Machine Learning

CONVOLUTION NEURAL NETWORK USING TENSORFLOW

담당교수 : 박철수,이혁준

제출일 : 2022. 06. 11.

학과 : 컴퓨터정보공학부

학번 : 2017202088

이름 : 신해담

1. 데이터 설명 및 목적

- Kaggle에서 제공하는 open dataset을 사용한다. Multi-class Weather Dataset 폴더 안에 Cloudy, Rain, Shine, Sunrise의 날씨 class에 해당하는 폴더가 존재하며, 각각의 폴더 내부에 image 파일이 존재한다. 이 이미지 데이터를 사용해서 CNN model을 학습하여 최대한 정확도가 높은 classifier를 구현한다.

2. 네트워크 구조 설명

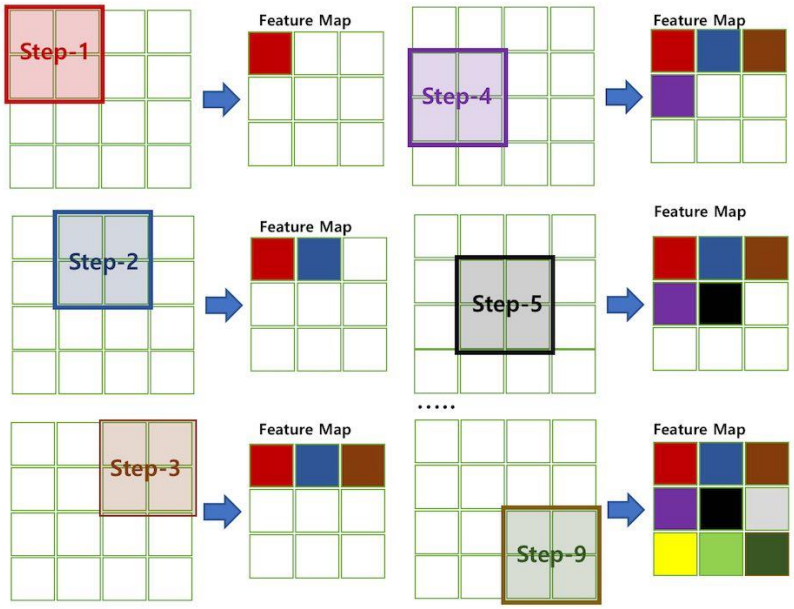
- **Summary**

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 249, 249, 64)      1792

max_pooling2d (MaxPooling2D  (None, 83, 83, 64)        0
)

conv2d_1 (Conv2D)            (None, 42, 42, 64)        36928

conv2d_2 (Conv2D)            (None, 42, 42, 64)        36928

max_pooling2d_1 (MaxPooling  (None, 21, 21, 64)        0
2D)

conv2d_3 (Conv2D)            (None, 11, 11, 128)       73856

conv2d_4 (Conv2D)            (None, 10, 10, 128)       65664

max_pooling2d_2 (MaxPooling  (None, 5, 5, 128)         0
2D)

conv2d_5 (Conv2D)            (None, 3, 3, 256)         295168

max_pooling2d_3 (MaxPooling  (None, 1, 1, 256)         0
2D)

flatten (Flatten)            (None, 256)               0

dropout (Dropout)            (None, 256)               0

dense (Dense)                (None, 2048)              526336

dense_1 (Dense)              (None, 4)                 8196

=================================================================
Total params: 1,044,868
Trainable params: 1,044,868
Non-trainable params: 0
```

- **입력 이미지**

  251x251 크기로 resize된 이미지 데이터를 사용한다.

- **Convolution Layer**



Convolution을 수행하는 layer이다. 이미지 파일을 전체가 아니라 일부분마다 filter와 dot product 연산을 하고 activation function을 적용해서 하나의 값을 얻는다. 하나의 filter로부터 생성된 결과를 activation map이라 한다.

1) Filter

데이터의 feature를 찾아내기 위한 공용 파라미터이다. 일반적으로 square matrix로 정의되며, 데이터를 순회하면서 dot product 연산을 하고 activation map을 생산한다.

2) Stride

filter가 데이터를 순회하는 간격을 stride라고 한다.

3) Padding

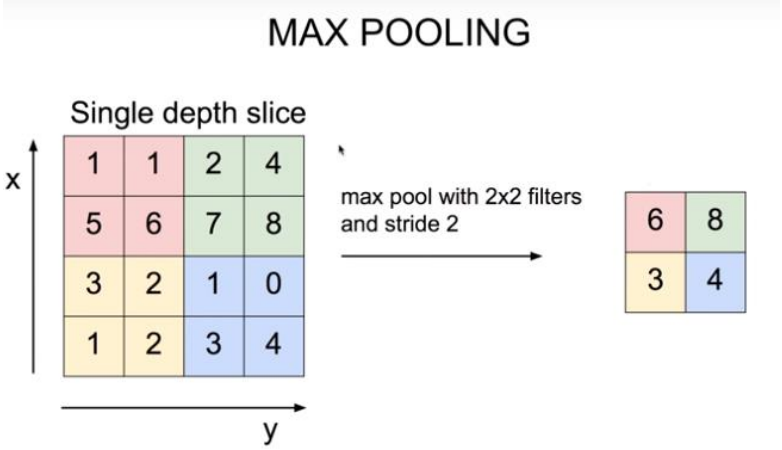
filter와 stride로 convolution하면 feature map의 크기는 입력 데이터보다 작아진다. 이를 방지하기 위해, 입력 데이터의 외각에 특정 데이터를 채워 넣는 방법이다. 보통 0을 넣는 zero padding을 사용한다.

4) Feature map size

출력 데이터의 width, height는 입력 데이터의 크기와 filter, stride, padding에 영향을 받고, depth는 filter 수와 같다.

$$k = \frac{N - F + 2P}{S} + 1$$
$$d = \#\ of\ filters$$

- **Pool Layer**



일정 영역의 pixel에서 하나의 값을 뽑아내는 layer이다. 위의 그림처럼 4x4 데이터를 2x2 filter, stride 2로 max pooling한다면 데이터의 각 2x2 영역마다 최대값을 선택해서 4x4의 pooling 결과를 얻을 수 있다.

Convolution layer를 통과한 데이터는 stride나 pooling을 해주지 않으면 filter 수만큼 feature가 많아져 overfitting될 수 있어 stride, pooling을 해서 feature를 조절해 준다.

- **Flatten Layer**

Feature extraction을 거치면 주요 특징만 추출되고, 추출된 주요 특징은 fully connected layer에 전달되어 학습된다. 이 때 fully connected layer에 전달하기 위해 1차원 자료로 바꿔주는 데 사용된다.

- **Dropout**

overfitting을 방지하기 위해 무작위 특정 노드를 0으로 만드는 정규화 기술이다.

- **Regularization**

model을 training하다 보면 overfitting이 발생해서 학습율이 떨어진다. Overfitting을 막는 가장 좋은 방법은 train data의 양을 늘리는 것이지만, train data을 추가로 확보하는 것이 어렵거나 불가능한 경우가 많다. 이 때 regularization을 사용한다.

통상적으로 cost function이나 error function이 작아지는 쪽으로 학습을 진행하는데, 계속 작아지는 쪽으로만 진행하다 보면 특정 가중치 값이 커져서 오히려 결과가 나빠지는 경우가 있다.

1) L2 Regularization

$$\tilde{J} = \frac{\alpha}{2}\omega^T\omega + J$$

위 수식에서 J는 cost function이다. α는 regularization 변수, ω는 가중치를 의미한다. L2 regularization을 사용하면 학습의 방향이 단순히 J가 작아지는 방향으로만 진행되지 않고, ω도 최소가 되는 방향으로 진행한다.

$\tilde{J}$를 가중치 ω로 편미분하면 아래와 같은 식이 나온다.

$$\omega \leftarrow (1 - \varepsilon\alpha)\omega - \varepsilon\nabla_\omega J$$

$(1-\varepsilon\alpha)\omega$는 값이 작아지는 방향으로 진행함을 알 수 있다.
이를 weight decay라고 하며, 특정 가중치가 비이상적으로 커져 학습에 큰 영향을 끼치는 것을 방지할 수 있다.

2) L1 Regularization

L1은 L2 regularization에서 2차항 대신 1차항이 들어간다.
L1은 통상적으로 상수 값을 빼도록 되어있어 작은 가중치가 0으로 수렴하며, 몇 개의 중요 가중치만 남는다. 따라서 몇 개의 의미 있는 값을 도출하려면 L1이 효과적이지만, 미분이 불가능한 점이 존재해서 gradient-based learning에 적용할 때엔 주의할 필요가 있다.

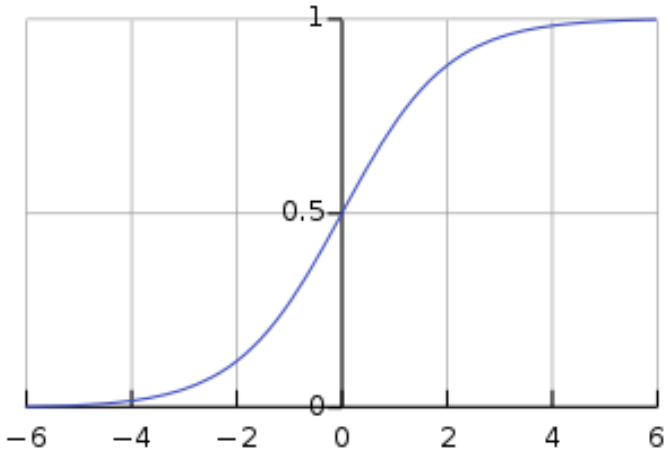
- **Fully Connected Layer(Dense Layer)**

CNN 과정의 마지막에서 classification을 결정하는 단계이다. Flatten layer에서 1차원 vector로 변환된 layer를 하나의 벡터로 연결해서 softmax로 분류한다.

- **Activation Function**

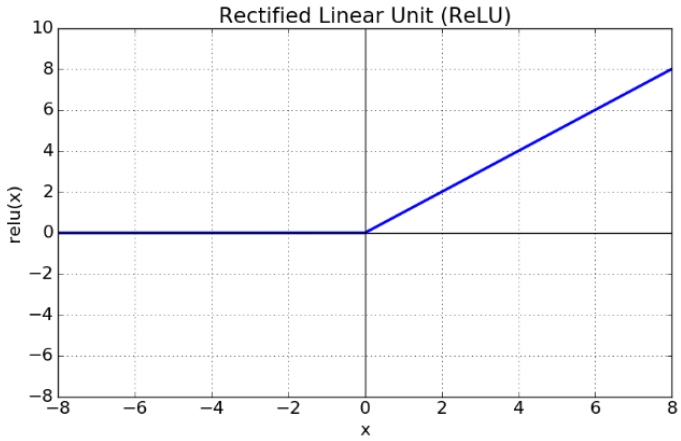
각 노드마다 값이 존재하고 뉴런마다 weight와 baias이 존재하여 다른 계층으로 넘어갈 때 각 노드에 weight를 곱하고 baias을 더한 값이 넘어가게 된다. 이 때 activation function이 특정 조건을 만족하면 활성화되었다는 신호를 다음 노드로 보내어 활성화하고, 조건을 만족하지 못했으면 해당 노드를 비활성화하는 함수이다.
수많은 activation function이 있어 상황에 따라 적절히 사용할 필요가 있다.

1) Sigmoid



음수면 0, 양수면 1을 출력하는 S형 함수이다. Sigmoid 함수는 음수 값을 0에 가깝게 표현하므로 입력 값이 최종 layer에서 미치는 영향이 적어지는 vanishing gradient problem이 발생한다. 또한 미분 값이 최대 0.25이고 양극에서 미분 값이 빠르게 0에 근사하므로, back-propagation 계산 과정에서 오차율 계산이 어려워 미분 값이 소실될 수 있다는 단점이 있다.

2) ReLu



음수면 0으로 비활성화, 양수면 해당 값을 출력한다. Sigmoid 함수에 비해 소실되는 기울기가 적고, 다른 비슷한 함수에 비해 많고 복잡한 데이터셋에서 더 빠르고 효과적인 훈련이 가능하다.

3) Softmax

input값을 [0,1] 사이의 값으로 정규화하여 출력하며, 출력값의 총합이 항상 1이 되는 특성을 가진 함수이다. 따라서 multi-class classification에서 softmax 함수를 자주 사용한다.

3. 코드 설명

- **상수 및 데이터 경로 선언**

```python
# 상수
BATCH_SIZE = 64
IMG_HEIGHT = 251
IMG_WIDTH = 251
NUM_CLASSES = 4

# 데이터셋 경로
data_dir = 'Multi-class Weather Dataset'
```

입력 데이터 경로와 크기, 클래스 수, 배치 크기를 선언한다.

- **이미지 데이터의 path, label로 구성된 dataframe 생성**

```python
# path, label list 초기화
train_filepaths = []
train_labels = []
test_filepaths = []
test_labels = []

classlist= os.listdir(data_dir) # data directory 안의 class directory list
for klass in classlist: # 각 class 폴더에 대해
    # class별로 list 초기화
    X = []
    y = []

    classpath=os.path.join(data_dir, klass) # class dir 경로
    file_list=os.listdir(classpath) # class dir에 들어있는 file list

    for f in file_list: # 각 file에 대해
        fpath=os.path.join(classpath, f)
        X.append(fpath) # filepath 추가
        y.append(klass) # label 추가(class 폴더명)

    # 8:2로 train 데이터와 test 데이터 분할
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=True)

    # train path, label과 test path, label
    train_filepaths.extend(X_train)
    train_labels.extend(y_train)
    test_filepaths.extend(X_test)
    test_labels.extend(y_test)

# train path, label로 dataframe 생성
Fseries=pd.Series(train_filepaths, name='filename')
Lseries=pd.Series(train_labels, name='class')
train_df=pd.concat([Fseries, Lseries], axis=1)
train_df = train_df.sample(frac=1).reset_index(drop=True)

# test path, label로 dataframe 생성
Fseries=pd.Series(test_filepaths, name='filename')
Lseries=pd.Series(test_labels, name='class')
test_df=pd.concat([Fseries, Lseries], axis=1)
test_df = test_df.sample(frac=1).reset_index(drop=True)

print(train_df.head(), len(train_df))
print(test_df.head(), len(test_df))
```

class별로 데이터를 8:2 비율로 분할해서 train, test의 dataframe을 생성한다.

```
                                          filename    class
0    Multi-class Weather Dataset\Cloudy\cloudy85.jpg   Cloudy
1       Multi-class Weather Dataset\Rain\rain126.jpg     Rain
2  Multi-class Weather Dataset\Sunrise\sunrise82.jpg  Sunrise
3  Multi-class Weather Dataset\Sunrise\sunrise83.jpg  Sunrise
4     Multi-class Weather Dataset\Shine\shine135.jpg    Shine 899
                                          filename    class
0     Multi-class Weather Dataset\Shine\shine151.jpg    Shine
1    Multi-class Weather Dataset\Cloudy\cloudy74.jpg   Cloudy
2  Multi-class Weather Dataset\Sunrise\sunrise85.jpg  Sunrise
3       Multi-class Weather Dataset\Rain\rain145.jpg     Rain
4   Multi-class Weather Dataset\Cloudy\cloudy284.jpg   Cloudy 226
```

train dataframe과 test dataframe이 899:226, 약 8:2로 나누어졌다.

- **이미지 데이터셋 생성**

```python
# 이미지 데이터 설정
train_datagen = ImageDataGenerator(
    rescale=1/255, # 0~255의 rgb를 0~1 사이의 값으로 일반화
    rotation_range=40, # 무작위 회전각도 범위
    #shear_range=0.2, # 층밀리기 강도
    zoom_range=0.2, # 무작위 줌 범위
    horizontal_flip=True, # input을 무작위로 가로로 뒤집음
    fill_mode='nearest', # 경계 바깥공간 채우기 모드
    validation_split=0.2) # 8:2로 training, validation 데이터 분할
test_datagen = ImageDataGenerator(rescale=1/255) # 테스트 데이터는 기본 설정

train_ds = train_datagen.flow_from_dataframe(
    train_df, # dataframe
    class_mode='categorical', # classification이 목적이므로 categorical
    subset="training", # training dataset
    target_size=(IMG_HEIGHT,IMG_WIDTH), # resize될 크기
    batch_size=BATCH_SIZE # batch 크기
)
val_ds = train_datagen.flow_from_dataframe(
    train_df,
    class_mode='categorical',
    subset="validation", # validation dataset
    target_size=(IMG_HEIGHT,IMG_WIDTH),
    batch_size=BATCH_SIZE
)
test_ds = test_datagen.flow_from_dataframe(
    test_df,
    class_mode='categorical',
    target_size=(IMG_HEIGHT,IMG_WIDTH),
    batch_size=BATCH_SIZE
)
```

train dataframe을 8:2로 분할해서 training dataset, validation dataset으로 나누어 생성한다. Test dataset은 모델 검증할 때 사용할 것이므로, 별도의 옵션 설정을 하지 않고 생성한다.

```
Found 720 validated image filenames belonging to 4 classes.
Found 179 validated image filenames belonging to 4 classes.
Found 226 validated image filenames belonging to 4 classes.
```

데이터를 training, validation, test로 나누어서 training dataset으로 training한 모델을 validation dataset으로 검증하여 모델의 overfitting을 방지하고 test dataset에 대한 정확도를 높여주는 것을 기대할 수 있다.

- **Keras**

  Keras deep learning library는 neural network 구성에 대한 보다 높은 수준의 접근 방식을 제공한다. Keras를 사용하면 복잡한 세부 사항이 아닌 전반적인 아키텍처 디자인에만 관심을 가질 수 있다.

- **Feed Forward and Back Propagation**

  다층 퍼셉트론 구조에서 입력층 > 은닉층 > 출력층으로 값의 흐름이 발생한다. 이 때 cost function의 최소값을 찾아가는 방법으로 gradient decent를 활용한다.

  Feed forward에 있어서 weight 값을 미세하게 변화시키면 cost function도 미세하게 변화된다. 그런데 매우 미세한 영역으로 국한할 때 weight의 미세변화와 이에 따른 cost function의 미세변화가 선형적인 관계를 가진다. 이로 인해 feed forward를 통해 계산된 오차의 각 weight에 따른 미세변화를 입력방향으로 back propagation하면서 weigh를 업데이트하고, 다시 입력 값을 이용해 feed forward로 새 오차를 계산하고, 다시 back propagation으로 weight를 업데이트하는 과정이 반복된다.

  Keras에서는 custom training loop를 생성하는 경우를 제외하면 고급 용도에서만 back propagation을 설정한다. Keras는 자동으로 back propagation을 수행하므로, fit method로 모델을 훈련시키는 것 외에 추가로 설정할 필요가 없다.

- **모델 정의**

```python
## 모델 정의
import tensorflow as tf
model = tf.keras.models.Sequential([
  # the input shape is the size of the image 251×251 with 3 bytes color
  # first convolution
  tf.keras.layers.Conv2D(64, (3,3), activation='relu', input_shape=(IMG_HEIGHT, IMG_WIDTH, 3)),
  tf.keras.layers.MaxPooling2D(3, 3), # first pooling
  # The second convolution
  tf.keras.layers.Conv2D(64, (3,3), padding='same', strides=2, activation='relu'),
  # The third convolution
  tf.keras.layers.Conv2D(64, (3,3), padding='same', activation='relu'),
  tf.keras.layers.MaxPooling2D(2, 2), # second pooling
  # The fourth convolution
  tf.keras.layers.Conv2D(128, (3,3), padding='same', strides=2, activation='relu'),
  # The fifth convolution
  tf.keras.layers.Conv2D(128, (2,2), activation='relu'),
  tf.keras.layers.MaxPooling2D(2, 2), # third pooling
  # The sixth convolution
  tf.keras.layers.Conv2D(256, (3,3), padding='same', strides=2, activation='relu'),
  tf.keras.layers.MaxPooling2D(3, 3), # fourth pooling
  # Flatten the results to feed into a DNN
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dropout(0.5),
  # 2048 neuron hidden layer
  tf.keras.layers.Dense(2048, activation='relu'),
  tf.keras.layers.Dense(NUM_CLASSES, activation='softmax') # 4 weather class
])
model.summary()
```

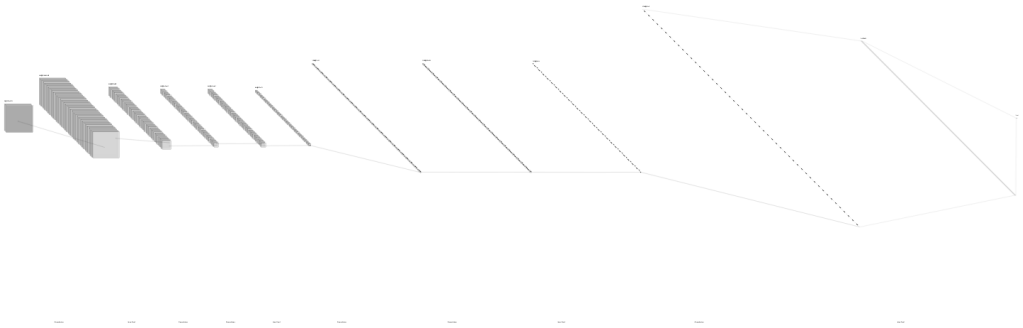
  2D convolution과 pooling, dropout, dense을 설정해서 모델을 정의한다. 활성함수는 sigmoid보다 빠르고 효과적인 ReLU를 사용했으며, 마지막 dense layer에서 softmax로 multi-class classification한다.

- **Feature Extraction**

Input Shape	Layer	Calc	Output Shape
251x251x3	64개의 3x3 filters	(251-3)+1 = 249	249x249x64
249x249x64	3x3 max pooling	249/3 = 83	83x83x64
83x83x64	64개의 3x3 filters 2 strides, 1 padding	(83+2-3)/2+1 = 42	42x42x64
42x42x64	64개의 3x3 filters 1 padding	(42+2-3)+1 = 42	42x42x64
42x42x64	2x2 max pooling	42/2 = 21	21x21x64
21x21x64	128개의 3x3 filters 2 strides, 1 padding	(21+2-3)/2+1 = 11	11x11x128
11x11x128	128개의 2x2 filters	(11-2)+1 = 10	10x10x128
10x10x128	2x2 max pooling	10/2 = 5	5x5x128
5x5x128	256개의 3x3 filters 2 strides, 1 padding	(5+2-3)/2+1 = 3	3x3x256
3x3x256	3x3 max pooling	3/3 = 1	**1x1x256**

251x251x3의 입력 이미지가 convolution과 pooling 과정을 거쳐 1x1x256의 feature maps가 된다. 이후 flatten, dropout, dense layer를 거쳐서 가장 확률이 높은 class를 output으로 분류한다.

- **NN-SVG를 사용한 네트워크 구조 시각화**



- **모델 컴파일**

```
# 모델 컴파일
model.compile(
    loss='categorical_crossentropy', # 손실함수
    optimizer='rmsprop', # 최적화함수
    metrics=['accuracy'] # 척도
)
```

모델을 컴파일한다.

1) optimization function

$$h_i \leftarrow \rho h_{i-1} + (1-\rho)\frac{\partial L_i}{\partial W} \odot \frac{\partial L_i}{\partial W}$$

RMSProp을 사용한다. 가중치 기울기를 단순 누적이 아니라 최신 기울기를 더 반영해서 극점에서의 학습과 local minimum 수렴 문제를 해결하며, hyper parameter p를 추가해서 h가 무한히 커지지 않게 한다.

2) loss function

categorical crossentropy를 사용한다. 다중 클래스 분류 문제를 해결할 때 자주 사용되며, 출력값으로 one-hot encoding된 결과를 얻을 수 있다. Softmax 활성화 함수와 함께 사용하여 각 클래스별 positive 확률값이 나오게 된다.

3) metrics

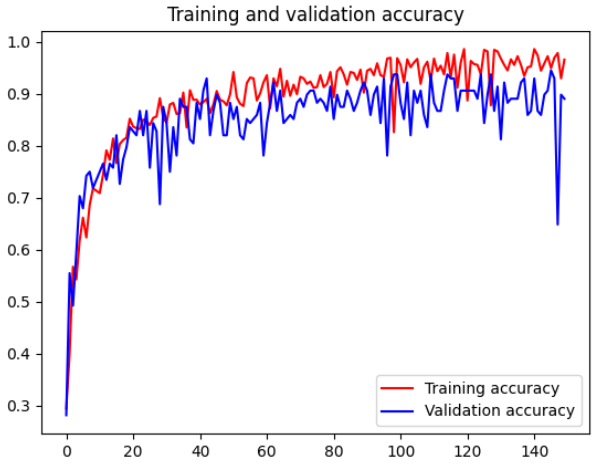
모델을 평가하기 위해 사용하는 값으로, 정확도(accuracy, TP+TN / Target)를 사용했다.

- **모델 학습**

```
# 모델 학습
history = model.fit(
    train_ds, # 학습할 training dataset
    validation_data=val_ds, # validation dataset
    batch_size=BATCH_SIZE,
    steps_per_epoch=len(train_df)*0.8//BATCH_SIZE,
    epochs=100,
    verbose=1,
    validation_steps=len(train_df)*0.2//BATCH_SIZE
)
```

train dataset, validation dataset으로 학습한다.

1) epochs



전체 dataset을 반복해서 학습할 횟수로, overfitting이나 underfitting되지 않도록 적절한 횟수가 필요하다. Epochs 150으로 학습해본 결과 plot은 위와 같으며, 약 100회를 넘기면 정확도가 증가하지 않고 학습의 의미가 없다 판단해서 100으로 결정했다.

2) batch size

한 step에서 학습할 데이터의 크기를 설정하는 파라미터이다. Batch 크기가 매우 크면 local minima에 빠질 수 있으며, 너무 작으면 global optimal value를 찾기 힘들다. 따라서 일정 mini batch 단위로 데이터를 나누어 학습하는 것이 좋다. Mini batch는 보통 32, 64를 사용하며, 여기서는 64를 사용했다.

3) steps per epoch

각 epoch별 step 수로 training dataset / batch size로 사용한다.

4) validation steps

validation dataset / batch size를 사용한다.

5) verbose

1로 설정하면 학습 진행상황을 볼 수 있다.

- **학습 결과 및 Plot**

```python
# 학습 결과 및 Plot 출력
import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend(loc=0)
plt.show()
```

matplotlib를 사용해서 학습 과정을 시각화한다.

- **정확도 평가**

```python
# test dataset으로 모델 정확도 평가
loss, accuracy = model.evaluate(
    test_ds,
    batch_size=BATCH_SIZE,
    steps=len(test_df)//BATCH_SIZE
)
print(loss, accuracy)
```

따로 구분해 놓았던 test dataset으로 학습된 모델을 평가한다.

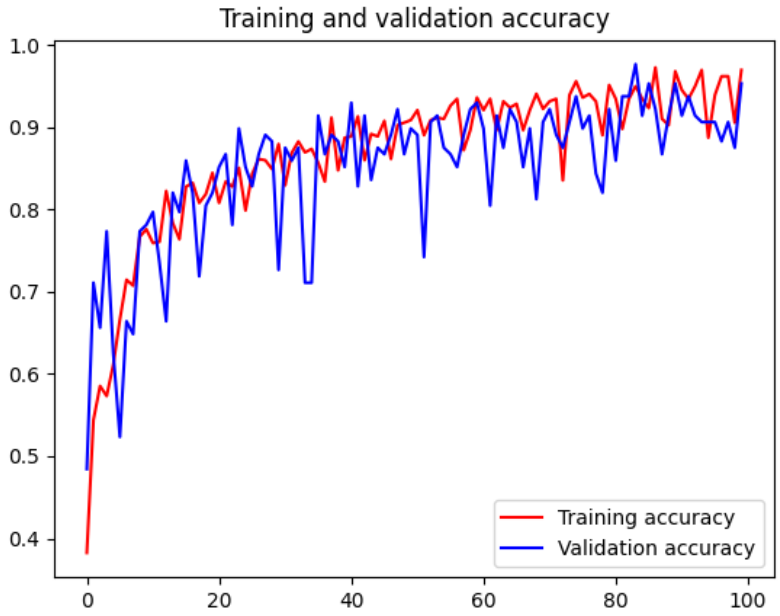
4. 실행 결과 및 Plot

- **정확도 향상 과정**

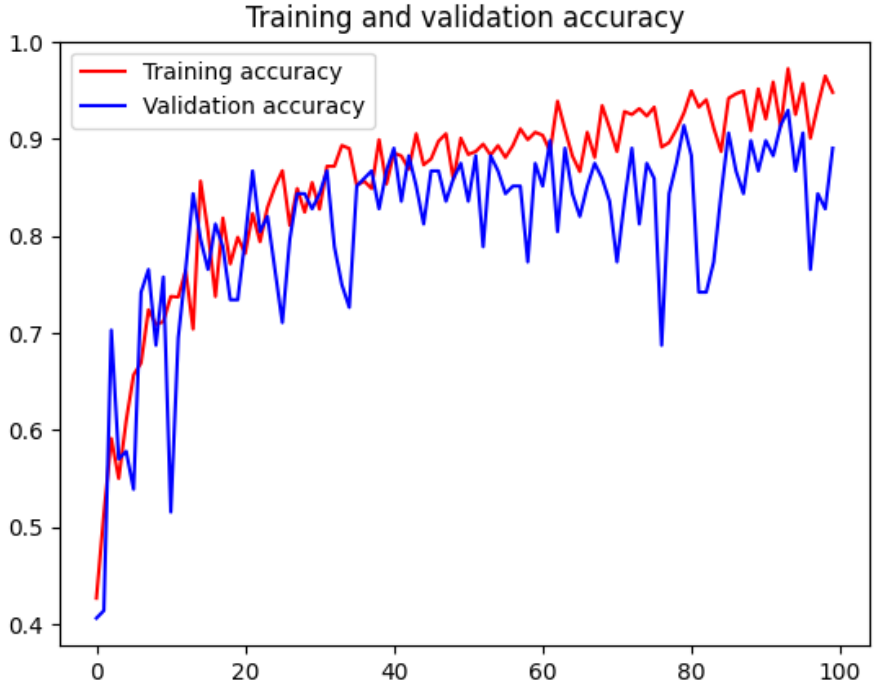
img_height	img_width	batch_size	epochs	steps_per_epoch	accuracy	val_accuracy	evlauate
150	150	40	15	15	0.83	0.8167	
150	150	40	20	15	0.8365	0.7	
300	300	40	15	15	0.8072	0.775	
300	300	40	10	20	0.799	0.7833	
300	300	40	15	20	0.8175	0.7833	
300	300	40	15	20	0.8268	0.7833	
300	300	40	25	10	0.81	0.825	
300	300	50	20	10	0.814	0.7933	
300	300	100	20	10	0.7814	0.7545	
150	150	50	20	10	0.8043	0.7933	0.8467
150	150	50	20	15	0.8264	0.7467	0.7533
150	150	50	15	15	0.7556	0.7933	0.7933
150	150	64	15	10	0.7828	0.6704	0.7396
225	225	50	15	15	0.7917	0.8067	0.86
225	225	50	15	15	0.7778	0.7667	0.7467
225	225	50	15	15	0.7875	0.8467	0.86
225	225	50	15	15			0.7333
225	225	50	50	15	0.9792	0.9067	0.94
225	225	50	100	10	0.942	0.9267	0.8933
225	225	50	70	15	0.9667	0.9133	0.8933
225	225	50	50	15	0.9681	0.8733	0.8467
225	225	50	75	15	0.975	0.7067	0.8067
225	225	50	60	15	0.9875	0.8933	0.8467
225	225	75	50	15	0.9875	0.8933	0.8467
225	225	72	50	10	0.9778	0.5642	0.5741
250	250	64	50	10	0.8615	0.8994	0.8802
250	250	64	70	10	0.9899	0.905	0.8958
250	250	64	90	10	0.9953	0.9218	0.9219
250	250	64	90	10	0.8851	0.9162	0.901
251	251	64	90	10	0.9459	0.9274	0.901

이미지 크기, batch 크기, epochs, steps_per_epoch 를 수정하면서 정확도를 측정했다. 파라미터를 결정한 다음 epochs 를 크게 잡아서 학습시켜보고, plot 을 관찰해서 epochs 횟수를 결정했다.

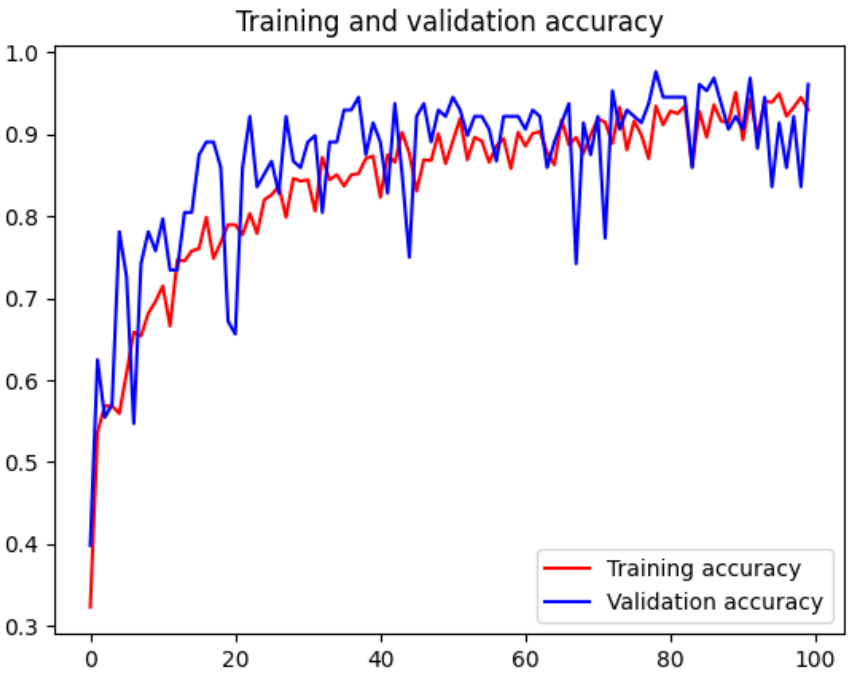
- **11회 학습한 결과 및 Plot**



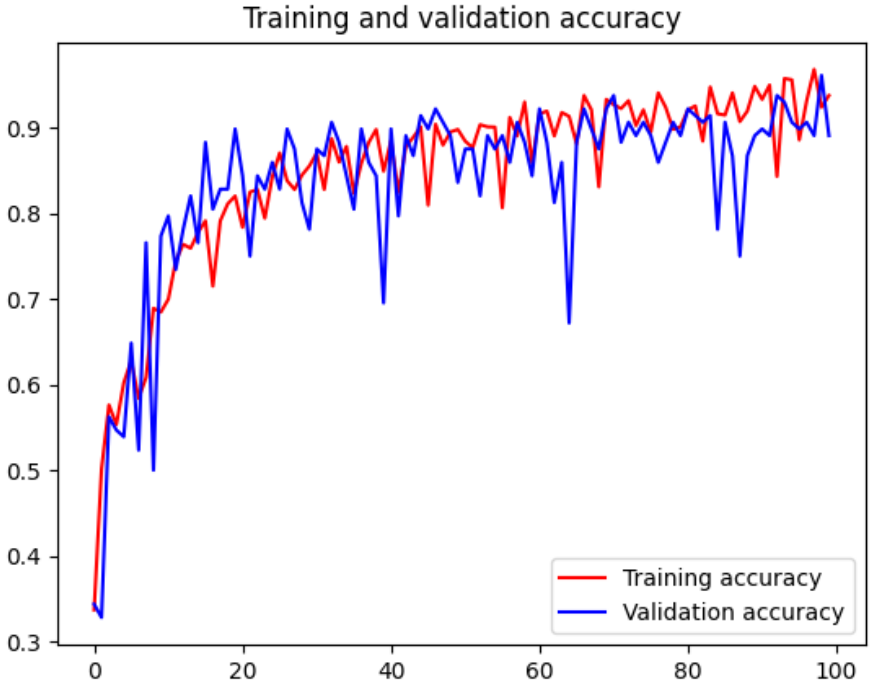

```
3/3 [==============================] - 2s 537ms/step - loss: 0.2400 - accuracy: 0.9583
0.2399529218673706 0.9583333134651184
```



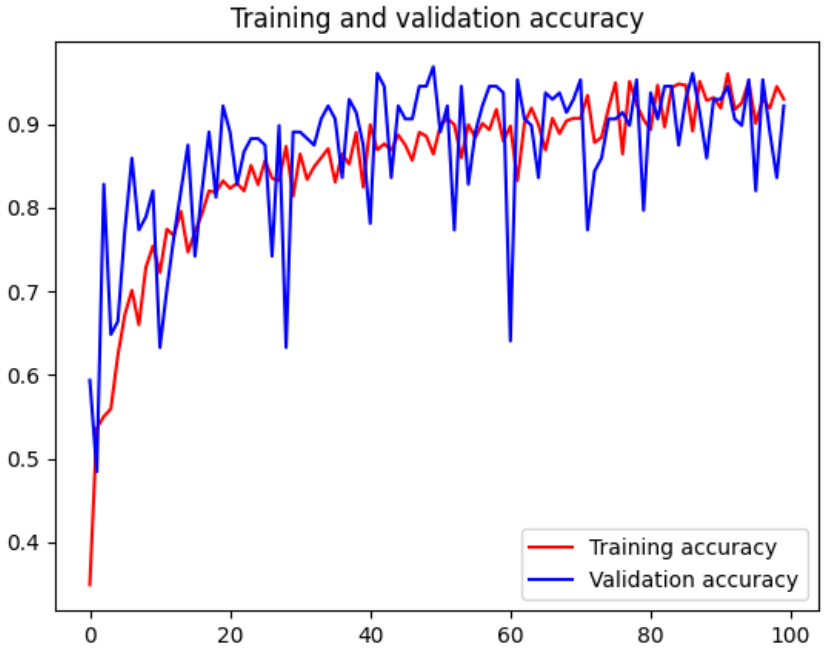
```
3/3 [==============================] - 2s 517ms/step - loss: 0.4322 - accuracy: 0.8750
0.43215346336364746 0.875
```



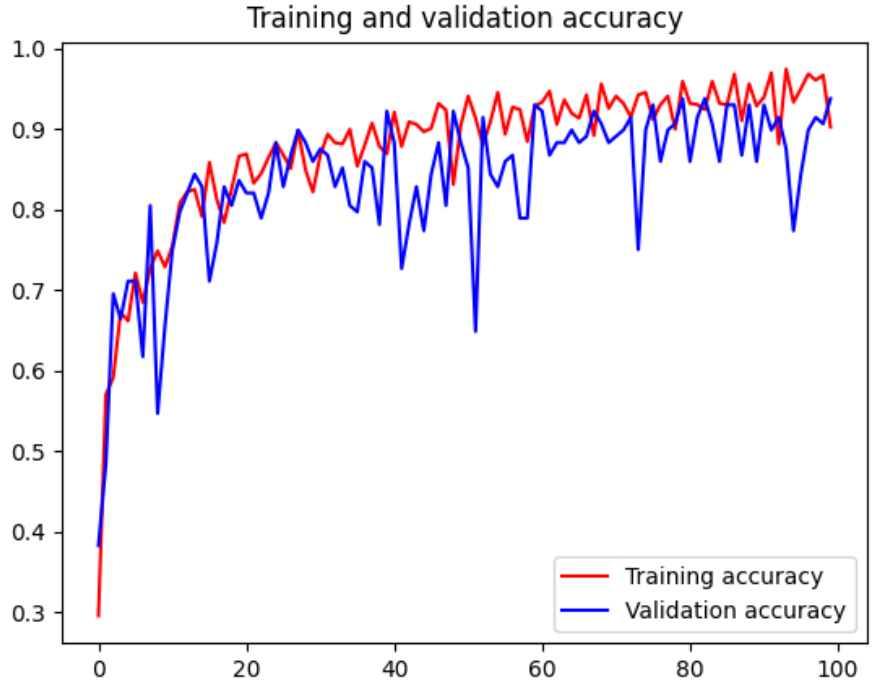

```
3/3 [==============================] - 1s 454ms/step - loss: 0.0966 - accuracy: 0.9635
0.0966324433684349 0.9635416865348816
```



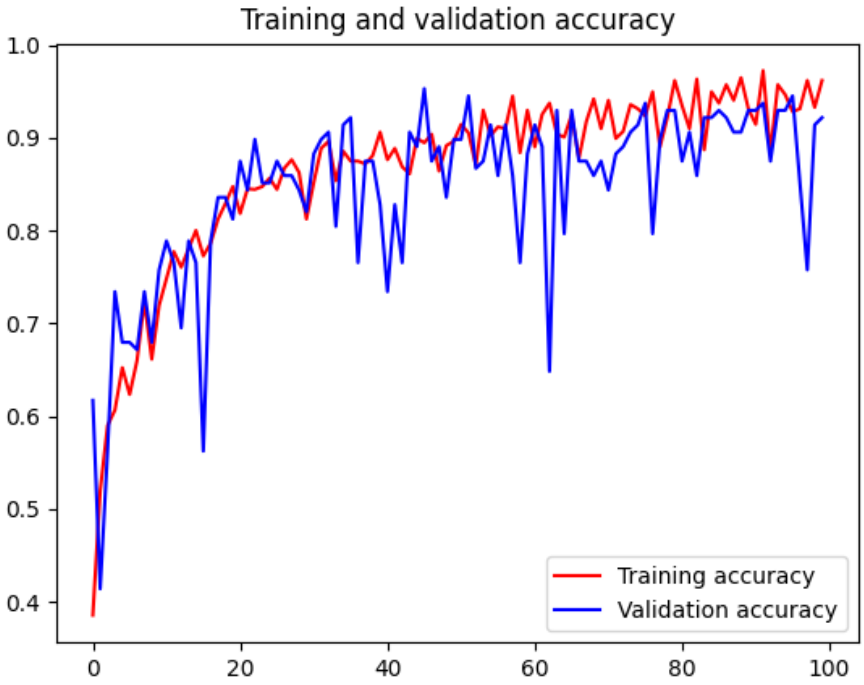

```
3/3 [==============================] - 1s 452ms/step - loss: 0.3417 - accuracy: 0.8750
0.34171780943870544 0.875
```



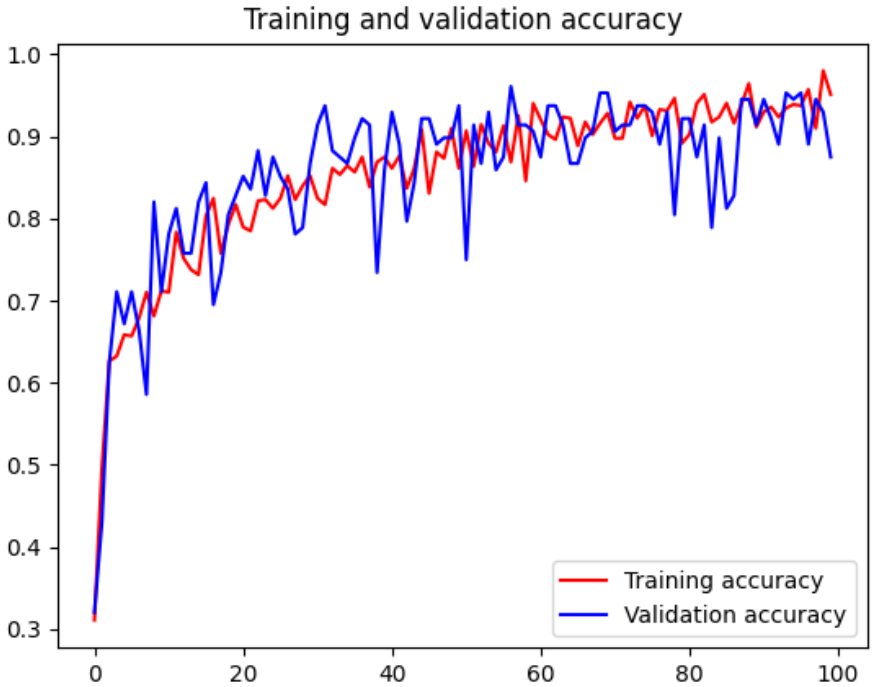

```
3/3 [==============================] - 1s 453ms/step - loss: 0.1488 - accuracy: 0.9375
0.1488213688135147 0.9375
```



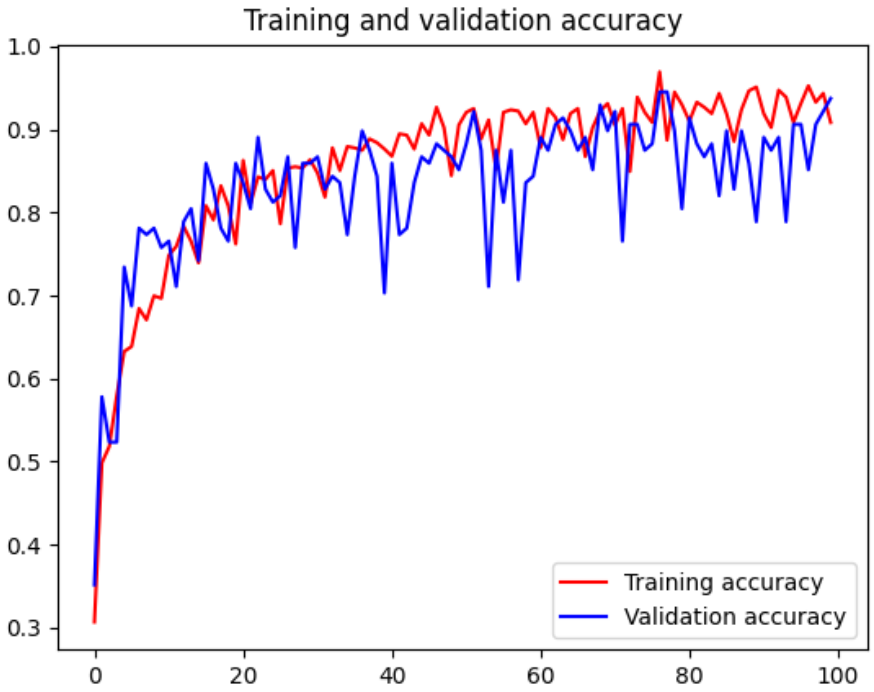

```
3/3 [==============================] - 2s 451ms/step - loss: 0.1557 - accuracy: 0.9635
0.15567879378795624 0.9635416865348816
```



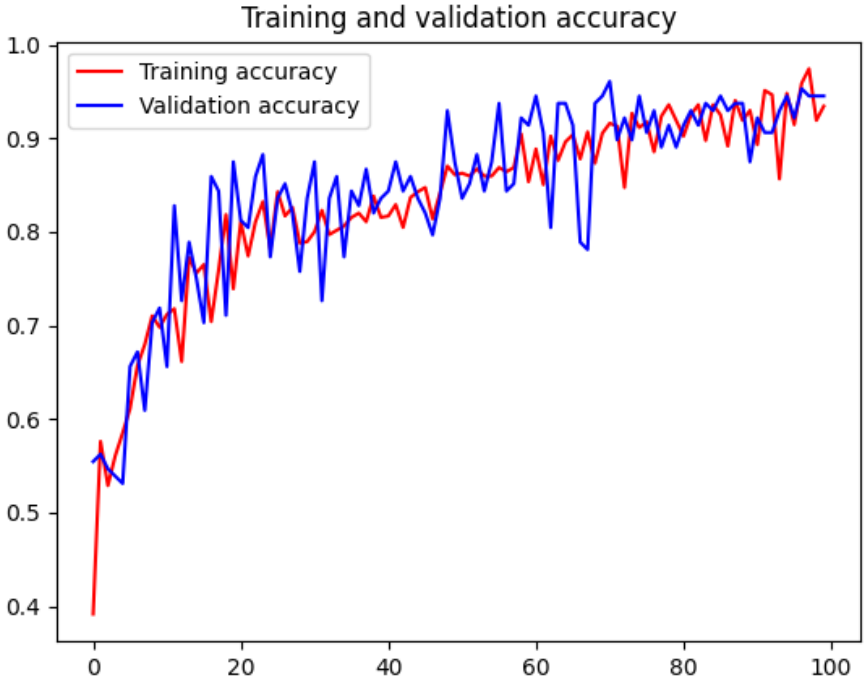

```
3/3 [==============================] - 3s 1s/step - loss: 0.4332 - accuracy: 0.8958
0.4332102835178375 0.8958333134651184
```



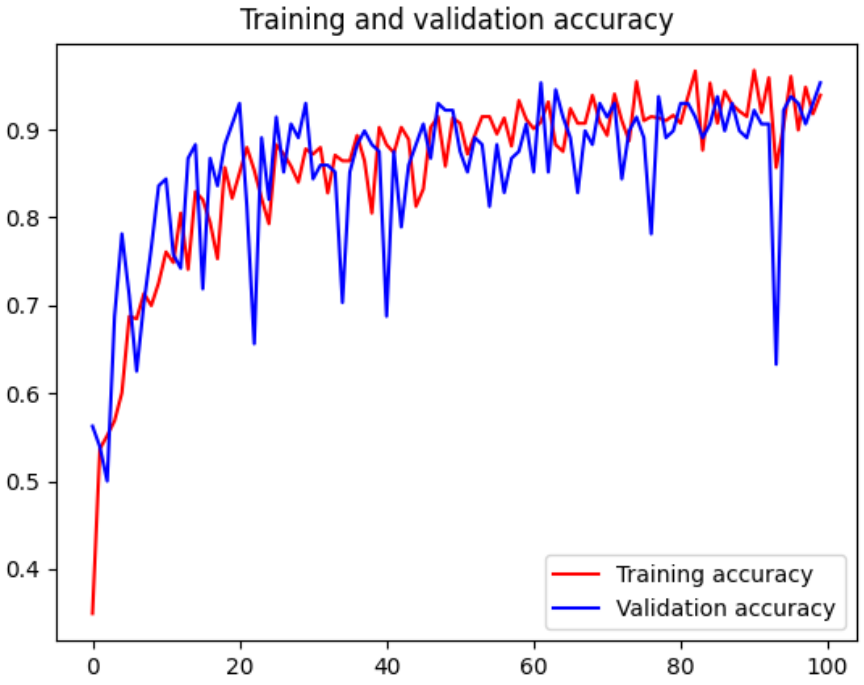

```
3/3 [==============================] - 3s 1000ms/step - loss: 0.2591 - accuracy: 0.9271
0.25913891196250916 0.9270833134651184
```




```
3/3 [==============================] - 2s 504ms/step - loss: 0.2950 - accuracy: 0.9167
0.2950291037559509 0.9166666865348816
```




```
3/3 [==============================] - 2s 469ms/step - loss: 0.3141 - accuracy: 0.9271
0.3141184151172638 0.9270833134651184
```




```
3/3 [==============================] - 2s 681ms/step - loss: 0.2106 - accuracy: 0.9323
0.21063262224197388 0.9322916865348816
```

- **Plot 분석**

  training accuracy가 증가하면 validation accuracy도 증가하는 경향을 보인다. training accuracy보다 validation accuracy에서 정확도가 종종 크게 튄다. Overfitting을 방지하도록 dropout layer, validation dataset 등의 설정해 두어서, training accuracy만 증가하고 validation accuracy가 감소하는 구간이 계속되지는 않는다.

- **11회 정확도 통계**

img_height	img_width	batch_size	epochs	steps_per_epoch	accuracy	val_accuracy	evlauate
251	251	64	100	11	0.939	0.9531	0.9323
					0.9345	0.9453	0.9271
					0.9085	0.9375	0.9167
							0.9271
					0.9619	0.9219	0.8958
					0.902	0.9375	0.9635
					0.9299	0.9219	0.9375
					0.9375	0.8906	0.875
					0.9299	0.9609	0.9635
					0.9482	0.8906	0.875
					0.9695	0.9531	0.9583
							0.924709091

편차가 있으나 85~95 사이의 정확도를 보이고 있다.

5. Reference

- Model training APIs (keras.io)
- 파이썬 딥러닝 - 07. Layer 의 종류 (tistory.com)
- Advanced Activations Layers - Keras Documentation
- Core Layers - Keras Documentation
- Sequential 모델 | TensorFlow Core
- python - What is batch size in neural network? - Cross Validated (stackexchange.com)
- 완전 연결 계층, Fully connected layer - 데이터 사이언스 사용 설명서 (tistory.com)
- CNN, Convolutional Neural Network 요약 (taewan.kim)
- [Part Ⅲ. Neural Networks 최적화] 2. Regularization
- python - Backward propagation in Keras? - Stack Overflow
- 이혁준. Lecture 10, 11 - Convolutional Neural Networks.