

강아지 vs 고양이 분류하기(1)

학습 내용

- 구글 드라이브와 연동하는 것을 실습해 봅니다.
- 개와 고양이의 이미지를 준비합니다.
- 신경망을 구축 후, 학습을 수행합니다.
- 모델 학습 후, 이를 저장하는 것을 대해 알아봅니다.

환경

- Google Colab (T4 GPU)

데이터(강아지 vs 고양이)

- url : <https://www.kaggle.com/c/dogs-vs-cats/data>
- test1.zip : 271.15MB
- train.zip : 543.16MB
- sampleSubmission.csv

목차

01. 데이터 준비하기
02. 신경망 구성하기
03. 데이터 전처리
04. 모델 학습
05. 훈련 후, 모델 저장

01. 데이터 준비하기

목차로 이동하기



- url : https://s3.amazonaws.com/book.keras.io/img/ch5/cats_vs_dogs_samples.jpg

- 25,000개의 강아지와 고양이 이미지
- 클래스마다 12,500개를 담고 있다. 압축(543MB크기)
- 클래스마다 1,000개의 샘플로 이루어진 훈련 세트
- 클래스마다 500개의 샘플로 이루어진 검증 세트
- 클래스마다 500개의 샘플로 이루어진 테스트 세트

In [1]: `import os, shutil`

구글 드라이브 연동하기

- 링크가 뜨면 해당 링크로 연결하여 연결 암호 정보를 빈칸에 넣어준다.

In [2]: `### 드라이브 마운트
from google.colab import drive
drive.mount('/content/drive')`

Mounted at /content/drive

- 구글 드라이브의 dataset/cats_dogs 의 위치에 데이터를 위치시킵니다.
- cp 명령을 이용하여 내 현재 구글 콜랩으로 복사해 오기.

In [3]: `# 데이터 셋 복사 및 확인
!cp -r '/content/drive/My Drive/dataset/cats_dogs' '/content/'
!ls -ls '/content/cats_dogs'`

```
total 833956
  88 -rw----- 1 root root    88903 Nov 14 13:49 sampleSubmission.csv
277664 -rw----- 1 root root 284321224 Nov 14 13:50 test1.zip
556204 -rw----- 1 root root 569546721 Nov 14 13:50 train.zip
```

파일 압축풀기

```
!rm -rf '/content/datasets/'
!unzip '/content/cats_dogs/test1.zip' -d '/content/datasets/' |
head -n 10
!unzip '/content/cats_dogs/train.zip' -d '/content/datasets/' |
head -n 10
```

```
In [43]: !rm -rf '/content/datasets/'
!unzip -q '/content/cats_dogs/test1.zip' -d '/content/datasets/'
!unzip -q '/content/cats_dogs/train.zip' -d '/content/datasets/'
```

파일 확인

```
In [44]: !ls -al '/content/datasets/train' | head -5
!ls -l '/content/datasets/train' | grep ^- | wc -l
!ls -al '/content/datasets/test1' | head -5
!ls -l '/content/datasets/test1' | grep ^- | wc -l
```

```
total 609284
drwxr-xr-x 2 root root 782336 Sep 20 2013 .
drwxr-xr-x 4 root root 4096 Nov 14 14:08 ..
-rw-r--r-- 1 root root 12414 Sep 20 2013 cat.0.jpg
-rw-r--r-- 1 root root 21944 Sep 20 2013 cat.10000.jpg
25000
total 304260
drwxr-xr-x 2 root root 270336 Sep 20 2013 .
drwxr-xr-x 4 root root 4096 Nov 14 14:08 ..
-rw-r--r-- 1 root root 54902 Sep 20 2013 10000.jpg
-rw-r--r-- 1 root root 21671 Sep 20 2013 10001.jpg
12500
```

- train 25000장, test 12500장

데이터 셋 경로 지정

```
In [47]: # 원본 데이터셋(학습용)을 압축 해제한 디렉터리 경로
ori_dataset_train_dir = '/content/datasets/train'
ori_dataset_test_dir = '/content/datasets/test1'

# 일부 소규모 데이터셋을 저장할 디렉터리
base_dir = '/content/datasets/cats_and_dogs_small'
```

```
In [48]: # 반복실행을 위해 디렉터리 삭제
if os.path.exists(base_dir):
    shutil.rmtree(base_dir)
os.mkdir(base_dir)
```

학습, 검증, 테스트 데이터 셋 디렉터리 준비(생성)

- base_dir = './datasets/cats_and_dogs_small'

```
In [49]: # 훈련, 검증, 테스트 분할을 위한 디렉터리
train_dir = os.path.join(base_dir, 'train')
os.mkdir(train_dir)
```

```
val_dir = os.path.join(base_dir, 'validation')
os.mkdir(val_dir)

test_dir = os.path.join(base_dir, 'test')
os.mkdir(test_dir)

print("학습용 : ", train_dir)
print("검증용 : ", train_dir)
print("테스트용 : ", train_dir)
```

학습용 : /content/datasets/cats_and_dogs_small/train
검증용 : /content/datasets/cats_and_dogs_small/train
테스트용 : /content/datasets/cats_and_dogs_small/train

```
In [50]: # 학습용 고양이 사진 디렉터리
train_cats_dir = os.path.join(train_dir, 'cats')
os.mkdir(train_cats_dir)

# 학습용 강아지 사진 디렉터리
train_dogs_dir = os.path.join(train_dir, 'dogs')
os.mkdir(train_dogs_dir)

print("학습용(고양이) : ", train_cats_dir)
print("학습용(강아지) : ", train_dogs_dir)
```

학습용(고양이) : /content/datasets/cats_and_dogs_small/train/cats
학습용(강아지) : /content/datasets/cats_and_dogs_small/train/dogs

```
In [51]: # 검증용 고양이 사진 디렉터리
val_cats_dir = os.path.join(val_dir, 'cats')
os.mkdir(val_cats_dir)

# 검증용 강아지 사진 디렉터리
val_dogs_dir = os.path.join(val_dir, 'dogs')
os.mkdir(val_dogs_dir)

print("검증용(고양이) : ", val_cats_dir)
print("검증용(강아지) : ", val_dogs_dir)
```

검증용(고양이) : /content/datasets/cats_and_dogs_small/validation/cats
검증용(강아지) : /content/datasets/cats_and_dogs_small/validation/dogs

```
In [52]: # 테스트용 고양이 사진 디렉터리
test_cats_dir = os.path.join(test_dir, 'cats')
os.mkdir(test_cats_dir)

# 테스트용 강아지 사진 디렉터리
test_dogs_dir = os.path.join(test_dir, 'dogs')
os.mkdir(test_dogs_dir)

print("테스트용(고양이) : ", test_cats_dir)
print("테스트용(강아지) : ", test_dogs_dir)
```

테스트용(고양이) : /content/datasets/cats_and_dogs_small/test/cats
테스트용(강아지) : /content/datasets/cats_and_dogs_small/test/dogs

```
In [53]: print(train_cats_dir, train_dogs_dir)
print(val_cats_dir, val_dogs_dir)
print(test_cats_dir, test_dogs_dir)
```

```
/content/datasets/cats_and_dogs_small/train/cats /content/datasets/cats_and_dogs_small/train/dogs
/content/datasets/cats_and_dogs_small/validation/cats /content/datasets/cats_and_dogs_small/validation/dogs
/content/datasets/cats_and_dogs_small/test/cats /content/datasets/cats_and_dogs_small/test/dogs
```

```
In [54]: !ls -al './datasets/cats_and_dogs_small/train' | head -5
!ls -al './datasets/cats_and_dogs_small/validation/' | head -5
!ls -al './datasets/cats_and_dogs_small/test/' | head -5
```

```
total 16
drwxr-xr-x 4 root root 4096 Nov 14 14:09 .
drwxr-xr-x 5 root root 4096 Nov 14 14:09 ..
drwxr-xr-x 2 root root 4096 Nov 14 14:09 cats
drwxr-xr-x 2 root root 4096 Nov 14 14:09 dogs
total 16
drwxr-xr-x 4 root root 4096 Nov 14 14:09 .
drwxr-xr-x 5 root root 4096 Nov 14 14:09 ..
drwxr-xr-x 2 root root 4096 Nov 14 14:09 cats
drwxr-xr-x 2 root root 4096 Nov 14 14:09 dogs
total 16
drwxr-xr-x 4 root root 4096 Nov 14 14:09 .
drwxr-xr-x 5 root root 4096 Nov 14 14:09 ..
drwxr-xr-x 2 root root 4096 Nov 14 14:09 cats
drwxr-xr-x 2 root root 4096 Nov 14 14:09 dogs
```

데이터 준비

- 개와 고양이의 각각의 이미지 준비
 - 학습용 이미지 1000장
 - 검증용 이미지 500장
 - 테스트용 이미지 500장

```
In [55]: # 처음 1,000개의 고양이 이미지를 train_cats_dir에 복사합니다
fnames = ['cat.{}.jpg'.format(i) for i in range(1000)]
for fname in fnames:
    src = os.path.join(ori_dataset_train_dir, fname)
    dst = os.path.join(train_cats_dir, fname)
    shutil.copyfile(src, dst)

# 다음 500개 고양이 이미지를 validation_cats_dir에 복사합니다
fnames = ['cat.{}.jpg'.format(i) for i in range(1000, 1500)]
for fname in fnames:
    src = os.path.join(ori_dataset_train_dir, fname)
    dst = os.path.join(val_cats_dir, fname)
    shutil.copyfile(src, dst)

# 다음 500개 고양이 이미지를 test_cats_dir에 복사합니다
fnames = ['cat.{}.jpg'.format(i) for i in range(1500, 2000)]
for fname in fnames:
    src = os.path.join(ori_dataset_train_dir, fname)
    dst = os.path.join(test_cats_dir, fname)
    shutil.copyfile(src, dst)

# 처음 1,000개의 강아지 이미지를 train_dogs_dir에 복사합니다
fnames = ['dog.{}.jpg'.format(i) for i in range(1000)]
for fname in fnames:
```

```

src = os.path.join(ori_dataset_train_dir, fname)
dst = os.path.join(train_dogs_dir, fname)
shutil.copyfile(src, dst)

# 다음 500개 강아지 이미지를 validation_dogs_dir에 복사합니다
fnames = ['dog.{}.jpg'.format(i) for i in range(1000, 1500)]
for fname in fnames:
    src = os.path.join(ori_dataset_train_dir, fname)
    dst = os.path.join(val_dogs_dir, fname)
    shutil.copyfile(src, dst)

# 다음 500개 강아지 이미지를 test_dogs_dir에 복사합니다
fnames = ['dog.{}.jpg'.format(i) for i in range(1500, 2000)]
for fname in fnames:
    src = os.path.join(ori_dataset_train_dir, fname)
    dst = os.path.join(test_dogs_dir, fname)
    shutil.copyfile(src, dst)

```

[훈련/검증/테스트]에 들어있는 사진의 개수를 카운트

```

In [59]: print('학습용 고양이 이미지 전체 개수:', len(os.listdir(train_cats_dir)))
         print('학습용 강아지 이미지 전체 개수:', len(os.listdir(train_dogs_dir)))

```

학습용 고양이 이미지 전체 개수: 1000
 학습용 강아지 이미지 전체 개수: 1000

```

In [60]: print('검증용 고양이 이미지 전체 개수:', len(os.listdir(val_cats_dir)))
         print('검증용 강아지 이미지 전체 개수:', len(os.listdir(val_dogs_dir)))

```

검증용 고양이 이미지 전체 개수: 500
 검증용 강아지 이미지 전체 개수: 500

```

In [61]: print('테스트용 고양이 이미지 전체 개수:', len(os.listdir(test_cats_dir)))
         print('테스트용 강아지 이미지 전체 개수:', len(os.listdir(test_dogs_dir)))

```

테스트용 고양이 이미지 전체 개수: 500
 테스트용 강아지 이미지 전체 개수: 500

- 훈련 이미지 : 2000개
- 검증 이미지 : 1000개
- 테스트 이미지 : 1000개

02. 신경망 구성하기

목차로 이동하기

- Conv2D (3x3) filter:32 - Maxpooling2D (2x2) stride 1
- Conv2D (3x3) filter:64 - Maxpooling2D (2x2) stride 1
- Conv2D (3x3) filter:128 - Maxpooling2D (2x2) stride 1
- Conv2D (3x3) filter:128 - Maxpooling2D (2x2) stride 1

```

In [62]: from tensorflow.keras import layers
         from tensorflow.keras import models

```

```

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))

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

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

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

model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

```

/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

```
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [63]: model.summary()

Model: "sequential"

Layer (type)	Output Shape	
conv2d (Conv2D)	(None, 148, 148, 32)	
max_pooling2d (MaxPooling2D)	(None, 74, 74, 32)	
conv2d_1 (Conv2D)	(None, 72, 72, 64)	
max_pooling2d_1 (MaxPooling2D)	(None, 36, 36, 64)	
conv2d_2 (Conv2D)	(None, 34, 34, 128)	
max_pooling2d_2 (MaxPooling2D)	(None, 17, 17, 128)	
conv2d_3 (Conv2D)	(None, 15, 15, 128)	
max_pooling2d_3 (MaxPooling2D)	(None, 7, 7, 128)	
flatten (Flatten)	(None, 6272)	
dense (Dense)	(None, 512)	
dense_1 (Dense)	(None, 1)	

Total params: 3,453,121 (13.17 MB)

Trainable params: 3,453,121 (13.17 MB)

Non-trainable params: 0 (0.00 B)

- 150 x 150 크기에서 7 x 7 크기의 특성 맵으로 줄어든다.

최적화 알고리즘, 손실 함수 선택

```
In [85]: from tensorflow.keras import optimizers

model.compile(loss='binary_crossentropy',
               optimizer=optimizers.RMSprop(learning_rate=1e-4),
               metrics=['accuracy'])
```

03. 데이터 전처리

목차로 이동하기

- 사진 파일을 읽기
- JPEG 콘텐츠를 RGB픽셀로 디코딩
- 부동 소수 타입의 텐서로 변환
- 픽셀 값(0~255)의 스케일을 [0,1]사이로 조정

```
In [81]: import tensorflow as tf

# 이미지 데이터셋 생성
train_dataset = tf.keras.utils.image_dataset_from_directory(
    train_dir,
    validation_split=0.2, # 검증 데이터 분할 비율
    subset="training",    # 훈련 데이터셋 선택
    seed=42,              # 일관된 분할을 위한 시드
    image_size=(150, 150), # 이미지 크기 조정
    batch_size=20,        # 배치 크기
    label_mode='binary'   # 이진 분류 레이블
)

val_dataset = tf.keras.utils.image_dataset_from_directory(
    train_dir,
    validation_split=0.2, # 검증 데이터 분할 비율
    subset="validation",  # 검증 데이터셋 선택
    seed=42,              # 일관된 분할을 위한 시드
    image_size=(150, 150), # 이미지 크기 조정
    batch_size=20,        # 배치 크기
    label_mode='binary'   # 이진 분류 레이블
)

# 데이터 정규화 레이어 추가
normalization_layer = tf.keras.layers.Rescaling(1./255)

# 데이터셋에 정규화 적용
train_dataset = train_dataset.map(lambda x, y: (normalization_layer(x), y))
val_dataset = val_dataset.map(lambda x, y: (normalization_layer(x), y))

# 데이터 증강 (선택적)
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.RandomRotation(0.2),
    tf.keras.layers.RandomZoom(0.2),
    tf.keras.layers.RandomFlip('horizontal')
])
```



```
# 데이터 증강을 데이터셋에 적용
train_dataset = train_dataset.map(lambda x, y: (data_augmentation(x, training=Tr
```

Found 2000 files belonging to 2 classes.
Using 1600 files for training.
Found 2000 files belonging to 2 classes.
Using 400 files for validation.

이미지 제너레이터를 활용하여 데이터와 라벨을 확인해보기

```
In [82]: # 클래스 이름 확인
class_names = train_dataset.class_names
print("클래스 이름:", class_names)

# 배치의 상세 정보
for data_batch, labels_batch in train_dataset:
    print('배치 데이터 형태:', data_batch.shape)
    print('배치 레이블 형태:', labels_batch.shape)

    # 각 배치의 첫 번째 이미지와 레이블 출력
    print('\n첫 번째 이미지 레이블:', labels_batch[0].numpy())

    # 이미지 시각화
    plt.figure(figsize=(10, 5))
    plt.imshow(data_batch[0].numpy())
    plt.title(f'Label: {labels_batch[0].numpy()} ({class_names[int(labels_batch[0].numpy())]})')
    plt.axis('off')
    plt.show()

    break
```

배치 데이터 크기: (20, 150, 150, 3)
배치 레이블 크기: (20, 1)

```
In [83]: labels_batch[0:3], data_batch[0]
```

```

Out[83]: (<tf.Tensor: shape=(3, 1), dtype=float32, numpy=
array([[0.],
       [1.],
       [1.]], dtype=float32)>,
<tf.Tensor: shape=(150, 150, 3), dtype=float32, numpy=
array([[[[0.09461719, 0.06148966, 0.07029568],
         [0.097324 , 0.05980401, 0.06671599],
         [0.09747081, 0.05755104, 0.0625195 ],
         ...,
         [0.5702627 , 0.5467332 , 0.6016352 ],
         [0.55451834, 0.53098893, 0.5858909 ],
         [0.5682866 , 0.5447572 , 0.59965914]],

        [[0.14719865, 0.09980871, 0.09984565],
         [0.15157649, 0.0996299 , 0.09617049],
         [0.1621139 , 0.10768132, 0.09996882],
         ...,
         [0.5667919 , 0.5432625 , 0.59816444],
         [0.54468 , 0.5211506 , 0.5760526 ],
         [0.55206823, 0.52853876, 0.5834408 ]],

        [[0.2748953 , 0.20972031, 0.19427161],
         [0.2883572 , 0.220711 , 0.2017009 ],
         [0.30628815, 0.23883374, 0.20942336],
         ...,
         [0.5667876 , 0.54325813, 0.59816015],
         [0.5558772 , 0.5323478 , 0.58724976],
         [0.5574515 , 0.5339221 , 0.58882403]],

        ...,

        [[0.08537801, 0.06996575, 0.07392462],
         [0.08897338, 0.07327435, 0.07720229],
         [0.09513363, 0.07944736, 0.08336893],
         ...,
         [0.28853357, 0.24144736, 0.26593095],
         [0.26779646, 0.21901414, 0.24282953],
         [0.2510249 , 0.20315312, 0.21883588]],

        [[0.10264582, 0.08378803, 0.0890668 ],
         [0.10427538, 0.08675198, 0.09145431],
         [0.10935295, 0.09366667, 0.09758824],
         ...,
         [0.27253336, 0.22317132, 0.2467561 ],
         [0.25824815, 0.20761782, 0.23401856],
         [0.24551691, 0.19642025, 0.21759367]],

        [[0.11240316, 0.08548269, 0.09416311],
         [0.10489219, 0.08216092, 0.08916031],
         [0.10252255, 0.08603838, 0.09041229],
         ...,
         [0.2546206 , 0.20598234, 0.22640693],
         [0.2456365 , 0.19521558, 0.22098811],
         [0.24422826, 0.19411796, 0.21895869]]], dtype=float32)>))

```

- 제너레이터는 이 배치를 무한정으로 만들어낸다.
- 타깃 폴더에 있는 이미지를 끝없이 반복한다.

```
In [84]: ## 경로에 이미지 데이터의 개수
num_cats_tr = len(os.listdir(train_cats_dir))
num_dogs_tr = len(os.listdir(train_dogs_dir))

num_cats_val = len(os.listdir(val_cats_dir))
num_dogs_val = len(os.listdir(val_dogs_dir))

total_train = num_cats_tr + num_dogs_tr
total_val = num_cats_val + num_dogs_val

print("학습용 데이터 : ", total_train)
print("검증용 데이터 : ", total_val)

batch_size = 20
epochs = 30
```

학습용 데이터 : 2000

검증용 데이터 : 1000

04. 모델 학습

















목차로 이동하기

```
In [86]: %%time

# 조기 종료 콜백
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss', # 모니터링할 지표
    patience=5,         # 5번의 에포크 동안 개선되지 않으면 중단
    restore_best_weights=True # 가장 좋은 가중치로 복원
)

# 모델 학습 시 콜백 추가
history = model.fit(
    train_dataset,
    epochs=30,
    validation_data=val_dataset,
    callbacks=[early_stopping],
    verbose=1 # 진행 상황 출력
)
```

```

Epoch 1/30
80/80  13s 137ms/step - accuracy: 0.5576 - loss: 0.6843 - val
_accuracy: 0.6100 - val_loss: 0.6518
Epoch 2/30
80/80  13s 159ms/step - accuracy: 0.5736 - loss: 0.6746 - val
_accuracy: 0.6275 - val_loss: 0.6313
Epoch 3/30
80/80  19s 135ms/step - accuracy: 0.5834 - loss: 0.6747 - val
_accuracy: 0.6450 - val_loss: 0.6261
Epoch 4/30
80/80  21s 137ms/step - accuracy: 0.6029 - loss: 0.6582 - val
_accuracy: 0.6575 - val_loss: 0.6144
Epoch 5/30
80/80  20s 136ms/step - accuracy: 0.6067 - loss: 0.6560 - val
_accuracy: 0.6225 - val_loss: 0.6323
Epoch 6/30
80/80  11s 136ms/step - accuracy: 0.6334 - loss: 0.6384 - val
_accuracy: 0.6650 - val_loss: 0.5826
Epoch 7/30
80/80  12s 143ms/step - accuracy: 0.6121 - loss: 0.6377 - val
_accuracy: 0.7250 - val_loss: 0.5698
Epoch 8/30
80/80  20s 133ms/step - accuracy: 0.6500 - loss: 0.6228 - val
_accuracy: 0.7050 - val_loss: 0.5544
Epoch 9/30
80/80  21s 140ms/step - accuracy: 0.6655 - loss: 0.6184 - val
_accuracy: 0.7000 - val_loss: 0.5443
Epoch 10/30
80/80  20s 136ms/step - accuracy: 0.6620 - loss: 0.6159 - val
_accuracy: 0.7300 - val_loss: 0.5421
Epoch 11/30
80/80  11s 132ms/step - accuracy: 0.6823 - loss: 0.5954 - val
_accuracy: 0.7400 - val_loss: 0.5197
Epoch 12/30
80/80  10s 130ms/step - accuracy: 0.6671 - loss: 0.5957 - val
_accuracy: 0.7150 - val_loss: 0.5362
Epoch 13/30
80/80  21s 131ms/step - accuracy: 0.6783 - loss: 0.5929 - val
_accuracy: 0.7050 - val_loss: 0.5387
Epoch 14/30
80/80  11s 137ms/step - accuracy: 0.6771 - loss: 0.5873 - val
_accuracy: 0.6700 - val_loss: 0.5818
Epoch 15/30
80/80  11s 131ms/step - accuracy: 0.6845 - loss: 0.5898 - val
_accuracy: 0.7275 - val_loss: 0.5401
Epoch 16/30
80/80  9s 117ms/step - accuracy: 0.7004 - loss: 0.5780 - val
accuracy: 0.7175 - val_loss: 0.5259
CPU times: user 3min 27s, sys: 3.98 s, total: 3min 31s
Wall time: 4min 1s

```

```

In [87]: ### CPU 30 epochs : 52분 55초
         ### GPU 30 epochs : 4분 1초

```

05. 훈련 후, 모델 저장

목차로 이동하기

```
In [91]: # model.save('cats_and_dogs_small_1.h5')
model.save('cats_and_dogs_small_1.keras')

# 추후 로드 할 경우
# loaded_model = tf.keras.models.load_model('model_name.keras')
```

훈련 데이터와 검증 데이터의 모델의 손실과 정확도

```
In [92]: import matplotlib.pyplot as plt
```

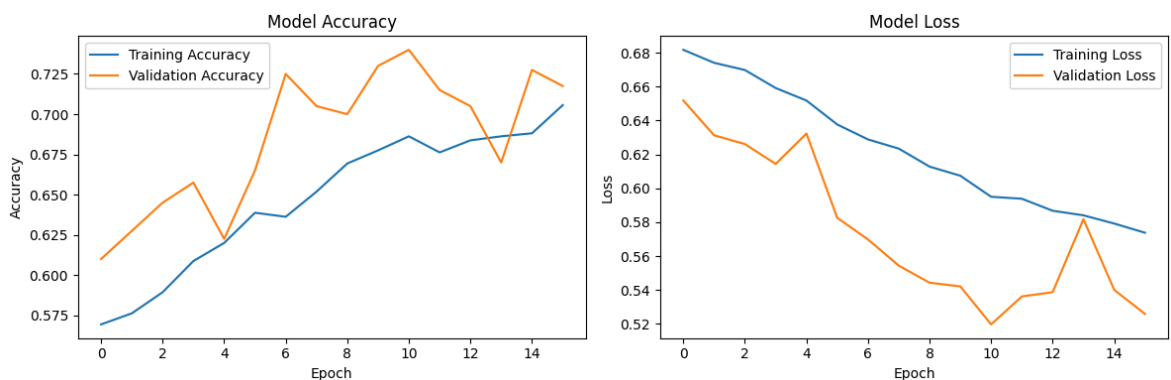
```
In [93]: import matplotlib.pyplot as plt

# 정확도와 손실 그래프 그리기
plt.figure(figsize=(12, 4))

# 정확도 그래프
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# 손실 그래프
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```



- 검증 손실(Validation Loss)은 지속적으로 감소하다가 10번째 에포크 이후 불안정한 변동을 보임.
- 훈련 손실(Training Loss)은 0.68에서 0.57까지 꾸준히 감소하는 경향을 보임.
- 검증 정확도(Validation Accuracy)는 훈련 정확도(Training Accuracy)보다 전반적으로 높게 유지되며, 최대 0.73까지 도달함.
- 비교적 훈련 샘플의 수(2,000개)가 적어 과대적합 위험이 있으나, 검증 정확도가 훈련 정확도보다 높게 유지되는 특이한 패턴을 보임.

- 드롭아웃이나 가중치 감소(L2규제)와 같은 과대적합을 감소시킬 수 있는 기법을 적용하기 전에, 현재 모델의 특이한 패턴에 대한 추가 분석이 필요함.

실습해 보기

1. 드롭아웃(Dropout) 적용

- 드롭아웃 레이어를 추가하여 과대적합을 방지하고, 최적의 드롭아웃 비율을 찾아보세요.
- **실습 목표:** 드롭아웃 비율을 0.2, 0.5, 0.7 등으로 설정하여 모델의 성능을 비교합니다.
- **코드 예시:**

```
from tensorflow.keras.layers import Dropout

model.add(Dropout(0.5)) # 드롭아웃 비율 50%
```

2. 가중치 감소(L2 규제) 실험

- L2 규제를 적용하여 모델의 복잡성을 줄이고, 과대적합을 방지하는 효과를 확인해보세요.
- **실습 목표:** L2 규제를 적용하여 모델의 성능을 비교합니다.
- **코드 예시:**

```
from tensorflow.keras.regularizers import l2

model.add(Dense(128, activation='relu',
kernel_regularizer=l2(0.01))) # L2 규제 적용
```

3. 데이터 증강(Data Augmentation) 활용

- 이미지 회전, 이동, 확대 등의 변형을 통해 데이터 다양성을 높이고, 모델의 일반화 능력을 향상시켜 보세요.
- **실습 목표:** ImageDataGenerator 를 사용하여 데이터 증강을 적용합니다.
- **코드 예시:**

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

4. 모델 아키텍처 조정

- 레이어의 수나 뉴런의 수를 조정하거나, 다른 종류의 레이어를 사용하여 모델 구조를 변경해 보세요.
- **실습 목표:** 레이어 수를 늘리고, 뉴런 수를 변경하여 성능을 개선해 봅니다.

- 코드 예시:

```
model.add(Dense(512, activation='relu')) # 뉴런 수 증가
```

5. 전이 학습(Transfer Learning) 적용

- 사전 훈련된 모델을 활용하여 성능을 개선하고, 적은 데이터로도 좋은 결과를 얻을 수 있는지 실험해 보세요.

- 실습 목표: VGG16 또는 ResNet 모델을 사용하여 전이 학습을 적용합니다.

- 코드 예시:

```
from tensorflow.keras.applications import import VGG16
```

```
base_model = VGG16(weights='imagenet', include_top=False,  
input_shape=(224, 224, 3))  
model.add(base_model)
```

- history
- 2024/11 code 실행 및 확인

In []: