이미지 증강(Image Augmentation)을 사용하여 CNN 학습

> Overfitting 대응책

```
import warnings
warnings.filterwarnings('ignore')
```

Import Tensorflow

```
import tensorflow

tensorflow.__version__
```

```
'2.5.0'
```

I. Google Drive Mount

- 'dogs_and_cats_small.zip' 디렉토리를 구글드라이브에 업로드

```
from google.colab import drive

drive.mount('/content/drive')
```

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

1) 구글 드라이브 마운트 결과 확인

```
!ls -l '/content/drive/My Drive/Colab Notebooks/datasets/dogs_and_cats_small.zip'
```

```
-rw------- 1 root root 90618980 Mar  4 04:51 '/content/drive/My Drive/Colab Notebooks/datasets/dogs_and_cats_small.zip'
```

2) unzip 'dogs_and_cats_small.zip'

```
!unzip /content/drive/My₩ Drive/Colab₩ Notebooks/datasets/dogs_and_cats_small.zip
```

```
  inflating: validation/dogs/dog.1443.jpg
  inflating: validation/dogs/dog.1444.jpg
  inflating: validation/dogs/dog.1445.jpg
  inflating: validation/dogs/dog.1446.jpg
  inflating: validation/dogs/dog.1447.jpg
  inflating: validation/dogs/dog.1448.jpg
  inflating: validation/dogs/dog.1449.jpg
  inflating: validation/dogs/dog.1450.jpg
  inflating: validation/dogs/dog.1451.jpg
  inflating: validation/dogs/dog.1452.jpg
  inflating: validation/dogs/dog.1453.jpg
  inflating: validation/dogs/dog.1454.jpg
  inflating: validation/dogs/dog.1455.jpg
  inflating: validation/dogs/dog.1456.jpg
  inflating: validation/dogs/dog.1457.jpg
  inflating: validation/dogs/dog.1458.jpg
  inflating: validation/dogs/dog.1459.jpg
  inflating: validation/dogs/dog.1460.jpg
  inflating: validation/dogs/dog.1461.jpg
  inflating: validation/dogs/dog.1462.jpg
  inflating: validation/dogs/dog.1463.jpg
  inflating: validation/dogs/dog.1464.jpg
  inflating: validation/dogs/dog.1465.jpg
  inflating: validation/dogs/dog.1466.jpg
  inflating: validation/dogs/dog.1467.jpg
  inflating: validation/dogs/dog.1468.jpg
  inflating: validation/dogs/dog.1469.jpg
  inflating: validation/dogs/dog.1470.jpg
  inflating: validation/dogs/dog.1471.jpg
  inflating: validation/dogs/dog.1472.jpg
```

```
inflating: validation/dogs/dog.1473.jpg
inflating: validation/dogs/dog.1474.jpg
inflating: validation/dogs/dog.1475.jpg
inflating: validation/dogs/dog.1476.jpg
inflating: validation/dogs/dog.1477.jpg
inflating: validation/dogs/dog.1478.jpg
inflating: validation/dogs/dog.1479.jpg
inflating: validation/dogs/dog.1480.jpg
inflating: validation/dogs/dog.1481.jpg
inflating: validation/dogs/dog.1482.jpg
inflating: validation/dogs/dog.1483.jpg
inflating: validation/dogs/dog.1484.jpg
inflating: validation/dogs/dog.1485.jpg
inflating: validation/dogs/dog.1486.jpg
inflating: validation/dogs/dog.1487.jpg
inflating: validation/dogs/dog.1488.jpg
inflating: validation/dogs/dog.1489.jpg

inflating: validation/dogs/dog.1490.jpg
inflating: validation/dogs/dog.1491.jpg
inflating: validation/dogs/dog.1492.jpg
inflating: validation/dogs/dog.1493.jpg
inflating: validation/dogs/dog.1494.jpg
inflating: validation/dogs/dog.1495.jpg
inflating: validation/dogs/dog.1496.jpg
inflating: validation/dogs/dog.1497.jpg
inflating: validation/dogs/dog.1498.jpg
inflating: validation/dogs/dog.1499.jpg
inflating: validation/dogs/dog.1500.jpg
```

```
!ls -l
```

```
total 20
drwx------ 5 root root 4096 Aug  6 00:16 drive
drwxr-xr-x 1 root root 4096 Jul 16 13:20 sample_data
drwxr-xr-x 4 root root 4096 Aug  6 00:16 test
drwxr-xr-x 4 root root 4096 Aug  6 00:16 train
drwxr-xr-x 4 root root 4096 Aug  6 00:16 validation
```

3) [Optional] Image Augmentation Test

- rotation_range = 40 : 0도에서 40도 사이에서 임의의 각도록 회전
- width_shift_range = 0.2 : 20% 픽셀 내외로 좌우 이동
- height_shift_range = 0.2 : 20% 픽셀 내외로 상하 이동
- shear_range = 0.2 : 0.2 라디안 내외로 시계 반대방향으로 변형
- zoom_range = 0.2 : 80%에서 120% 범위에서 확대/축소
- horizontal_flip = True : 수평방향 뒤집기
- vertical_flip = True : 수직방향 뒤집기
- fill_mode = 'nearest' : 주변 픽셀로 이미지 채우기

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

datagen = ImageDataGenerator(rotation_range = 40,
                             width_shift_range = 0.2,
                             height_shift_range = 0.2,
                             shear_range = 0.2,
                             zoom_range = 0.2,
                             horizontal_flip = True,
                             vertical_flip = True,
                             fill_mode = 'nearest')
```

```
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt
import os

train_cats_dir = train_dir = os.path.join('train', 'cats')
fnames = sorted([os.path.join(train_cats_dir, fname) for fname in os.listdir(train_cats_dir)])

# 테스트 이미지 선택
img_path = fnames[77]

# 이미지 읽고 크기 변경
```

```
img = image.load_img(img_path, target_size=(150, 150))

# (150, 150, 3) 배열 변환
x = image.img_to_array(img)

# (1, 150, 150, 3) 변환
x = x.reshape((1,) + x.shape)

# 랜덤하게 변환된 이미지 배치 생성
i = 0

for batch in datagen.flow(x, batch_size=1):
    plt.figure(i)
    imgplot = plt.imshow(image.array_to_img(batch[0]))
    i += 1
    if i % 4 == 0:
        break

plt.show()
```

II. Data Preprocessing

1) Image_File Directory Setting

- train_dir
- valid_dir
- test_dir

```
train_dir = 'train'
valid_dir = 'validation'
test_dir = 'test'
```

2) ImageDataGenerator( ) & flow_from_directory( )

- Normalization & Augmentation
  - ImageDataGenerator( )
- Resizing & Generator
  - flow_from_directory( )

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

# With Augmentation
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   rotation_range = 40,
                                   width_shift_range = 0.2,
                                   height_shift_range = 0.2,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip=True,
                                   vertical_flip = True,
                                   fill_mode = 'nearest')

# Without Augmentation
valid_datagen = ImageDataGenerator(rescale = 1./255)



# With Augmentation
train_generator = train_datagen.flow_from_directory(
                  train_dir,
                  target_size = (150, 150),
                  batch_size = 20,
                  class_mode = 'binary')

# Without Augmentation
valid_generator = valid_datagen.flow_from_directory(
                  valid_dir,
                  target_size = (150, 150),
                  batch_size = 20,
                  class_mode = 'binary')
```

```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
```

III. CNN Keras Modeling

1) Model Define

- Feature Extraction & Classification
    - Dropout Layer

```
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.Dropout(0.5))
model.add(layers.Dense(512, activation = 'relu'))
model.add(layers.Dense(1, activation = 'sigmoid'))
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dropout (Dropout)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 512)               3211776
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
_________________________________________________________________
```

2) Model Compile

- 모델 학습방법 설정

```
model.compile(loss = 'binary_crossentropy',
              optimizer = 'adam',
              metrics = ['accuracy'])
```

3) Model Fit

- 약 30분

- epochs : 60 -> 100

```
%%time

Hist_dandc = model.fit(train_generator,
                       steps_per_epoch = 100,
                       epochs = 100,
                       validation_data = valid_generator,
                       validation_steps = 50)
```

```
Epoch 73/100
100/100 [==============================] - 17s 170ms/step - loss: 0.4564 - accuracy: 0.7840 - val_loss: 0.5100 - val_accuracy: 0.7590
Epoch 74/100
100/100 [==============================] - 18s 182ms/step - loss: 0.4653 - accuracy: 0.7790 - val_loss: 0.5228 - val_accuracy: 0.7250
Epoch 75/100
100/100 [==============================] - 17s 172ms/step - loss: 0.4448 - accuracy: 0.7960 - val_loss: 0.4692 - val_accuracy: 0.7760
Epoch 76/100
100/100 [==============================] - 17s 169ms/step - loss: 0.4408 - accuracy: 0.7995 - val_loss: 0.4691 - val_accuracy: 0.7830
Epoch 77/100
100/100 [==============================] - 17s 172ms/step - loss: 0.4373 - accuracy: 0.7965 - val_loss: 0.4927 - val_accuracy: 0.7760
Epoch 78/100
100/100 [==============================] - 21s 208ms/step - loss: 0.4382 - accuracy: 0.7890 - val_loss: 0.4604 - val_accuracy: 0.7790
Epoch 79/100
100/100 [==============================] - 17s 172ms/step - loss: 0.4477 - accuracy: 0.7930 - val_loss: 0.4307 - val_accuracy: 0.7960
Epoch 80/100
100/100 [==============================] - 17s 169ms/step - loss: 0.4313 - accuracy: 0.7975 - val_loss: 0.5342 - val_accuracy: 0.7570
Epoch 81/100
100/100 [==============================] - 18s 177ms/step - loss: 0.4383 - accuracy: 0.8015 - val_loss: 0.4838 - val_accuracy: 0.7610
Epoch 82/100
100/100 [==============================] - 18s 177ms/step - loss: 0.4169 - accuracy: 0.8105 - val_loss: 0.4591 - val_accuracy: 0.7820

Epoch 83/100
100/100 [==============================] - 17s 170ms/step - loss: 0.4303 - accuracy: 0.7990 - val_loss: 0.5133 - val_accuracy: 0.7600
Epoch 84/100
100/100 [==============================] - 17s 170ms/step - loss: 0.4235 - accuracy: 0.7990 - val_loss: 0.4605 - val_accuracy: 0.7730
Epoch 85/100
100/100 [==============================] - 18s 180ms/step - loss: 0.4251 - accuracy: 0.7985 - val_loss: 0.4811 - val_accuracy: 0.7940
Epoch 86/100
100/100 [==============================] - 18s 176ms/step - loss: 0.4345 - accuracy: 0.7970 - val_loss: 0.4564 - val_accuracy: 0.7820
Epoch 87/100
100/100 [==============================] - 17s 170ms/step - loss: 0.4213 - accuracy: 0.8120 - val_loss: 0.4546 - val_accuracy: 0.7900
Epoch 88/100
100/100 [==============================] - 17s 170ms/step - loss: 0.4305 - accuracy: 0.8055 - val_loss: 0.4390 - val_accuracy: 0.7890
Epoch 89/100
100/100 [==============================] - 18s 182ms/step - loss: 0.4108 - accuracy: 0.8120 - val_loss: 0.4377 - val_accuracy: 0.7990
Epoch 90/100
100/100 [==============================] - 18s 176ms/step - loss: 0.4005 - accuracy: 0.8190 - val_loss: 0.4575 - val_accuracy: 0.7700
Epoch 91/100
100/100 [==============================] - 18s 181ms/step - loss: 0.4136 - accuracy: 0.8085 - val_loss: 0.5526 - val_accuracy: 0.7660
Epoch 92/100
100/100 [==============================] - 17s 175ms/step - loss: 0.4183 - accuracy: 0.8080 - val_loss: 0.4273 - val_accuracy: 0.8040
Epoch 93/100
100/100 [==============================] - 17s 169ms/step - loss: 0.4008 - accuracy: 0.8040 - val_loss: 0.4683 - val_accuracy: 0.7820
Epoch 94/100
100/100 [==============================] - 17s 174ms/step - loss: 0.4115 - accuracy: 0.8175 - val_loss: 0.4489 - val_accuracy: 0.7930
Epoch 95/100
100/100 [==============================] - 18s 181ms/step - loss: 0.3991 - accuracy: 0.8245 - val_loss: 0.4521 - val_accuracy: 0.7870
Epoch 96/100
100/100 [==============================] - 17s 170ms/step - loss: 0.4029 - accuracy: 0.8170 - val_loss: 0.4968 - val_accuracy: 0.7730
Epoch 97/100
100/100 [==============================] - 17s 169ms/step - loss: 0.4085 - accuracy: 0.8115 - val_loss: 0.5462 - val_accuracy: 0.7640
Epoch 98/100
100/100 [==============================] - 18s 178ms/step - loss: 0.3867 - accuracy: 0.8250 - val_loss: 0.4701 - val_accuracy: 0.7870
Epoch 99/100
100/100 [==============================] - 18s 178ms/step - loss: 0.4017 - accuracy: 0.8160 - val_loss: 0.4401 - val_accuracy: 0.7770
Epoch 100/100
100/100 [==============================] - 17s 169ms/step - loss: 0.3889 - accuracy: 0.8190 - val_loss: 0.4472 - val_accuracy: 0.7860
CPU times: user 32min 48s, sys: 31.1 s, total: 33min 20s
Wall time: 29min 47s
```

4) 학습 결과 시각화

- Loss Visualization

```
import matplotlib.pyplot as plt

epochs = range(1, len(Hist_dandc.history['loss']) + 1)

plt.figure(figsize = (8, 6))
```

```
plt.figure(figsize = (9, 6))
plt.plot(epochs, Hist_dandc.history['loss'])
plt.plot(epochs, Hist_dandc.history['val_loss'])

plt.title('Training & Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(['Training Loss', 'Validation Loss'])
plt.grid()
plt.show()
```

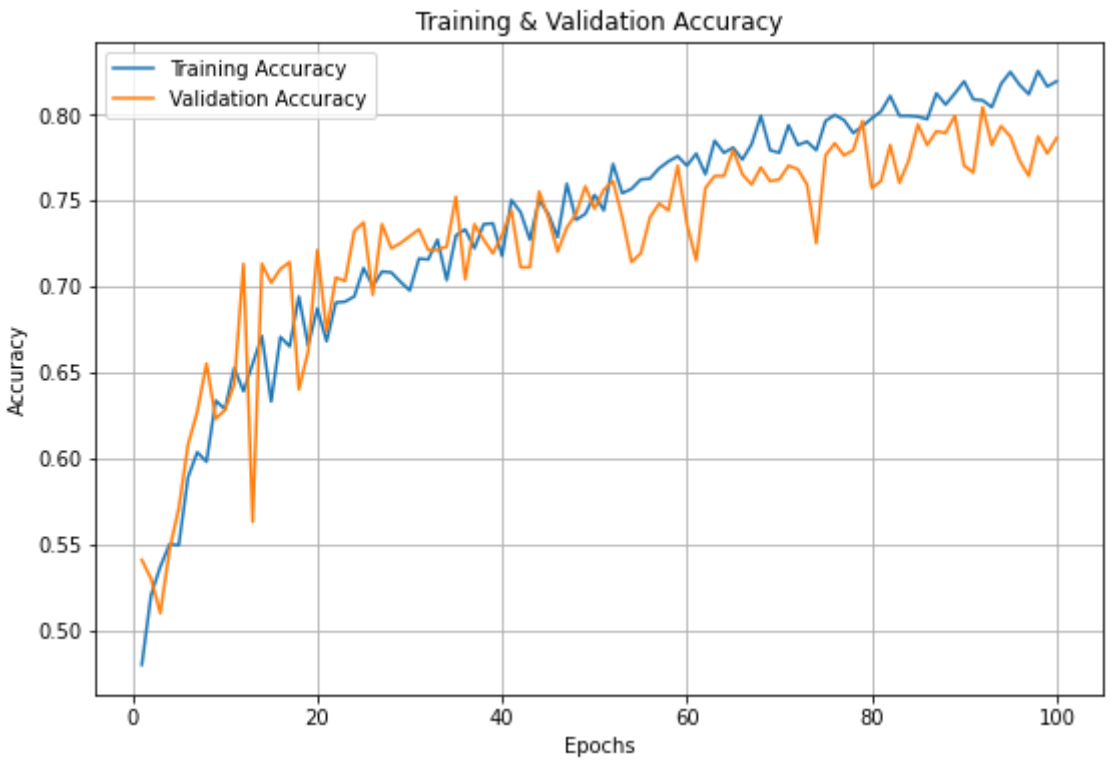
- Accuracy Visualization

```
import matplotlib.pyplot as plt

epochs = range(1, len(Hist_dandc.history['loss']) + 1)

plt.figure(figsize = (9, 6))
plt.plot(epochs, Hist_dandc.history['accuracy'])
plt.plot(epochs, Hist_dandc.history['val_accuracy'])

plt.title('Training & Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(['Training Accuracy', 'Validation Accuracy'])
plt.grid()
plt.show()
```



5) Model Evaluate

- test_generator

```
test_datagen = ImageDataGenerator(rescale = 1./255)

test_generator = test_datagen.flow_from_directory(
                 test_dir,
                 target_size = (150, 150),
                 batch_size = 20,
                 class_mode = 'binary')
```

```
Found 1000 images belonging to 2 classes.
```

- Loss & Accuracy

```
loss, accuracy = model.evaluate(test_generator,
                                steps = 50)

print('Loss = {:.5f}'.format(loss))
print('Accuracy = {:.5f}'.format(accuracy))
```

```
50/50 [==============================] - 3s 51ms/step - loss: 0.4764 - accuracy: 0.7880
Loss = 0.47637
Accuracy = 0.78800
```

IV. Model Save & Load to Google Drive

1) Google Drive Mount

```
from google.colab import drive

drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

2) Model Save

```
model.save('/content/drive/My Drive/Colab Notebooks/models/003_dogs_and_cats_augmentation.h5')
```

```
!ls -l /content/drive/My Drive/Colab Notebooks/models
```

```
total 81088
-rw------- 1 root root    34600 Aug  5 23:41 001_Model_iris.h5
-rw------- 1 root root 41498696 Aug  6 00:11 002_dogs_and_cats_small.h5
-rw------- 1 root root 41499544 Aug  6 00:46 003_dogs_and_cats_augmentation.h5
```

3) Model Load

```
from tensorflow.keras.models import load_model

model_google = load_model('/content/drive/My Drive/Colab Notebooks/models/003_dogs_and_cats_augmentation.h5')
```

```
loss, accuracy = model_google.evaluate(test_generator,
                                       steps = 50)

print('Loss = {:.5f}'.format(loss))
print('Accuracy = {:.5f}'.format(accuracy))
```

```
50/50 [==============================] - 3s 52ms/step - loss: 0.4764 - accuracy: 0.7880
```

```
Loss = 0.47637
Accuracy = 0.78800
```

#

#

#

The End

#

#

#

✓ 3초 오전 9:47에 완료됨