미세조정

최석재 *lingua@naver.com*

파인튜닝 (미세조정)

- 앞에서 수행한 방법은 완전연결층을 제외한
- 사전학습모델의 모든 층의 가중치를 그대로 사용한 전이학습의 방법

- 층의 일부분을 현재 데이터에 맞게 재학습시키는 파인튜닝의 방법도 사용할 수 있다
- 전이학습과 파인튜닝 중 어떤 방식이 더 좋은지는 모델과 데이터에 따라 다르다
- 기본적으로 층의 학습 가능 여부를 다루는 부분을 다음과 같이 조절함으로 가능하다

```
• vgg.trainable = True                  # 우선 모든 층의 가중치를 학습이 가능한 형태로 만듦
• for layer in vgg.layers[:-4]:         # 마지막에서 네 번째 층까지는 가중치 동결
    layer.trainable = False
```

※ 전이학습과 비교하여 코드의 차이는 두 슬라이드에서만 있음

케라스 사전학습 모델

- 사전 학습 모델로 이미지 분류

  - Keras는 이미지 분류용의 다양한 사전 학습 모델을 가지고 있다
  - 이들은 대부분 1,000 class를 갖는 ImageNet 데이터로부터 학습되었다

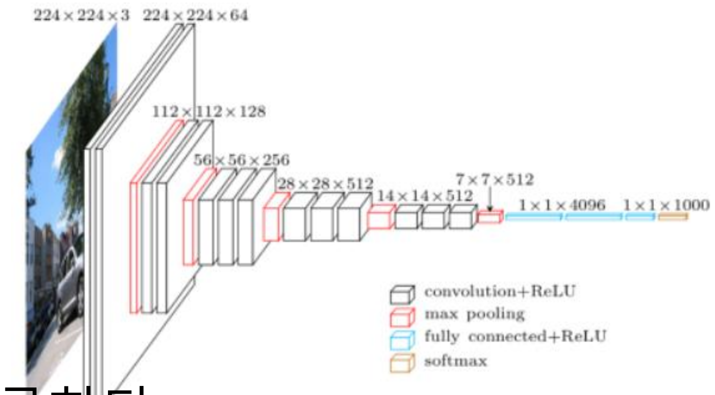
  - https://keras.io/api/applications/

Available models

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4
ResNet152	232	76.6%	93.1%	60.4M	311	127.4	6.5
ResNet152V2	232	78.0%	94.2%	60.4M	307	107.5	6.6
InceptionV3	92	77.9%	93.7%	23.9M	189	42.2	6.9
InceptionResNetV2	215	80.3%	95.3%	55.9M	449	130.2	10.0
MobileNet	16	70.4%	89.5%	4.3M	55	22.6	3.4
MobileNetV2	14	71.3%	90.1%	3.5M	105	25.9	3.8
DenseNet121	33	75.0%	92.3%	8.1M	242	77.1	5.4
DenseNet169	57	76.2%	93.2%	14.3M	338	96.4	6.3
DenseNet201	80	77.3%	93.6%	20.2M	402	127.2	6.7
NASNetMobile	23	74.4%	91.9%	5.3M	389	27.0	6.7

※ 데이터는 cats_dogs를 사용하며,
full data는 _original_jpg_backup 폴더에 있음

VGG19



- 이미지 분류용 사전 학습 모델인 VGG 19를 이용하여 이미지를 분류한다

- VGG19의 최종 출력층은 1,000 class를 분류할 수 있다
- input size는 (224, 224)이다

- https://keras.io/api/applications/vgg/

VGG19 function [source]

```
tf.keras.applications.VGG19(
    include_top=True,
    weights="imagenet",
    input_tensor=None,
    input_shape=None,
    pooling=None,
    classes=1000,
    classifier_activation="softmax",
)
```

Instantiates the VGG19 architecture.

Reference

- Very Deep Convolutional Networks for Large-Scale Image Recognition (ICLR 2015)

For image classification use cases, see this page for detailed examples.

For transfer learning use cases, make sure to read the guide to transfer learning & fine-tuning.

The default input size for this model is 224x224.

Note: each Keras Application expects a specific kind of input preprocessing. For VGG19, call `tf.keras.applications.vgg19.preprocess_input` on your inputs before passing them to the model. `vgg19.preprocess_input` will convert the input images from RGB to BGR, then will zero-center each color channel with respect to the ImageNet dataset, without scaling.

ImageNet 1000 Class

- https://deeplearning.cms.waikato.ac.nz/user-guide/class-maps/IMAGENET/

IMAGENET 1000 Class List

This is used by most pretrained models included in **WekaDeeplearning4j**.

Back to Inference Tutorial

Class ID	Class Name
0	tench, Tinca tinca
1	goldfish, Carassius auratus
2	great white shark, white shark, man-eater, man-eating shark, Carcharodon caharias',
3	tiger shark, Galeocerdo cuvieri
4	hammerhead, hammerhead shark
5	electric ray, crampfish, numbfish, torpedo
6	stingray
7	cock
8	hen
9	ostrich, Struthio camelus
10	brambling, Fringilla montifringilla
11	goldfinch, Carduelis carduelis
12	house finch, linnet, Carpodacus mexicanus
13	junco, snowbird
14	indigo bunting, indigo finch, indigo bird, Passerina cyanea

구글 드라이브와 연결

from google.colab import auth
auth.authenticate_user()

- from google.colab import drive
- drive.mount('/content/gdrive')

관련 패키지 임포트

- import numpy as np
- import pandas as pd
- import matplotlib.pyplot as plt
- from tensorflow.keras.layers import Input, Lambda, Dense, Flatten
- from tensorflow.keras.models import Model
- from tensorflow.keras.applications.vgg19 import VGG19
- from tensorflow.keras.preprocessing.image import ImageDataGenerator

데이터 경로 설정

- folder = '/content/gdrive/MyDrive/pytest_img/cats_dogs’

- train_dir = folder+"/train"
- validation_dir = folder+"/validation"
- test_dir = folder+"/test"

flow_from_directory()를 이용한 데이터 증식

- 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)
```

※ validation과 test 데이터는 증식을 하면 안된다
※ 여기서는 flow_from_directory()를 이용할 것이며,
한 batch 마다 하나의 증식된 이미지가 생성된다 (100 batch인 경우, 100장 당 1장)
따라서 2000 data / 100 batch = 20 aug. images
100 epoch면, 20 aug. images x 100 epochs = 2000 aug. images

- validation_datagen = ImageDataGenerator(rescale=1./255)
- test_datagen = ImageDataGenerator(rescale=1./255)

데이터 주입

VGG19는 입력데이터의 size가 (224, 224)일 것을 요구한다

- train_generator = train_datagen.flow_from_directory(
  train_dir, target_size=(224, 224), batch_size=100, class_mode='binary',
  classes=['cats', 'dogs'])

이진분류에서는 binary
다중분류에서는 categorical

※ 메모리가 부족하면 batch_size=20 정도로 조정해본다
단, batch_size=20 x steps_per_epoch=20 이므로 2000장 중 400장만 학습하는 것임

- validation_generator = validation_datagen.flow_from_directory(
  validation_dir, target_size=(224, 224), batch_size=100, class_mode='binary',
  classes=['cats', 'dogs'])

- test_generator = test_datagen.flow_from_directory(
  test_dir, target_size=(224, 224), batch_size=100, class_mode='binary',
  classes=['cats', 'dogs'])

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

VGG 19 최종층 제거 전

- 우측과 같은 VGG19 모델에서
- include_top=False 를 통해 마지막 부분을 제거하고
- 이진분류용 Dense층을 추가할 것임

- Dense 층은 이미지의 일반적인 특징을 잘 포착하지 못해 일반적으로 재사용하지 않는다

최종층 제거 전

input_1 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv4 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv4 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv4 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000

이 부분이 제거될 것

파인튜닝 : 모델 설계

include_top=False를 통해 최종 완전연결층을 제거하고, 가중치를 가져온다

- vgg = VGG19(input_shape=[224, 224, 3], weights='imagenet', include_top=False)

- vgg.trainable = True   # 우선 모든 층의 가중치를 학습이 가능한 형태로 만듦
- for layer in vgg.layers[:-4]:   # 마지막에서 네 번째 층까지는 가중치 동결
    layer.trainable = False

결과를 Flatten 하여 Dense 층에 붙일 수 있게 한다

- x = Flatten()(vgg.output)

Flatten된 결과를 Dense층에 입력하여, 분류기 형태가 되게 한다
이진분류이므로 출력층 노드는 1, 활성화함수는 sigmoid이다. 다중분류에서는 softmax를 사용한다

- prediction = Dense(1, activation='sigmoid')(x)

FunctionalAPI를 이용하여 모델을 구성한다

- model = Model(inputs=vgg.input, outputs=prediction)
- model.summary()

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0

block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792

block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928

block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0

block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856

block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584

block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0

block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168

block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080

block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080

block3_conv4 (Conv2D)        (None, 56, 56, 256)       590080

block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0

block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160

block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808

block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808

block4_conv4 (Conv2D)        (None, 28, 28, 512)       2359808

block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0

block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808

block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808

block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808

block5_conv4 (Conv2D)        (None, 14, 14, 512)       2359808

block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0

flatten (Flatten)            (None, 25088)             0

dense (Dense)                (None, 1)                 25089

=================================================================
Total params: 20,049,473
Trainable params: 25,089
Non-trainable params: 20,024,384
_________________________________________________________________
```

파인튜닝 : 모델 컴파일

- 파인튜닝 시에는 학습되는 층의 가중치 학습이 조금씩 일어나게 하는 것이 좋다
- 학습률을 기본값보다 낮춰서 천천히 학습이 진행되도록 한다
- 학습률을 조절하려면 다음과 같이 옵티마이저 클래스를 불러 사용한다

- import tensorflow as tf
- model.compile(loss='binary_crossentropy', optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.00001), metrics=['acc'])

모델 훈련

- model.compile(loss='binary_crossentropy', optimizer='rmsprop', metrics=['acc'])

- history = model.fit(

```
        train_generator,
        steps_per_epoch=20,
        epochs=100,
        validation_data=validation_generator,
        validation_steps=10)
```

정확도 확인

- acc = history.history['acc']
- val_acc = history.history['val_acc']
- loss = history.history['loss']
- val_loss = history.history['val_loss']

- print('Accuracy of each epoch:', np.round(acc))
- print()
- print('Validation Accuracy of each epoch:', np.round(val_acc))

파인튜닝 결과

Accuracy of each epoch: [0.651, 0.698, 0.764, 0.796, 0.814, 0.846, 0.832, 0.846, 0.847, 0.852, 0.876, 0.878, 0.866, 0.888, 0.890, 0.876, 0.897, 0.868, 0.898, 0.896, 0.897, 0.912, 0.891, 0.919, 0.913, 0.909, 0.910, 0.915, 0.908, 0.908, 0.922, 0.912, 0.926, 0.928, 0.921, 0.915, 0.923, 0.924, 0.922, 0.930, 0.934, 0.905, 0.933, 0.926, 0.928, 0.939, 0.930, 0.930, 0.927, 0.937, 0.932, 0.936, 0.940, 0.944, 0.936, 0.944, 0.936, 0.939, 0.936, 0.942, 0.928, 0.953, 0.937, 0.940, 0.943, 0.951, 0.937, 0.959, 0.942, 0.948, 0.952, 0.935, 0.953, 0.957, 0.954, 0.960, 0.955, 0.949, 0.949, 0.950, 0.964, 0.941, 0.966, 0.954, 0.957, 0.955, 0.946, 0.955, 0.950, 0.953, 0.951, 0.962, 0.961, 0.957, 0.965, 0.955, 0.963, 0.963, 0.959, 0.958]

Validation Accuracy of each epoch: [0.754, 0.822, 0.808, 0.842, 0.854, 0.858, 0.862, 0.880, 0.904, 0.884, 0.896, 0.896, 0.904, 0.902, 0.920, 0.908, 0.892, 0.908, 0.902, 0.926, 0.928, 0.896, 0.922, 0.904, 0.930, 0.928, 0.916, 0.902, 0.942, 0.906, 0.926, 0.944, 0.932, 0.918, 0.938, 0.924, 0.926, 0.920, 0.932, 0.942, 0.936, 0.942, 0.926, 0.940, 0.930, 0.936, 0.940, 0.958, 0.942, 0.938, 0.946, 0.944, 0.940, 0.946, 0.926, 0.926, 0.928, 0.932, 0.928, 0.928, 0.926, 0.932, 0.954, 0.926, 0.950, 0.936, 0.954, 0.942, 0.944, 0.950, 0.950, 0.940, 0.934, 0.950, 0.952, 0.946, 0.942, 0.934, 0.954, 0.952, 0.940, 0.952, 0.948, 0.938, 0.936, 0.950, 0.948, 0.958, 0.940, 0.948, 0.952, 0.934, 0.954, 0.956, 0.938, 0.938, 0.918, 0.950, 0.952, 0.940]

손실값 확인

- print('Loss of each epoch:', np.round(loss, 3))
- print()
- print('Validation Loss of each epoch:', np.round(val_loss, 3))

파인튜닝 결과

```
Loss of each epoch: [0.638 0.584 0.526 0.474 0.435 0.386 0.376 0.353 0.354 0.339 0.303 0.313
 0.309 0.277 0.278 0.286 0.244 0.284 0.253 0.249 0.25  0.247 0.251 0.205 0.22  0.216 0.236 0.219
0.227 0.212 0.199 0.211 0.191 0.197 0.196 0.203  0.19  0.179 0.191 0.17  0.184 0.211 0.167 0.189
0.178 0.162 0.171 0.16  0.175 0.155 0.167 0.162 0.156 0.141 0.158 0.159 0.161 0.157 0.158 0.15
 0.157 0.123 0.153 0.143 0.145 0.128 0.154 0.11  0.144 0.134 0.124 0.156 0.124 0.109 0.124 0.114
0.114 0.121 0.124 0.118 0.098 0.137 0.095 0.118 0.11 0.115 0.124 0.112 0.123 0.112 0.129 0.105 0.1
0.11  0.106 0.1 0.095 0.099 0.108 0.103]

Validation Loss of each epoch: [0.576 0.496 0.458 0.4   0.357 0.336 0.329 0.274 0.245 0.263 0.25  0.242
 0.24  0.241 0.198 0.229 0.252 0.216 0.252 0.2   0.182 0.204 0.212 0.225 0.189 0.19  0.201 0.248 0.159
 0.235 0.159 0.14  0.164 0.21  0.183 0.148 0.158 0.193 0.174 0.17  0.159 0.139 0.185 0.148 0.156 0.158
 0.148 0.127 0.123 0.161 0.143 0.144 0.145 0.145 0.156 0.144 0.148 0.193 0.197 0.18 0.162 0.207 0.131
 0.208 0.122 0.159 0.123 0.164 0.144 0.111 0.117 0.137 0.195 0.125 0.136 0.142 0.156 0.14  0.144 0.141
 0.169 0.123 0.125 0.145 0.187 0.116 0.139 0.129 0.181 0.116 0.128 0.159 0.123 0.119 0.123 0.131 0.205
 0.136 0.151 0.147]
```

정확도 그래프 확인

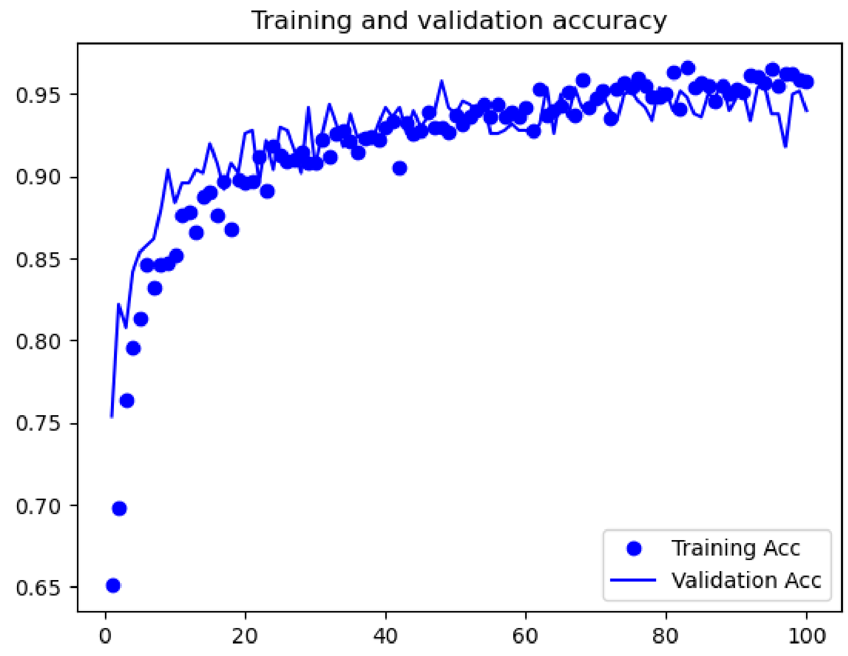
- import matplotlib.pyplot as plt

- epochs = range(1, len(acc) +1)

- plt.plot(epochs, acc, 'bo', label='Training Acc')
- plt.plot(epochs, val_acc, 'b', label='Validation Acc')
- plt.title('Training and validation accuracy')
- plt.legend()

- plt.show()

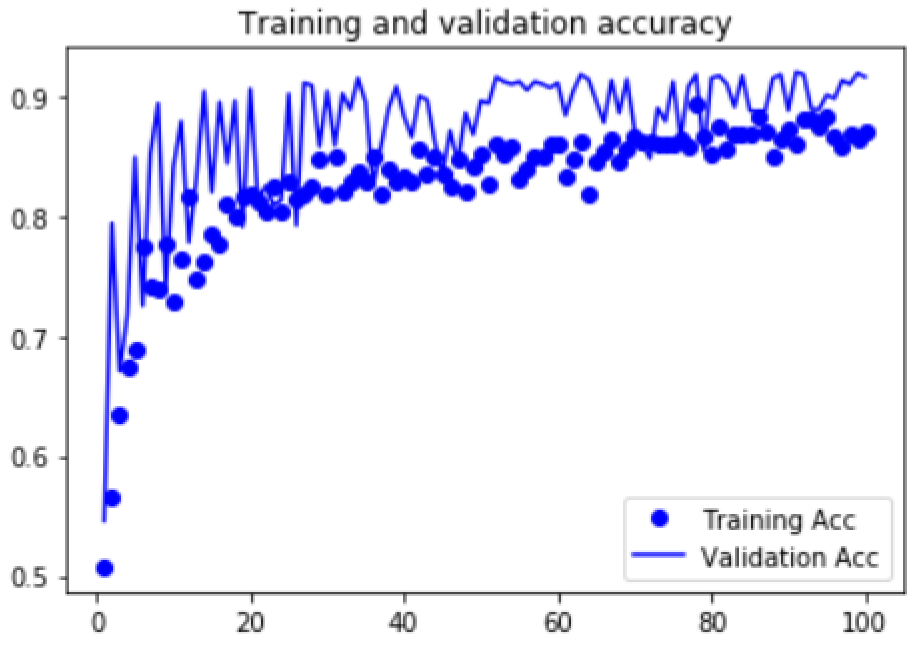
파인튜닝 결과



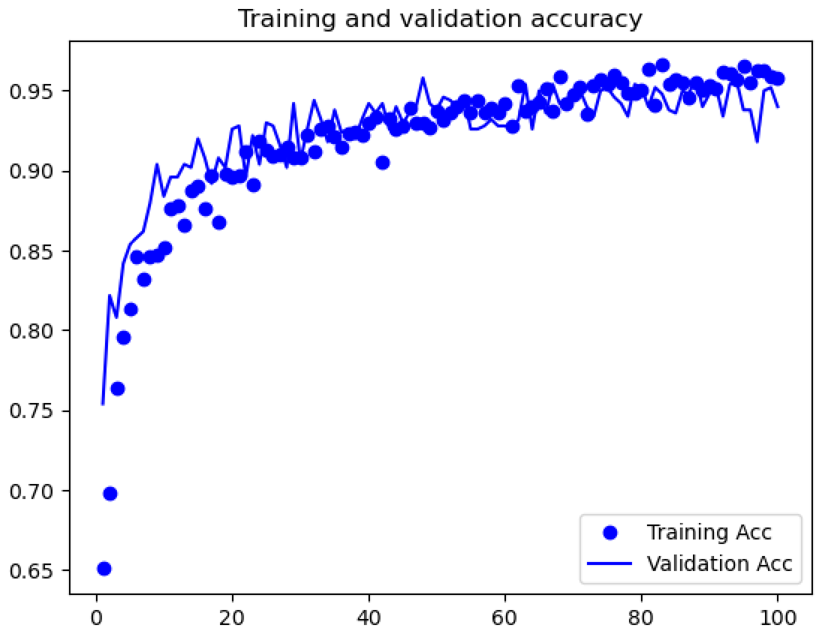
정확도 그래프 비교

- 전이학습과 파인튜닝 비교

전이학습 모델



파인튜닝 모델



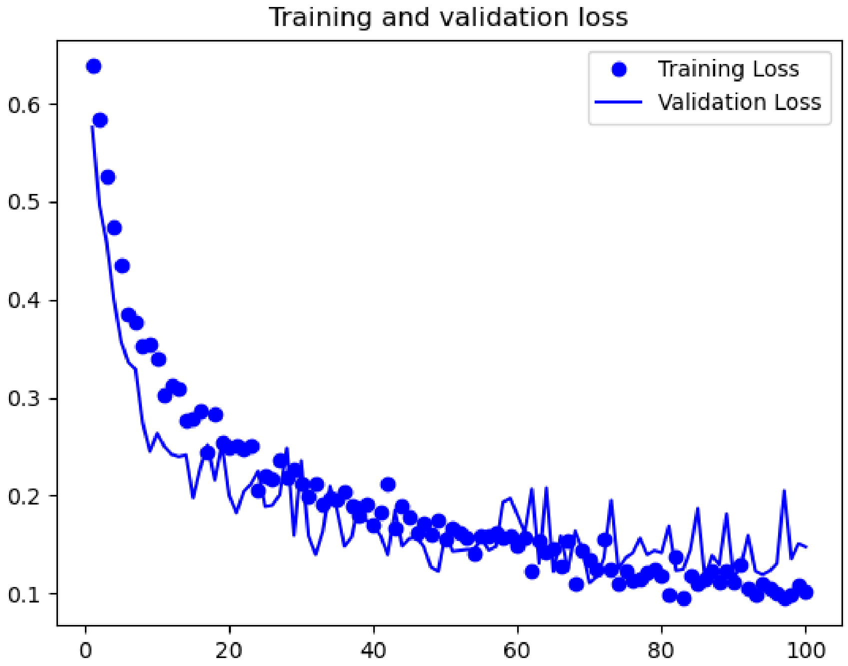
손실값 그래프 확인

- plt.figure()

- plt.plot(epochs, loss, 'bo', label='Training Loss')
- plt.plot(epochs, val_loss, 'b', label='Validation Loss')
- plt.title('Training and validation loss')
- plt.legend()

- plt.show()

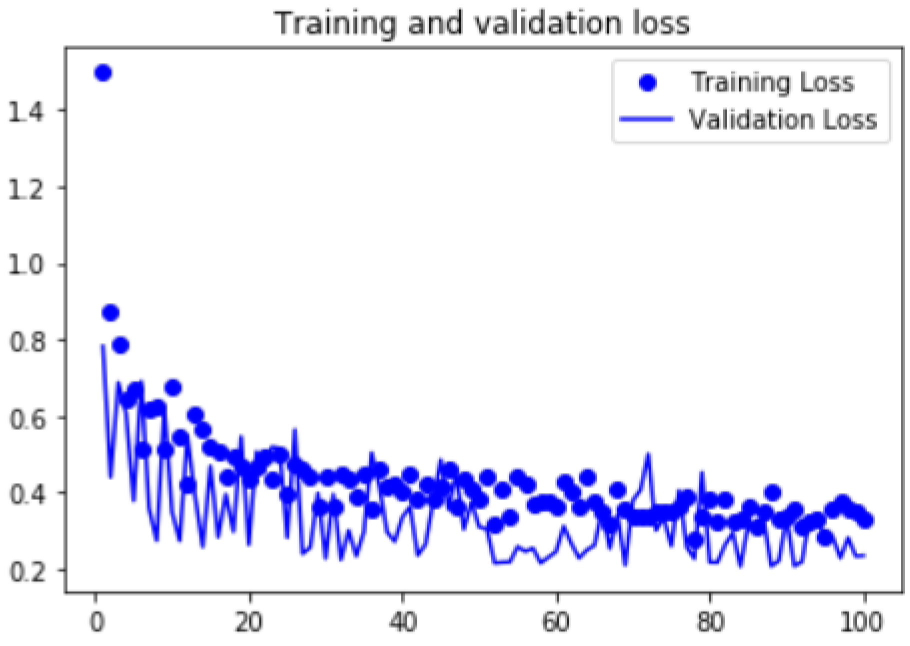
파인튜닝 결과



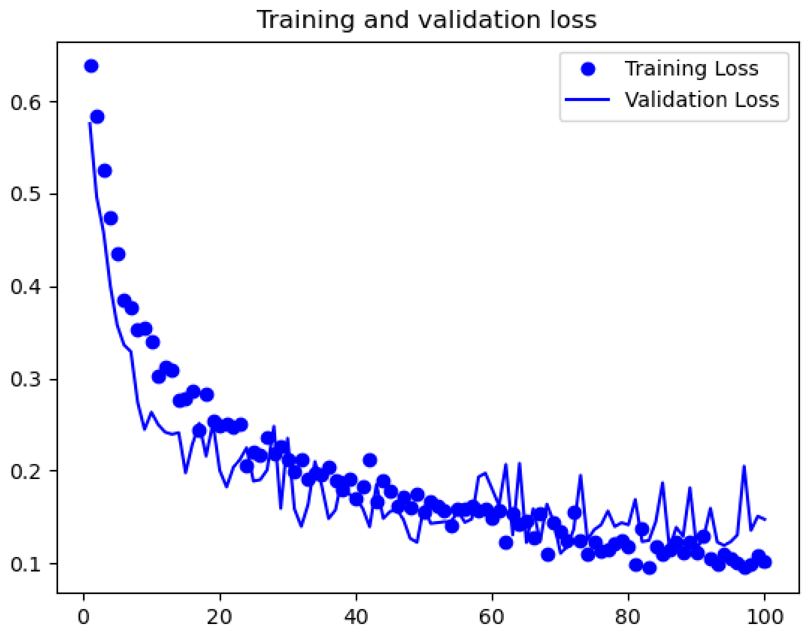
손실값 그래프 비교

- 전이학습과 파인튜닝 비교

전이학습 모델



파인튜닝 모델



테스트 데이터 평가

- model.evaluate(test_generator)

```
loss: 0.1343 - acc: 0.9430
```

loss 0.237 → 0.134
accuracy 0.907 → 0.943
로 파인튜닝의 경우가 더 좋았다