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__
```

```
'2.5.0'
```

- GPU 설정 확인

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

!nvidia-smi
```

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

Fri Aug  6 00:00:28 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.42.01    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 T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   42C    P0    26W /  70W |    222MiB / 15109MiB |      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
```

```
  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 00:01 drive
drwxr-xr-x 1 root root 4096 Jul 16 13:20 sample_data
drwxr-xr-x 4 root root 4096 Aug  6 00:01 test
drwxr-xr-x 4 root root 4096 Aug  6 00:01 train
drwxr-xr-x 4 root root 4096 Aug  6 00:01 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([0., 1., 1., 0., 1., 1., 1., 1., 0., 1., 1., 1., 0., 0., 0., 1., 0.,
       1., 0., 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)
```

```
100/100 [==============================] - 10s 98ms/step - loss: 1.0751e-05 - accuracy: 1.0000 - val_loss: 3.1576 - val_accuracy: 0.7140
Epoch 33/60
100/100 [==============================] - 10s 98ms/step - loss: 9.3623e-06 - accuracy: 1.0000 - val_loss: 3.2032 - val_accuracy: 0.7170
Epoch 34/60
100/100 [==============================] - 10s 96ms/step - loss: 8.2684e-06 - accuracy: 1.0000 - val_loss: 3.2331 - val_accuracy: 0.7160
Epoch 35/60
100/100 [==============================] - 9s 91ms/step - loss: 7.2307e-06 - accuracy: 1.0000 - val_loss: 3.2624 - val_accuracy: 0.7170
Epoch 36/60
100/100 [==============================] - 9s 91ms/step - loss: 6.5145e-06 - accuracy: 1.0000 - val_loss: 3.2914 - val_accuracy: 0.7140
Epoch 37/60
100/100 [==============================] - 9s 91ms/step - loss: 5.7538e-06 - accuracy: 1.0000 - val_loss: 3.3244 - val_accuracy: 0.7180
Epoch 38/60
100/100 [==============================] - 9s 89ms/step - loss: 5.1275e-06 - accuracy: 1.0000 - val_loss: 3.3558 - val_accuracy: 0.7140
Epoch 39/60
100/100 [==============================] - 9s 95ms/step - loss: 4.6287e-06 - accuracy: 1.0000 - val_loss: 3.3633 - val_accuracy: 0.7150
Epoch 40/60
100/100 [==============================] - 10s 96ms/step - loss: 4.2193e-06 - accuracy: 1.0000 - val_loss: 3.3941 - val_accuracy: 0.7120
Epoch 41/60
100/100 [==============================] - 9s 94ms/step - loss: 3.8226e-06 - accuracy: 1.0000 - val_loss: 3.4084 - val_accuracy: 0.7140
Epoch 42/60
100/100 [==============================] - 9s 90ms/step - loss: 3.4778e-06 - accuracy: 1.0000 - val_loss: 3.4293 - val_accuracy: 0.7150
Epoch 43/60
100/100 [==============================] - 9s 92ms/step - loss: 3.1713e-06 - accuracy: 1.0000 - val_loss: 3.4513 - val_accuracy: 0.7160
Epoch 44/60
100/100 [==============================] - 9s 91ms/step - loss: 2.8984e-06 - accuracy: 1.0000 - val_loss: 3.4736 - val_accuracy: 0.7130
Epoch 45/60
100/100 [==============================] - 9s 90ms/step - loss: 2.6650e-06 - accuracy: 1.0000 - val_loss: 3.4982 - val_accuracy: 0.7120
Epoch 46/60
100/100 [==============================] - 10s 96ms/step - loss: 2.4687e-06 - accuracy: 1.0000 - val_loss: 3.5073 - val_accuracy: 0.7140
Epoch 47/60
100/100 [==============================] - 10s 96ms/step - loss: 2.2647e-06 - accuracy: 1.0000 - val_loss: 3.5343 - val_accuracy: 0.7100
Epoch 48/60
100/100 [==============================] - 10s 96ms/step - loss: 2.1079e-06 - accuracy: 1.0000 - val_loss: 3.5488 - val_accuracy: 0.7110
Epoch 49/60
100/100 [==============================] - 9s 93ms/step - loss: 1.9645e-06 - accuracy: 1.0000 - val_loss: 3.5658 - val_accuracy: 0.7110
Epoch 50/60
100/100 [==============================] - 10s 99ms/step - loss: 1.8076e-06 - accuracy: 1.0000 - val_loss: 3.5859 - val_accuracy: 0.7110
Epoch 51/60
100/100 [==============================] - 10s 96ms/step - loss: 1.6846e-06 - accuracy: 1.0000 - val_loss: 3.5982 - val_accuracy: 0.7100
Epoch 52/60
100/100 [==============================] - 9s 91ms/step - loss: 1.5669e-06 - accuracy: 1.0000 - val_loss: 3.6105 - val_accuracy: 0.7160
Epoch 53/60
100/100 [==============================] - 9s 91ms/step - loss: 1.4397e-06 - accuracy: 1.0000 - val_loss: 3.6397 - val_accuracy: 0.7110
Epoch 54/60
100/100 [==============================] - 9s 91ms/step - loss: 1.3590e-06 - accuracy: 1.0000 - val_loss: 3.6545 - val_accuracy: 0.7110
Epoch 55/60
100/100 [==============================] - 9s 92ms/step - loss: 1.2669e-06 - accuracy: 1.0000 - val_loss: 3.6603 - val_accuracy: 0.7130
Epoch 56/60
100/100 [==============================] - 9s 91ms/step - loss: 1.1785e-06 - accuracy: 1.0000 - val_loss: 3.6865 - val_accuracy: 0.7110
Epoch 57/60
100/100 [==============================] - 10s 98ms/step - loss: 1.1038e-06 - accuracy: 1.0000 - val_loss: 3.6957 - val_accuracy: 0.7130
Epoch 58/60
100/100 [==============================] - 10s 96ms/step - loss: 1.0321e-06 - accuracy: 1.0000 - val_loss: 3.7084 - val_accuracy: 0.7120
Epoch 59/60
100/100 [==============================] - 10s 95ms/step - loss: 9.6650e-07 - accuracy: 1.0000 - val_loss: 3.7208 - val_accuracy: 0.7110
Epoch 60/60
100/100 [==============================] - 9s 91ms/step - loss: 9.0702e-07 - accuracy: 1.0000 - val_loss: 3.7426 - val_accuracy: 0.7110
CPU times: user 11min 29s, sys: 19.8 s, total: 11min 49s
Wall time: 9min 52s
```

4) 학습 결과 시각화

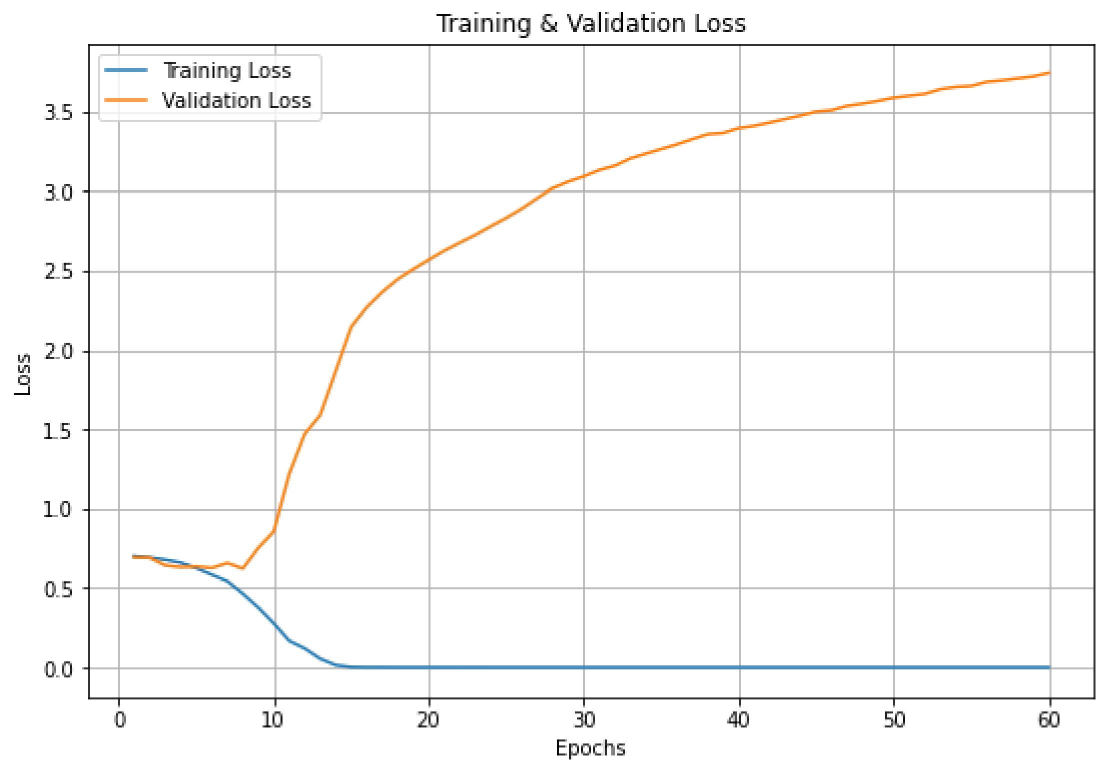
- Loss Visualization

```
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()
```



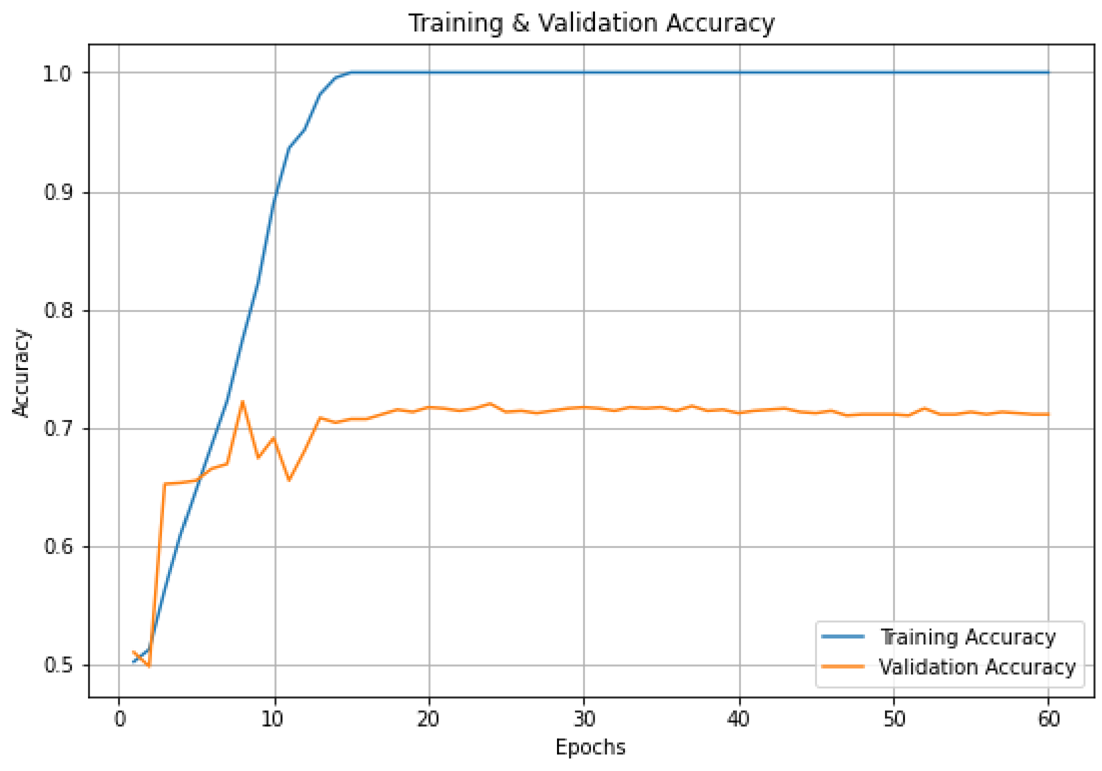
- Accuracy Visualization

```
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 55ms/step - loss: 3.7631 - accuracy: 0.6900
Loss = 3.76313
Accuracy = 0.69000
```

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  5 23:41 001_Model_iris.h5
-rw------- 1 root root 41498696 Aug  6 00:11 002_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 56ms/step - loss: 3.7631 - accuracy: 0.6900
Loss = 3.76313
Accuracy = 0.69000
```

#

#

#

The End

```
#

#

#
```

✓ 3초 오전 9:11에 완료됨