이미지 데이터 셋을 이용한 CNN Modeling

> ## Google Drive Mount

Dogs and Cats Image_Data

- Train_Data : 2000(1000_Dogs, 1000_Cats)
- Valid_Data : 1000(500_Dogs, 500_Cats)
- Test_Data : 1000(500_Dogs, 500_Cats)

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

Import Tensorflow & Keras

- import TensorFlow

```
import tensorflow as tf

tf.__version__
```

```
'2.6.0'
```

- GPU 설정 확인

```
print('GPU Information -', tf.test.gpu_device_name(), '₩n')

!nvidia-smi
```

```
GPU Information - /device:GPU:0

Wed Sep  1 06:31:36 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.57.02    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla K80           Off  | 00000000:00:04.0 Off |                    0 |
| N/A   39C    P0    58W / 149W |    121MiB / 11441MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

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
```

- 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'
```

II. Data Preprocessing

1) Unzip 'dogs_and_cats_small.zip'

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

```
Archive:  /content/drive/My Drive/Colab Notebooks/datasets/dogs_and_cats_small.zip
  inflating: test/cats/cat.1501.jpg
  inflating: test/cats/cat.1502.jpg
  inflating: test/cats/cat.1503.jpg
  inflating: test/cats/cat.1504.jpg
  inflating: test/cats/cat.1505.jpg
  inflating: test/cats/cat.1506.jpg
  inflating: test/cats/cat.1507.jpg
  inflating: test/cats/cat.1508.jpg
  inflating: test/cats/cat.1509.jpg
  inflating: test/cats/cat.1510.jpg
  inflating: test/cats/cat.1511.jpg
  inflating: test/cats/cat.1512.jpg
  inflating: test/cats/cat.1513.jpg
  inflating: test/cats/cat.1514.jpg
  inflating: test/cats/cat.1515.jpg
  inflating: test/cats/cat.1516.jpg
  inflating: test/cats/cat.1517.jpg
  inflating: test/cats/cat.1518.jpg
  inflating: test/cats/cat.1519.jpg
  inflating: test/cats/cat.1520.jpg
  inflating: test/cats/cat.1521.jpg
  inflating: test/cats/cat.1522.jpg
  inflating: test/cats/cat.1523.jpg
  inflating: test/cats/cat.1524.jpg
  inflating: test/cats/cat.1525.jpg
  inflating: test/cats/cat.1526.jpg
  inflating: test/cats/cat.1527.jpg
  inflating: test/cats/cat.1528.jpg
  inflating: test/cats/cat.1529.jpg
  inflating: test/cats/cat.1530.jpg
  inflating: test/cats/cat.1531.jpg
  inflating: test/cats/cat.1532.jpg
  inflating: test/cats/cat.1533.jpg
  inflating: test/cats/cat.1534.jpg
  inflating: test/cats/cat.1535.jpg
  inflating: test/cats/cat.1536.jpg
  inflating: test/cats/cat.1537.jpg
  inflating: test/cats/cat.1538.jpg
  inflating: test/cats/cat.1539.jpg
  inflating: test/cats/cat.1540.jpg
  inflating: test/cats/cat.1541.jpg
  inflating: test/cats/cat.1542.jpg
  inflating: test/cats/cat.1543.jpg
  inflating: test/cats/cat.1544.jpg
  inflating: test/cats/cat.1545.jpg
  inflating: test/cats/cat.1546.jpg
  inflating: test/cats/cat.1547.jpg
  inflating: test/cats/cat.1548.jpg
  inflating: test/cats/cat.1549.jpg
  inflating: test/cats/cat.1550.jpg
  inflating: test/cats/cat.1551.jpg
  inflating: test/cats/cat.1552.jpg
  inflating: test/cats/cat.1553.jpg
  inflating: test/cats/cat.1554.jpg
  inflating: test/cats/cat.1555.jpg
  inflating: test/cats/cat.1556.jpg
  inflating: test/cats/cat.1557.jpg
  inflating: test/cats/cat.1558.jpg
```

```
!ls -l
```

```
total 20
drwx------ 5 root root 4096 Sep  1 06:32 drive
drwxr-xr-x 1 root root 4096 Aug 25 13:35 sample_data
drwxr-xr-x 4 root root 4096 Sep  1 06:33 test
drwxr-xr-x 4 root root 4096 Sep  1 06:33 train
drwxr-xr-x 4 root root 4096 Sep  1 06:33 validation
```

2) Image_File Directory Setting

- train_dir
- valid_dir
- test_dir

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

3) ImageDataGenerator( ) & flow_from_directory( )

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

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

train_datagen = ImageDataGenerator(rescale = 1./255)
valid_datagen = ImageDataGenerator(rescale = 1./255)

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

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.
```

4) Test train_generator

```
for data_batch, labels_batch in train_generator:
    print('배치 데이터 크기:', data_batch.shape)
    print('배치 레이블 크기:', labels_batch.shape)
    break
```

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

```
labels_batch
```

```
array([1., 1., 1., 0., 0., 1., 1., 1., 1., 0., 0., 1., 0., 0., 1., 1., 0.,
       1., 1., 1.], dtype=float32)
```

III. CNN Keras Modeling

1) Model Define

- Feature Extraction & Classification

```
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'))
```

```
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
_________________________________________________________________
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

- 모델 학습 수행
  - 약 10분

```
%%time

Hist_dandc = model.fit(train_generator,
                       steps_per_epoch = 100,
                       epochs = 60,
```

```
                  validation_data = valid_generator,
                  validation_steps = 50)
```

```
Epoch 1/60
100/100 [==============================] - 41s 112ms/step - loss: 0.6951 - accuracy: 0.5015 - val_loss: 0.6909 - val_accuracy: 0.5280
Epoch 2/60
100/100 [==============================] - 11s 110ms/step - loss: 0.6760 - accuracy: 0.5745 - val_loss: 0.6988 - val_accuracy: 0.5000
Epoch 3/60
100/100 [==============================] - 11s 110ms/step - loss: 0.6790 - accuracy: 0.5550 - val_loss: 0.6538 - val_accuracy: 0.6250
Epoch 4/60
100/100 [==============================] - 11s 108ms/step - loss: 0.6648 - accuracy: 0.5975 - val_loss: 0.6635 - val_accuracy: 0.5790
Epoch 5/60
100/100 [==============================] - 11s 106ms/step - loss: 0.6406 - accuracy: 0.6200 - val_loss: 0.6377 - val_accuracy: 0.6440
Epoch 6/60
100/100 [==============================] - 11s 108ms/step - loss: 0.6011 - accuracy: 0.6815 - val_loss: 0.5987 - val_accuracy: 0.6690
Epoch 7/60
100/100 [==============================] - 11s 110ms/step - loss: 0.5750 - accuracy: 0.7035 - val_loss: 0.6147 - val_accuracy: 0.6750
Epoch 8/60
100/100 [==============================] - 11s 110ms/step - loss: 0.5223 - accuracy: 0.7380 - val_loss: 0.5414 - val_accuracy: 0.7230
Epoch 9/60
100/100 [==============================] - 11s 110ms/step - loss: 0.4851 - accuracy: 0.7545 - val_loss: 0.5461 - val_accuracy: 0.7230
Epoch 10/60
100/100 [==============================] - 11s 110ms/step - loss: 0.4167 - accuracy: 0.8085 - val_loss: 0.5924 - val_accuracy: 0.7150
Epoch 11/60
100/100 [==============================] - 11s 110ms/step - loss: 0.3514 - accuracy: 0.8405 - val_loss: 0.6683 - val_accuracy: 0.6980
Epoch 12/60
100/100 [==============================] - 11s 111ms/step - loss: 0.2678 - accuracy: 0.8820 - val_loss: 0.7170 - val_accuracy: 0.7240
Epoch 13/60
100/100 [==============================] - 11s 110ms/step - loss: 0.1817 - accuracy: 0.9290 - val_loss: 0.7675 - val_accuracy: 0.7440
Epoch 14/60
100/100 [==============================] - 11s 109ms/step - loss: 0.1633 - accuracy: 0.9360 - val_loss: 0.9546 - val_accuracy: 0.7450
Epoch 15/60
100/100 [==============================] - 11s 111ms/step - loss: 0.0954 - accuracy: 0.9655 - val_loss: 0.9750 - val_accuracy: 0.7250
Epoch 16/60
100/100 [==============================] - 11s 111ms/step - loss: 0.0724 - accuracy: 0.9795 - val_loss: 1.1126 - val_accuracy: 0.7360
Epoch 17/60
100/100 [==============================] - 11s 110ms/step - loss: 0.0264 - accuracy: 0.9920 - val_loss: 1.2606 - val_accuracy: 0.7500
Epoch 18/60
100/100 [==============================] - 11s 109ms/step - loss: 0.0442 - accuracy: 0.9850 - val_loss: 1.3328 - val_accuracy: 0.7390
Epoch 19/60
100/100 [==============================] - 11s 110ms/step - loss: 0.0189 - accuracy: 0.9950 - val_loss: 1.4302 - val_accuracy: 0.7300
Epoch 20/60
100/100 [==============================] - 11s 111ms/step - loss: 0.0026 - accuracy: 1.0000 - val_loss: 1.5340 - val_accuracy: 0.7500
Epoch 21/60
100/100 [==============================] - 11s 109ms/step - loss: 9.5646e-04 - accuracy: 1.0000 - val_loss: 1.6257 - val_accuracy: 0.7400
Epoch 22/60
100/100 [==============================] - 11s 110ms/step - loss: 4.7233e-04 - accuracy: 1.0000 - val_loss: 1.7278 - val_accuracy: 0.7420
Epoch 23/60
100/100 [==============================] - 11s 111ms/step - loss: 2.8982e-04 - accuracy: 1.0000 - val_loss: 1.8221 - val_accuracy: 0.7420
Epoch 24/60
100/100 [==============================] - 11s 110ms/step - loss: 1.7959e-04 - accuracy: 1.0000 - val_loss: 1.9174 - val_accuracy: 0.7340
Epoch 25/60
100/100 [==============================] - 11s 110ms/step - loss: 1.2972e-04 - accuracy: 1.0000 - val_loss: 2.0457 - val_accuracy: 0.7340
Epoch 26/60
100/100 [==============================] - 11s 112ms/step - loss: 8.8792e-05 - accuracy: 1.0000 - val_loss: 2.1088 - val_accuracy: 0.7350
Epoch 27/60
100/100 [==============================] - 11s 110ms/step - loss: 6.7335e-05 - accuracy: 1.0000 - val_loss: 2.1625 - val_accuracy: 0.7300
Epoch 28/60
100/100 [==============================] - 11s 111ms/step - loss: 5.1872e-05 - accuracy: 1.0000 - val_loss: 2.2197 - val_accuracy: 0.7310
Epoch 29/60
100/100 [==============================] - 11s 108ms/step - loss: 4.3213e-05 - accuracy: 1.0000 - val_loss: 2.2585 - val_accuracy: 0.7340
Epoch 30/60
```

4) 학습 결과 시각화

- Loss Visualization

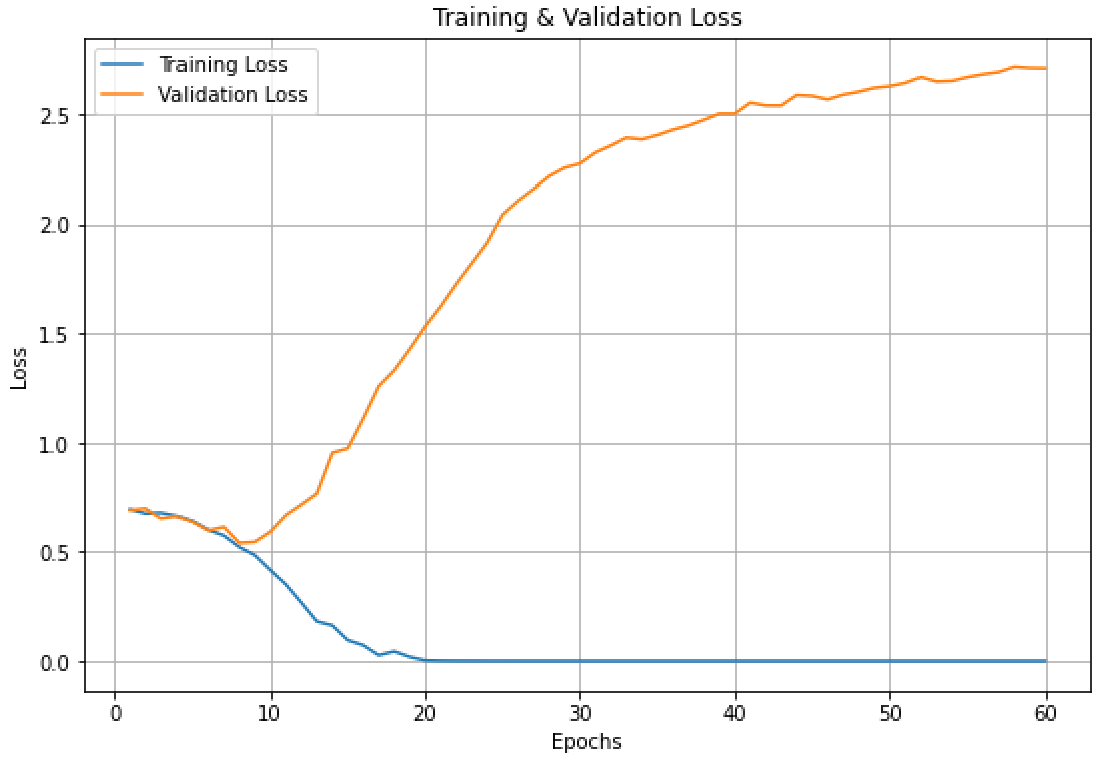
```
import matplotlib.pyplot as plt

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

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



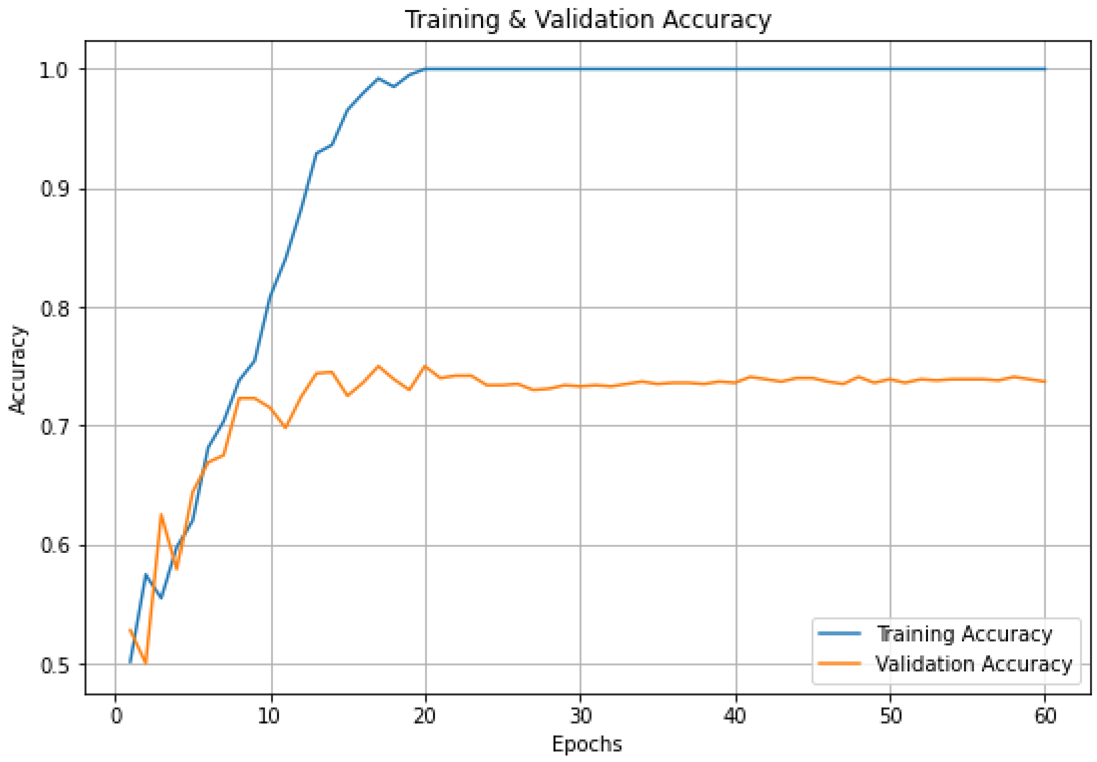
- Accuracy Visualization

```
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 67ms/step - loss: 3.3354 - accuracy: 0.7360
Loss = 3.33541
Accuracy = 0.73600
```

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/002_dogs_and_cats_small.h5')
```

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

```
total 40561
-rw------- 1 root root    34600 Aug 31 00:45 001_Model_iris.h5
-rw------- 1 root root 41498696 Sep  1 07:55 002_dogs_and_cats_small.h5
```

3) Model Load

```
from tensorflow.keras.models import load_model

model_small = load_model('/content/drive/My Drive/Colab Notebooks/models/002_dogs_and_cats_small.h5')
```

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

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

```
50/50 [==============================] - 3s 65ms/step - loss: 3.3354 - accuracy: 0.7360
Loss = 3.33541
Accuracy = 0.73600
```

#

#

#

The End

#
#
#