손실, 정확도 비교
Fig5. Comparison of single-layer neural network and deep neural network loss and accuracy

4.3 합성곱 신경망(CNN)으로 전환

다층 신경망의 한계를 극복하기 위해 합성곱 신경망으로 모델을 전환했다. 합성곱 신경망(Convolutional Neural Network, CNN)은 이미지 공간 정보를 효율적으로 학습할 수 있도록 설계된 딥러닝 아키텍처로, 컨볼루션 신경층(Convolution layer), 풀링 신경층(Pooling layer), 정류선형유닛(Rectified linear unit; ReLU)을 핵심 요소로 가진다.[6] CNN은 이미지 데이터를 처리하는 데 매우 적합한 구조로, 공간적 구조를 유지하면서 학습할 수 있다.

본 연구에서 설계한 CNN 모델은 LeNet-5 모델을 참고해 설계했다(모델3). LeNet-5 모델은 CNN을 처음 개발한 얀 르쿤(Yann Lecun) 연구팀이 1998년에 개발한 CNN알고리즘 이름이다.

[6] 문성은, 장수범, 이정혁, 이종석. (2016). 기계학습 및 딥러닝 기술동향, 정보와 통신 : 한국통신학회지 = Information & communications magazine, v.33 no.10. pp.49 - 56 (p.53).
https://scienceon.kisti.re.kr/commons/util/originalView.do?cn=JAKO201631641550186&oCn=JAKO201631641550186&dbt=JAKO&journal=NJOU00422884

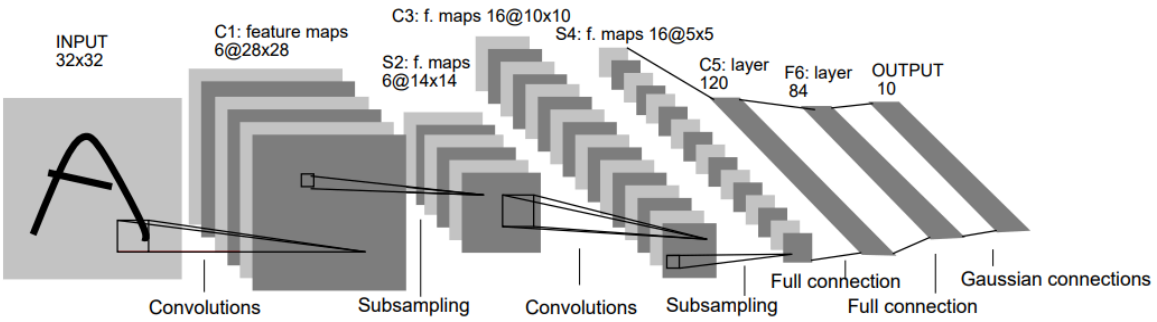


그림6. LeNet-5의 구조

Fig6. Architecture of LeNet-5[7]

CNN으로 전환하면서 모델의 입력 데이터 형태도 변경되었다. 기존의 다층 신경망에서는 입력 데이터를 1차원 형태의 벡터(32x32x3 이미지를 펼친 3072개의 값)로 변환하여 처리하였으나, CNN에서는 이미지의 공간적 정보를 유지할 수 있도록 3차원 텐서의 형태(32x32x3)를 그대로 사용한다.

$$y[m,n] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} x[k,l]h[m-k,n-l]$$

식1. 일반적인 2D 합성곱.

Eq1. General 2D convolution.

```
cnn = Sequential(
    name='LeNet_5',
    layers=[
    Input(shape=(train_scaled.shape[1:]), name='Input'),
    Conv2D(6, (5, 5), padding='valid', activation='relu', name='C1'),
    MaxPooling2D(pool_size=(2, 2), name='S2'),
    Conv2D(16, (5, 5), padding='valid', activation='relu', name='C3'),
    MaxPooling2D(pool_size=(2, 2), name='S4'),
    Conv2D(120, (5, 5), padding='valid', activation='relu', name='C5'),
    Flatten(name='Flatten'),
    Dense(84, activation='relu', name='FC6'),
    Dense(10, activation='softmax', name='Output')
    ]
)

cnn.summary()
```

[7] ^1 Yann LeCun. (1998). Gradient-Based Learning Applied to Document Recognition. https://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf

Model: "LeNet_5"

Layer (type)	Output Shape	Param #
C1 (Conv2D)	(None, 28, 28, 6)	456
S2 (MaxPooling2D)	(None, 14, 14, 6)	0
C3 (Conv2D)	(None, 10, 10, 16)	2,416
S4 (MaxPooling2D)	(None, 5, 5, 16)	0
C5 (Conv2D)	(None, 1, 1, 120)	48,120
Flatten (Flatten)	(None, 120)	0
FC6 (Dense)	(None, 84)	10,164
Output (Dense)	(None, 10)	850

Total params: 62,006 (242.21 KB)
Trainable params: 62,006 (242.21 KB)
Non-trainable params: 0 (0.00 B)

위 처럼 LeNet-5 구조를 쓰면 파라미터의 수는 총 62,006개가 된다. 결코 적다고 할 수 있는 파라미터 숫자이지만 성능을 확인해보면 다음과 같다.

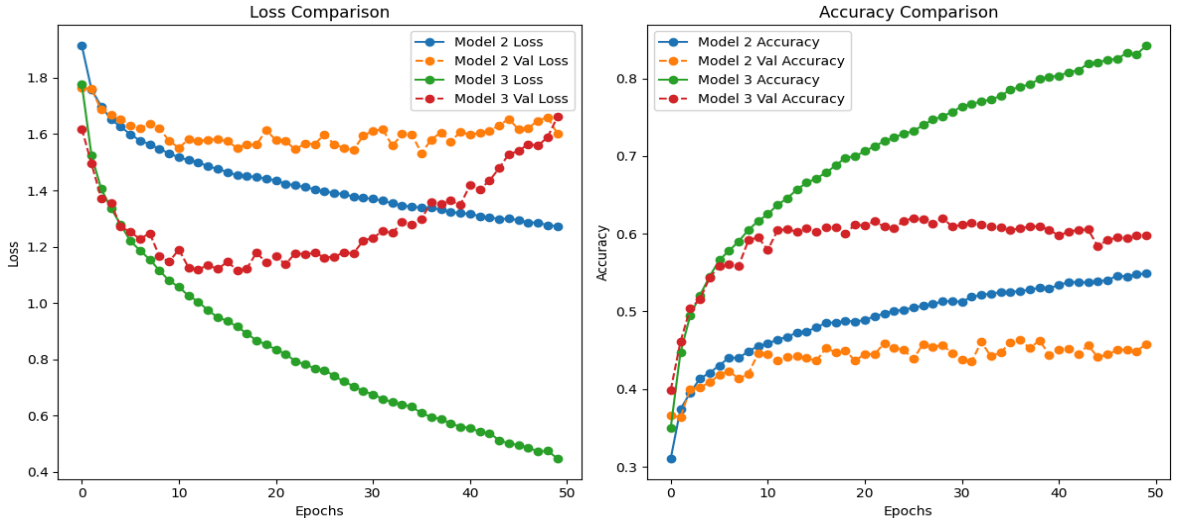


그림7. 심층신경망과 CNN의 손실, 정확도 비교
Fig7. Comparison of loss and accuracy of deep neural network and CNN

기존 심층 신경망과 비교하면 정확도는 40%정도 증가하고, 손실은 감소한 모습을 볼 수 있다. 하지만, 위 그래프를 보면 에포크가 증가할 수록 손실이 점점 증가하고, 정확도가 미세하지만 감소하는 것을 확인할 수 있다. 반면, 훈련세트에 대한 정확도는 증가하고 손실은 계속 감소하는 모습을 보인다. 이는 과대적합이 일어났다고 판단할 수 있는 근거가 된다.

과대적합을 막기 전 전체적인 성능 향상을 위해 기존 모델에 커널 사이즈를 줄이고, 커널의 깊이와 완전연결층인 밀집층의 뉴런 수도 늘렸다.

4.4 과대적합

과대적합(Overfitting)이란 모델이 훈련 데이터에는 높은 성능을 보이지만, 새로운 데이터(검증 데이터나 테스트 데이터 등)에는 일반화 성능이 떨어지는 현상을 말한다. 이는

모델이 훈련 데이터의 노이즈나 세부적인 패턴까지 지나치게 학습해 본질적인 데이터 구조를 제대로 파악하지 못할 때 발생한다. 과대적합이 일어나는 요인은 다양할 수 있는데, 모델이 복잡할 때, 데이터가 부족할 때, 데이터의 특정 클래스가 너무 많거나 적을 때, 적절하지 않은 하이퍼파라미터가 적용됐을 때 등이 있다. 과대적합을 방지하는 방법으로는 대표적으로 드롭아웃, 데이터 증강, 규제, 조기 종료, 충분한 데이터 확보 등이 있다.

```
dpout = Sequential(
    name='dpout',
    layers=[
        Input(shape=(train_scaled.shape[1:]), name='Input'),
        Conv2D(32, (3, 3), padding='same', activation='relu', name='C1'),
        MaxPooling2D(pool_size=(2, 2),name='S2'),
        Dropout(0.25, name='D3'),
        Conv2D(64, (3, 3), padding='same', activation='relu', name='C4'),
        Conv2D(64, (3, 3), padding='same', activation='relu', name='C5'),
        MaxPooling2D(pool_size=(2, 2), name='S6'),
        Dropout(0.25, name='D7'),
        Flatten(name='Flatten'),
        Dense(512, activation='leaky_relu', name='FC8'),
        Dropout(0.5, name='D9'),
        Dense(10, activation='softmax', name='Output')
    ])

dpout.summary()
```

Model: "dpout"

Layer (type)	Output Shape	Param #
C1 (Conv2D)	(None, 32, 32, 32)	896
S2 (MaxPooling2D)	(None, 16, 16, 32)	0
D3 (Dropout)	(None, 16, 16, 32)	0
C4 (Conv2D)	(None, 16, 16, 64)	18,496
C5 (Conv2D)	(None, 16, 16, 64)	36,928
S6 (MaxPooling2D)	(None, 8, 8, 64)	0
D7 (Dropout)	(None, 8, 8, 64)	0
Flatten (Flatten)	(None, 4096)	0
FC8 (Dense)	(None, 512)	2,097,664
D9 (Dropout)	(None, 512)	0
Output (Dense)	(None, 10)	5,130

Total params: 2,159,114 (8.24 MB)
Trainable params: 2,159,114 (8.24 MB)
Non-trainable params: 0 (0.00 B)

그림8. 드롭아웃과 컨볼루션 레이어를 추가한 모델 구조 플롯
Fig8. Model architecture plot with dropout and convolutional layers added

4.4.1 드롭아웃

드롭아웃은 뉴런의 일부를 랜덤하게 비활성화하여 학습하는 기법이다. 이는 뉴런 간의 의존성을 줄이고, 모델이 과도하게 복잡한 패턴을 학습하는 것을 방지한다. 이는 연산량이 크게 증가하지 않으면서 과대적합을 방지한다는 장점을 가지고있다. 다음 플롯은 드롭아웃을 추가한 모델(모델4)와 모델3의 손실, 정확도 비교 플롯이다.

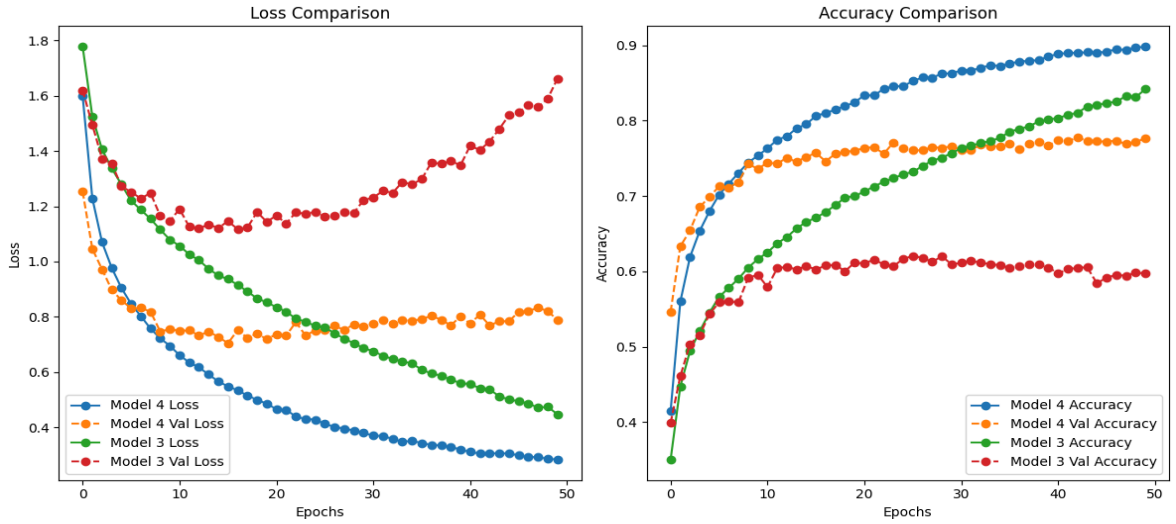


그림9. 드롭아웃을 추가한 모델과 그렇지 않은 모델 손실, 정확도 비교
Fig9. Comparison of loss and accuracy of models with and without dropouts

모델 3과 비교하면 손실은 더욱 낮아졌고, 정확도는 약 18%p 정도 증가하였다. 가장 눈에 띄는 점은 이 뿐 아니라, 손실의 변화이다. 모델 3의 손실은 감소하다가 일정 에포크 이후로는 증가하는 모습을 보이고 있다. 드롭아웃을 추가한 모델 4도 특정 에포크 이후로 손실이 증가하는 모습을 보이고는 있지만, 모델 3에 비해 그 변화가 현저히 적어진 것을 확인할 수 있다. 하지만, 위에서 보다싶이 드롭아웃을 추가한 것 만으로는 과대적합이 완전하게 잡히지는 않았다.

4.4.2 데이터증강

과대적합은 훈련 데이터 즉, 훈련 예제가 부족할 때도 발생한다. 모델이 더 많은 데이터에 학습하면서 일반화 성능을 높이면 과대적합이 억제될 수 있다. 하지만, 새로운 훈련 데이터를 획보한다는 것은 일반적으로 시간과 비용이 많이 드는 작업이다. 반면, 데이터 증강은 손쉽게 데이터 다양성을 늘려주는 방법론 중 하나이다.

데이터 증강에는 대표적으로 좌우로 반전을 하는 수평 뒤집기, 특정 각도만큼 돌리는 회전, 일부를 자르고 크기를 재조정하는 Cropping, 밝기를 무작위로 변형하는 등의 방법이 있다. 이를 통해 하나의 이미지로 다양한 방법들을 조합하여 여러장의 이미지로 만들 수 있다. 본 연구에서는 Keras 전처리 레이어인 `RandomFlip`[8], `RandomRotation`[9], `RandomZoom`[10]을 활용해 데이터 증강을 구현할 것이다. 또한, 원본 데이터와 증강 데이터 병합을 하는 함수를 만들어 원본과 증강 데이터를 적절하게 섞어 성능 저하를 막았다(모델5).[11]

[8] https://keras.io/api/layers/preprocessing_layers/image_augmentation/random_flip/
[9] https://keras.io/api/layers/preprocessing_layers/image_augmentation/random_rotation/
[10] https://keras.io/api/layers/preprocessing_layers/image_augmentation/random_zoom/
[11] Kim DongHwan, hwan1111. (2024). augment_and_combine_data [Source Code]. https://github.com/hwan1111/Computer-Vision/blob/835b5731a6c78fcea29ad7d10e48ee42c044f225/augment_and_combine_data.py

```
augment = Sequential(
    name='augment',
    layers=[
        Input(shape=(train_scaled.shape[1:]), name='Input'),
        Conv2D(32, (3, 3), padding='same', activation='relu', name='C1'),
        MaxPooling2D(pool_size=(2, 2), name='S2'),
        Dropout(0.25, name='D3'),
        Conv2D(64, (3, 3), padding='same', activation='relu', name='C4'),
        Conv2D(64, (3, 3), padding='same', activation='relu', name='C5'),
        MaxPooling2D(pool_size=(2, 2), name='S6'),
        Dropout(0.25, name='D7'),
        Flatten(name='Flatten'),
        Dense(512, activation='leaky_relu', name='FC8'),
        Dropout(0.5, name='D9'),
        Dense(10, activation='softmax', name='Output')
    ]
)

augment.summary()
```

Model: "augment"

Layer (type)	Output Shape	Param #
C1 (Conv2D)	(None, 32, 32, 32)	896
S2 (MaxPooling2D)	(None, 16, 16, 32)	0
D3 (Dropout)	(None, 16, 16, 32)	0
C4 (Conv2D)	(None, 16, 16, 64)	18,496
C5 (Conv2D)	(None, 16, 16, 64)	36,928
S6 (MaxPooling2D)	(None, 8, 8, 64)	0
D7 (Dropout)	(None, 8, 8, 64)	0
Flatten (Flatten)	(None, 4096)	0
FC8 (Dense)	(None, 512)	2,097,664
D9 (Dropout)	(None, 512)	0
Output (Dense)	(None, 10)	5,130

Total params: 2,159,114 (8.24 MB)
Trainable params: 2,159,114 (8.24 MB)
Non-trainable params: 0 (0.00 B)

```
augment_hist = augment.fit(
    train_combined, target_combined,
    batch_size=128, epochs=50,
    validation_data=(val_scaled, val_target),
    verbose=0)
```

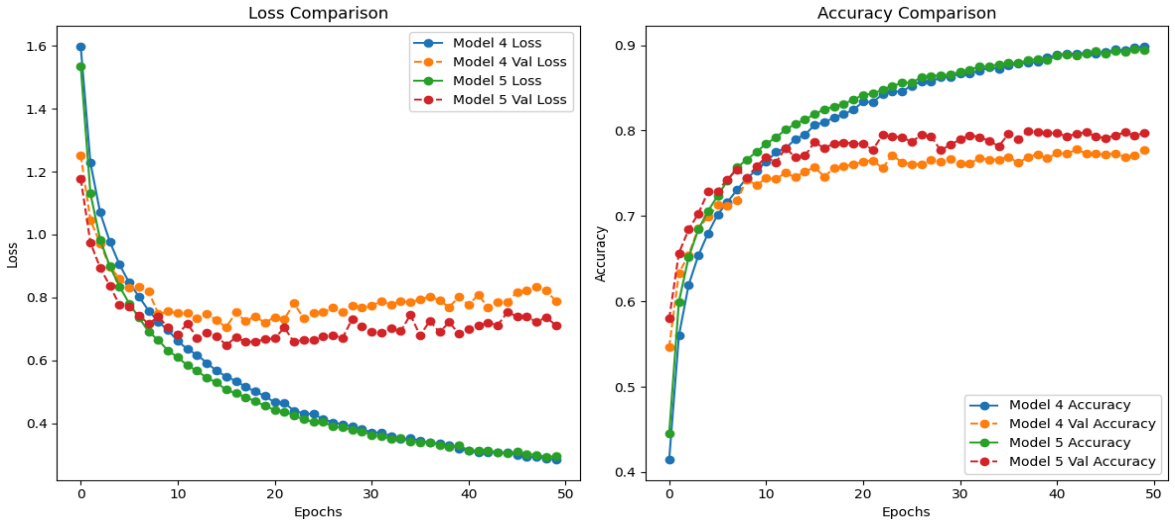


그림10. 데이터 증강을 추가한 모델과 그렇지 않은 모델 손실, 정확도 비교
Fig10. Comparison of loss and accuracy of models with and without data augmentation

데이터 증강 레이어를 추가한 결과 정확도는 소폭 증가하고, 손실도 소폭 감소하였다. 손실의 증가 정도도 데이터 증강을 하지 않은 모델과 비교하였을 때 증가 폭이 감소한 것을 볼 수 있다. 혼동행렬과 PR커브를 확인해본다면 아래와 같다. 대부분의 클래스에서 좋은 성능을 보이는 반면, 새, 고양이, 개가 다른 클래스에 비해 성능이 낮다는 것을 확인할 수 있다.

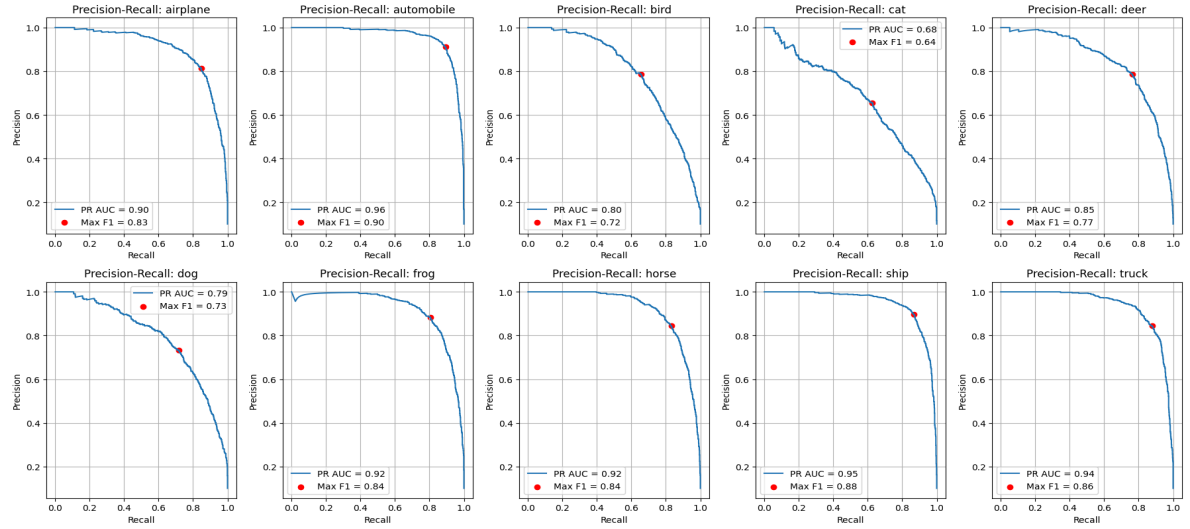
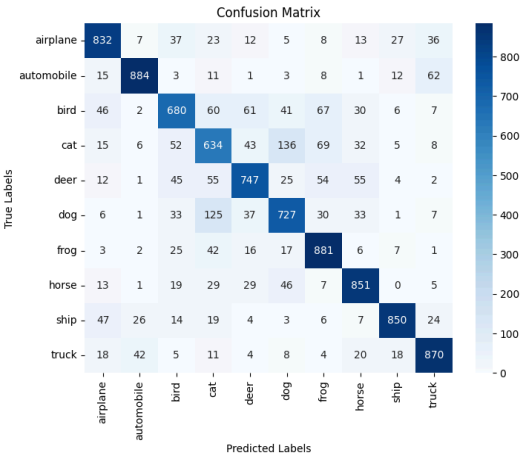


그림11.(위) 데이터 증강을 추가한 모델의 PR커브와 최대 F1-Score를 표시
Fig11.(Up) Displays the maximum F1-Score point within the PR curve of the model with added data augmentation.

그림 12.(아래) 데이터 증강을 추가한 모델의 혼동행렬
Fig12.(Down) Confusion matrix of model with added data augmentation



왼쪽 혼동행렬을 보게된다면, 고양이를 개로 예측한 비율이 약 13.6%이며, 개를 고양이라고 예측한 비율은 12.5%로 나타났다. 이는 고양이와 개의 생김새가 유사하여 모델이 구분하기 어려운 샘플이 존재할 가능성을 시사한다. 또한, 새 클래스의 경우 비행기, 고양이, 사슴, 개, 개구리, 말로 예측된 비율이 고르게 분포되어 있다. 이를 통해 새와 같은 날개를 가진 비행기, 그리고

동물들과의 분류에서 모델 성능이 떨어진다고 추측할 수 있다. 특히, 날개를 가진 비행기와 동물 중 개와 고양이를 구분하는 데 있어 성능이 더 저조한 것을 확인할 수 있다.

4.4.3 L2 정규화

위 방법들을 사용했음에도 원하는 성능과 과대적합이 완전히 잡혔다고 할 수는 없다. 일반화 성능 증가와 일반화 성능증가를 위해 정규화(Regularization)을 적용했다. 그 중 L2 정규화는 과대적합을 방지하기 위해 손실 함수에 가중치의 제곱합을 패널티로 추가하는 기법이다. 이는 모델의 복잡도를 제한하고, 과도한 가중치 증가를 억제하여 안정적인 학습을 돕는다. L2 정규화는 다음과 같은 형태로 손실 함수에 패널티 항을 추가한다.

$$L = L_0 + \lambda \sum_i w_i^2$$

식2. L2 정규화
Eq2. L2 Regularization

L2 정규화는 가중치가 큰 값으로 치우치는 것을 방지하여 모델이 훈련 데이터의 노이즈를 과도하게 학습하지 않도록 한다. 본 연구에서는 첫 번째 컨볼루션(C1), 두 번째 컨볼루션(C4), 첫 번째 완전연결층(FC8)에 l2 정규화를 적용하였다. 이를 통해 컨볼루션 레이어와 완전 연결층에서 발생할 수 있는 과대적합을 효과적으로 방지하였다. 정규화 계수로는 0.0001을 적용하였으며, 반복적인 작업을 통해 드롭아웃과 최적의 조합을 발견하여 모델을 결정하였다.

```
add_l2 = Sequential(
    name='add_l2',
    layers=[
        Input(shape=(train_scaled.shape[1:]), name='Input'),
        Conv2D(32, (3, 3), padding='same', activation='relu',
kernel_regularizer=l2(0.0001), name='C1'),
        MaxPooling2D(pool_size=(2, 2), name='S2'),
        Dropout(0.25, name='D3'),
        Conv2D(64, (3, 3), padding='same', activation='relu',
kernel_regularizer=l2(0.0001), name='C4'),
        Conv2D(64, (3, 3), padding='same', activation='relu', name='C5'),
        MaxPooling2D(pool_size=(2, 2), name='S6'),
        Dropout(0.25, name='D7'),
        Flatten(name='Flatten'),
        Dense(512, activation='leaky_relu', kernel_regularizer=l2(0.0001), name='FC8'),
        Dropout(0.5, name='D9'),
        Dense(10, activation='softmax', name='Output')
    ]
)

add_l2.summary()
```

Model: "add_l2"

Layer (type)	Output Shape	Param #
C1 (Conv2D)	(None, 32, 32, 32)	896
S2 (MaxPooling2D)	(None, 16, 16, 32)	0
D3 (Dropout)	(None, 16, 16, 32)	0
C4 (Conv2D)	(None, 16, 16, 64)	18,496
C5 (Conv2D)	(None, 16, 16, 64)	36,928
S6 (MaxPooling2D)	(None, 8, 8, 64)	0
D7 (Dropout)	(None, 8, 8, 64)	0
Flatten (Flatten)	(None, 4096)	0
FC8 (Dense)	(None, 512)	2,097,664
D9 (Dropout)	(None, 512)	0
Output (Dense)	(None, 10)	5,130

Total params: 2,159,114 (8.24 MB)
Trainable params: 2,159,114 (8.24 MB)
Non-trainable params: 0 (0.00 B)

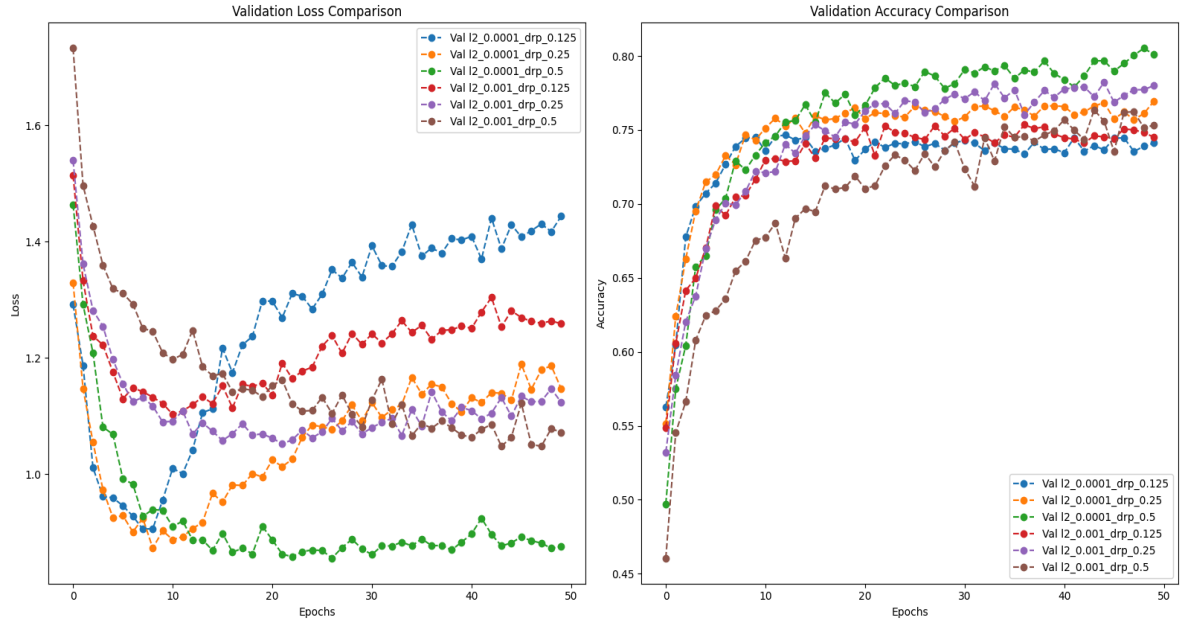


그림13. 드롭아웃, 데이터 증강, L2 정규화를 순차적으로 추가한 모델 정확도, 손실

Fig13. Model accuracy and loss with sequential addition of dropout, data augmentation, and L2 regularization

위 그래프를 보게 된다면 성능은 L2 정규화보다는 드롭아웃 계수에 더욱 민감하게 반응한다는 것을 볼 수 있다. 특히, 드롭아웃 계수가 높을수록(예: 0.5) 손실이 더 안정적으로 감소하며 검증 정확도가 전반적으로 높은 값을 유지하는 경향을 보여준다. 이는 드롭아웃 계수가 높아질수록 뉴런 간의 의존성이 줄어들어 모델이 일반화 성능을 더 잘 확보할 수 있기 때문으로 해석된다.

반면, 드롭아웃 계수가 낮을수록(예: 0.125), 과대적합 영향으로 손실 초기에는 빠르게 감소하지만, 이후 손실이 안정적으로 감소하지 못하거나 정확도가 상대적으로 낮아지는

경향을 보인다. 이와 비교해 L2 규제의 변화는 모델 성능에 상대적으로 덜 민감한 영향을 미치는 것으로 나타났다. 이는 가중치 크기를 제한하는 L2 규제보다 뉴런 비활성화를 통한 과대적합 방지 효과가 더 크게 작용했기 때문으로 해석된다.

따라서, 본 연구에서는 L2 규제와 드롭아웃을 적절히 조합하되, 드롭아웃 계수를 충분히 높게 설정하는 방향으로 최적화를 진행했다.

4.4.4 배치 정규화

배치 정규화(Batch Normalization)는 모델 학습 과정에서 각 미니배치의 입력값을 정규화하여 평균과 분산을 일정하게 유지하는 기법이다. 이는 학습 속도를 향상시키고, 과대적합을 방지하는 데 영향을 미친다. 배치 정규화는 아래와 같은 형태로 입력값을 정규화한다.

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

식3. 배치 정규화
Eq3. Batch Normalization

μ_B와 σ_B^2는 미니배치의 평균과 분산, ϵ은 안정성을 위한 작은 값(분산이 0일때를 대비한)이다. 배치 정규화는 각 레이어의 입력값 분포를 정규화하여 레이어 간 입력 분포 변화를 줄인다. 이는 학습 속도를 증가시키고, 가중치 초기화의 민감도를 완화한다. 또한, 학습 과정에서 이동 평균과 이동 분산을 사용해 추론 시 정규화를 수행한다.

본 연구에서는 컨볼루션 층과 완전 연결 층 사이에 적절히 배치 정규화를 삽입하였다. 첫 번째 배치 정규화는 컨볼루션 레이어(C4) 이후에 삽입(BN5)하여 합성곱 레이어의 출력값을 정규화하였다. BN5는 활성화 함수(ReLU) 적용 전 배치 정규화를 수행함으로써 입력값의 분포를 조정하였다. 중첩된 컨볼루션 레이어 사이에는 배치 정규화를 적용하지 않았으며, 이는 해당 레이어의 활성화 함수가 마지막 합성곱 블록의 Max Pooling 이후 바로 이어지는 구조이기 때문이다. Max Pooling은 이미 입력값의 공간적 정보를 요약하고 크기를 줄이는 역할을 수행하며, 배치 정규화 없이도 출력값의 분포를 안정적으로 유지할 수 있다. 이는 배치 정규화의 추가 사용이 과도한 연산을 초래할 수 있고, 오히려 학습 과정의 효율성을 저하시킬 가능성이 있기 때문이다. 또한, MaxPooling 이후의 출력값은 다음 레이어인 드롭아웃(D8)에서 과대적합 방지 처리가 되므로, 지나친 억제 효과를 줄이기 위해 배치 정규화를 생략하였다. Flatten 이후의 배치 정규화(BN9)는 완전 연결층(FC10)의 입력값을 정규화하여 레이어 간의 입력 분포 변화를 최소화하였다. 이는 FC10의 학습 효율을 높이고, 과대적합을 방지하는 데 기여하였다.

```
final_model = Sequential(
    name='Final',
    layers=[
        Input(shape=(train_combined.shape[1:]), name='Input'),
        Conv2D(32, (3, 3), padding='same', activation='relu',
kernel_regularizer=l2(0.0001), name='C1'),
        MaxPooling2D(pool_size=(2, 2), name='S2'),
        Dropout(0.5, name='D3'),
        Conv2D(64, (3, 3), padding='same', activation='relu',
kernel_regularizer=l2(0.0001), name='C4'),
        BatchNormalization(name='BN5'),
        Conv2D(64, (3, 3), padding='same', activation='relu',
```

```
kernel_regularizer=l2(0.0001), name='C6'),
        MaxPooling2D(pool_size=(2, 2), name='S7'),
        Dropout(0.5, name='D8'),
        Flatten(name='Flatten'),
        BatchNormalization(name='BN9'),
        Dense(512, activation='relu', kernel_regularizer=l2(0.0001), name='FC10'),
        Dropout(0.5, name='D11'),
        Dense(10, activation='softmax', name='Output')
    ]
)

final_model.summary()
```

Model: "Final"

Layer (type)	Output Shape	Param #
C1 (Conv2D)	(None, 32, 32, 32)	896
S2 (MaxPooling2D)	(None, 16, 16, 32)	0
D3 (Dropout)	(None, 16, 16, 32)	0
C4 (Conv2D)	(None, 16, 16, 64)	18,496
BN5 (BatchNormalization)	(None, 16, 16, 64)	256
C6 (Conv2D)	(None, 16, 16, 64)	36,928
S7 (MaxPooling2D)	(None, 8, 8, 64)	0
D8 (Dropout)	(None, 8, 8, 64)	0
Flatten (Flatten)	(None, 4096)	0
BN9 (BatchNormalization)	(None, 4096)	16,384
FC10 (Dense)	(None, 512)	2,097,664
D11 (Dropout)	(None, 512)	0
Output (Dense)	(None, 10)	5,130

Total params: 6,510,624 (24.84 MB)
Trainable params: 2,167,434 (8.27 MB)
Non-trainable params: 8,320 (32.50 KB)
Optimizer params: 4,334,870 (16.54 MB)

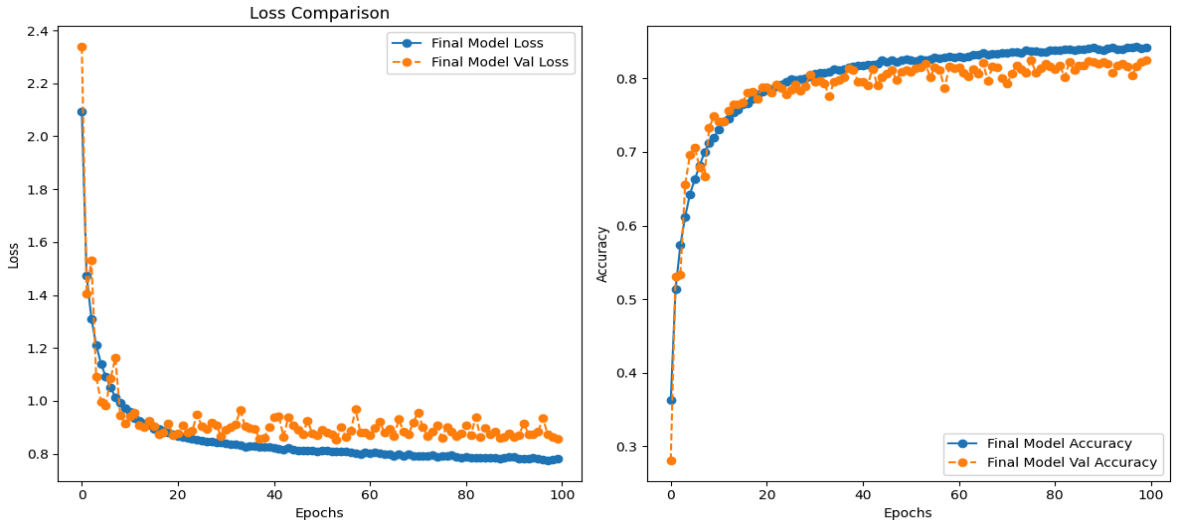


그림14. 배치 정규화를 추가한 모델(최종 모델)

Fig14 Model with batch normalization added (Final Model)

배치 정규화를 추가한 결과, 검증 손실과 검증 정확도가 훈련 데이터셋에 거의 근접할 정도로 가까워졌다는 것을 확인할 수 있다. 이는 과대적합이 발생하지 않고, 모델이 훈련

데이터 패턴을 잘 학습하면서도 검증 데이터에 대해 일반화 성능을 유지하고 있음을 의미한다. 즉, 전반적인 훈련이 잘 이루어졌다는 상태를 의미하기도 한다.

5. 결과 분석

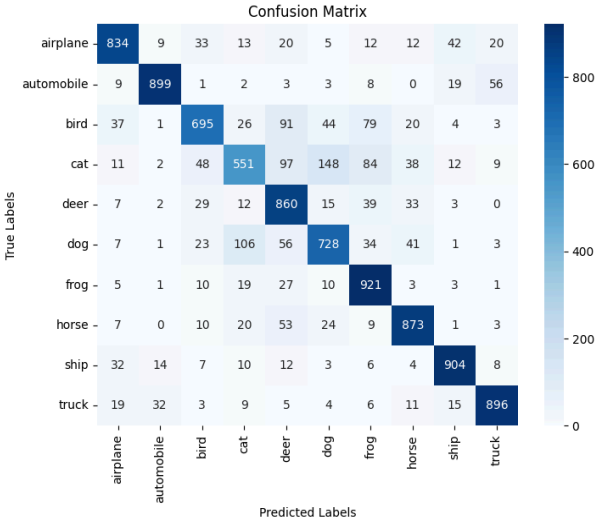
최종 모델에서 목표치인 정확도 80%수준을 넘는 결과가 나왔다. 검증 데이터에서 뿐만 아니라, 테스트 데이터에서도 81.61%의 정확도를 달성하였다. 최종 모델의 구조는 다음 그림과 같다.

그림15. 최종 모델의 구조
Fig15 Architecture of the final model

총 파라미터 개수는 647만개 정도로, 훈련 가능 파라미터는 약 216만개, 옵티마이저 파라미터는 약 431만개 정도이다. 이는 이미지 분류 분야의 대다수 모델과 비교할 때 상대적으로 적은 파라미터 수를 가지면서도, 높은 정확도를 달성했다는 점에서 의미가 크다. 모델의 효율성과 성능 모두를 만족시킨 결과라고 볼 수 있다.

5.1 세부 성능 분석

그림 16. 최종모델 혼동행렬
Fig 16.The Confusion Matrix of Final Model



하지만, 전체 평균적인 성능은 괜찮지만 아쉬운 성능을 보여주는 클래스가 있다. 왼쪽 그림을 보게 된다면 개와 고양이 클래스에서 서로 고양이와 개라고 잘못 예측한 수가 많다. 뿐만 아니라, 동물들끼리 잘못 예측한 경우가 눈에 띈다. 이러한 경우는 다양한 가능성이 존재한다. 대표적으로 클래스간 유사성, 데이터 불균형, 모델의 학습 한계, 데이터 전처리 및 증강의 한계 등이 있을 수 있다. 해당 모델에서는 이러한 성능 저하가 클래스간 유사성과 입력 데이터의 한계와 이미지 해상도 한계 때문이라고 추측된다.

5.2 향후 개선 방안

향후 모델을 개선한다면 첫 번째, 유사 클래스 간 혼동을 줄이기 위해, 더 많은 컨볼루션 레이어와 완전 연결층 레이어를 추가하여 고수준 특징을 학습하도록 개선할 수 있을 것이다. 두 번째, 이렇게 더 깊은 레이어를 설계할 경우 발생할 수 있는 과대적합을 방지하기 위해서는 기존의 드롭아웃과 배치 정규화를 더욱 적극적으로 적용할 필요가 있을 것이다.

마지막으로 데이터 증강 강화로 고양이와 개의 중요한 특징(얼굴 구조, 귀의 형태 등)이 왜곡되지 않도록 데이터 증강 기법을 조정할 수 있을 것이다. 이는 데이터 증강 범위를 제한하거나 클래스별 증강 전략을 다르게 설계해 해결할 수 있을 것이다.

6. 결론

본 연구에서는 CIFAR-10 데이터셋을 대상으로 한 이미지 분류 모델을 설계하고 최적화하여, 테스트 데이터에서 81.61%의 정확도를 달성하였다. 이는 초기 설계 단계에서 설정한 목표치인 80%를 초과 달성한 결과로, 모델 설계와 최적화 과정에서 적용된 기법들의 유효성을 입증했다. 특히, 약 647만개의 파라미터로 비교적 경량화된 구조를 유지하면서 높은 성능을 달성해 연구의 효율성을 입증하였다.

그러나 분석 결과, 일부 클래스 간의 성능 편차가 확인되었다. 특히 고양이와 개 클래스에서 높은 혼동율이 관찰되었으며, 이는 클래스 간 시각적 유사성과 데이터셋의 낮은 해상도에서 기인한 것으로 보인다.

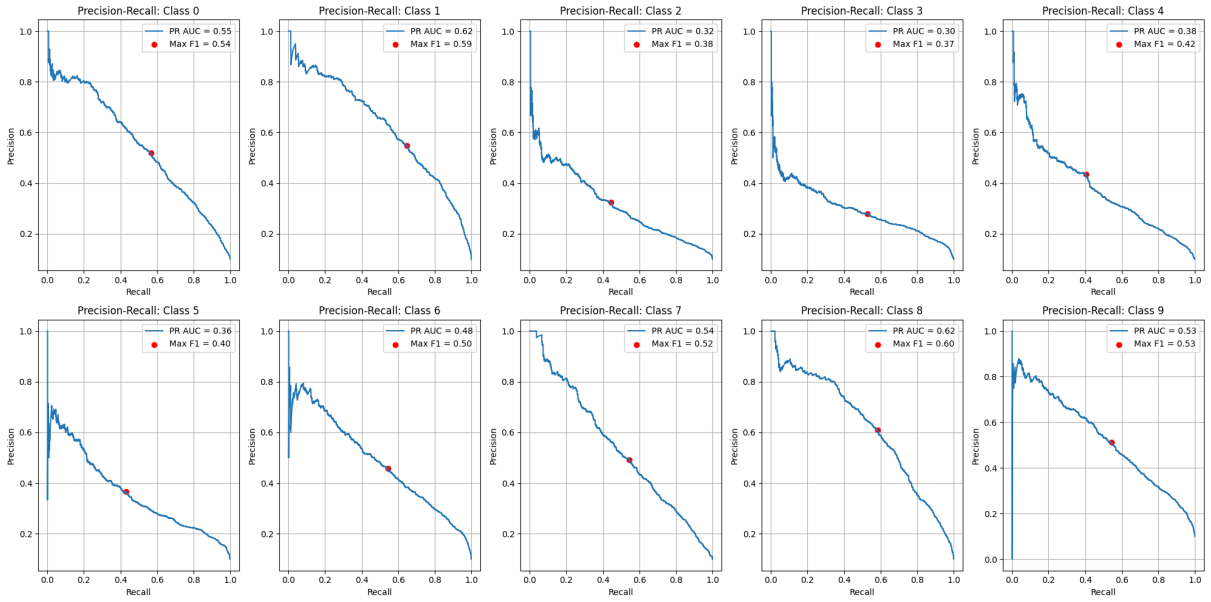
이러한 한계는 데이터 증강 기법 및 모델의 구조 개선을 통해 해결할 여지가 있다.

결론적으로 본 연구의 결과는 효율적인 모델 설계의 중요성을 입증하며, 제안된 개선 방향은 향후 더 복잡한 이미지 분석 문제로 확장 가능성을 제공한다.

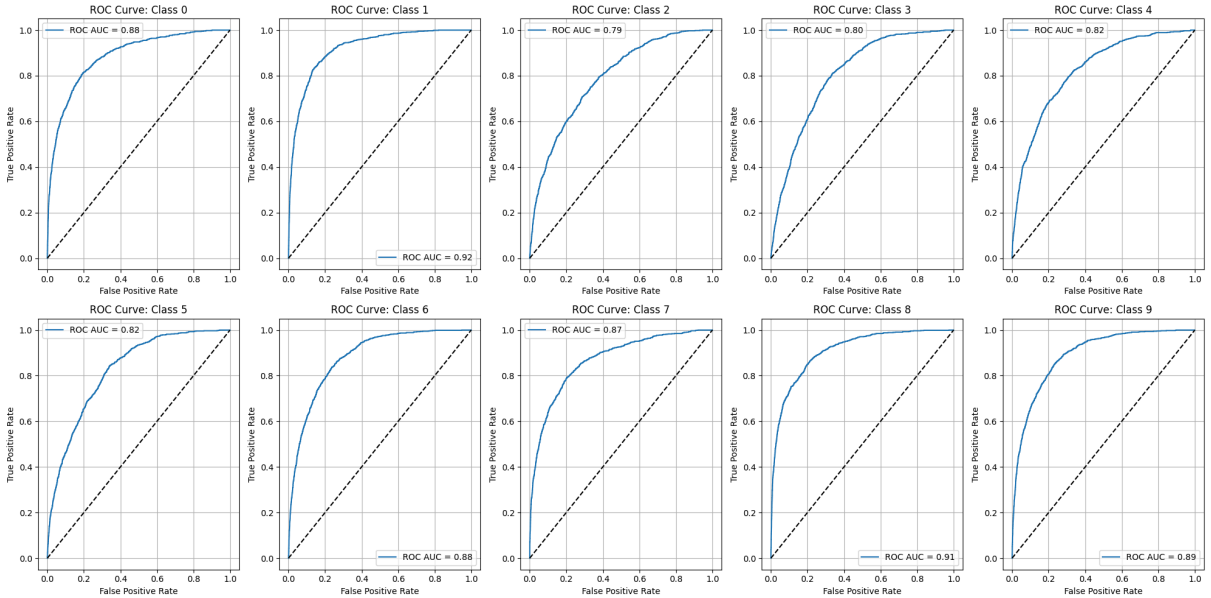
부록. Appendix

A. 모델 2 성능 지표(Model 2 Performance Indicators)

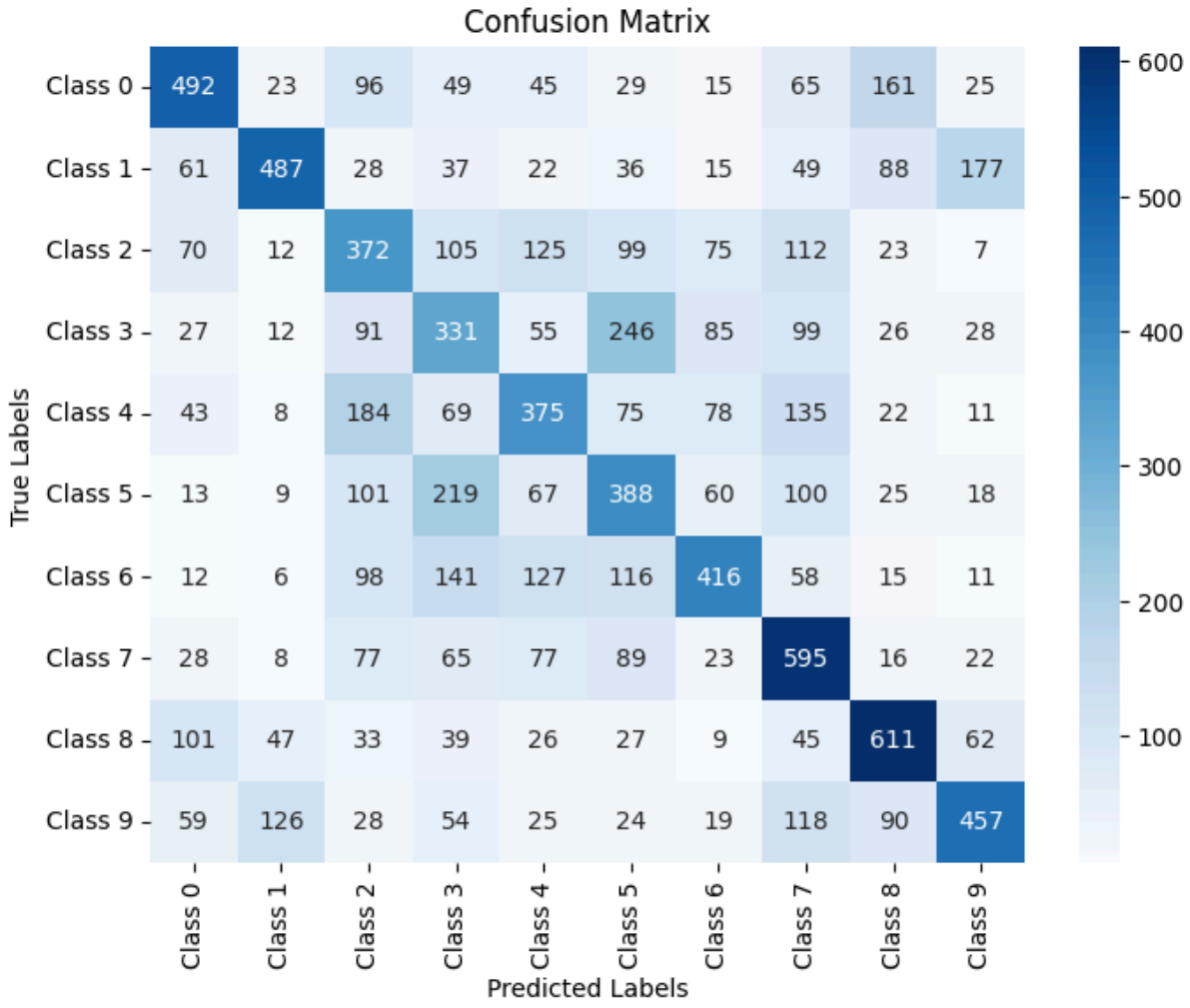
1. PR-Curve



2. ROC-AUC



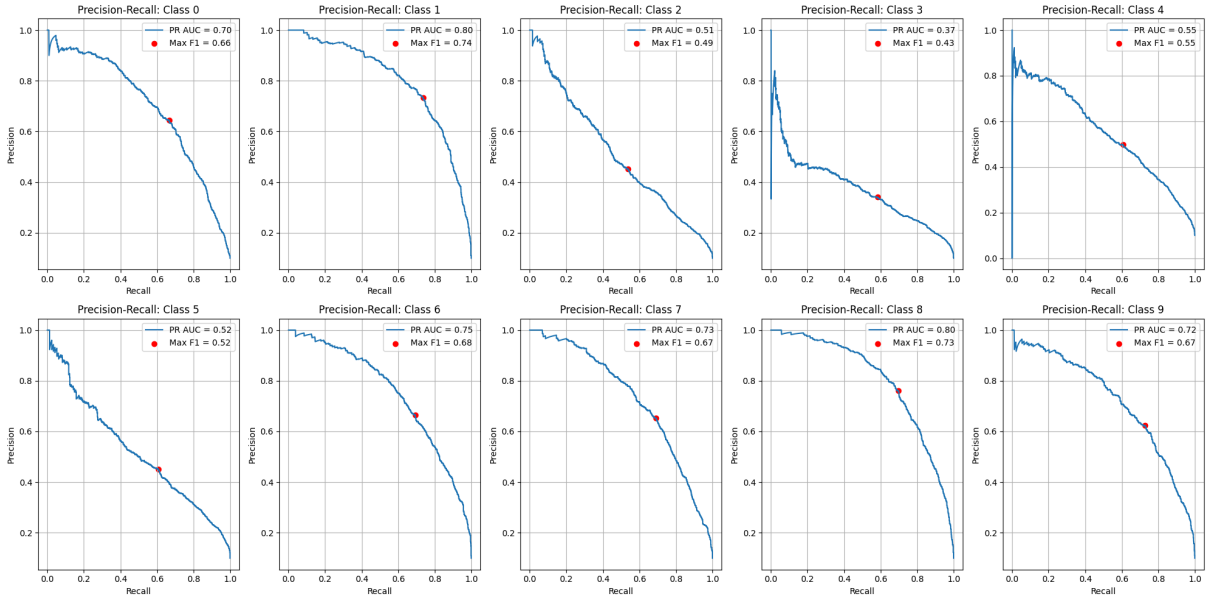
3. Confusion Matrix



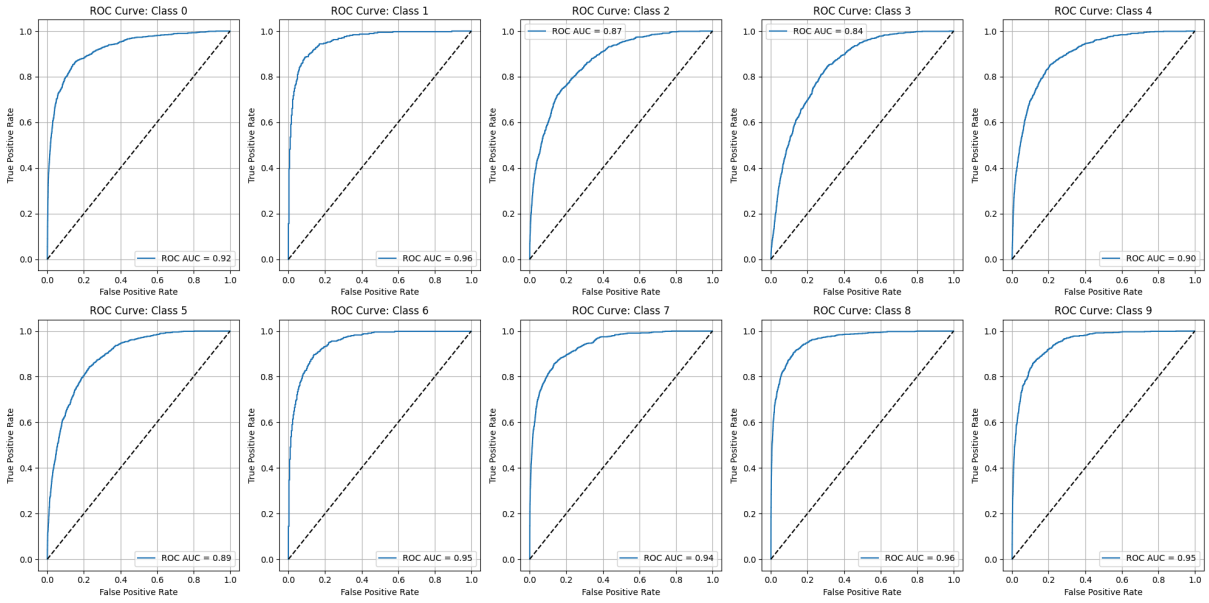
4. Accuracy&Loss History JSON
[model2_history.json]

B. 모델 3 성능 지표(Model 3 Performance Indicators)

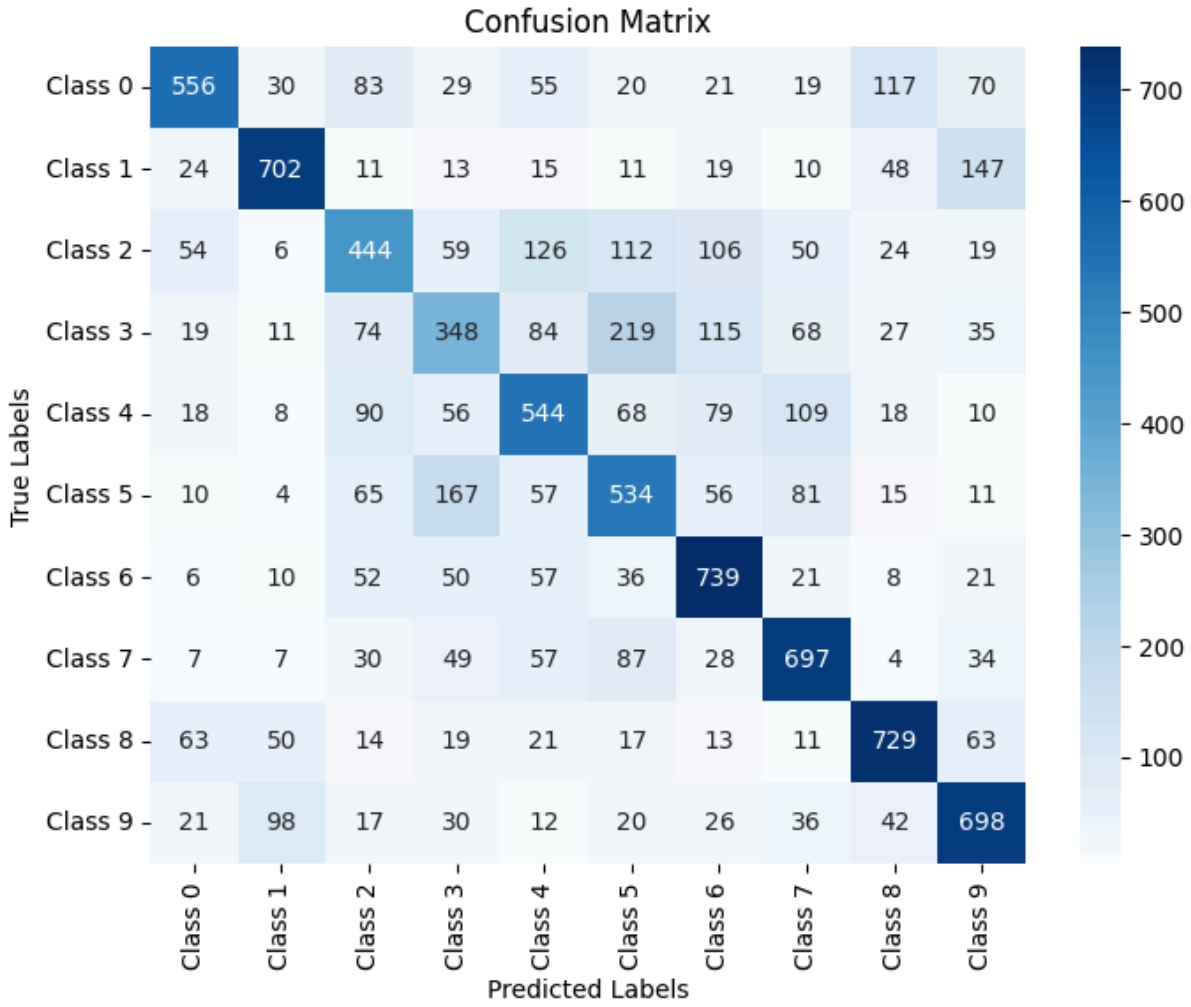
5. PR-Curve



6. ROC-AUC



7. Confusion Matrix

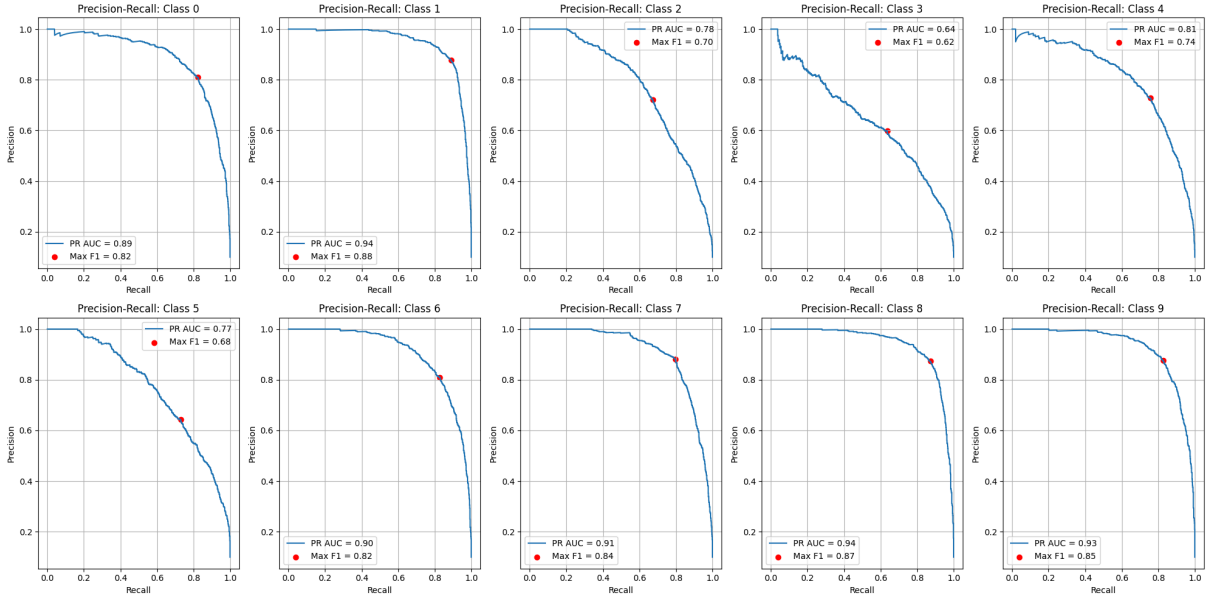


8. Accuracy&Loss History JSON

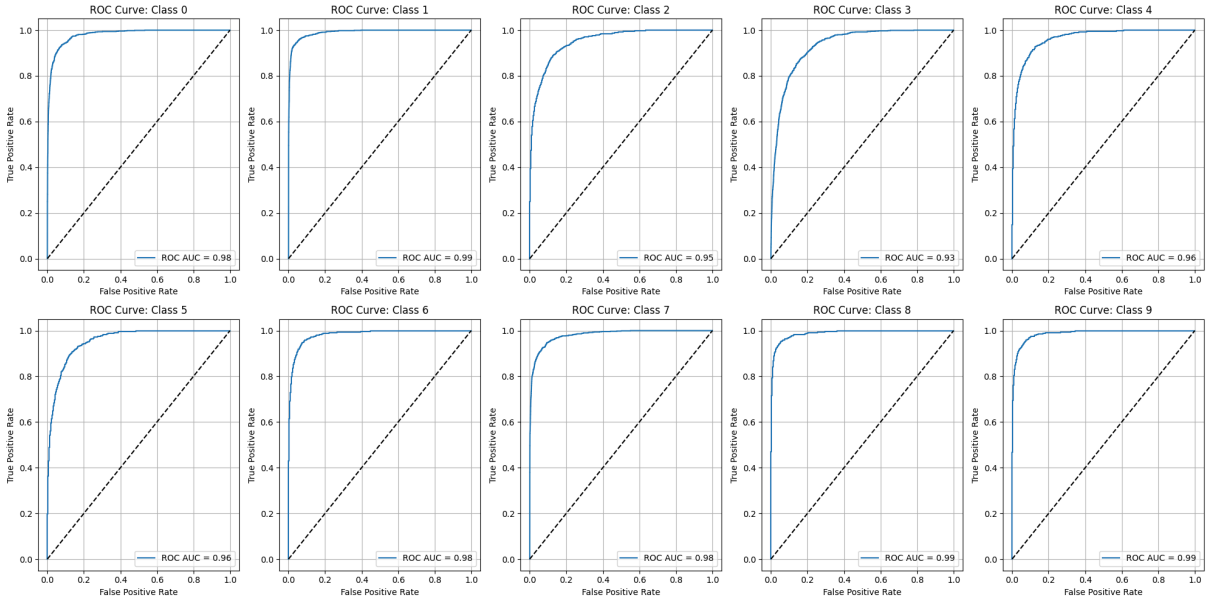
[model3_history.json]

C. 모델 4 성능 지표(Model 4 Performance Indicators)

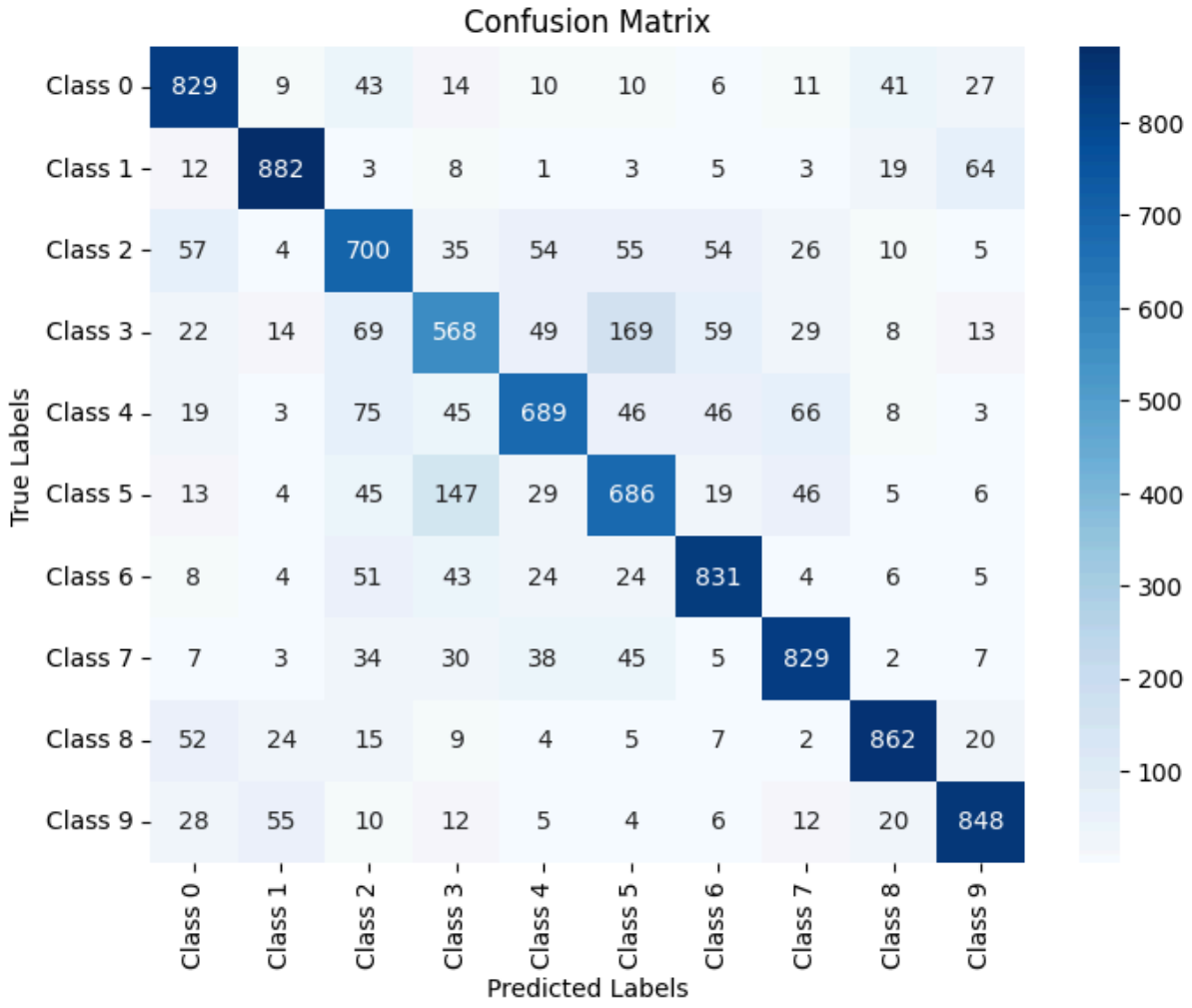
9. PR-Curve



10. ROC-AUC



11. Confusion Matrix

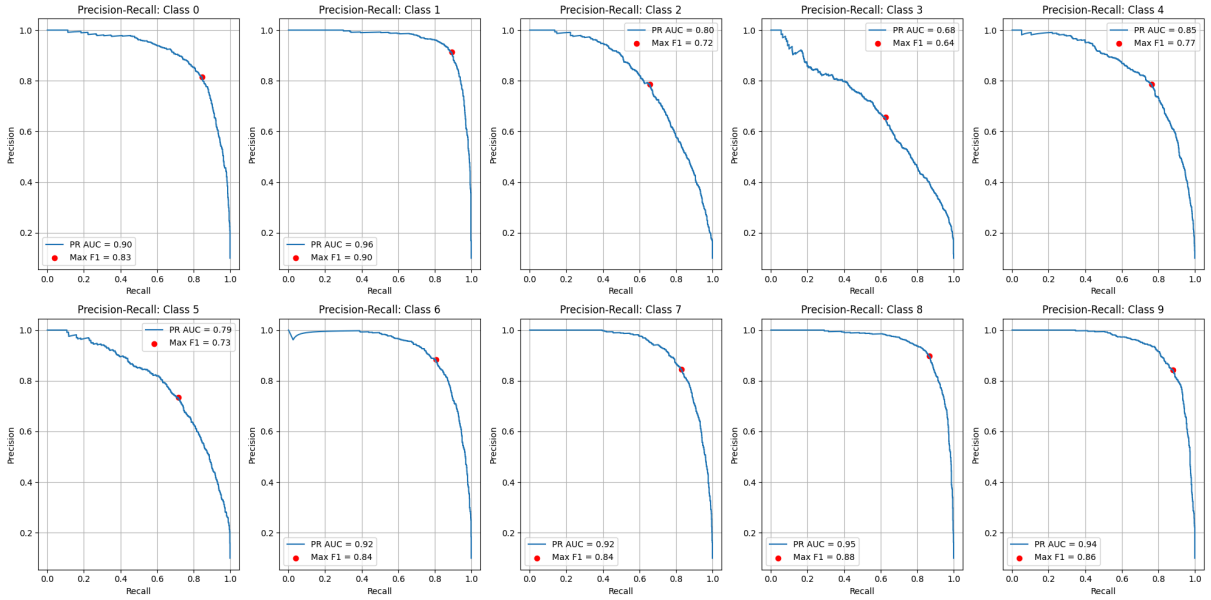


12. Accuracy&Loss History JSON

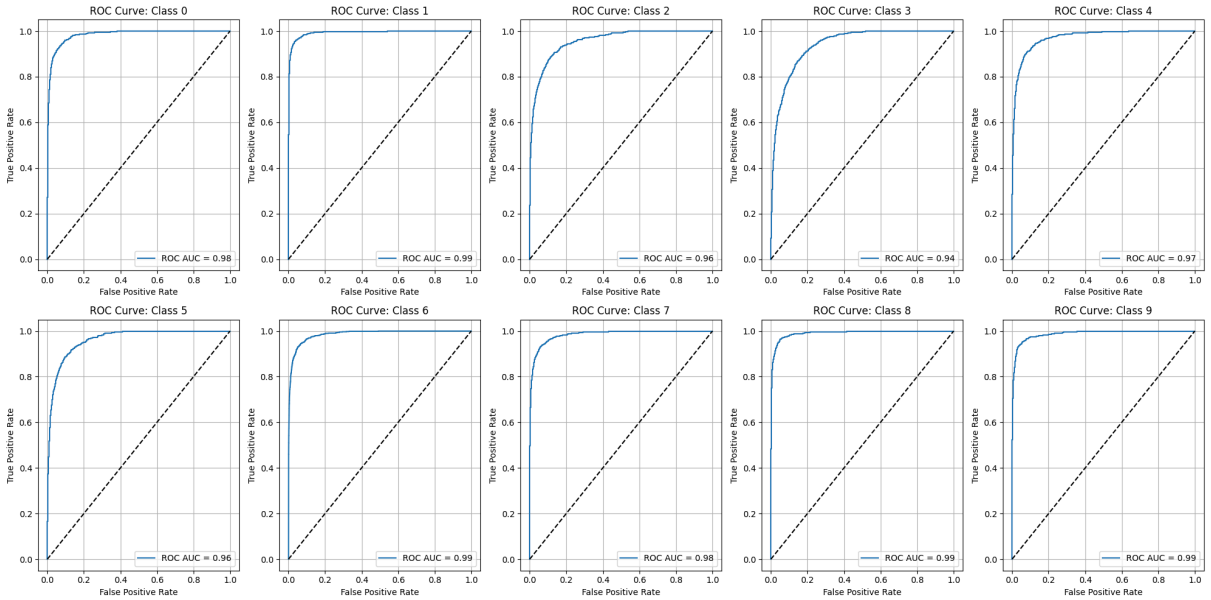
[model4_history.json]

D. 모델 5 성능 지표(Model 5 Performance Indicators)

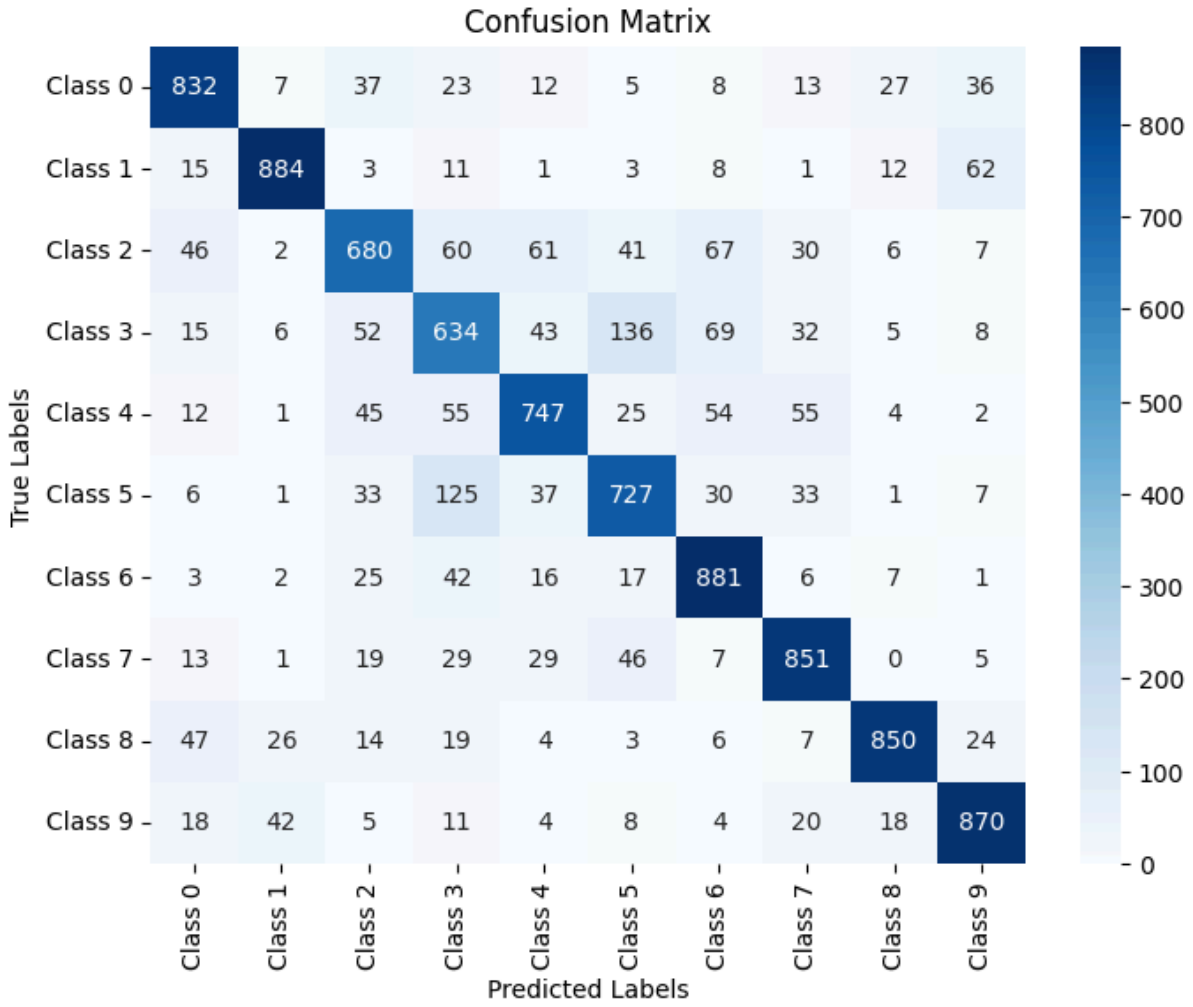
13. PR-Curve



14. ROC-AUC



15. Confusion Matrix

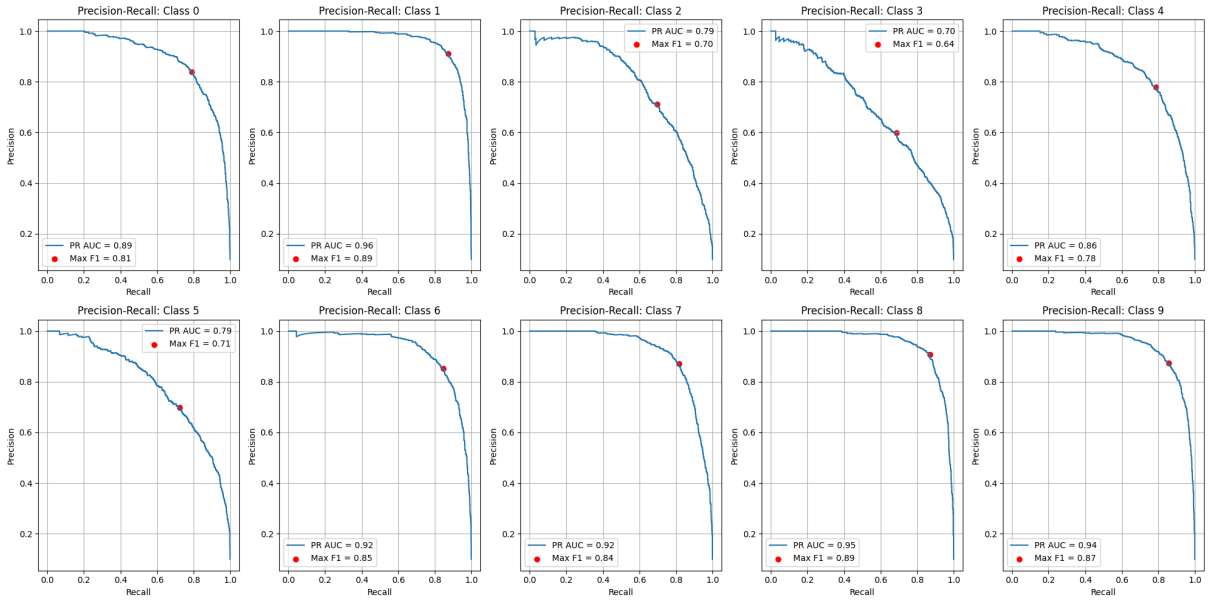


16. Accuracy&Loss History JSON

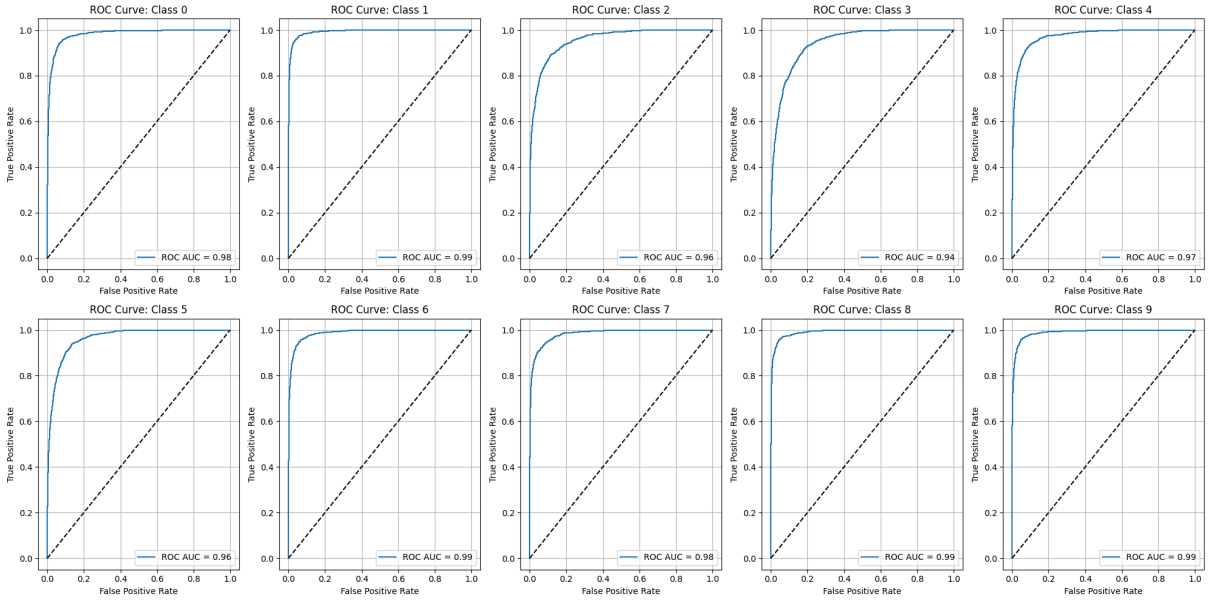
model5_history.json

E. 모델 6 성능 지표(Model 6 Performance Indicators)

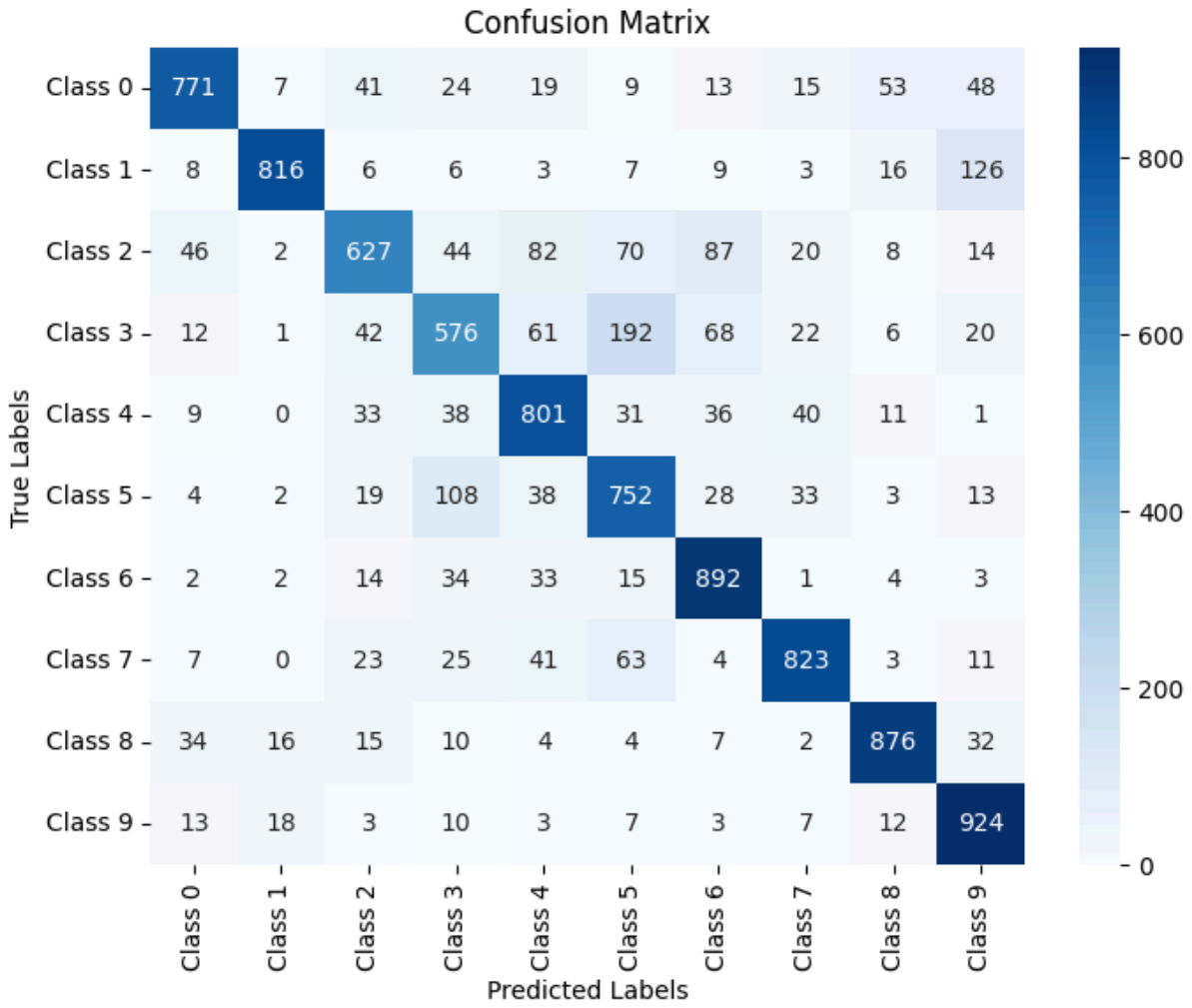
17. PR-Curve



18. ROC-AUC



19. Confusion Matrix

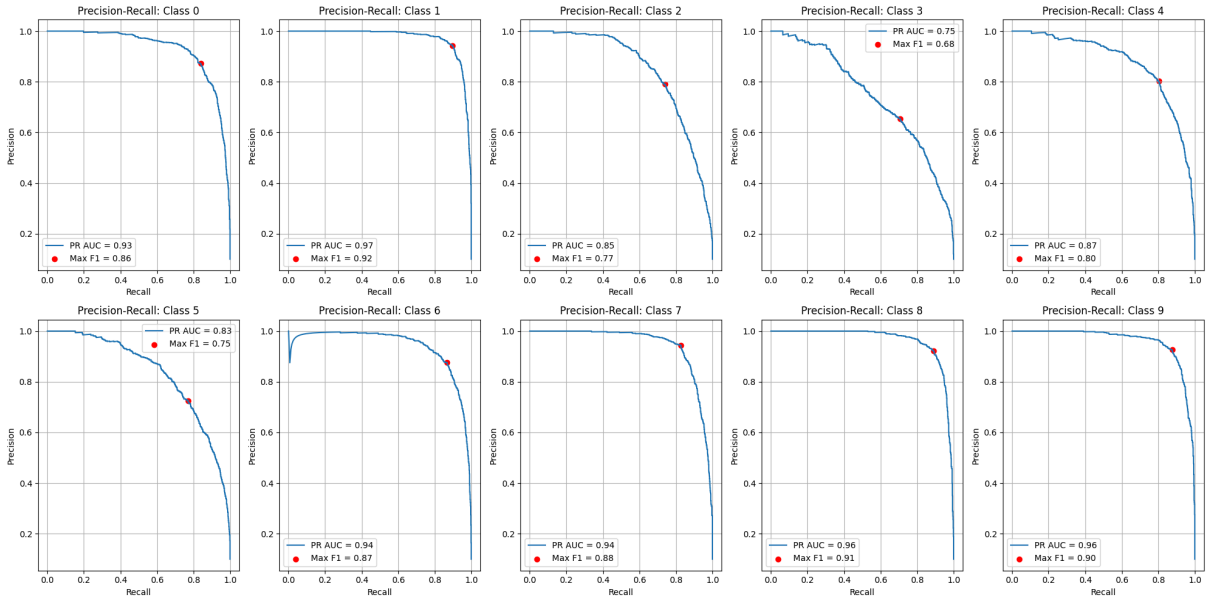


20. Accuracy&Loss History JSON

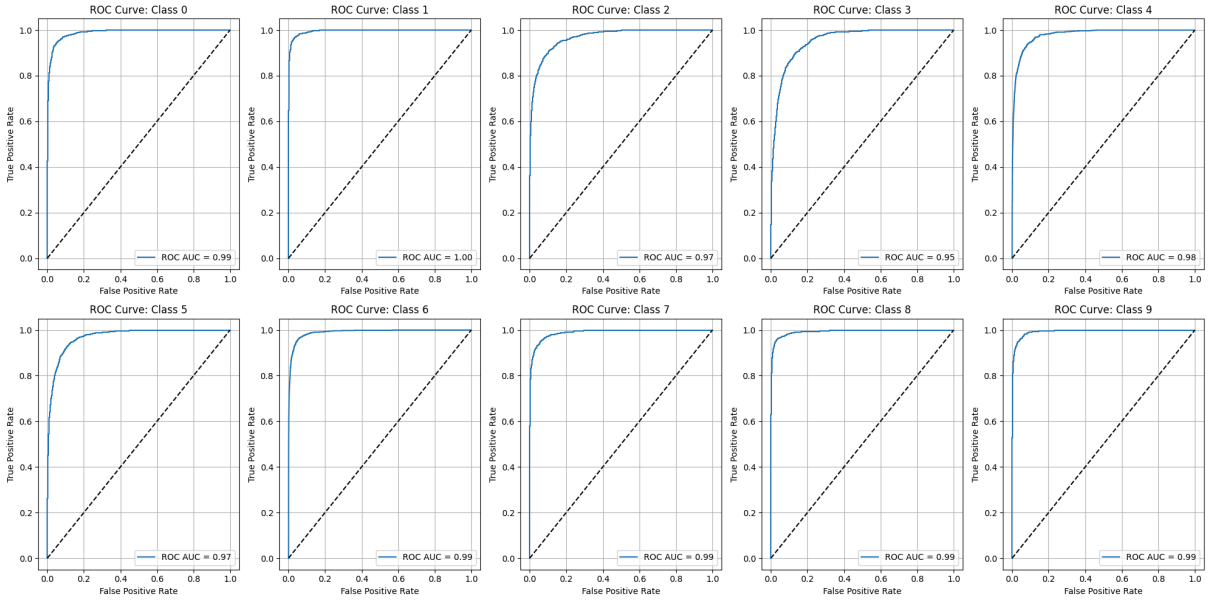
[model6_history.json]

F. 최종 모델 성능 지표(Final Model Performance Indicators)

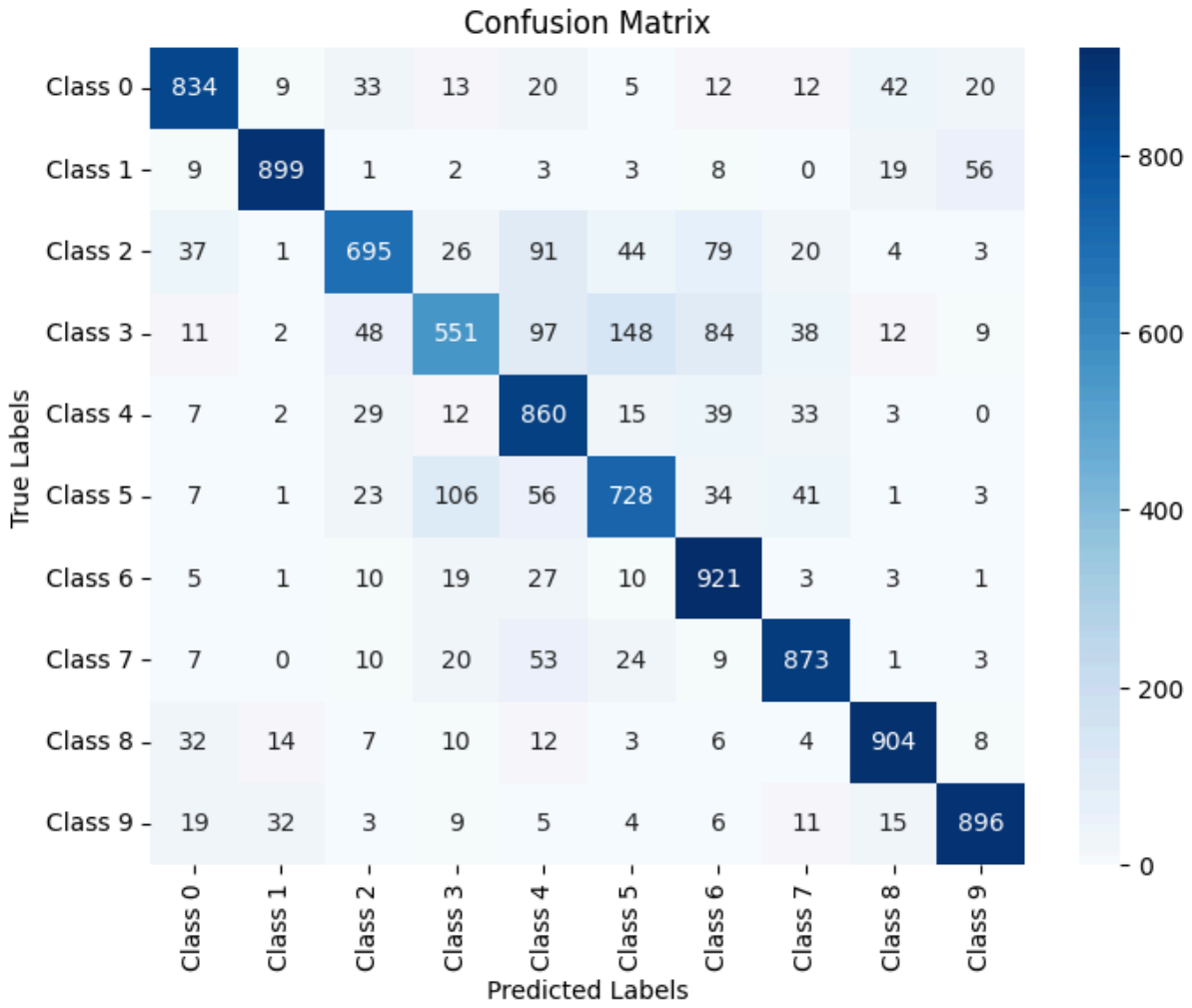
21. PR-Curve



22. ROC-AUC



23. Confusion Matrix



24. Accuracy&Loss History JSON

[batch_model_history.json]