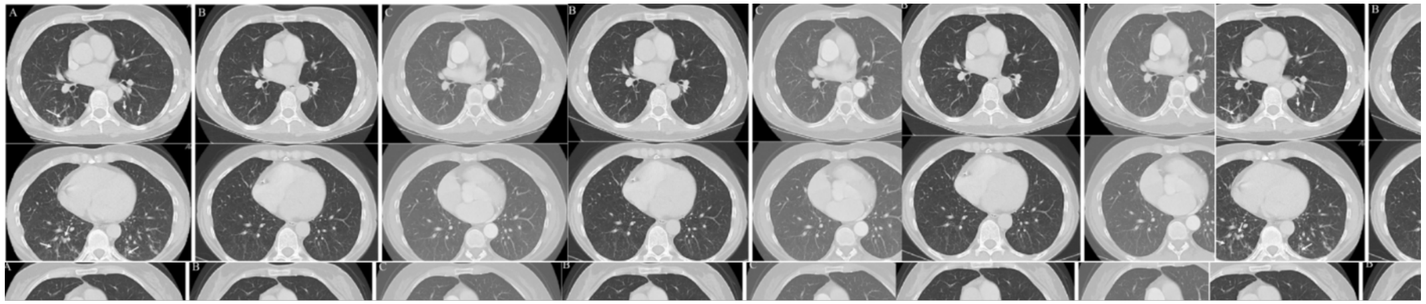
컴퓨터네트워크 과제2

20220502 유민서

Datasets



[Kaggle Chest CT-Scan images Dataset]
https://www.kaggle.com/datasets/mohamedhanyyy/chest-ctscan-images/data

폐암 데이터의 4진분류

Data Preparation

(1) Resize & grayscale

전이학습으로 ResNet을 사용하기 위해 해당 모델에서 사용하는 입력값으로 전처리하고, 이미지를 모두 RGB로 통일하는 방법으로 전처리

```
[3]
resnet_process = tf.keras.Sequential([
    layers.Resizing(224, 224),
    layers.Lambda(lambda image: tf.image.grayscale_to_rgb(image))
])

train_process = train.map(lambda x, y: (resnet_process(x), y))
valid_process = valid.map(lambda x, y: (resnet_process(x), y))
test_process = test.map(lambda x, y: (resnet_process(x), y))
```

Data Preparation

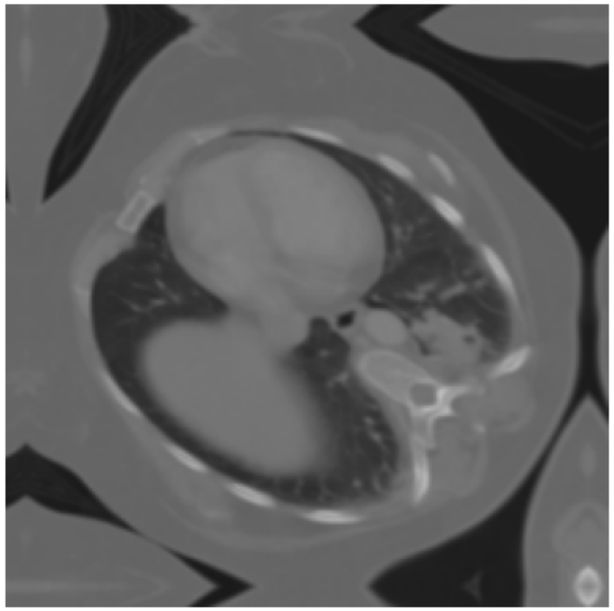
(2) Data Augmentation

tf.keras.layers의 다양한 증강 방법 중 뒤집기, 돌리기, 대비 조정, 확대, 이미지 이동의 방법으로 부족한 이미지 데이터를 보충함.

```python
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.RandomFlip("horizontal"),
    tf.keras.layers.RandomRotation(0.3),
    tf.keras.layers.RandomContrast(0.3),
    tf.keras.layers.RandomZoom(0.3),
    tf.keras.layers.RandomTranslation(0.3, 0.3)
])

# 훈련 데이터 증강
train_aug = train_process.map(lambda x, y:
  (data_augmentation(x, training=True), y))
```

<증강을 시행한 이미지>

adenocarcinoma_left.lower.lobe_T2_N0_M0_Ib



ResNet-50 Transfer learning

<최적의 하이퍼파라미터 탐색>

머신러닝의 gridsearchCV를 응용하여 kerastuner를 이용해 변경 가능한 하이퍼파라미터와 그 값을 리스트화한 뒤, 조건/반복문으로 최적의 변수값을 찾아낼 수 있도록 함

```python
#gridsearch와 유사한 방법으로 가능한 파라미터 설정 후 반복문/조건문 이용
class CNNHyperModel(kt.HyperModel):
    # 최적의 hyperparameter를 찾을 기본적 모델을 정의
    def build(self, hp):
        model = Sequential()
        resnet = tf.keras.applications.ResNet50(weights="imagenet", include_top=False, input_shape=(224, 224, 3))
        model.add(resnet)
        model.add(BatchNormalization())

        # Conv2D 레이어 개수 1~3개 중 최적 / dropout 여부 최적값 확인
        for i in range(hp.Int('num_layers', 1, 3)):
            model.add(Conv2D(filters=hp.Int(f'filters_{i}', 32, 512, step=32),
                             kernel_size=hp.Choice(f'kernel_size_{i}', [3, 5]),
                             activation='relu',
                             padding='same'))
            if hp.Boolean(f'dropout_{i}'):
                model.add(Dropout(rate=0.5))

        model.add(Flatten())

        # Dense 레이어 개수 1~3개 중 최적 / dropout 여부 최적값 확인
        for i in range(hp.Int('num_dense_layers', 1, 3)):  # Choose between 1 to 3 Dense layers
            model.add(Dense(units=hp.Int(f'units_{i}', 64, 512, step=64),
                            activation='relu'))
            model.add(Dropout(rate=0.5))

        model.add(Dense(4, activation='softmax'))

        # 최적화 방법 및 파라미터 최적값 확인
        optimizer = hp.Choice('optimizer', ['adam', 'sgd', 'adamax'])
        if optimizer == 'adam':
            opt = Adam(learning_rate=hp.Float('lr', 1e-4, 1e-2, sampling='log'))
        elif optimizer == 'sgd':
            opt = SGD(learning_rate=hp.Float('lr', 1e-4, 1e-2, sampling='log'), momentum=0.9)
        else:
            opt = Adamax(learning_rate=hp.Float('lr', 1e-4, 1e-2, sampling='log'))

        model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
        return model

hypermodel = CNNHyperModel()

tuner = kt.RandomSearch(
    hypermodel, objective='val_accuracy', max_trials=10, executions_per_trial=1,
    directory='my_dir', project_name='hyperparameter_tuning'
)

earlystop = EarlyStopping(monitor='val_loss', patience=5, mode='min', verbose=1)
lrlow = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=3, mode='min', verbose=1)

tuner.search(train_aug, epochs=10, validation_data=valid_process, callbacks=[earlystop, lrlow])

best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
```

간단한 transfer learning

VGGNet 기반의 간단한 전이학습 모델로 학습

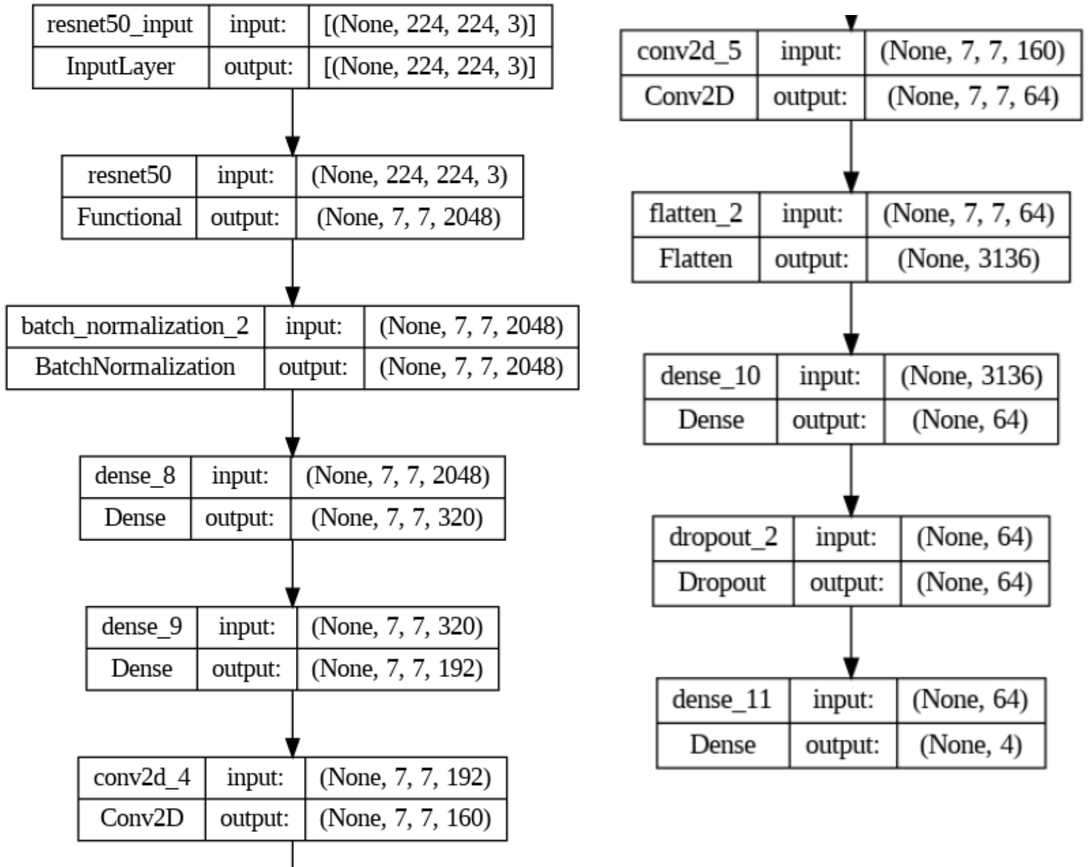
```python
# vggnet을 사용한 코드
from tensorflow.keras.applications import VGG16

vgg_preprocess = tf.keras.Sequential([
    layers.Resizing(180, 180),
    layers.Lambda(lambda image: tf.image.grayscale_to_rgb(image))
])

# 전처리 함수를 각 데이터셋에 적용
train_vgg_process = train.map(lambda x, y: (vgg_preprocess(x), y))
valid_vgg_process = valid.map(lambda x, y: (vgg_preprocess(x), y))
test_vgg_process = test.map(lambda x, y: (vgg_preprocess(x), y))

# 전처리된 훈련데이터를 증강
train_vgg_aug = train_vgg_process.map(lambda x, y:(data_augmentation(x, training=True), y))

# VGG16 모델 로드 이후 간단한 모델 디자인
base_model = tf.keras.applications.vgg16.VGG16(weights="imagenet", include_top=False, input_shape=(180, 180, 3))
model = Sequential()
model.add(base_model)
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(4, activation='softmax'))

# 모든 층을 학습 가능하게 설정
for layer in base_model.layers:
    layer.trainable = True
```

Test Accuracy: 0.4984
Test Loss: 1.1011

ResNet-50 Transfer learning

<모델 구조>

I2정규화를 이용해 과적합 방지
kernel_regularizer=I2(0.001)를 레이어 파라미터에 추가

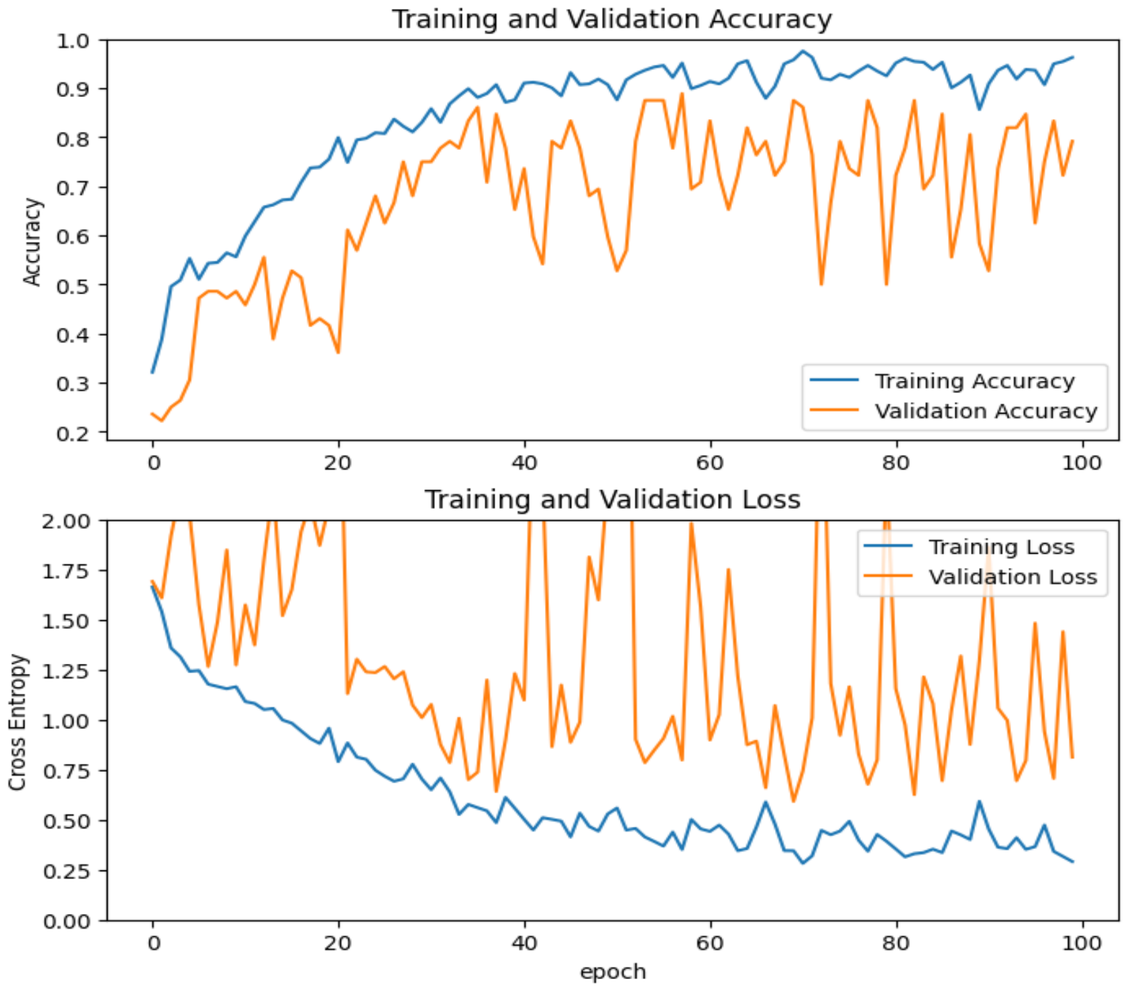


<학습 방법>

compile: 최적화 방법 - Adam, 학습률은 search했던 값
손실함수 - categorical_crossentropy, 평가지표 - 정확도
fit: 50 epoch동안 학습

```python
# l2 0.001
tf.random.set_seed(1234)
# 전이학습 base모델
resnet = tf.keras.applications.ResNet50(weights="imagenet", include_top=False,
                                        input_shape=(224, 224, 3))

# 나머지 모델구조 디자인
model = Sequential()
model.add(resnet)
model.add(BatchNormalization())   # 배치정규화
model.add(Dense(320, activation='relu'))
model.add(Dense(192, activation='relu'))
model.add(Conv2D(160, (5, 3), activation='relu', kernel_regularizer=l2(0.001),
                 padding='same'))
model.add(Conv2D(64, (5, 3), activation='relu', kernel_regularizer=l2(0.001),
                 padding='same'))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4, activation='softmax'))

model.compile(loss="categorical_crossentropy", optimizer=Adam(0.00011845329206998704),
              metrics=["accuracy"])
history_1 = model.fit(train_aug, validation_data=valid_process, epochs=50)
```

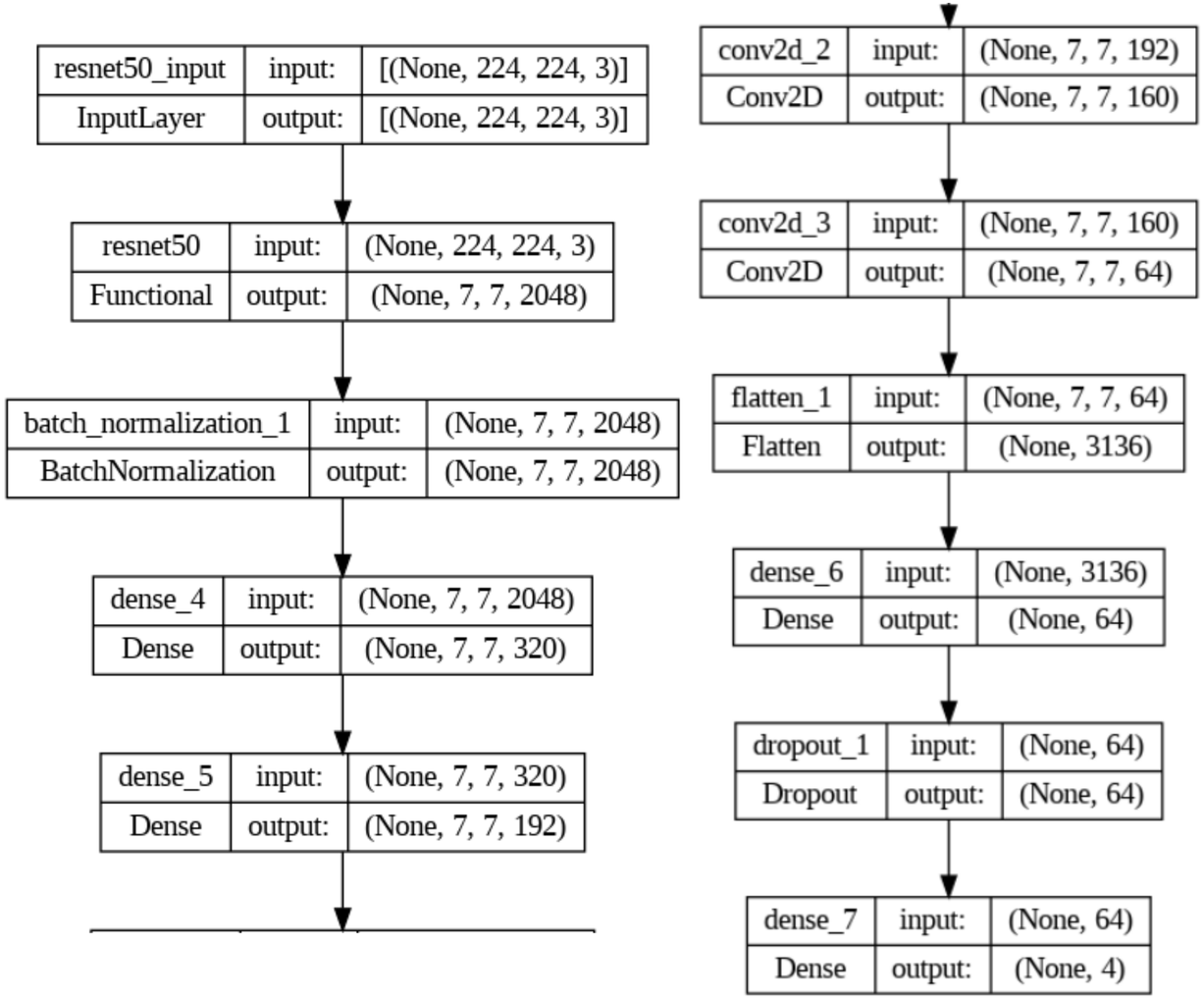
ResNet-50 Transfer learning



100 epoch동안 학습
lloss: 0.2906 - accuracy: 0.9625
val_loss: 0.8122 - val_accuracy: 0.7917

Fine-tuning

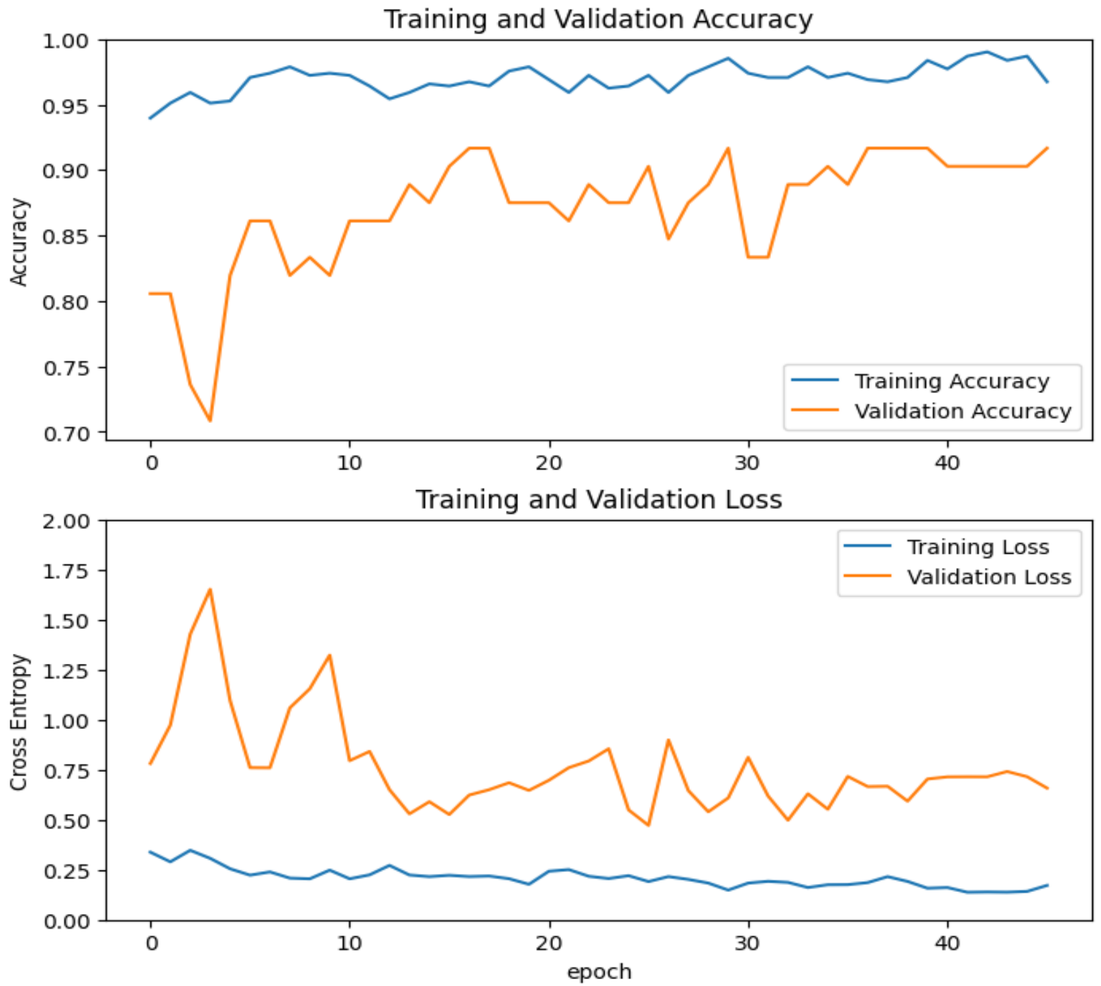
<모델 구조>



<학습 방법>

compile: 최적화 방법 - Adam, 학습률 0.0001
손실함수 - categorical_crossentropy, 평가지표 - 정확도
fit: 100 epoch동안 학습
callback으로 검증 손실이 20번의 epoch 동안 개선되지 않으면 훈련을 중단하고(EarlyStopping), 검증 손실이 15번의 epoch 동안 개선되지 않으면 학습률을 0.1배로 축소(ReduceLROnPlateau)

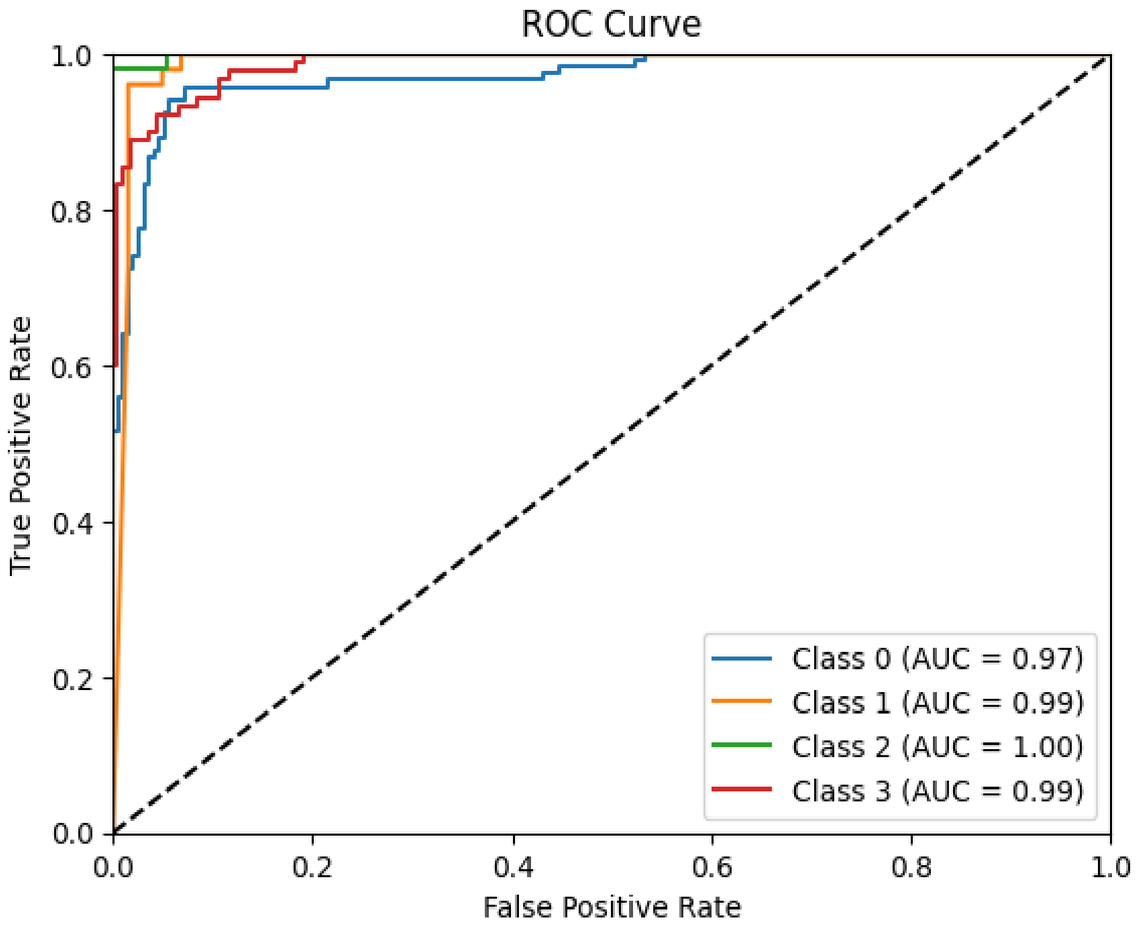
Fine-tuning



loss: 0.1716 - accuracy: 0.9674
val_loss: 0.6577 - val_accuracy: 0.9167

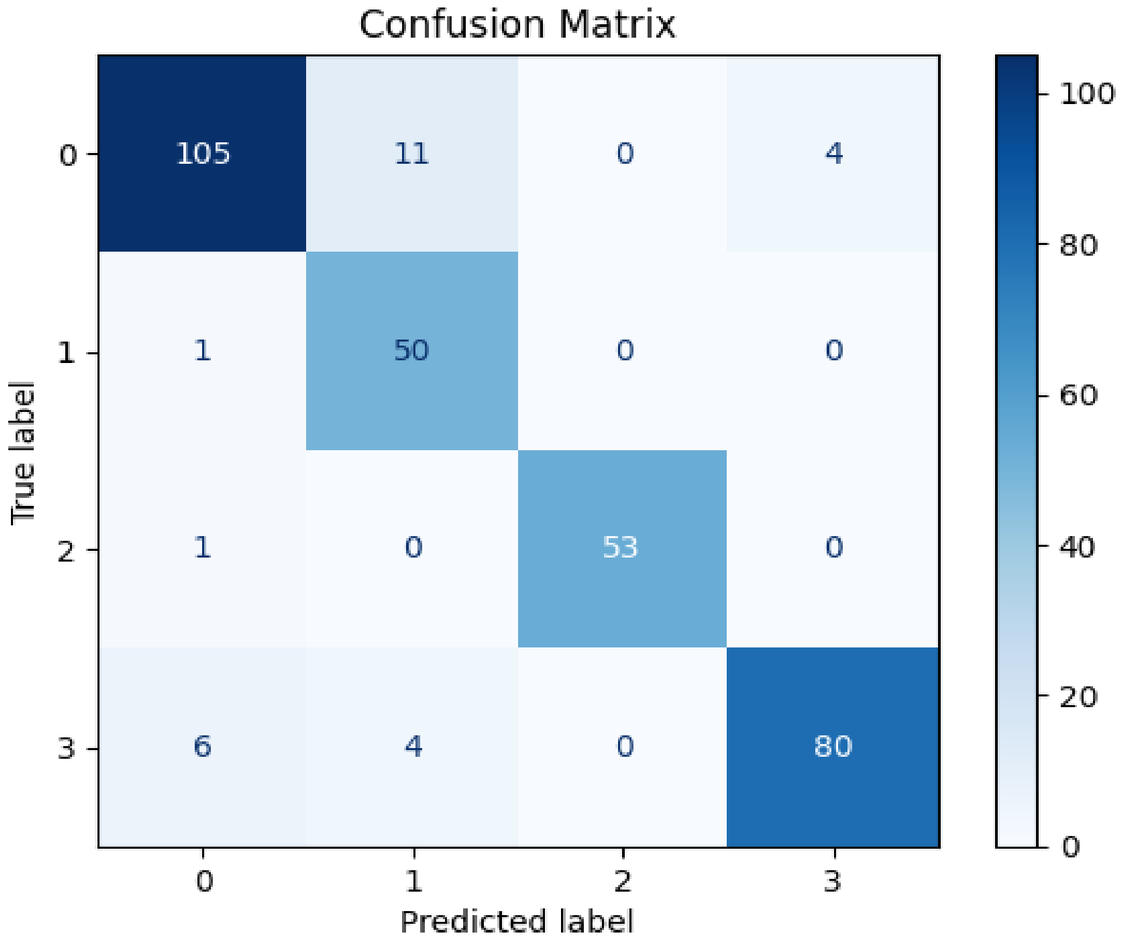
Test Accuracy: 0.914, Test Loss: 0.626

ROC curve



AUC값은 1이 최대이며, 클수록 모델의 성능이 좋다고 평가한다. 모든 클래스에서 AUC 값이 0.9 이상으로 기록되기에 모델의 예측 성능은 우수하다고 판단할 수 있다.

confusion matrix



ACC(class 0)
Precision: 0.943
Recall: 0.833
F1 Score: 0.884

normal(class 1)
Precision: 0.754
Recall: 0.961
F1 Score: 0.846

LCC(class 2)
Precision: 1.000
Recall: 0.981
F1 Score: 0.990

SCC(class 3)
Precision: 0.900
Recall: 0.900
F1 Score: 0.900

precision: 양성 중 진양성의 확률
recall: 예측을 맞춘 사람들 중 진양성인 확률
f1 score정밀도와 재현율을 모두 고려한 점수

GradCAM - 설명가능한 인공지능

데이터가 의료 분야라는 점을 고려하여 XAI 모델 구축을 위해 GradCAM으로 어떤 부분에서 폐암을 예측하였는지 설명가능할 수 있도록 모델을 보강함.