## 텐서플로워 신경망 만들기

- CNN 신경망 이해
- 고양이와 개의 분류를 CNN을 이용하여 구현해 보기(2)
- 추가 성능 개선시켜보기

## 학습 내용

- 데이터 로드
- ImageGenerator를 생성
- 모델 학습
- 모델 평가
- 모델 저장 및 불러오기

## 목차

## 01. 라이브러리 가져오기

목차로 이동하기

In [4]:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout, MaxPooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator

import os, sys
import numpy as np
import matplotlib.pyplot as plt
```

In [5]:

```
print(tf.__version__)
print(np.__version__)
print(sys.version)
```

```
2.10.0
1.21.5
3.9.12 (main, Apr  4 2022, 05:22:27) [MSC v.1916 64 bit (AMD64)]
```

## 데이터 가져오기

In [6]:

```
dat_URL = 'https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip'

path_to_zip = tf.keras.utils.get_file('cats_and_dogs.zip',
                                      origin=dat_URL, extract=True)

PATH = os.path.join(os.path.dirname(path_to_zip), 'cats_and_dogs_filtered')
print(PATH)
```

```
C:\Users\totofriend\.keras\datasets\cats_and_dogs_filtered
```

## 모델을 위한 지정 경로 지정(학습, 검증)

In [7]:

```
## 경로 지정
train_dir = os.path.join(PATH, 'train')
val_dir = os.path.join(PATH, 'validation')

train_cats_dir = os.path.join(train_dir, 'cats')  # directory with our training cat pictures
train_dogs_dir = os.path.join(train_dir, 'dogs')  # directory with our training dog pictures
val_cats_dir = os.path.join(val_dir, 'cats')  # directory with our validation cat pictures
val_dogs_dir = os.path.join(val_dir, 'dogs')  # directory with our validation dog pictures

## 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
```

In [8]:

```
print('total training cat images:', num_cats_tr)
print('total training dog images:', num_dogs_tr)
print("--")
print('total validation cat images:', num_cats_val)
print('total validation dog images:', num_dogs_val)
print("--")
print("Total training images:", total_train)
print("Total validation images:", total_val)
```

```
total training cat images: 1000
total training dog images: 1000
--
total validation cat images: 500
total validation dog images: 500
--
Total training images: 2000
Total validation images: 1000
```

## 02. 이미지 확인

목차로 이동하기

In [9]:

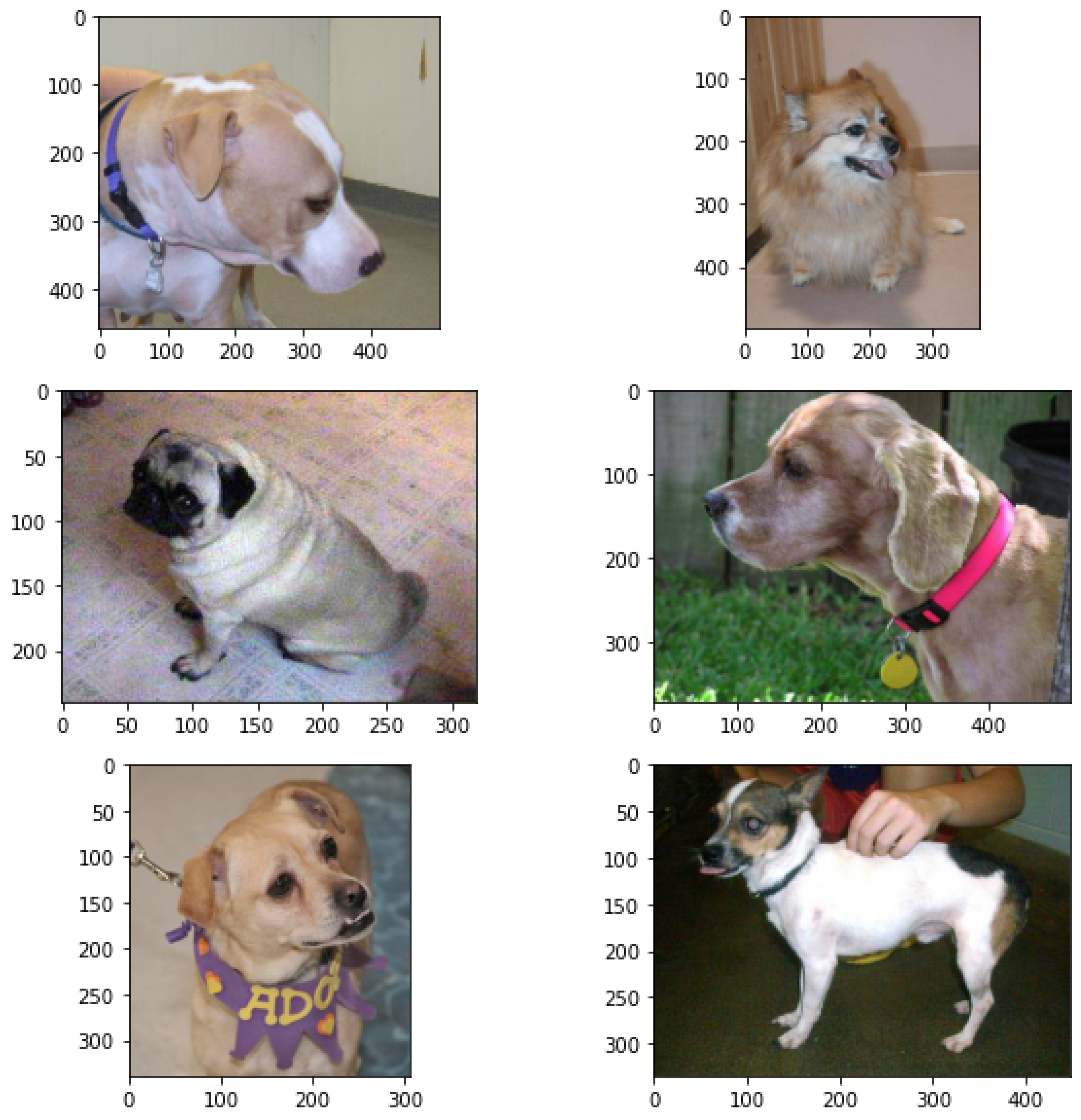
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

In [10]:

```
#  'dog.838.jpg', 'dog.967.jpg', 'dog.126.jpg', 'dog.102.jpg', 'dog.563.jpg', 'dog.214.jpg']
plt.figure(figsize=(10, 10))
img_idx = [838, 967, 126, 102, 563, 214]

cnt = 1
for i in img_idx:
  imgpath = PATH + "/train/dogs/dog." + str(i) + ".jpg"
  img = mpimg.imread(imgpath)
  print(img.shape)
  plt.subplot(3,2,cnt)
  plt.imshow(img)
  cnt += 1
```

```
(458, 499, 3)
(499, 375, 3)
(240, 319, 3)
(373, 499, 3)
(339, 305, 3)
(335, 448, 3)
```



In [11]:

```
batch_size = 20
epochs = 15
IMG_HEIGHT = 150
IMG_WIDTH = 150
```

## 이미지 데이터 제너레이터

In [12]:

```
# 학습용 데이터에 대한 제너레이터(Generator)
train_img_generator = ImageDataGenerator(rescale=1./255)

# 검증용 데이터에 대한 제너레이터(Generator)
val_img_generator = ImageDataGenerator(rescale=1./255)
```

- directory : 타깃 디렉터리
- shuffle : 데이터 섞기
- target_size : 모든 이미지를 지정된 이미지로 변경 (150 x 150)
- class_mode : binary_crossentropy 손실을 사용하기에 이진 레이블이 필요
  - 이진 분류의 경우는 binary
  - 다중 분류의 경우는 categorical, sparse를 사용.

## 객체 생성 후, 폴더의 이미지를 넘파이 배열 객체로 전달.

- flow_from_directory : 데이터가 담긴 디렉터리 안의 폴더를 확인하여 이미지 배열 데이터 생성.

In [13]:

```
train_data_gen = train_img_generator.flow_from_directory(batch_size=batch_size,
                                                   directory=train_dir,
                                                   shuffle=True,
                                                   target_size=(IMG_HEIGHT, IMG_WIDTH),
                                                   class_mode='binary')

val_data_gen = val_img_generator.flow_from_directory(batch_size=batch_size,
                                               directory=val_dir,
                                               target_size=(IMG_HEIGHT, IMG_WIDTH),
                                               class_mode='binary')
```

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

## 03. 모델 구축 및 학습

목차로 이동하기

In [14]:

```
model = Sequential([
    Conv2D(16, 3, padding='same', activation='relu', input_shape=(IMG_HEIGHT, IMG_WIDTH ,3)),
    MaxPooling2D(),
    Conv2D(32, 3, padding='same', activation='relu'),
    MaxPooling2D(),
    Conv2D(64, 3, padding='same', activation='relu'),
    MaxPooling2D(),
    Flatten(),
    Dense(512, activation='relu'),
    Dense(1, activation='sigmoid')
])

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

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 150, 150, 16)      448

 max_pooling2d (MaxPooling2D  (None, 75, 75, 16)       0
 )

 conv2d_1 (Conv2D)           (None, 75, 75, 32)        4640

 max_pooling2d_1 (MaxPooling  (None, 37, 37, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 37, 37, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 18, 18, 64)       0
 2D)

 flatten (Flatten)           (None, 20736)             0

 dense (Dense)               (None, 512)               10617344

 dense_1 (Dense)             (None, 1)                 513

=================================================================
Total params: 10,641,441
Trainable params: 10,641,441
Non-trainable params: 0
_________________________________________________________________
```

## 모델 학습

- [].fit_generator() 메서드는 fit메서드와 동일하되 데이터 제너레이터를 사용할 수 있음.
  - steps_per_epoch : 하나의 에포크 단위로 제너레이터로부터 얼마나 많은 샘플을 뽑을 것인가?
    - 지정된 횟수만큼 경사 하강법 단계를 실행한 후, 훈련 프로세스는 다음 에포크로 넘어간다.

In [15]:

```
print("학습용 데이터 : ", total_train)
print("검증용 데이터 : ", total_val)
print("1 epoch(train) : ", total_train // batch_size)
print("1 epoch(val) : ", total_val // batch_size)
print("batch size : ", batch_size)
print("epochs = ", epochs)
```

```
학습용 데이터 :  2000
검증용 데이터 :  1000
1 epoch(train) :  100
1 epoch(val) :  50
batch size :  20
epochs =  15
```

- 현재 fit으로 학습이 가능하며, fit_generator도 사용 되었었음.

In [16]:

```
%%time

### 모델 학습
history = model.fit(    #  model.fit_generator(
    train_data_gen, # 학습용 데이터
    steps_per_epoch=total_train // batch_size,
    epochs=epochs,
    validation_data=val_data_gen,             # 검증 데이터
    validation_steps=total_val // batch_size   # 검증 데이터 스텝
)
```

```
Epoch 1/15
100/100 [==============================] - 26s 255ms/step - loss: 0.8062 - accuracy:
0.5260 - val_loss: 0.6593 - val_accuracy: 0.6440
Epoch 2/15
100/100 [==============================] - 24s 243ms/step - loss: 0.6501 - accuracy:
0.6355 - val_loss: 0.6540 - val_accuracy: 0.5810
Epoch 3/15
100/100 [==============================] - 24s 236ms/step - loss: 0.5868 - accuracy:
0.6965 - val_loss: 0.5784 - val_accuracy: 0.7140
Epoch 4/15
100/100 [==============================] - 24s 238ms/step - loss: 0.5056 - accuracy:
0.7480 - val_loss: 0.5702 - val_accuracy: 0.7220
Epoch 5/15
100/100 [==============================] - 24s 237ms/step - loss: 0.4187 - accuracy:
0.8090 - val_loss: 0.5661 - val_accuracy: 0.7270
Epoch 6/15
100/100 [==============================] - 24s 238ms/step - loss: 0.3423 - accuracy:
0.8535 - val_loss: 0.5924 - val_accuracy: 0.7320
Epoch 7/15
100/100 [==============================] - 24s 238ms/step - loss: 0.2424 - accuracy:
0.8995 - val_loss: 0.6806 - val_accuracy: 0.7370
Epoch 8/15
100/100 [==============================] - 24s 238ms/step - loss: 0.1627 - accuracy:
0.9395 - val_loss: 0.7745 - val_accuracy: 0.7340
Epoch 9/15
100/100 [==============================] - 24s 239ms/step - loss: 0.0973 - accuracy:
0.9685 - val_loss: 0.9576 - val_accuracy: 0.7330
Epoch 10/15
100/100 [==============================] - 24s 236ms/step - loss: 0.0772 - accuracy:
0.9760 - val_loss: 1.0749 - val_accuracy: 0.6850
Epoch 11/15
100/100 [==============================] - 24s 237ms/step - loss: 0.0486 - accuracy:
0.9840 - val_loss: 1.3200 - val_accuracy: 0.7110
Epoch 12/15
100/100 [==============================] - 24s 237ms/step - loss: 0.0303 - accuracy:
0.9920 - val_loss: 1.3180 - val_accuracy: 0.7090
Epoch 13/15
100/100 [==============================] - 24s 235ms/step - loss: 0.0105 - accuracy:
0.9990 - val_loss: 1.5725 - val_accuracy: 0.7240
Epoch 14/15
100/100 [==============================] - 24s 235ms/step - loss: 0.0023 - accuracy:
1.0000 - val_loss: 1.7049 - val_accuracy: 0.7280
Epoch 15/15
100/100 [==============================] - 24s 236ms/step - loss: 9.1471e-04 - accur
acy: 1.0000 - val_loss: 1.7873 - val_accuracy: 0.7310
CPU times: total: 52min 19s
Wall time: 5min 58s
```

## 04. 학습된 모델 저장 및 불러오기

목차로 이동하기

### 학습 후, 모델 저장

In [17]:

```
model.evaluate(val_data_gen)
```

```
50/50 [==============================] - 3s 64ms/step - loss: 1.7873 - accuracy: 0.7
310
```

Out[17]:

```
[1.7873262166976929, 0.7310000061988831]
```

- 학습된 모델 아키텍쳐와 학습된 모델 가중치 저장.

In [19]:

```
model.save('cats_and_dogs_small_1.h5')
```

In [20]:

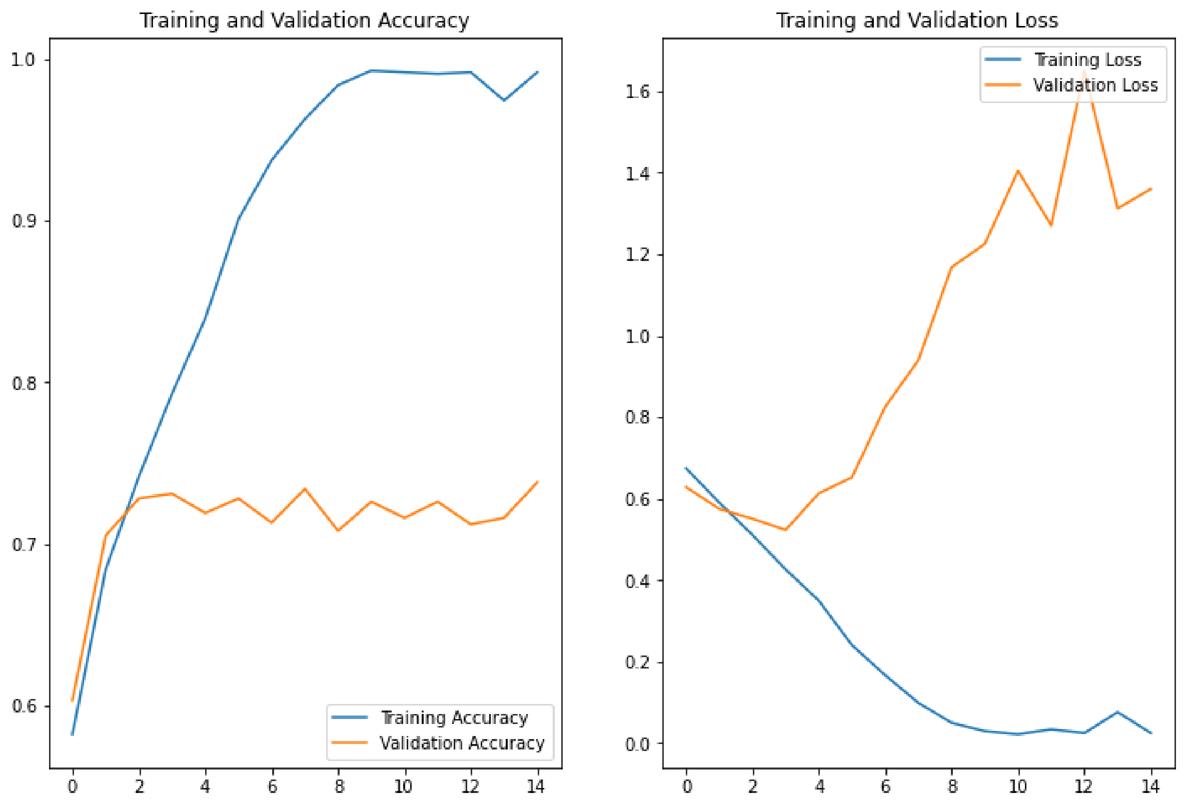
```
### 결과 시각화
acc = history.history['accuracy']
loss = history.history['loss']

val_acc = history.history['val_accuracy']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(12, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



**학습된 모델을 다시 불러와 5epoch를 더 학습 시킨 후, 평가를 수행해 보자.**

In [21]:

```
from tensorflow.keras.models import load_model
model2 = load_model("cats_and_dogs_small_1.h5")
model2
```

Out[21]:

```
<keras.engine.sequential.Sequential at 0x24895257c40>
```

In [24]:

```
epochs = 5
```

In [23]:

```
%%time

### 모델 학습
hist = model2.fit(    #  model.fit_generator(
    train_data_gen, # 학습용 데이터
    steps_per_epoch=total_train // batch_size,
    epochs=5,
    validation_data=val_data_gen,              # 검증 데이터
    validation_steps=total_val // batch_size   # 검증 데이터 스텝
)
```

```
Epoch 1/5
100/100 [==============================] - 25s 245ms/step - loss: 0.0059 - accuracy:
0.9990 - val_loss: 1.5655 - val_accuracy: 0.7380
Epoch 2/5
100/100 [==============================] - 26s 261ms/step - loss: 0.0015 - accuracy:
1.0000 - val_loss: 1.6401 - val_accuracy: 0.7450
Epoch 3/5
100/100 [==============================] - 25s 248ms/step - loss: 4.0299e-04 - accur
acy: 1.0000 - val_loss: 1.7490 - val_accuracy: 0.7450
Epoch 4/5
100/100 [==============================] - 24s 243ms/step - loss: 2.3251e-04 - accur
acy: 1.0000 - val_loss: 1.7873 - val_accuracy: 0.7470
Epoch 5/5
100/100 [==============================] - 24s 245ms/step - loss: 1.6189e-04 - accur
acy: 1.0000 - val_loss: 1.8173 - val_accuracy: 0.7480
CPU times: total: 17min 32s
Wall time: 2min 4s
```

In [25]:

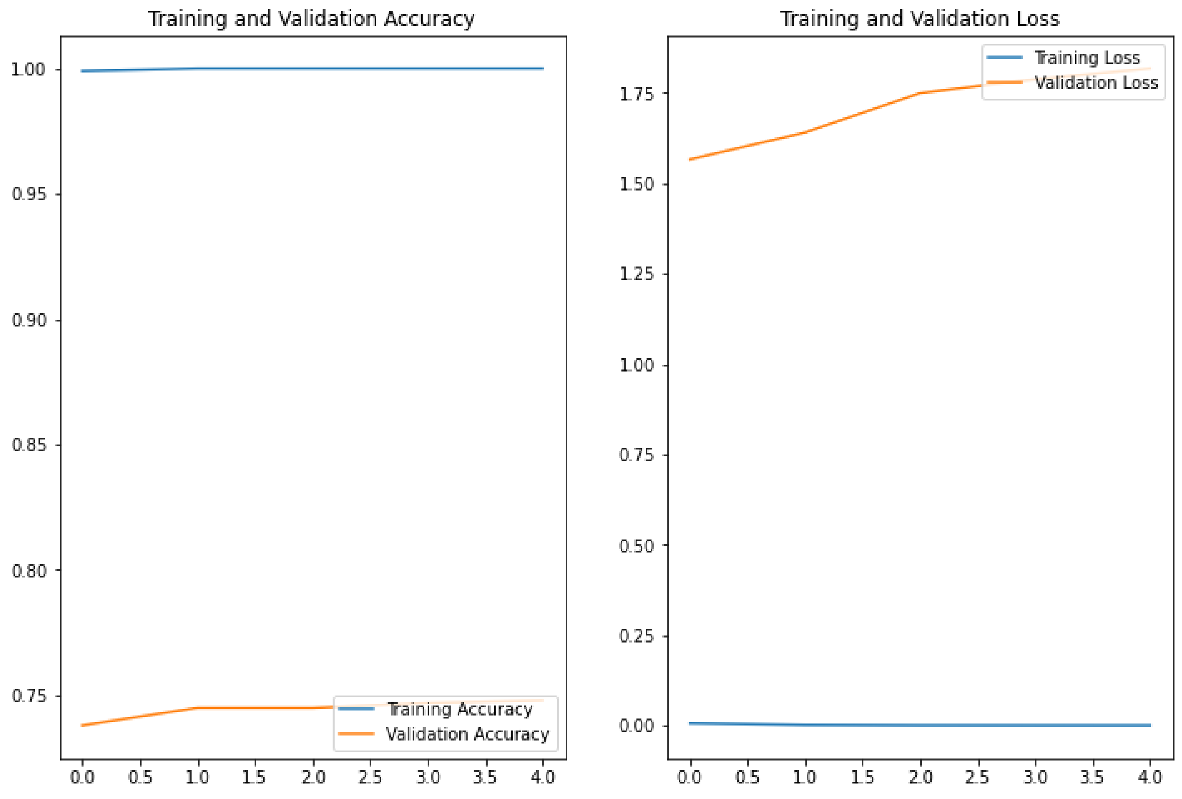
```
### 결과 시각화
acc = hist.history['accuracy']
loss = hist.history['loss']

val_acc = hist.history['val_accuracy']
val_loss = hist.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(12, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```


Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Training and Validation Loss
Training Loss
Validation Loss