REPORT

Dogs vs. Cats
6조 팀 프로젝트 결과 보고서

제출일	2022. 12. 06	팀 원	201984002 김다현
과 목	딥러닝기초[01]		202084021 양희훈
담당교수	안정호		202104268 이경수
팀 명	6조		201804235 이현중

목 차

Ⅰ. A1 모델

1. 구현 개요

A1 모델은 랜덤한 데이터 3000개를 사용하였으며, 모델을 구현 시 maxpooling, drop out, flatten 레이어를 사용하였다. 최종 데이터를 보다 효율적으로 확인하기 위해 그래프를 이용하여 시각적으로 결과를 나타냈다.

2. 구현 상세

2.1 패키지 설치와 데이터 준비

```
[ ] import os
    import os.path as osp
    import numpy as np
    import matplotlib.pyplot as plt
```

```
[ ] from google.colab import drive        # Google Drive를 Colab에 연결해서 데이터 로딩하기
    drive.mount('/content/drive')         # 드라이브 마운트
```

```
Mounted at /content/drive
```

구글 드라이브에 colab을 연결하여 데이터를 가져온다.

```
[ ] from keras.preprocessing.image import ImageDataGenerator
    import numpy as np
    import pandas as pd
    from keras.utils import to_categorical
    from sklearn.model_selection import train_test_split
    import matplotlib.pyplot as plt
    import random
    import os
    print(os.listdir('/content/drive/MyDrive/deep_learning/팀플/data/'))
```

```
['sampleSubmission.csv', 'test1.zip', 'train.zip']
```

```
[ ] import zipfile
    train_zip = zipfile.ZipFile('/content/drive/MyDrive/deep_learning/팀플/data/train.zip')
    train_zip.extractall("/kaggle/working/dataset")
    train_zip.close()

    test_zip = zipfile.ZipFile('/content/drive/MyDrive/deep_learning/팀플/data/test1.zip')
    test_zip.extractall("/kaggle/working/dataset")
    test_zip.close()
```

필요한 라이브러리와 데이터를 불러오고 경로내에 있는 파일의 이름을 list형태로 반환한다.
zipfile 라이브러리를 사용해 압축을 해제한다.

```
[ ]
    filenames = os.listdir("/kaggle/working/dataset/train")
    categories = []
    for filename in filenames:
        category = filename.split('.')[0]
        if category == 'dog':
            categories.append(1)
        else:
            categories.append(0)

    df = pd.DataFrame({
        'filename': filenames,
        'category': categories
    })
```

```
[ ] # 3000개만
    df = df.sample(n=3000)
```

```
[ ] # 데이터 형상 관련 상수 정의
    IMAGE_WIDTH=150
    IMAGE_HEIGHT=150
    IMAGE_SIZE=(IMAGE_WIDTH, IMAGE_HEIGHT)                #1
    IMAGE_CHANNELS=3
```

Training data를 준비하는 단계이다.
3000개의 데이터를 임의로 선택하게 설정하고 이미지 사이즈를 150*150으로 할당해주었다.

2.2 모델구성

```
import tensorflow as tf



model = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(32, (3,3), activation='relu',  input_shape=(IMAGE_WIDTH, IMAGE_HEIGHT, IMAGE_CHANNELS)),
  tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2,2),
  tf.keras.layers.Dropout(0.25),
  tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2,2),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(2, activation='sigmoid')
])

model.summary()
```

```
[ ] from tensorflow.keras import optimizers

    model.compile(optimizer=optimizers.RMSprop(lr=0.001),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

model마다 순차적으로 레이어를 쌓아주고 인자마다 값을 넣어준 후 pooing의 크기를 설정해주었다. optimizer는 RMSprop를 사용하였다.

```
df["category"] = df["category"].replace({0: 'cat', 1: 'dog'})
```

```
train_df, validate_df = train_test_split(df, test_size=0.3333, random_state=42)
train_df = train_df.reset_index(drop=True)
validate_df = validate_df.reset_index(drop=True)
```

손실함수를 categorical_crossentropy로 설정하였기 때문에 0을 cat으로 1을 dog로 바꿔주었다.
train_test_split을 사용하여 66%를 학습데이터, 33%를 검증 데이터로 나눠주었다.

```
#이미지 데이터 전처리

#Traning Generator
train_datagen = ImageDataGenerator(
    rotation_range=15,
    rescale=1./255,
    shear_range=0.1,
    zoom_range=0.2,
    horizontal_flip=True,
    width_shift_range=0.1,
    height_shift_range=0.1
)
```

```
train_generator = train_datagen.flow_from_dataframe(
    train_df,
    "/kaggle/working/dataset/train",
    x_col='filename',
    y_col='category',
    target_size=IMAGE_SIZE,
    class_mode='categorical',
    batch_size=batch_size
)
```

트레이닝 제너레이터를 설정하였다.
전처리 과정으로 이미지를 1/255로 스케일을 조정하였으며 랜덤으로 각도, 확대범위를 정해주었다.
앞서 만든 트레이닝 제너레이터를 사용하여 train_data를 Numpy Array Iterator로 만들었다.

```
# Validation Generator
validation_datagen = ImageDataGenerator(rescale=1./255)
validation_generator = validation_datagen.flow_from_dataframe(
    validate_df,
    "/kaggle/working/dataset/train",
    x_col='filename',
    y_col='category',
    target_size=IMAGE_SIZE,
    class_mode='categorical',
    batch_size=batch_size
)
```

마찬가지로 검증 데이터 또한 같은 방식으로 전처리 및 Numpy Array Iterator로 만들었다.

```
epochs=10
history = model.fit_generator(
    train_generator,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=total_validate//batch_size,
    steps_per_epoch=total_train//batch_size,
    verbose = 1
)
```

에포크를 10번으로 하여 학습을 진행했다. steps_per_epoch는 한 세대의 종료를 선언하고 다음 세대를 시작하기까지 단계(샘플 배치)의 총 개수 validation_steps는 정지 전 검증할 단계(샘플 배치)의 총 개수로 한 epoch 종료 시마다 검증할 때 사용되는 검증 스텝 수를 나타낸다.

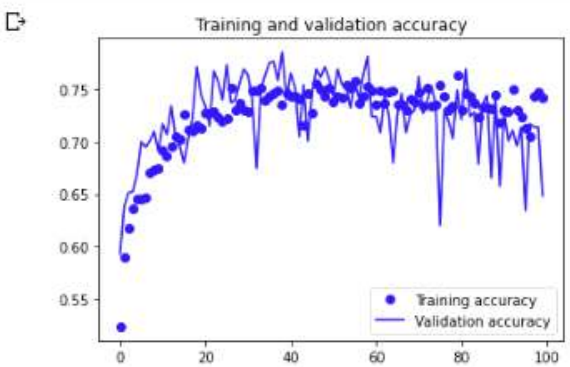
```
Epoch 1/10
133/133 [==============================] - 320s 2s/step - loss: 0.8559 - accuracy: 0.5330 - val_loss: 0.6859 - val_accuracy: 0.5444
Epoch 2/10
133/133 [==============================] - 323s 2s/step - loss: 0.7049 - accuracy: 0.5935 - val_loss: 0.6418 - val_accuracy: 0.6212
Epoch 3/10
133/133 [==============================] - 317s 2s/step - loss: 0.6457 - accuracy: 0.6332 - val_loss: 0.6081 - val_accuracy: 0.6778
Epoch 4/10
133/133 [==============================] - 325s 2s/step - loss: 0.6351 - accuracy: 0.6564 - val_loss: 0.5749 - val_accuracy: 0.7232
Epoch 5/10
133/133 [==============================] - 317s 2s/step - loss: 0.6094 - accuracy: 0.6816 - val_loss: 0.6088 - val_accuracy: 0.6586
Epoch 6/10
133/133 [==============================] - 314s 2s/step - loss: 0.6111 - accuracy: 0.6680 - val_loss: 0.5558 - val_accuracy: 0.7303
Epoch 7/10
133/133 [==============================] - 316s 2s/step - loss: 0.5999 - accuracy: 0.6882 - val_loss: 0.5555 - val_accuracy: 0.7303
Epoch 8/10
133/133 [==============================] - 316s 2s/step - loss: 0.5884 - accuracy: 0.6992 - val_loss: 0.5889 - val_accuracy: 0.6889
Epoch 9/10
133/133 [==============================] - 318s 2s/step - loss: 0.5835 - accuracy: 0.6957 - val_loss: 0.5497 - val_accuracy: 0.7384
Epoch 10/10
133/133 [==============================] - 319s 2s/step - loss: 0.5689 - accuracy: 0.7068 - val_loss: 0.5780 - val_accuracy: 0.7182
```

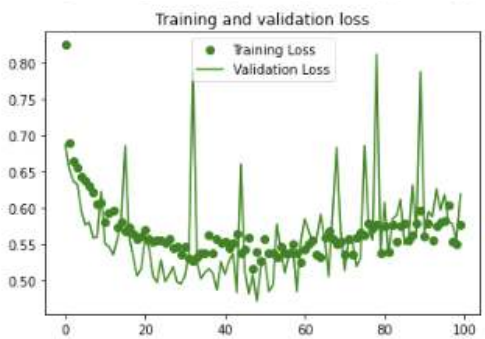
학습 결과 처음에는 정확도가 50% 정도였지만 Epoch가 늘어나 정확도가 70%까지 올랐다.

3. 구현 결과

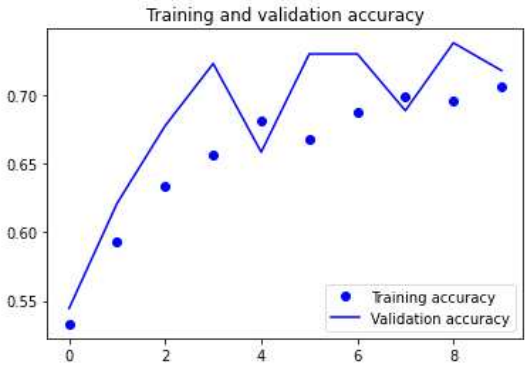
```
Epoch 94/100
133/133 [==============================] - 18s 137ms/step - loss: 0.5748 - accuracy: 0.7300 - val_loss: 0.6269 - val_accuracy: 0.6960
Epoch 95/100
133/133 [==============================] - 18s 137ms/step - loss: 0.5813 - accuracy: 0.7244 - val_loss: 0.5973 - val_accuracy: 0.7141
Epoch 96/100
133/133 [==============================] - 18s 137ms/step - loss: 0.5833 - accuracy: 0.7128 - val_loss: 0.6190 - val_accuracy: 0.6343
Epoch 97/100
133/133 [==============================] - 18s 137ms/step - loss: 0.6039 - accuracy: 0.7053 - val_loss: 0.5825 - val_accuracy: 0.7172
Epoch 98/100
133/133 [==============================] - 18s 137ms/step - loss: 0.5545 - accuracy: 0.7441 - val_loss: 0.5783 - val_accuracy: 0.7141
Epoch 99/100
133/133 [==============================] - 18s 137ms/step - loss: 0.5498 - accuracy: 0.7481 - val_loss: 0.5560 - val_accuracy: 0.7141
Epoch 100/100
133/133 [==============================] - 18s 137ms/step - loss: 0.5780 - accuracy: 0.7426 - val_loss: 0.6196 - val_accuracy: 0.6485
```

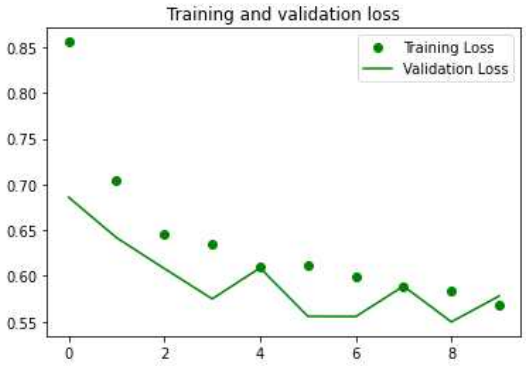
100번의 Epoch를 했을때 오히려 정확도가 낮아졌다.





너무 많은 학습을 하여 과적합이 된 것을 확인했다.
Epoch를 10번으로 학습하여 결과를 냈다.





정확도는 증가하다 더이상 증가하지 않는다. loss또한 감소하다 적당한 선에서 멈춘다. 더 학습할 경우 과적합의 우려가 있어 학습을 멈췄다. A1의 모델경우 정확도 약 70%의 성능을 가진다.

```
submission = pd.read_csv('/content/drive/MyDrive/deep_learning/팀플/data/sampleSubmission.csv')
```

```
test = os.listdir("/kaggle/working/dataset/test1")

test_df = pd.DataFrame({'filename': test})

nb_samples = test_df.shape[0]
```

submission 파일과 test 파일을 가져왔다.

```
predict = model.predict_generator(test_generator, steps=np.ceil(nb_samples/batch_size))
```

앞서 만든 모델을 이용해 예측을 진행하였다.

```
submission_df = test_df.copy()
submission_df['id'] = submission_df['filename'].str.split('.').str[0]
submission_df['label'] = submission_df['category']
submission_df.drop(['filename', 'category'], axis=1, inplace=True)
```

앞서 불러온 submission의 열은 id와 label이였고 test데이터의 번호를 id로 예측된 결과 0, 1을 label로 똑같이 만들어 주었다.

Ⅱ. A2 모델

1. 구현 개요

Ⅱ번 항목인 A1 모델의 구현 결과 과대적합 문제가 발견되었다. 또한 전체 데이터를 확인하는 과정에서 소수의 불필요한 데이터가 포함되어 있음을 확인하였다. 따라서 A2 모델에서는 이러한 불필요한 이미지를 사전에 삭제하고, 문제가 되었던 과대적합을 해결하기 위해 모델 구현 과정에서의 Dropout 사용, 데이터에 대한 데이터 증식 처리를 진행하였다.

2. 구현 상세

2.1 패키지 설치와 데이터 준비

```
[ ] import os
    import os.path as osp
    import numpy as np
    import matplotlib.pyplot as plt
```

```
from google.colab import drive        # Google Drive를 Colab에 연결해서 데이터 로딩하기
drive.mount('/content/drive')         # 드라이브 마운트
```

```
Mounted at /content/drive
```

```
[ ] from keras.preprocessing.image import ImageDataGenerator   #keras로부터 ImageDataGenerator를 불러옵니다
    import numpy as np
    import pandas as pd
    from keras.utils import to_categorical                     #데이터를 다루는데 필요한 라이브러리를 불러옵니다.
    from sklearn.model_selection import train_test_split
    import matplotlib.pyplot as plt
    import random
    import os
    print(os.listdir('/content/drive/MyDrive/deep_learning/팀플/data/'))   #os.listdir() 메서드는 경로 내에 있는 파일의
                                                                          #이름을 리스트의 형태로 반환합니다.
```

```
['sampleSubmission.csv', 'test1.zip', 'train.zip']
```

다운 받은 케글 데이터를 구글 드라이브에 업로드하여 데이터를 준비한다.
os라이브러리를 통하여 listdir을 사용해 파일이 잘 받아왔는지 이름을 list형태로 받아와서 확인한다.

```
import zipfile
#파일 압출 풀기
original_base_dir = '/content/datasets'
os.mkdir(original_base_dir)
dataset_zip = zipfile.ZipFile('/content/drive/MyDrive/deep_learning/팀플/data/test1.zip')
dataset_zip.extractall(original_base_dir)
dataset_zip.close()

dataset_zip = zipfile.ZipFile('/content/drive/MyDrive/deep_learning/팀플/data/train.zip')
dataset_zip.extractall(original_base_dir)
dataset_zip.close()
```

zipfile 라이브러리를 사용해 압축을 해제한다.

```
[ ] import os
    labels=['dogs', 'cats']
    for label in labels:
        os.makedirs(os.path.join('train', label), exist_ok = True)
```

```
[ ] !find /content/datasets/train -name 'dog.*' -exec mv {} train/dogs/ ₩; #파일 이름을 기준으로 자료를 각각에 폴더에 이동시켜서 나눠주기
    !find /content/datasets/train -name 'cat.*' -exec mv {} train/cats/ ₩;
```

train셋 label을 dogs,cats로 분류한 뒤 파일이름을 기준으로 이미지를 각각의 폴더에 이동시켜서 나눠준다.

2.2 불필요한 데이터 삭제

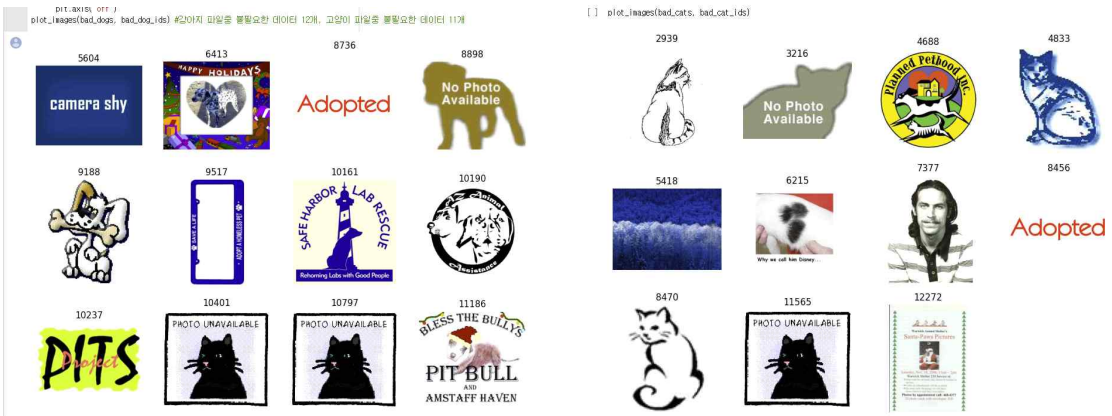
```
bad_dog_ids = [5604, 6413, 8736, 8898, 9188, 9517, 10161, #사전 테스트를 통해 불필요한 이미지 목록
               10190, 10237, 10401, 10797, 11186]

bad_cat_ids = [2939, 3216, 4688, 4833, 5418, 6215, 7377,
               8456, 8470, 11565, 12272]
```

사전에 데이터를 확인하는 과정에서 불필요한 이미지 목록이 있음을 확인하였다.
이를 제거하기 위해 이미지들의 번호를 리스트 형태로 bad_dog_ids와 bad_cat_ids에 할당한다.

```
import os
import matplotlib.image as mpimg
#불필요한 이미지 확인 과정
def load_images(ids, categ):
  images = []
  dirname = categ+'s' # dog -> dogs
  for theid in ids:
    fname = 'train/{dirname}/{categ}.{theid}.jpg'.format(
        dirname=dirname,
        categ=categ,
        theid=theid
    )
    im = mpimg.imread(fname)
    images.append(im)
  return images
bad_dogs = load_images(bad_dog_ids, 'dog')
bad_cats = load_images(bad_cat_ids, 'cat')
def plot_images(images, ids):
    ncols, nrows = 4, 3
    fig = plt.figure( figsize=(ncols*3, nrows*3), dpi=90)
    for i, (mpimg, theid) in enumerate(zip(images,ids)):
      plt.subplot(nrows, ncols, i+1)
      plt.imshow(mpimg)
      plt.title(str(theid))
      plt.axis('off')
```

앞서 선언한 리스트 번호 변수를 이용하여 강아지 파일 중 불필요한 데이터 12개와 고양이 파일 중 불필요한 데이터에 해당하는 이미지 파일 11개를 확인하였다.



```
[ ]  subsets=['cats', 'dogs'] #남은 이미지들을 하나의 폴더로 모아줌
     for subset in subsets:
         category = subset[:-1]
         if category == 'cat':
             ids = bad_cat_ids
         else:
             ids = bad_dog_ids

         for id in ids:
             fname = category +'.'+ str(id) + '.jpg'
             dirty_img = os.path.join('train',subset,fname)
             print(dirty_img)
             os.remove(dirty_img)
             print("clean up data")
```

확인한 불필요 데이터를 os.remove()함수를 사용해 제거해준다.

2.3 글로벌 변수 선언과 정답 설정

```
# 데이터 형상 관련 상수 정의
FAST_RUN = False
IMAGE_WIDTH=150
IMAGE_HEIGHT=150
IMAGE_SIZE=(IMAGE_WIDTH, IMAGE_HEIGHT)
IMAGE_CHANNELS=3
```

이번 프로젝트에서 사용할 이미지의 크기를 150,150으로 지정해주고 이미지 RBG값을 위해 IMAGE_CHANNELS은 3으로 지정해주었다.

```
[ ]  filenames = os.listdir("/content/datasets/train")
     categories = []
     for filename in filenames:
         category = filename.split('.')[0]
         if category == 'dog':
             categories.append(1)
         else:
             categories.append(0)

     df = pd.DataFrame({
         'filename': filenames,
         'category': categories
     })
```

Os.listdir을 사용하여 파일이름을 리스트 형태로 받아온 후 카테고리를 파일이름에 따라서 강아지는 1로 고양이는 0으로 분류해준다.

```
df.head() #데이터 5개를 불러와 잘 분류되여있는지 확인
```

	filename	category
0	cat.9402.jpg	0
1	cat.2156.jpg	0
2	dog.3070.jpg	1
3	cat.286.jpg	0
4	dog.10072.jpg	1

맨 위의 데이터 5개를 불러와 파일 이름에 맞춰 잘 분류되었는지 확인해준다.

```
[ ]  print(df['category'].value_counts()) #카테고리 분류한 데이터의 갯수 확인
```

```
0    12489
1    12488
Name: category, dtype: int64
```

카테고리 분류한 전체 데이터를 확인한 결과 강아지는 12489개 고양이는 12488개의 이미지로 확인 되었다.
누락된 파일 없이 잘 분류되었음을 알 수 있다.

2.4 신경망 모델 구현

```
from keras.backend import dropout
from keras.layers.normalization.batch_normalization_v1 import BatchNormalization
from keras import layers, models
#컨볼류션 2d와 maxpooling을 번갈아 사용
#중간에 BatchNormalization을 통해 학습하는 과정 자체를 전체적으로 안정화하여 학습 속도를 가속 시켜줌
#dropout을 통해 오버피팅을 방지
#Flatten을 통해 3차원을 1차원 배열로 바꿔줌
model = models.Sequential()
model.add(layers.Conv2D(32, (3,3), activation='relu', input_shape=(IMAGE_WIDTH,IMAGE_HEIGHT,IMAGE_CHANNELS)))
model.add(BatchNormalization())
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Dropout(0.25))

model.add(layers.Conv2D(64, (3,3), activation='relu'))
model.add(BatchNormalization())
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Dropout(0.25))

model.add(layers.Conv2D(128, (3,3), activation='relu'))
model.add(BatchNormalization())
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Dropout(0.25))

model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(BatchNormalization())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(2, activation='sigmoid'))

model.summary()
```

신경망 모델은 총 4개의 은닉층으로 구성되도록 설계했으며 컨볼루젼2d와 멕스풀링을 번갈아 사용하였고 중간에 배치노말라이제이션을 통해 학습하는 과정 자체를 전체적으로 안정화하여 학습 속도를 가속 시켰다.
드랍아웃을 통해 과적합을 막고 마지막에 플래튼을 통해 3차원을 1차원으로 바꾸어주었다.
또한 해당 프로젝트는 이진분류 문제를 다루므로 출력층에서는 시그모이드 함수를 이용하여 마무리하였다.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 32)      896

batch_normalization (BatchN  (None, 148, 148, 32)      128
ormalization)

max_pooling2d (MaxPooling2D  (None, 74, 74, 32)       0
)

dropout (Dropout)            (None, 74, 74, 32)        0

conv2d_1 (Conv2D)            (None, 72, 72, 64)        18496

batch_normalization_1 (Batc  (None, 72, 72, 64)        256
hNormalization)

max_pooling2d_1 (MaxPooling  (None, 36, 36, 64)        0
2D)

dropout_1 (Dropout)          (None, 36, 36, 64)        0

conv2d_2 (Conv2D)            (None, 34, 34, 128)       73856

batch_normalization_2 (Batc  (None, 34, 34, 128)       512
hNormalization)

max_pooling2d_2 (MaxPooling  (None, 17, 17, 128)       0
2D)

dropout_2 (Dropout)          (None, 17, 17, 128)       0

flatten (Flatten)            (None, 36992)             0

dense (Dense)                (None, 512)               18940416

batch_normalization_3 (Batc  (None, 512)               2048
hNormalization)

dropout_3 (Dropout)          (None, 512)               0

dense_1 (Dense)              (None, 2)                 1026

=================================================================
Total params: 19,037,634
Trainable params: 19,036,162
Non-trainable params: 1,472
_________________________________________________________________
```

model.summary() 함수를 통해 확인한 모델의 상세 구조이다.
약 1천 8백만개의 파라미터를 학습하도록 설계하였다.
입력층의 Shape의 값 148, 148은 개, 고양이 사진의 가로 세로 크기를 나타낸다.
맨 아래 출력층에서의 shape값은 강아지, 고양이의 두 개 케이스를 출력하기 위해 2로 설정하였다.

```
[ ]  from tensorflow.keras import optimizers
     #RMSprop은 옵티마이저의 한 종류로 일반적으로 순환 신경망(RNN)에 많이 사용됨
     #categorical_crossentropy는 label이 원-핫 인코딩 된 형태 즉 label이 class를 나타내는 one-hot vector를 값으로 가질 때 사용됨
     model.compile(optimizer=optimizers.RMSprop(lr=0.001),
                   loss='categorical_crossentropy',
                   metrics=['accuracy'])
```

```
/usr/local/lib/python3.8/dist-packages/keras/optimizers/optimizer_v2/rmsprop.py:135: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(RMSprop, self).__init__(name, **kwargs)
```

모델을 컴파일할 때는 옵티마이저로 RMSprop를 사용하였다.
RMSProp를 통해 기울기를 단순 누적하지 않고 지수 가중 이동 평균을 사용하여 최신 기울기들이 더 크게 반영되도록 하였다.

Loss에 categorical_crossentropy를 사용한 이유는 결과값이 원-핫 인코딩 된 상태, 즉 label이 class를 나타내는 one-hot vector 값을 가지기 때문이다.

2.5 콜백, Early Stopping, Learning Rate 조정 정의

```
[ ]  from keras.callbacks import EarlyStopping, ReduceLROnPlateau
```

```
[ ]  earlystop = EarlyStopping(patience=10) #학습하는 동안 loss값이 줄어들지 않으면 stop시켜줌
```

모델이 과적합 되는 것을 막기 위해 earlystopping 시켜 주었다.

```
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy',
                                            patience=2,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)
```

monitor은 ReduceLROnPlateau의 기준이 되는 값으로 'val_loss'를 사용해 val_loss가 더 이상 감소되지 않을 경우 ReduceLROnPlateau을 적용한다.
patience은 Training이 진행됨에도 더이상 monitor되는 값의 개선이 없을 경우, 최적의 monitor 값을 기준으로 몇 번의 epoch을 진행하고, learning rate를 조절할 지의 값을 나타낸다.
verbose가 1일 경우 EarlyStopping이 적용될 때, 화면에 적용되었다고 나타나게 한다.
factor은 Learning rate를 얼마나 감소시킬 지 정하는 인자값으로 현재 lr이 0.001이고 factor가 0.5이므로, 콜백함수가 실행된다면 그 다음 lr은 0.0005가 된다.
min_lr은 Learning rate의 하한선을 지정하여 lr이 0.00001 밑으로 내려가지 않게 지정한다.

```
[ ]  callbacks = [earlystop, learning_rate_reduction]
```

```
df["category"] = df["category"].replace({0: 'cat', 1: 'dog'})
```

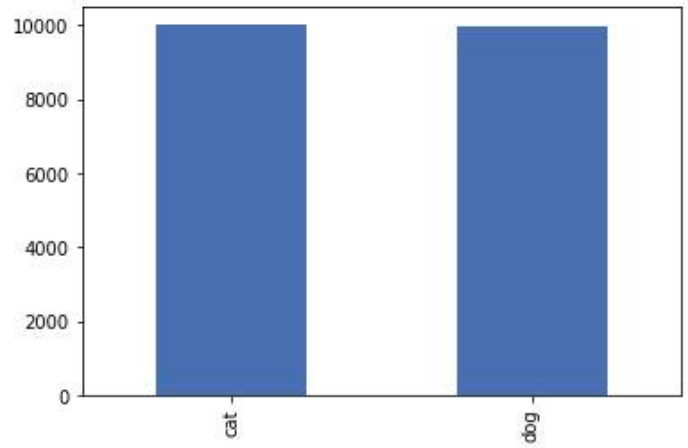
```
[ ]  train_df, validate_df = train_test_split(df, test_size=0.20, random_state=42)
     train_df = train_df.reset_index(drop=True)
     validate_df = validate_df.reset_index(drop=True)
```

앞서 정의한 두 콜백 클래스를 callbacks 변수에 할당한 다음, 추후 one-hot 인코딩으로 카테고리를 변환시키기 위해 기존의 파일 이름을 기준으로 강아지와 고양이를 저장해 놓은 dataframe에 대해서 category를 강아지의 경우 1, 고양이의 경우 0으로 변경해주었다.
이어서 train,validate를 정의하여 전체 train 데이터 중 20%를 쪼개어 validate_df에 넣어줌으로서 검증을 위한 데이터 셋을 준비하였다.
여기서 random_state는 호출 할때마다 동일한 학습/테스트용 데이터 세트를 생성해주기 위한 난수로 임의 값인 42를 할당하였다.

2.6 학습셋과 검증셋에 대한 데이터 분포 확인

```
[ ]  train_df['category'].value_counts().plot.bar()
     print(train_df['category'].value_counts())
     #확인
```

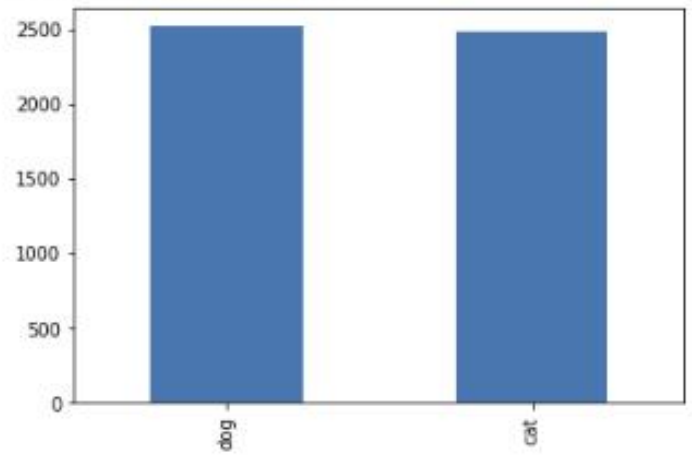
```
cat    10012
dog     9969
Name: category, dtype: int64
```



Train set의 내부를 확인한 결과 고양이가 10012개 강아지가 9969개 존재함을 알 수 있다.

```
validate_df['category'].value_counts().plot.bar()
print(validate_df['category'].value_counts())
```

```
dog    2519
cat    2477
Name: category, dtype: int64
```



앞서 잘라낸 20% validate set의 내부를 확인한 결과 강아지와 고양이의 분포가 잘 되어 있음을 알 수 있다.

```
[ ]  total_train = train_df.shape[0]
     total_validate = validate_df.shape[0]
     batch_size=15
```

이제 학습 데이터와 검증 데이터를 확인하기 위해 batch_size를 15로 지정해 15개씩 샘플로 가중치를 계산하였다.
배치사이즈가 클수록 많은 데이터를 저장해두어야 하므로 용량이 커야 한다.
반면, 배치사이즈가 작으면 학습은 촘촘하게 되겠지만 계속 레이블과 비교하고, 가중치를 업데이트하는 과정을 거치면서 시간이 오래 걸린다는 장단점이 있다. 따라서 적당한 값으로 15를 지정해주었다.

2.7 데이터 전처리와 데이터 증식

```
#학습 데이터 뻥튀기 하기
train_datagen = ImageDataGenerator(
    rotation_range=15,      #rotation_range는 랜덤하게 사진을 회전시킬 각도 범위
    rescale=1./255,         #이미지를 1/255로 스케일을 조정
    shear_range=0.1,        #shear_range는 랜덤하게 전단 변환을 적용할 각도 범위
    zoom_range=0.2,         #zoom_range는 랜덤하게 사진을 확대할 범위
    horizontal_flip=True,   #horizontal_flip은 랜덤하게 이미지를 수평으로 뒤집는다./수평 대칭을 가정할 수 있을 때 사용
    width_shift_range=0.1,  #width_shift_range와
    height_shift_range=0.1  #height_shift_range는 사진을 수평과 수직으로 랜덤하게 평행 이동시킬 범위
)

train_generator = train_datagen.flow_from_dataframe(
    train_df,
    "/content/datasets/train",
    x_col='filename',                   # filename를 열이름
    y_col='category',                   # category를 열이름
    target_size=IMAGE_SIZE,             #이미지 사이즈
    class_mode='categorical',           #y값 변화방법
    batch_size=batch_size               #배치사이즈
)
```

```
Found 19981 validated image filenames belonging to 2 classes.
```

```
validation_datagen = ImageDataGenerator(rescale=1./255)         #이미지를 1/255로 스케일을 조정
validation_generator = validation_datagen.flow_from_dataframe(
    validate_df,
    "/content/datasets/train",
    x_col='filename',
    y_col='category',
    target_size=IMAGE_SIZE,
    class_mode='categorical',
    batch_size=batch_size
)
#validation이미지도 마찬가지로 작업을 해줍니다.
#4996개의 validation이미지가 2classes로 됨
```

```
Found 4996 validated image filenames belonging to 2 classes.
```

학습과 검증에 사용할 데이터는 네트워크에 주입되기 전에 부동 소수 타입의 텐서로 적절하게 전처리 되어 있어야 한다. 프로젝트에 사용할 데이터가 JPEG 파일로 되어 있으므로 이를 전처리 하기 위해 다음과 같은 과정이 필요하다. 먼저 네트워크에 주입하기 위해 사진 파일을 읽어와 JPEG 콘텐츠를 RGB 픽셀 값으로 디코딩한 다음, 부동 소수 타입의 텐서로 변환하여 작은 입력값을 선호하는 신경망의 특성에 맞추어 픽셀 값(0에서 255 사이)의 스케일을 [0, 1] 사이로 조정해야 한다. 이러한 이미지 전처리를 자동으로 처리하기 위해 디스크에 있는 이미지 파일을 전처리 된 배치 텐서로 자동으로 바꾸어주는 ImageDataGenerator 클래스를 사용하여 파이썬 제너레이터를 만들어 주었다.

또한 이전 A1 모델에서 발견되었던 과대 적합 문제를 해결하기 위해 데이터 증식을 사용하였다. 과대적합은 학습할 샘플이 너무 적어 새로운 데이터에 일반화할 수 있는 모델을 훈련시킬 수 없기 때문에 발생하는 문제이다. 이를 해결하기 위해서는 드롭아웃이나 가중치 감소(L2 규제)와 같은 여러 가지 기법들을 사용할 수 있다. 이미 앞서 모델을 구성하면서 드롭아웃과 배치 정규화를 사용하였기 때문에 우리는 과대적합 문제를 해결할 수 있는 또 다른 방법을 적용하고자 했다.

학습 샘플 부족 문제를 근본적으로 해결하려면 부족한 샘플 수를 늘려야 한다. 즉, 더욱 많은 데이터가 주어지게 하여 데이터 분포의 가능한 측면을 모델에게 학습시켜야 한다. 따라서 우리는 기존의 훈련 샘플로부터 더 많은 훈련 데이터를 생성하는 데이터 증식을 사용하여 샘플을 늘려보았다. 이를 구현하기 위해 전처리 과정과 더불어 ImageDataGenerator가 읽은 이미지에 이미지 회전, 줌, 상하/좌우 반이라는 여러 종류의 랜덤 변환을 적용하도록 설정하였다.

여기서 rotation_range는 랜덤하게 사진을 회전시킬 각도 범위이다. -rotation_range ~ +rotation_range 범위를 가진다. rescale은 데이터 전처리 과정의 일부로서 픽셀값 조절에 사용하였다. width_shift_range와 height_shift_range는 전체 넓이와 높이에 대한 비율로 사진을 수평과 수직으로 랜덤하게 평행 이동시킬 범위를 의미한다. 정수가 입력되면 (-width_shift_range ~ + width_shift_range)범위, 실수가 입력되면 [-width_shift_range ~ width_shift_range) 범위를 가진다. shear_range는 랜덤하게 전단 변환을 적용할 각도 범위이다. rotation_range로 회전할 때 y축 방향으로 각도를 증가시킨다. zoom_range는 랜덤하게 사진을 확대할 범위이다. 1-zoom_range ~ 1+zoom_range범위를 가진다. horizontal_flip은 랜덤하게 이미지를 수평으로 뒤집는다.

```
#위에 데이터 부풀리기가 잘 되었는지 확인합니다.
example_df = train_df.sample(n=1).reset_index(drop=True)
example_generator = train_datagen.flow_from_dataframe(
    example_df,
    "/content/datasets/train",
    x_col='filename',
    y_col='category',
    target_size=IMAGE_SIZE,
    class_mode='categorical'
)
```

```
Found 1 validated image filenames belonging to 1 classes.
```

앞서 적용한 변환이 잘 적용되는지 확인하기 위해 증식을 수행하였다.

```
#작업된 데이터 확인
plt.figure(figsize=(12, 12))
for i in range(0, 15):
  plt.subplot(5, 3, i+1)
  for X_batch, Y_batch in example_generator:
      image = X_batch[0]
      plt.imshow(image)
      break
plt.tight_layout()
plt.show()
```

증식한 이미지를 화면에 나타낸 결과 동일한 이미지에 대해 좌우 반전, 줌과 같은 변환 과정이 성공적으로 수행되었음을 알 수 있다.

2.8 학습

```
#학습시작
#46번째 에포크에서 콜백 됩니다.
epochs = 3 if FAST_RUN else 50
history = model.fit_generator(
    train_generator,
    epochs = epochs,
    validation_data=validation_generator,
    validation_steps=total_validate//batch_size,
    steps_per_epoch=total_train//batch_size,
    callbacks=callbacks
)
```

```
Epoch 35/50
1332/1332 [==============================] - 166s 124ms/step - loss: 0.2426 - accuracy: 0.8998 - val_loss: 0.1867 - val_accuracy: 0.9303 - lr: 1.0000e-05
Epoch 36/50
1332/1332 [==============================] - 168s 126ms/step - loss: 0.2530 - accuracy: 0.8953 - val_loss: 0.1813 - val_accuracy: 0.9343 - lr: 1.0000e-05
Epoch 37/50
1332/1332 [==============================] - 171s 128ms/step - loss: 0.2492 - accuracy: 0.8994 - val_loss: 0.1837 - val_accuracy: 0.9317 - lr: 1.0000e-05
Epoch 38/50
1332/1332 [==============================] - 171s 128ms/step - loss: 0.2462 - accuracy: 0.8979 - val_loss: 0.1869 - val_accuracy: 0.9311 - lr: 1.0000e-05
Epoch 39/50
1332/1332 [==============================] - 180s 135ms/step - loss: 0.2435 - accuracy: 0.8998 - val_loss: 0.1857 - val_accuracy: 0.9305 - lr: 1.0000e-05
Epoch 40/50
1332/1332 [==============================] - 175s 131ms/step - loss: 0.2541 - accuracy: 0.8950 - val_loss: 0.1865 - val_accuracy: 0.9299 - lr: 1.0000e-05
Epoch 41/50
1332/1332 [==============================] - 170s 128ms/step - loss: 0.2458 - accuracy: 0.8969 - val_loss: 0.1861 - val_accuracy: 0.9315 - lr: 1.0000e-05
Epoch 42/50
1332/1332 [==============================] - 167s 125ms/step - loss: 0.2520 - accuracy: 0.8978 - val_loss: 0.1858 - val_accuracy: 0.9309 - lr: 1.0000e-05
Epoch 43/50
1332/1332 [==============================] - 170s 127ms/step - loss: 0.2461 - accuracy: 0.8998 - val_loss: 0.1829 - val_accuracy: 0.9327 - lr: 1.0000e-05
Epoch 44/50
1332/1332 [==============================] - 169s 127ms/step - loss: 0.2520 - accuracy: 0.8956 - val_loss: 0.1899 - val_accuracy: 0.9301 - lr: 1.0000e-05
Epoch 45/50
1332/1332 [==============================] - 169s 127ms/step - loss: 0.2485 - accuracy: 0.8964 - val_loss: 0.1843 - val_accuracy: 0.9331 - lr: 1.0000e-05
Epoch 46/50
1332/1332 [==============================] - 176s 132ms/step - loss: 0.2430 - accuracy: 0.8993 - val_loss: 0.1846 - val_accuracy: 0.9323 - lr: 1.0000e-05
```

지금까지 구축한 모델을 가지고 학습을 시작하였다.
총 50번의 epoch를 목표로 진행하다가 31번째 Epoch에서 ReduceLROnPlateau콜백이 일어나고, 최종적으로 46번째 epoch에서 학습을 멈추게 된다.

```
model.save_weights("A2model.h5")
#모델 저장
```

학습 결과를 자세히 살펴보기 전 추후 Competition에서 사용하기 위해 학습한 모델의 가중치를 저장하였다.

3. 구현 결과

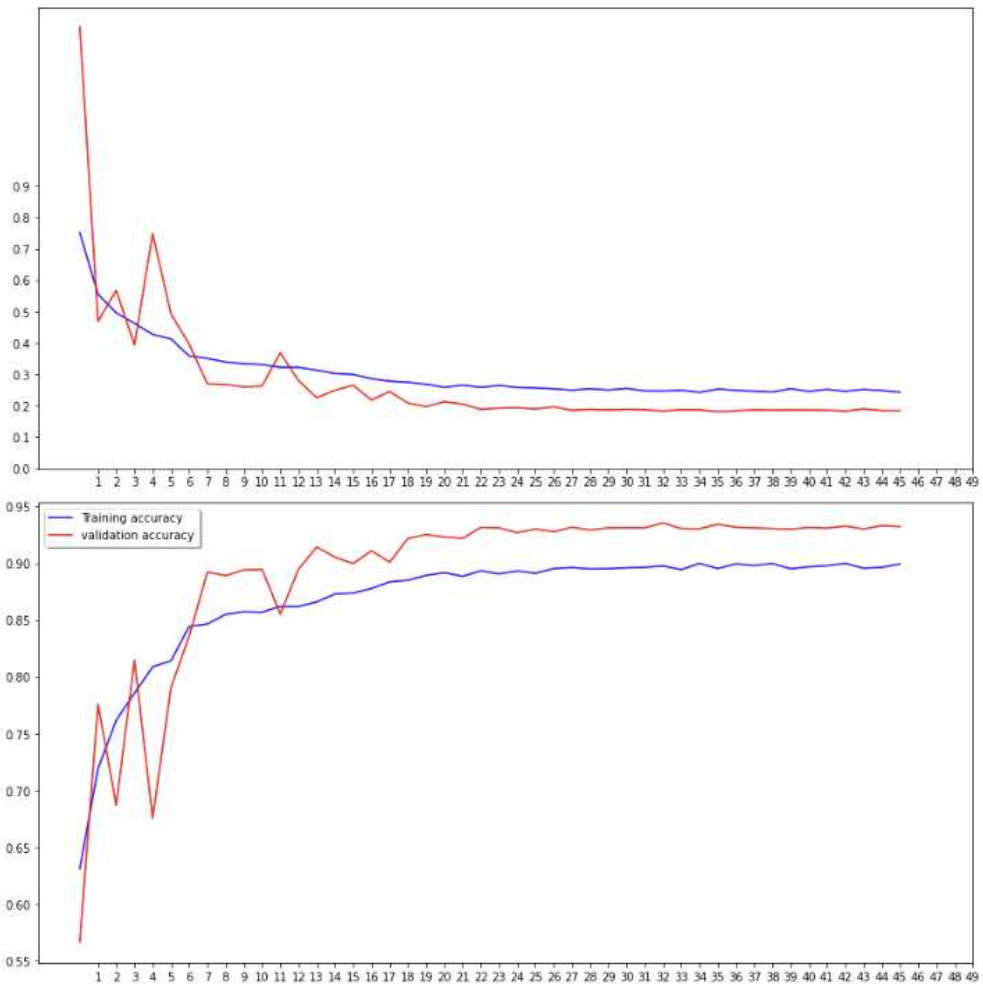
3.1 결과 확인

그동안 학습시킨 내용을 확인하기 위해 train loss, validation loss와 train accuracy, validation accuracy 그래프를 화면에 출력시킨다.

```
#그래프 확인
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 12))
ax1.plot(history.history['loss'], color='b', label="Training loss")
ax1.plot(history.history['val_loss'], color='r', label="validation loss")
ax1.set_xticks(np.arange(1, epochs, 1))
ax1.set_yticks(np.arange(0, 1, 0.1))

ax2.plot(history.history['accuracy'], color='b', label="Training accuracy")
ax2.plot(history.history['val_accuracy'], color='r', label="validation accuracy")
ax2.set_xticks(np.arange(1, epochs, 1))

legend = plt.legend(loc='best', shadow=True)
plt.tight_layout()
plt.show()
```



그래프에서 파란색은 학습셋에 대한 손실값과 정확도를 나타내고, 빨간색은 검증셋에 대한 손실값과 정확도를 나타낸다. 손실값을 나타내는 첫 번째 그래프를 살펴보면 처음 학습 시점에서부터는 오답률이 높다가, 학습이 지속될수록 점차 낮아짐을 볼 수 있다. 그에 따라서 검증셋에 대한 손실도 점차 낮아지고 있다

두 번째로 정확도를 나타내는 그래프를 살펴보면 처음 학습 시점에서는 정확도가 낮지만 점차 epoch가 늘어남에 따라

증가함을 알 수 있다. 학습 초기에 검증 그래프가 급격히 높아지거나 낮아지는 경우가 존재하기는 하지만 두 train과 validation에 대한 결괏값에 큰 차이가 있지 않은 것으로 보아 A1에서 발생했던 과대적합 문제는 해결된 것으로 보인다. 또한 학습 역시 원만하게 잘 이뤄졌음을 확인할 수 있다.

```
#결과확인
history_dict = history.history
print("final train accuracy : {:.4f}".format(history_dict['accuracy'][-1]))
print("final val accuracy : {:.4f}".format(history_dict['val_accuracy'][-1]))
```

```
final train accuracy : 0.8993
final val accuracy : 0.9323
```

최종 학습 정확도는 0.8993, 검증 정확도는 0.9323을 달성하였다.
이제 학습된 모델을 test 데이터에 적용하여 최종적인 평가를 진행하고자 한다.

3.2 테스트

```
#test데이터 불러오기
test_filenames = os.listdir("/content/datasets/test1/")
test_df=pd.DataFrame({
    'filename': test_filenames
})

nb_samples = test_df.shape[0]
```

```
#test데이터 사이즈 맞추기
test_gen=ImageDataGenerator(rescale=1./255)
test_generator = test_gen.flow_from_dataframe(
    test_df,
    "/content/datasets/test1/",
    x_col='filename',
    y_col=None,
    class_mode=None,
    target_size=IMAGE_SIZE,
    batch_size=batch_size,
    shuffle=False
)
```

```
Found 12500 validated image filenames.
```

```
predict = model.predict_generator(test_generator, steps=np.ceil(nb_samples/batch_size))
#아까 학습된 모델에 테스트 데이터 넣어보기
```

```
<ipython-input-38-52619fd72ccc>:1: UserWarning: `Model.predict_generator` is deprecated and will be removed in a future version. Please use `Model.predict`, which supports generators.
  predict = model.predict_generator(test_generator, steps=np.ceil(nb_samples/batch_size))
```

test1 디렉토리에 있는 강아지와 고양이 사진을 사용하여 평가를 진행한다.
기존의 검증 셋과 마찬가지로 테스트를 위한 데이터를 준비한다.
이어서 이전에 학습한 모델을 생성한 테스트 셋에 적용한다.

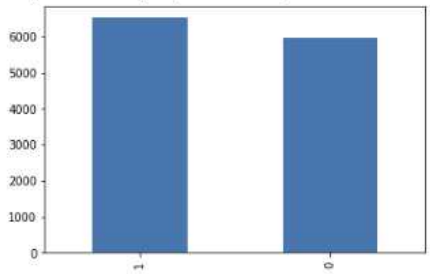
```
test_df['category']=np.argmax(predict, axis=-1)
#예측 결과가 고양이10% 강아지90% 이런식으로 나오기 때문에 개와 고양이일 확률중 보다 큰값에 해당하는 레이블을 선택해서 값을 치환합니다.
```

```
label_map = dict((v,k) for k,v in train_generator.class_indices.items())
test_df['category'] = test_df['category'].replace(label_map)
```

```
test_df['category']=test_df['category'].replace({'dog':1, 'cat':0})
```

```
test_df['category'].value_counts().plot.bar()
#개와 고양이를 어느정도 비율로 예측했는지 확인합니다.
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb29c5dff10>
```



예측 결과에는 강아지, 고양이 클래스에 대한 확률이 담겨있다. 편리하게 정확도를 검증하기 위해 이러한 강아지와 고양이일 확률 중 더 큰 값에 해당하는 레이블을 선택하게끔 값을 치환하였다. 즉, 만약 강아지일 확률이 0.83이고 고양이일 확률이 0.37이라면 더 큰 값을 가지는 강아지를 뜻하는 dog label에 해당 데이터를 할당하였다.
이를 위해 기존에 dog, cat으로 분류하였던 category 데이터를 다시 강아지는 1, 고양이는 0으로 변경해주었다.
그리고 정확한 검증을 위해 어느 정도 비율로 각 클래스를 예측하였는지를 시각화하여 살펴보았다.

```
#예측한 결과를 눈으로 확인
import keras.utils as image

sample_test=test_df.head(18)
sample_test.head()
plt.figure(figsize=(12, 24))
for index, row in sample_test.iterrows():
  filename=row['filename']
  category=row['category']
  img=image.load_img("/content/datasets/test1/"+filename, target_size=IMAGE_SIZE)
  plt.subplot(6, 3, index+1)
  plt.imshow(img)
  plt.xlabel(filename+'('+"{}".format(category)+')')
plt.tight_layout()
plt.show()
```

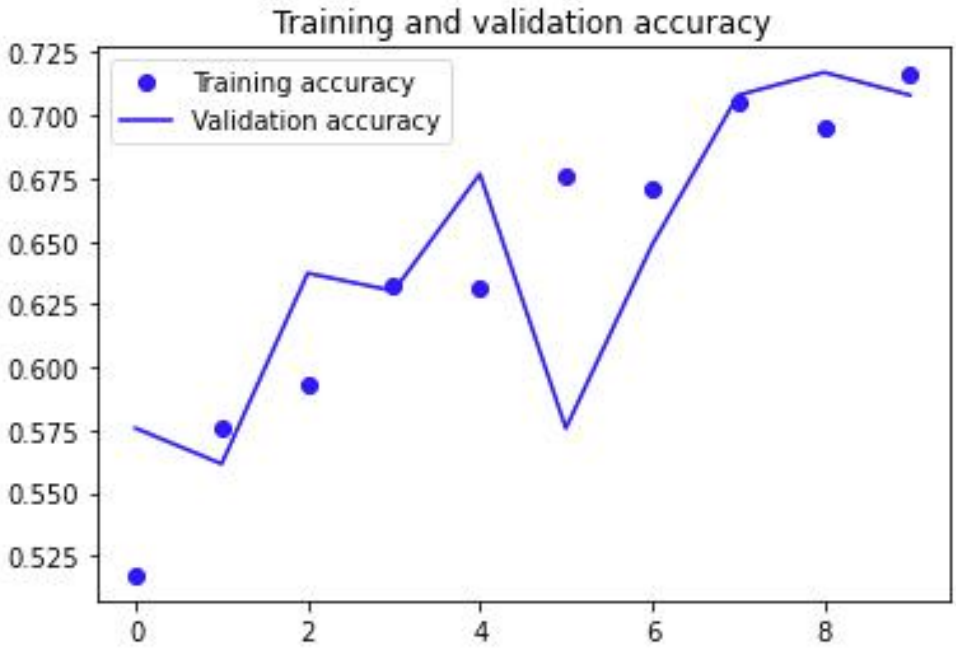
테스트 데이터를 각각 성공적으로 분류하였음을 확인할 수 있다.

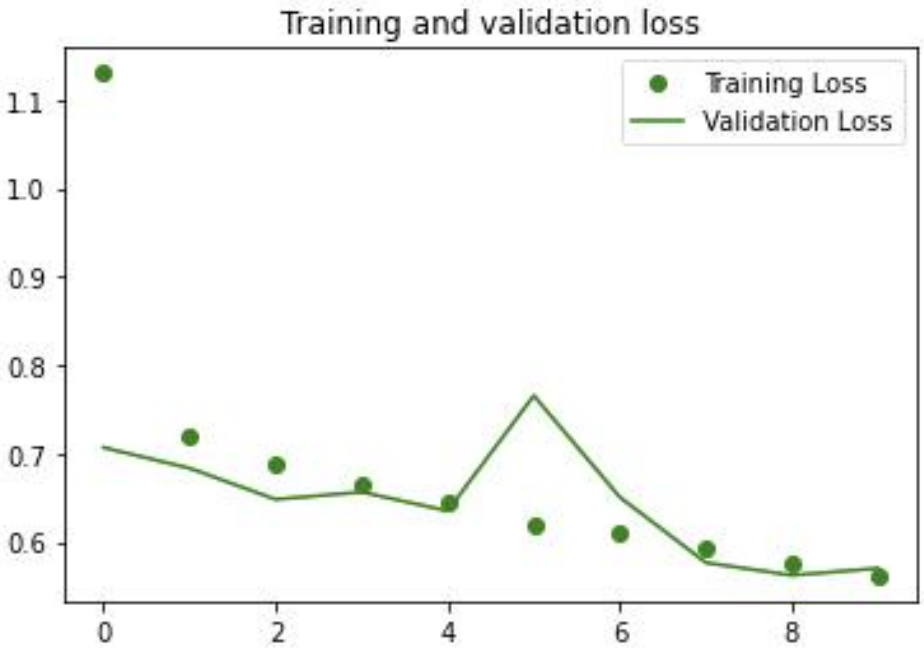
Ⅲ. 결과 분석

지금까지 구현한 A1 모델과 A2 모델의 결과를 바탕으로 두 모델을 비교한 내용은 다음과 같다.

```
Epoch 10/10
133/133 [==============================] - 316s 2s/step - loss: 0.5617 - accuracy: 0.7168 - val_loss: 0.5703 - val_accuracy: 0.7081
```

먼저 A1모델의 경우 최종 손실값이 0.56, 학습 정확도가 0.71, 검증 손실값이 0.57, 검증 정확도가 0.70을 기록했다. 전체적인 값의 추이는 그래프로 확인해볼 수 있다.





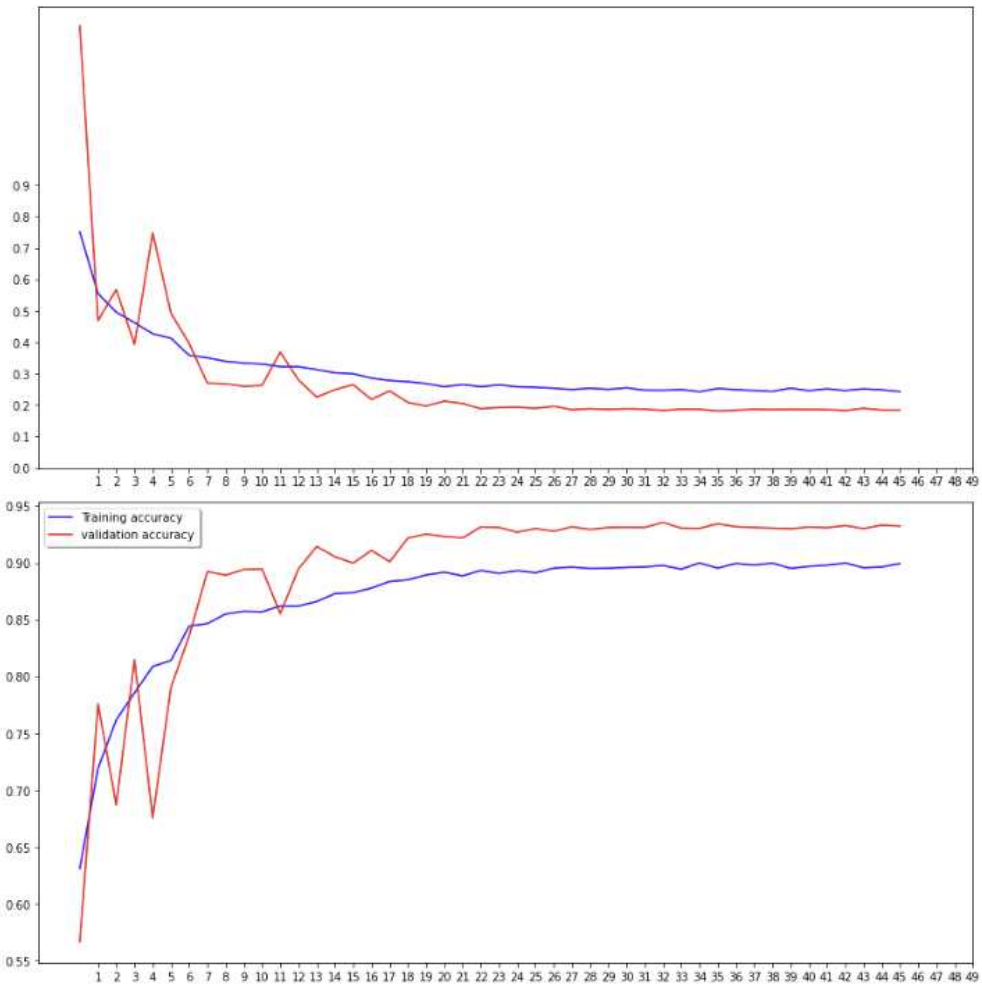
그래프를 통해 전체 epoch에 따른 손실값과 정확도를 확인해보았을 때, 훈련 정확도는 시간이 지남에 따라 대부분 선형적으로 증가하여 상향하는 추세에 있다. 반대로 검증 정확도는 급격히 감소하였다가 반등하기도 하는 등 불안정한 추이를 보이면서 대체로 훈련 정확도보다 낮은 현상을 보였다.

구현 중 여러 번 epoch를 변경하여 실행해 보았는데 epoch 수가 늘어날수록 훈련 정확도보다 검증 정확도가 현저히 낮아지는 과대적합 현상이 두드러지게 발견되었다. 우리는 이러한 현상이 일어난 가장 큰 원인은 비교적 적은 훈련 샘

플의 수(랜덤하게 선택한 3,000개의 데이터) 때문이라고 분석하였다.

이를 바탕으로 A2 설계시에는 이러한 과대적합 문제 해결을 최우선 과제로 두고 작업했다. 각 에포크당 A2 모델의 손실값과 정확도 추이는 다음과 같다.

```
Epoch 35/50
1332/1332 [==============================] - 166s 124ms/step - loss: 0.2426 - accuracy: 0.8998 - val_loss: 0.1867 - val_accuracy: 0.9303 - lr: 1.0000e-05
Epoch 36/50
1332/1332 [==============================] - 168s 126ms/step - loss: 0.2530 - accuracy: 0.8953 - val_loss: 0.1813 - val_accuracy: 0.9343 - lr: 1.0000e-05
Epoch 37/50
1332/1332 [==============================] - 171s 128ms/step - loss: 0.2492 - accuracy: 0.8994 - val_loss: 0.1837 - val_accuracy: 0.9317 - lr: 1.0000e-05
Epoch 38/50
1332/1332 [==============================] - 171s 128ms/step - loss: 0.2462 - accuracy: 0.8979 - val_loss: 0.1869 - val_accuracy: 0.9311 - lr: 1.0000e-05
Epoch 39/50
1332/1332 [==============================] - 180s 135ms/step - loss: 0.2435 - accuracy: 0.8998 - val_loss: 0.1857 - val_accuracy: 0.9305 - lr: 1.0000e-05
Epoch 40/50
1332/1332 [==============================] - 175s 131ms/step - loss: 0.2541 - accuracy: 0.8950 - val_loss: 0.1865 - val_accuracy: 0.9299 - lr: 1.0000e-05
Epoch 41/50
1332/1332 [==============================] - 170s 128ms/step - loss: 0.2458 - accuracy: 0.8969 - val_loss: 0.1861 - val_accuracy: 0.9315 - lr: 1.0000e-05
Epoch 42/50
1332/1332 [==============================] - 167s 125ms/step - loss: 0.2520 - accuracy: 0.8978 - val_loss: 0.1858 - val_accuracy: 0.9309 - lr: 1.0000e-05
Epoch 43/50
1332/1332 [==============================] - 170s 127ms/step - loss: 0.2461 - accuracy: 0.8998 - val_loss: 0.1829 - val_accuracy: 0.9327 - lr: 1.0000e-05
Epoch 44/50
1332/1332 [==============================] - 169s 127ms/step - loss: 0.2520 - accuracy: 0.8956 - val_loss: 0.1899 - val_accuracy: 0.9301 - lr: 1.0000e-05
Epoch 45/50
1332/1332 [==============================] - 169s 127ms/step - loss: 0.2485 - accuracy: 0.8964 - val_loss: 0.1843 - val_accuracy: 0.9331 - lr: 1.0000e-05
Epoch 46/50
1332/1332 [==============================] - 176s 132ms/step - loss: 0.2430 - accuracy: 0.8993 - val_loss: 0.1846 - val_accuracy: 0.9323 - lr: 1.0000e-05
```



A2 모델에서는 학습 데이터에서 불필요한 데이터를 제거하고 과대적합을 방지하기 위해 부족한 학습 데이터 문제를 해결하는 데이터 증식을 사용하였다. 또한 배치 정규화, Dropout을 사용하여 과적합을 막기 위해 일부 가중치만 사용하는 정규화 작업을 하였다. 그 결과 두 데이터 세트에 대한 손실값이 고르게 감소하였으며 훈련 곡선이 검증 곡선에 가깝게 따라가는 모습을 확인 할 수 있었다. 정확도 추이 역시 훈련 셋보다 검증 셋에서의 정확도가 더 높게 관찰되었다. 이를 통해 과대적합 문제가 해소되었으며, 기존 A1 모델 대비 23% 향상된 93%의 검증 정확도를 달성하였다.

Ⅳ. 참고문헌

[1] 코딩크루, 딥러닝으로 개와 고양이를 분류하는 모델을 만들어 보자.,
https://codingcrews.github.io/2019/01/17/cat-dog/

[2] UYSIM, Keras CNN Dog or Cat Classification,
https://www.kaggle.com/code/uysimty/keras-cnn-dog-or-cat-classification

[3] OMKAR C GODE, Cats vs Dogs - Image classification Pytorch CNN,
https://www.kaggle.com/code/omkarcgode/cats-vs-dogs-image-classification-pytorch-cnn

[4] PRIYANSHU SAHU, dogs_vs_cats, https://www.kaggle.com/code/priyanshusahu23/dogs-vs-cats

[5] 동화책, 소규모 데이터셋으로 심층신경망 학습하기 (feat.Keras) [3탄], https://dacon.io/codeshare/4445

[6] 딥러닝, 딥러닝으로 고양이와 개를 구분해보자., https://provia.tistory.com/79

[7] sseunghyuns, Dogs vs. Cats(Classification),
https://sseunghyuns.github.io/classification/2021/01/01/dogvscat/

[8] 별준, [Tensorflow][Kaggle] Cats vs. Dogs Classification(수정 : 2020-12-07),
https://junstar92.tistory.com/119

Ⅴ. 연구일지

<table>
<tr><th colspan="4">연 구 일 지</th></tr>
<tr><td>팀원</td><td colspan="3">김다현, 양희훈, 이경수, 이현중</td></tr>
<tr><td>주차</td><td>1 주차</td><td>활동기간</td><td>2022.11.15. ~ 11.21</td></tr>
<tr><td>연구 목표</td><td colspan="3">1. 역할 분담과 연구 계획 논의
2. 참고 자료 조사와 분석</td></tr>
<tr><td rowspan="4">연구 결과</td><td>회차</td><td>일시</td><td>기록</td></tr>
<tr><td>1</td><td>2022.11.15</td><td>(역할 분담 및 연구 계획 논의)</td></tr>
<tr><td>2</td><td>2022.11.15~11.21</td><td>(서로 공부해온 캐글 및 자료 공유 , 자료 분석)</td></tr>
<tr><td colspan="3">1 + α</td></tr>
<tr><th colspan="4">세부 활동 내역</th></tr>
<tr><th colspan="4">1주차</th></tr>
<tr><td>활동 주제</td><td colspan="3">역할 분담</td></tr>
<tr><td>활동 내용</td><td colspan="3">-팀장을 선출함.
-일정 논의 결과 이번 주말까지 시간을 두고 서로 자유롭게 연구 주제에 대해 파악하고 공부해 보는 시간을 가지도록 함.
-서로 준비해온 자료를 공유하고 가볍게 자료분석을 함.
-다음 수업까지 자유롭게 프로젝트내용 분석해오기로 함.
-전반적인 A1코드 생성 및 주석</td></tr>
<tr><td rowspan="4">팀원 역할</td><td colspan="3">김다현 : 팀원</td></tr>
<tr><td colspan="3">양희훈 : 팀장</td></tr>
<tr><td colspan="3">이경수 : 팀원</td></tr>
<tr><td colspan="3">이현중 : 팀원</td></tr>
</table>

<table>
<tr><th colspan="4">연 구 일 지</th></tr>
<tr><td>팀원</td><td colspan="3">김다현, 양희훈, 이경수, 이현중</td></tr>
<tr><td>주차</td><td>2 주차</td><td>활동기간</td><td>2022.11.22. ~ 11.28</td></tr>
<tr><td>연구 목표</td><td colspan="3">1. 참고 자료 조사를 정리 및 테스트
2. cats and dogs 데이터 학습
3. 중간 발표 준비</td></tr>
<tr><td rowspan="7">연구 결과</td><td>회차</td><td>일시</td><td>기록</td></tr>
<tr><td>1</td><td>2022.11.22</td><td>(참고자료를 대화하면서 공유하고 가볍게 테스트를 함)</td></tr>
<tr><td>2</td><td>2022.11.25</td><td>(cats and dogs데이터를 학습 시켜보고 중간발표 준비)</td></tr>
<tr><td>3</td><td>2022.11.26</td><td>(데이터 학습 및 중간발표 준비)</td></tr>
<tr><td>4</td><td>2022.11.27</td><td>(데이터 학습 및 중간발표 준비)</td></tr>
<tr><td>5</td><td>2022.11.28</td><td>(데이터 학습 및 중간발표 준비)</td></tr>
<tr><td>활동시간 합계</td><td colspan="2">8 + α</td></tr>
<tr><th colspan="4">세부 활동 내역</th></tr>
<tr><th colspan="4">2주차</th></tr>
<tr><td>활동 주제</td><td colspan="3">데이터 학습 및 중간발표 준비</td></tr>
<tr><td>활동 내용</td><td colspan="3">- 준비해왔던 cats and dogs자료들을 서로 공유하고 코드를 직접 입력하면서 테스트
- 데이터 전처리 방법 및 CNN기술에 대해 학습을 함
- 의견을 합쳐서 선호하는 방법으로 프로젝트 진행을 함
- 중간 발표를 위해 역할을 분담함.
- A1모델 구현 및 설계</td></tr>
<tr><td rowspan="4">팀원 역할</td><td colspan="3">김다현 : 자료정리 및 모델 설계</td></tr>
<tr><td colspan="3">양희훈 : ppt제작 , 자료조사</td></tr>
<tr><td colspan="3">이경수 : 자료분석 , 대본 및 발표</td></tr>
<tr><td colspan="3">이현중 : 자료분석 및 정리</td></tr>
</table>

<table>
<tr><th colspan="4">연 구 일 지</th></tr>
<tr><td>팀원</td><td colspan="3">김다현, 양희훈, 이경수, 이현중</td></tr>
<tr><td>주차</td><td>3 주차</td><td>활동기간</td><td>2022.11.29. ~ 12.05</td></tr>
<tr><td>연구 목표</td><td colspan="3">1. A1모델 구현
2. A2 모델 구현
3. 최종 발표 준비</td></tr>
<tr><td rowspan="5">연구 결과</td><td>회차</td><td>일시</td><td>기록</td></tr>
<tr><td>1</td><td>2022.11.29</td><td>(참고자료를 대화하면서 공유하고 가볍게 테스트를 함)</td></tr>
<tr><td>2</td><td>2022.11.30.~12.04</td><td>(최종적으로 A1 , A2 모델 구현을 하였고 오류는 없는지 수정, submisson수행, 최종발표 준비)</td></tr>
<tr><td>3</td><td>2022.12.05</td><td>(최종발표 준비 및 점검)</td></tr>
<tr><td>활동시간 합계</td><td colspan="2">4 + α</td></tr>
<tr><th colspan="4">세부 활동 내역</th></tr>
<tr><th colspan="4">3주차</th></tr>
<tr><td>활동 주제</td><td colspan="3">A1,A2 모델 최종 구현, Competition 코드 작성, 최종발표준비</td></tr>
<tr><td>활동 내용</td><td colspan="3">-A1모델과 세부 평가항목 구현완료
-A2모델과 세부 평가항목 구현완료
-A1 , A2 수정 및 보완
-Competition 코드 작성
-주석 및 ppt제작
-최종발표 준비</td></tr>
<tr><td rowspan="4">팀원 역할</td><td colspan="3">김다현 : 전체적인 A1, A2 모델 설계, 보고서 작성</td></tr>
<tr><td colspan="3">양희훈 : 발표 자료 제작, A1 주석, 보고서 작성</td></tr>
<tr><td colspan="3">이경수 : 주석 및 코드 수정, 보고서 작성</td></tr>
<tr><td colspan="3">이현중 : 전체적인 A1 모델 설계, 주석 및 코드수정, 보고서 작성</td></tr>
</table>

Ⅵ. 팀원 소감

김다현	졸업 전 마지막 팀 프로젝트였던 만큼 무사히 잘 마무리 할 수 있어 기쁘다. 특히 협업을 통해 스스로 부족한 점을 돌아볼 수 있어 좋았다. 이번 경험을 바탕으로 졸업 후 현업에서도 적극적으로 문제를 해결하는 개발자가 될 수 있도록 노력하겠다.
양희훈	팀플은 이번이 처음이었는데 베이스가 없던 상태에서 하다 보니 많은 부분을 팀원에게 기댔던 것 같습니다. 중간고사 전까지 이해가 잘되지 않았었는데 실전으로 직접 경험해 보니 더욱 이해가 잘 됐던 것 같고, 도움이 많이 되진 않았겠지만 믿고 데려가 주신 팀원분들이 감사했고, 혼자보단 여럿이 낫다는 것을 느꼈습니다. 좋은 경험을 하게 해주셔서 감사합니다.
이경수	이번 팀플을 통해 수업에서 배웠던 점을 실제로 활용하는 기회가 되서 공부한 것을 복습하는 기회가 되었고 발표 준비를 하면서 코드에 자세한 부분까지 다 어떻게 왜 필요한지 알수있었습니다. 또한 수업때 배우지 않은 딥러닝 모델의 성능을 향상시키는법 등등을 찾아가며 공부하는 기회가 되서 재미있었습니다. 팀원 분들과 소통하며 서로 물어보고 부족한점을 보안해주며 프로젝트를 진행하며 혼자 작업할 때 보다 팀으로 활동하는것에 중요성을 느끼는 시간이라 좋았습니다.
이현중	처음 배워보는 딥러닝이였던 만큼 어려운 부분이 많았다. 혼자 했다면 포기할 수도 있었겠지만, 팀 프로젝트인 만큼 포기하지 않고 끝까지 최선을 다하여 결과를 낼 수 있었던 것 같다. 이번 경험을 바탕으로 딥러닝에 대해 많은 지식을 쌓을 수 있었고 좋은 기반을 닦을 수 있었던 것 같다.