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

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 [4]:
```
### 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 [5]:
```
# 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 15:59 sampleSubmission.csv
277664 -rw------- 1 root root 284321224 Nov 14 16:00 test1.zip
556204 -rw------- 1 root root 569546721 Nov 14 16:00 train.zip
```

파일 압축풀기

```
!rm -rf '/content/datasets/'
!unzip -q '/content/cats_dogs/test1.zip' -d '/content/datasets/'
!unzip -q '/content/cats_dogs/train.zip' -d '/content/datasets/'
```

In [6]:
```
!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 [7]:
```
!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 609276
drwxr-xr-x 2 root root 782336 Sep 20  2013 .
drwxr-xr-x 4 root root   4096 Nov 14 16:01 ..
-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 304264
drwxr-xr-x 2 root root 270336 Sep 20  2013 .
drwxr-xr-x 4 root root   4096 Nov 14 16:01 ..
-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 [9]:
```
# 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 [10]:
```
# 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 [11]:
```
# 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 [12]:
```
# 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 [13]:
```
# 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 [14]:
```
# 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 [15]:
```
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 [16]:
```
!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 16:02 .
drwxr-xr-x 5 root root 4096 Nov 14 16:02 ..
drwxr-xr-x 2 root root 4096 Nov 14 16:02 cats
drwxr-xr-x 2 root root 4096 Nov 14 16:02 dogs
total 16
drwxr-xr-x 4 root root 4096 Nov 14 16:02 .
drwxr-xr-x 5 root root 4096 Nov 14 16:02 ..
drwxr-xr-x 2 root root 4096 Nov 14 16:02 cats
drwxr-xr-x 2 root root 4096 Nov 14 16:02 dogs
total 16
drwxr-xr-x 4 root root 4096 Nov 14 16:02 .
drwxr-xr-x 5 root root 4096 Nov 14 16:02 ..
drwxr-xr-x 2 root root 4096 Nov 14 16:02 cats
drwxr-xr-x 2 root root 4096 Nov 14 16:02 dogs
```

데이터 준비

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

In [17]:
```
# 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 [19]:
```
print('학습용 고양이 이미지 전체 개수:', len(os.listdir(train_cats_dir)))
print('학습용 강아지 이미지 전체 개수:', len(os.listdir(train_dogs_dir)))

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

print('테스트용 고양이 이미지 전체 개수:', len(os.listdir(test_cats_dir)))
print('테스트용 강아지 이미지 전체 개수:', len(os.listdir(test_dogs_dir)))
```

```
학습용 고양이 이미지 전체 개수: 1000
학습용 강아지 이미지 전체 개수: 1000
검증용 고양이 이미지 전체 개수: 500
검증용 강아지 이미지 전체 개수: 500
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 [20]:
```
import torch
import torch.nn as nn

class CNNModel(nn.Module):
```

```
    def __init__(self):
        super(CNNModel, self).__init__()
        self.features = nn.Sequential(
            # 첫 번째 Conv 블록
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),

            # 두 번째 Conv 블록
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),

            # 세 번째 Conv 블록
            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),

            # 네 번째 Conv 블록
            nn.Conv2d(128, 128, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )

        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(128 * 9 * 9, 512),
            nn.ReLU(),
            nn.Linear(512, 1),
            nn.Sigmoid()
        )

    def forward(self, x):
        x = self.features(x)
        x = self.classifier(x)
        return x
```

In [23]:

```
from torchsummary import summary

model = CNNModel()
# GPU를 사용하는 경우
model = model.cuda()  # 또는 model = model.to('cuda')
summary(model, input_size=(3, 150, 150))  # 입력 이미지 크기 지정
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 32, 150, 150]             896
              ReLU-2         [-1, 32, 150, 150]               0
         MaxPool2d-3           [-1, 32, 75, 75]               0
            Conv2d-4           [-1, 64, 75, 75]          18,496
              ReLU-5           [-1, 64, 75, 75]               0
         MaxPool2d-6           [-1, 64, 37, 37]               0
            Conv2d-7          [-1, 128, 37, 37]          73,856
              ReLU-8          [-1, 128, 37, 37]               0
         MaxPool2d-9          [-1, 128, 18, 18]               0
           Conv2d-10          [-1, 128, 18, 18]         147,584
             ReLU-11          [-1, 128, 18, 18]               0
        MaxPool2d-12            [-1, 128, 9, 9]               0
          Flatten-13                [-1, 10368]               0
           Linear-14                  [-1, 512]       5,308,928
             ReLU-15                  [-1, 512]               0
           Linear-16                    [-1, 1]             513
          Sigmoid-17                    [-1, 1]               0
================================================================
Total params: 5,550,273
Trainable params: 5,550,273
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.26
Forward/backward pass size (MB): 22.31
Params size (MB): 21.17
Estimated Total Size (MB): 43.74
----------------------------------------------------------------
```

- PyTorch 모델의 구조와 파라미터 수를 보여줍니다. 각 레이어(Conv2d, MaxPool2d, Linear, Sigmoid)는 모델의 구성 요소를 나타내며,
- Output Shape은 각 레이어에서 출력되는 텐서의 크기입니다. Param #은 각 레이어에서 학습할 수 있는 파라미터의 수를 나타냅니다.
  - 예를 들어, 첫 번째 Conv2d 레이어는 32개의 필터를 사용하고, 이에 대한 파라미터 수는 896입니다.
- 전체 모델은 5,550,273개의 학습 가능한 파라미터를 가지고 있습니다.
- 최종적으로 150x150은 7x7의 크기에 도달합니다.

```
입력 크기: 150x150 (컬러 이미지, 채널 수 3)
첫 번째 Conv2D (32, 3x3):
출력 크기: (150 - 3) / 1 + 1 = 148 → 출력 크기: 148x148
첫 번째 MaxPooling2D (2x2):
출력 크기: 148 / 2 = 74 → 출력 크기: 74x74

두 번째 Conv2D (64, 3x3):
출력 크기: (74 - 3) / 1 + 1 = 72 → 출력 크기: 72x72
두 번째 MaxPooling2D (2x2):
출력 크기: 72 / 2 = 36 → 출력 크기: 36x36
...
네 번째 Conv2D (128, 3x3):
출력 크기: (17 - 3) / 1 + 1 = 15 → 출력 크기: 15x15
네 번째 MaxPooling2D (2x2):
출력 크기: 15 / 2 = 7 → 출력 크기: 7x7
```

모델 컴파일 및 손실 함수, 최적화 함수 설정

In [25]:
```
import torch.optim as optim  # torch.optim 임포트 추가

# 3. 학습 설정
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = CNNModel().to(device)
criterion = nn.BCELoss()
optimizer = optim.RMSprop(model.parameters(), lr=1e-4)

print(device, model)
```

```
cuda CNNModel(
  (features): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=Fals
e)
    (3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=Fals
e)
    (6): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU()
    (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=Fals
e)
    (9): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (10): ReLU()
    (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=Fal
se)
  )
  (classifier): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=10368, out_features=512, bias=True)
    (2): ReLU()
    (3): Linear(in_features=512, out_features=1, bias=True)
    (4): Sigmoid()
  )
)
```

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 [26]:
```
from torch.utils.data import DataLoader
from torchvision import datasets, transforms

# 데이터 변환 정의
transform = transforms.Compose([
    transforms.Resize((150, 150)),
```

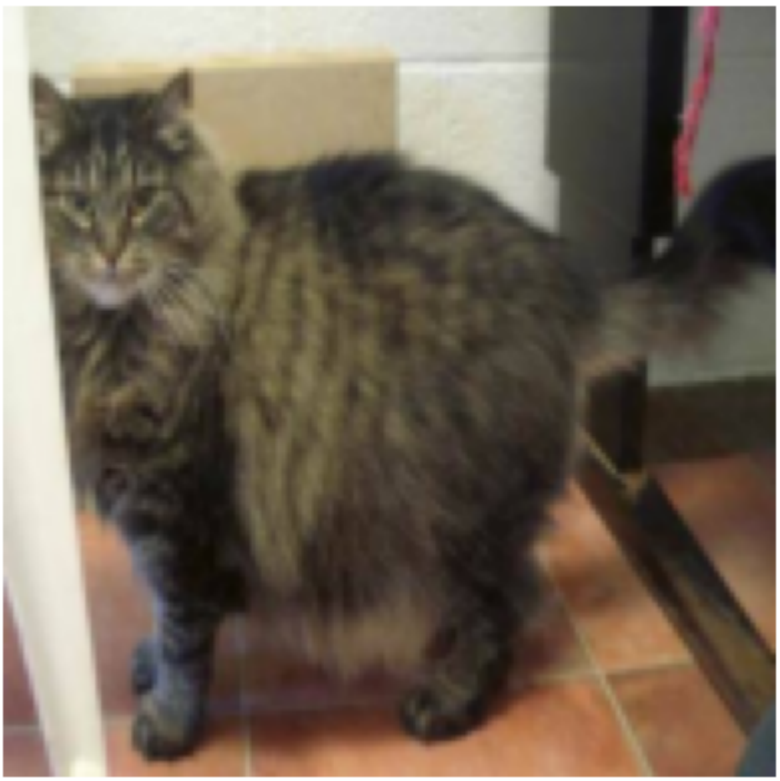
```
    transforms.ToTensor(),
])

# 데이터셋 로드
train_dataset = datasets.ImageFolder('/content/datasets/cats_and_dogs_small/trai
val_dataset = datasets.ImageFolder('/content/datasets/cats_and_dogs_small/valida

# DataLoader 정의
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
```

이미지 제너레이터을 활용하여 데이터와 라벨을 확인해보기

In [27]:
```
import torch
import matplotlib.pyplot as plt

# 클래스 이름 확인
class_names = train_dataset.classes  # train_dataset.classes 사용
print("클래스 이름:", class_names)

# 배치의 상세 정보
for data_batch, labels_batch in train_loader:  # train_loader로 배치 처리
    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].item())  # .item()으로 sca

    # 이미지 시각화
    plt.figure(figsize=(10, 5))
    # 데이터는 [C, H, W] 순서이므로 채널 순서를 [H, W, C]로 바꿔야 plt에서 제대로
    plt.imshow(data_batch[0].permute(1, 2, 0).numpy())  # [C, H, W] -> [H, W, C]
    plt.title(f'Label: {labels_batch[0].item()} ({class_names[labels_batch[0].it
    plt.axis('off')
    plt.show()

    break  # 첫 번째 배치만 처리
```

```
클래스 이름: ['cats', 'dogs']
배치 데이터 형태: torch.Size([32, 3, 150, 150])
배치 레이블 형태: torch.Size([32])

첫 번째 이미지 레이블: 0
```

Label: 0 (cats)



In [28]:
```
labels_batch[0:3], data_batch[0]
```

Out[28]:
```
(tensor([0, 0, 0]),
 tensor([[[0.9569, 0.9608, 0.9569,  ..., 0.4157, 0.4039, 0.4000],
          [0.9608, 0.9647, 0.9569,  ..., 0.4157, 0.4118, 0.4000],
          [0.9686, 0.9686, 0.9608,  ..., 0.4196, 0.4157, 0.4000],
          ...,
          [0.6235, 0.6196, 0.6157,  ..., 0.2627, 0.2549, 0.2510],
          [0.6314, 0.6275, 0.6235,  ..., 0.2549, 0.2549, 0.2627],
          [0.6510, 0.6510, 0.6471,  ..., 0.2549, 0.2549, 0.2510]],

         [[0.9608, 0.9647, 0.9647,  ..., 0.4196, 0.4078, 0.4039],
          [0.9647, 0.9686, 0.9647,  ..., 0.4196, 0.4157, 0.4000],
          [0.9725, 0.9725, 0.9686,  ..., 0.4235, 0.4196, 0.4039],
          ...,
          [0.4706, 0.4706, 0.4627,  ..., 0.2941, 0.2784, 0.2784],
          [0.4824, 0.4784, 0.4745,  ..., 0.2824, 0.2706, 0.2745],
          [0.5020, 0.5020, 0.4980,  ..., 0.2784, 0.2667, 0.2627]],

         [[0.9373, 0.9333, 0.9216,  ..., 0.4000, 0.3882, 0.3922],
          [0.9412, 0.9373, 0.9216,  ..., 0.4000, 0.3961, 0.3882],
          [0.9490, 0.9412, 0.9255,  ..., 0.4039, 0.4000, 0.3922],
          ...,
          [0.4392, 0.4392, 0.4314,  ..., 0.3176, 0.3176, 0.3216],
          [0.4431, 0.4392, 0.4353,  ..., 0.3020, 0.3176, 0.3294],
          [0.4588, 0.4588, 0.4549,  ..., 0.2980, 0.3176, 0.3216]]]))
```

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

In [29]:
```
## 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 [32]:

```
# 4. 학습 함수
def train_model(model, train_loader, val_loader, criterion, optimizer, num_epoch
    history = {'train_loss': [], 'train_acc': [], 'val_loss': [], 'val_acc': []}

    for epoch in range(num_epochs):
        # 학습 모드
        model.train()
        running_loss = 0.0
        correct = 0
        total = 0

        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.float().to(device)

            optimizer.zero_grad()
            outputs = model(inputs).squeeze()
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()
            predicted = (outputs > 0.5).float()
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

        train_loss = running_loss / len(train_loader)
        train_acc = correct / total

        # 검증 모드
        model.eval()
        val_loss = 0.0
        correct = 0
        total = 0

        with torch.no_grad():
            for inputs, labels in val_loader:
                inputs, labels = inputs.to(device), labels.float().to(device)
```

```
                outputs = model(inputs).squeeze()
                loss = criterion(outputs, labels)

                val_loss += loss.item()
                predicted = (outputs > 0.5).float()
                total += labels.size(0)
                correct += (predicted == labels).sum().item()

        val_loss = val_loss / len(val_loader)
        val_acc = correct / total

        print(f'Epoch {epoch+1}/{num_epochs}')
        print(f'Train Loss: {train_loss:.4f} Train Acc: {train_acc:.4f}')
        print(f'Val Loss: {val_loss:.4f} Val Acc: {val_acc:.4f}')

        history['train_loss'].append(train_loss)
        history['train_acc'].append(train_acc)
        history['val_loss'].append(val_loss)
        history['val_acc'].append(val_acc)

    return history
```

In [33]:

```
%%time

# 5. 모델 학습
history = train_model(model, train_loader, val_loader, criterion, optimizer)
```

```
Epoch 1/30
Train Loss: 0.6854 Train Acc: 0.5520
Val Loss: 0.6809 Val Acc: 0.5770
Epoch 2/30
Train Loss: 0.6773 Train Acc: 0.5825
Val Loss: 0.6626 Val Acc: 0.6140
Epoch 3/30
Train Loss: 0.6458 Train Acc: 0.6275
Val Loss: 0.6270 Val Acc: 0.6560
Epoch 4/30
Train Loss: 0.6209 Train Acc: 0.6680
Val Loss: 0.6044 Val Acc: 0.6710
Epoch 5/30
Train Loss: 0.5874 Train Acc: 0.6840
Val Loss: 0.7261 Val Acc: 0.5670
Epoch 6/30
Train Loss: 0.5607 Train Acc: 0.7095
Val Loss: 0.5992 Val Acc: 0.6610
Epoch 7/30
Train Loss: 0.5391 Train Acc: 0.7320
Val Loss: 0.5626 Val Acc: 0.7070
Epoch 8/30
Train Loss: 0.5212 Train Acc: 0.7475
Val Loss: 0.5796 Val Acc: 0.6910
Epoch 9/30
Train Loss: 0.5077 Train Acc: 0.7545
Val Loss: 0.5870 Val Acc: 0.6860
Epoch 10/30
Train Loss: 0.4889 Train Acc: 0.7715
Val Loss: 0.5494 Val Acc: 0.7250
Epoch 11/30
Train Loss: 0.4745 Train Acc: 0.7655
Val Loss: 0.5501 Val Acc: 0.7190
Epoch 12/30
Train Loss: 0.4597 Train Acc: 0.7765
Val Loss: 0.5799 Val Acc: 0.6940
Epoch 13/30
Train Loss: 0.4513 Train Acc: 0.7915
Val Loss: 0.5393 Val Acc: 0.7370
Epoch 14/30
Train Loss: 0.4296 Train Acc: 0.7995
Val Loss: 0.5368 Val Acc: 0.7390
Epoch 15/30
Train Loss: 0.4189 Train Acc: 0.8060
Val Loss: 0.5529 Val Acc: 0.7330
Epoch 16/30
Train Loss: 0.4029 Train Acc: 0.8330
Val Loss: 0.5343 Val Acc: 0.7310
Epoch 17/30
Train Loss: 0.3865 Train Acc: 0.8335
Val Loss: 0.5734 Val Acc: 0.7170
Epoch 18/30
Train Loss: 0.3732 Train Acc: 0.8350
Val Loss: 0.5507 Val Acc: 0.7180
Epoch 19/30
Train Loss: 0.3619 Train Acc: 0.8355
Val Loss: 0.5837 Val Acc: 0.7180
Epoch 20/30
Train Loss: 0.3511 Train Acc: 0.8495
Val Loss: 0.7438 Val Acc: 0.6690
```

```
Epoch 21/30
Train Loss: 0.3342 Train Acc: 0.8600
Val Loss: 0.5625 Val Acc: 0.7360
Epoch 22/30
Train Loss: 0.3258 Train Acc: 0.8620
Val Loss: 0.6362 Val Acc: 0.7150
Epoch 23/30
Train Loss: 0.3027 Train Acc: 0.8625
Val Loss: 0.5789 Val Acc: 0.7400
Epoch 24/30
Train Loss: 0.2857 Train Acc: 0.8825
Val Loss: 0.8041 Val Acc: 0.6560
Epoch 25/30
Train Loss: 0.2838 Train Acc: 0.8820
Val Loss: 0.5932 Val Acc: 0.7430
Epoch 26/30
Train Loss: 0.2622 Train Acc: 0.8905
Val Loss: 0.6345 Val Acc: 0.7200
Epoch 27/30
Train Loss: 0.2576 Train Acc: 0.9000
Val Loss: 0.6311 Val Acc: 0.7270
Epoch 28/30
Train Loss: 0.2296 Train Acc: 0.9150
Val Loss: 0.6818 Val Acc: 0.7250
Epoch 29/30
Train Loss: 0.2192 Train Acc: 0.9200
Val Loss: 0.6425 Val Acc: 0.7400
Epoch 30/30
Train Loss: 0.1924 Train Acc: 0.9280
Val Loss: 0.7449 Val Acc: 0.7220
CPU times: user 5min 26s, sys: 3.72 s, total: 5min 30s
Wall time: 5min 32s
```

In [ ]:
```
### GPU 30 epochs : 5분 32초
```

05. 훈련 후, 모델 저장

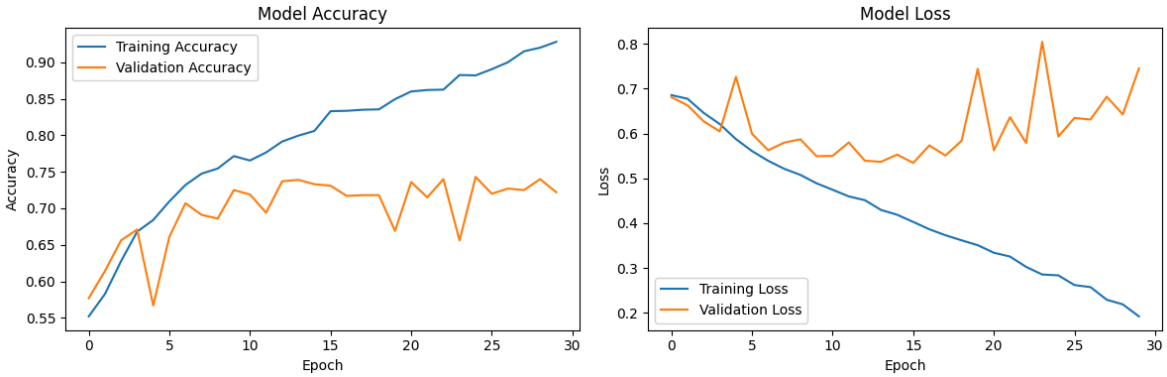
목차로 이동하기

In [34]:
```
# 6. 모델 저장
torch.save(model.state_dict(), 'cats_and_dogs_pytorch.pth')
```

훈련 데이터와 검증 데이터의 모델의 손실과 정확도

In [36]:
```
import matplotlib.pyplot as plt

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

# 정확도 그래프
plt.subplot(1, 2, 1)
plt.plot(history['train_acc'], label='Training Accuracy')
plt.plot(history['val_acc'], 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['train_loss'], label='Training Loss')
plt.plot(history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

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



- 1. 검증 손실(Validation Loss) 분석
  - 약 5번째 에포크에서 검증 손실이 최소값에 도달
  - 이후 더 이상의 성능 향상 없이 오히려 불안정하게 변동
  - 오렌지색 선이 불규칙하게 움직이는 것이 이를 보여줌
- 2. 훈련 손실(Training Loss) 분석
  - 파란색 선이 지속적으로 감소하며 거의 0에 가까워짐
  - 이는 모델이 훈련 데이터에 과도하게 맞춰지고 있음을 의미
- 03.과적합 문제 해결 방안
  - 적은 훈련 데이터(2,000개)로 인한 과적합 현상 발생
  - 다음과 같은 정규화 기법 적용 필요:
    - Dropout 층 추가
    - L2 가중치 규제 적용
    - 데이터 증강(Data Augmentation) 고려

실습해 보기

- 다양한 방법을 이용해서 성능을 개선시키는 것을 해 보기

- history
- 2024/11 code 실행 및 확인

In [ ]: