전이학습

최석재 *lingua@naver.com*

사전 학습 모델 *Pre-Training Model*

- 사전 학습 모델이란 대규모 데이터와 모델로 사전에 학습을 마친 모델
- 네트워크 가중치가 학습되어 있기 때문에 많은 경우 좋은 성능을 낸다

- 사전 학습 모델의 두 가지 종류
  - 파인튜닝과 전이학습의 구분이 명확치 않아졌다
  - 최근에는 파인튜닝은 백본 모델(파운데이션 모델)에 데이터를 추가하여 업데이트하는 것을
  - 전이학습은 학습된 파라미터를 그대로 사용하고, 출력 방법만 바꿔 다른 모델에 적용한다는 의미를 가진다

  - 또는 파인튜닝은 특정 작업에 맞도록 모델을 훈련하는 것을,
  - 전이학습은 사전학습된 모델을 다른 종류의 예측에 사용하는 것을 가리키기도 한다

  - 이보다 전에는 추가 학습하는 부분이 어디인지에 따라 구분하였다
  - 여기서는 일단 다음의 정의를 따르기로 하나, 굳이 둘을 구분하지 않고 파인튜닝이라고 해도 무방하다

사전 학습 모델 *Pre-Training Model*

- 다음과 같이 정리한다

① 파인튜닝 fine tuning
  - 사전 학습된 모델의 네트워크 가중치를 초깃값으로 사용
  - 이 초깃값을 바탕으로 모델을 처음 또는 중간층부터 학습

② 전이학습 transfer learning
  - 사전 학습된 모델의 네트워크 가중치를 최종 고정된 것으로 사용
  - 다만, 현재 데이터에 맞게 모델의 마지막 부분만 수정하고, 학습
  - 일부 가중치만 수정하므로 빠르게 학습된다

사전학습모델의 전부가 아닌, 일부층부터 학습하는 경우도 파인튜닝이라고 한다
- 먼저 학습된 층들은 일반적인 특징을 갖는 반면, 나중에 학습되는 층들은 보다 특화된 특성을 학습하기 때문에,
- 새로운 문제에 재활용할 때는 나중에 학습되는 층들만 조정하는 것이 유리하다
- 먼저 학습된 층들로 올라갈수록 파인 튜닝에 대한 효과가 감소한다

전이학습

- 전통적으로 학습 알고리즘은 특정 과제를 해결하기 위하여 고안되었다
- 그러나 해결해야 하는 과제는 매우 다양하고 많으며,
- 그때마다 대량의 데이터를 확보하여 새로 모델을 학습해야 한다는 것은 큰 부담

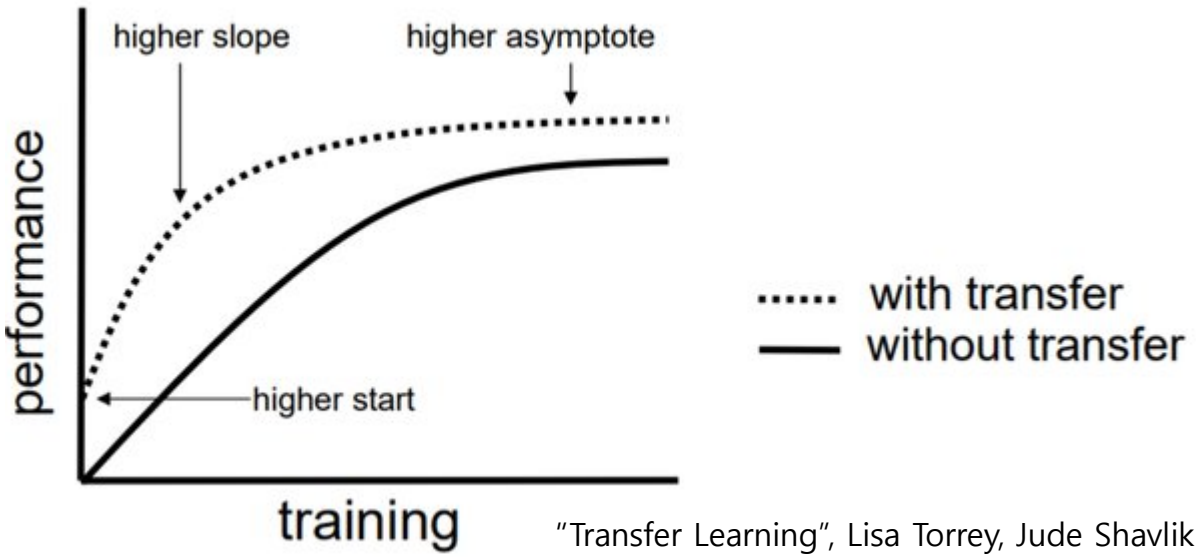
- 사람은 서로 다른 과제 간에 지식을 전이하는 능력이 있다
- 어떤 과제를 통해 배운 지식이 있다면 이를 활용하여 관련된 작업을 해결할 수 있다
- 그렇게 하는 것이 더 쉽고, 더 빠르고, 더 좋은 결과를 낸다

- 전이학습의 아이디어는 이와 같이 고립된 학습 패러다임을 극복하고
- 하나의 과제에서 습득한 지식을 활용해 관련된 새로운 과제를 해결하자는 것

전이학습의 효과

- 전이학습은 사람의 경우와 비슷하게 더 빠른 시간 안에 더 좋은 결과를 낳는다
- 전이학습을 사용한 경우와 그렇지 않은 경우를 비교해보면,

- 우선 학습 초기부터 성능이 좋고 (higher start)
- 학습에 따른 성능 향상 속도가 빠르며 (higher slope)
- 최종 성능도 더 좋다 (higher asymptote)



"Transfer Learning", Lisa Torrey, Jude Shavlik (2009)

언제 동결할 것인가

- 파인튜닝과 전이학습의 용어가 혼용되어 쓰이는 경우가 많은데,
- 최근에는 파인튜닝은 특정 작업에 맞도록 모델을 훈련하는 것을,
- 전이학습은 사전학습된 모델을 다른 종류의 예측에 사용하는 것을 가리킬 때가 많다
- 중요한 것은 학습된 가중치를 어느 단계에서 동결할 것인가이다

- 만약 적용하려는 도메인이 사전학습된 모델과 유사도가 높으면,
- 최종 분류층에 가까운 곳에서 동결하면 된다 (즉, 대부분의 층을 동결)

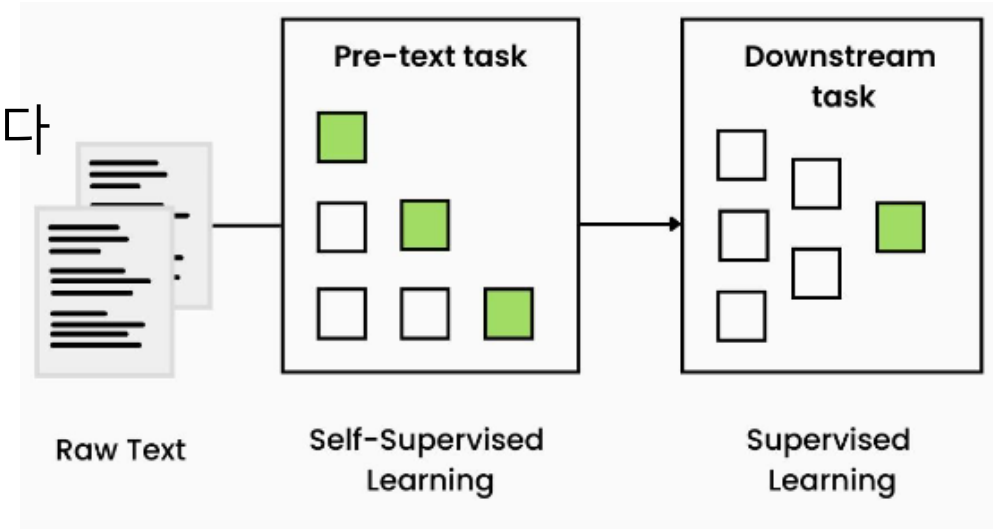
- 만약 유사도가 낮으며, 학습할 데이터가 많이 있다면
- 거의 동결하지 않고, 학습된 가중치로부터 시작하여 전체 층을 학습하면 된다

Downstream Task

- 사전 학습시 사용된 문제를 사전 학습 문제 pre-train task라 하고,
- 사전 학습한 가중치를 활용해 학습하고자 하는 문제를 하위 문제 downstream task라 한다

- 예를 들어, ImageNet과 같은 대규모 데이터로 이미지에 대한 이해 즉, 가중치를 얻었고,
- 이 가중치로부터 시작하여 교통 사고 감지 모델을 만들었다면

- 대규모 데이터 학습이 pre-train task가 되고,
- 교통 사고 감지를 위한 학습이 downstream task가 된다



케라스 사전학습 모델

- 사전 학습 모델로 이미지 분류

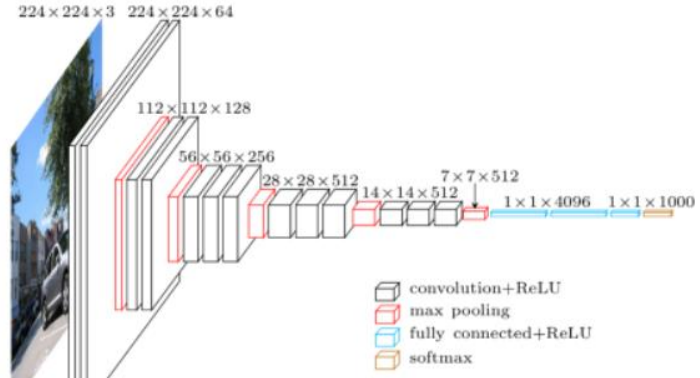
  - Keras는 이미지 분류용의 다양한 사전 학습 모델을 가지고 있다
  - 이들은 대부분 1,000 class를 갖는 ImageNet 데이터로부터 학습되었다

  - 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

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.

ImageNet은 (224, 224) 사이즈로 되어 있어 딥러닝 모델의 표준 크기로 사용된다

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')

※ 데이터는 cats_dogs를 사용하며,
full data는 _original_jpg_backup 폴더에 있음

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

- 앞에서 충분한 결과를 보지못했던 cats_dogs 파일을 다시 사용한다

- 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 batch마다 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 기대한다

- 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 공간 정보를 소실하여 이미지의 특징을 잘 포착하지 못한다

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

```
from tensorflow.keras.applications.vgg19 import VGG19
model = VGG19()
model.summary()          # 모델 구조 출력
```

모델 설계

include_top=False를 통해 완전연결층을 제거하고, 사전학습된 가중치를 가져온다

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

모델의 모든 층에 대하여 훈련이 되지 않도록 한다

- for layer in vgg.layers:
  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 sigmoid. 다중분류에서는 softmax를 사용한다

- prediction = Dense(1, activation='sigmoid')(x) ← 현재의 Dense 층만 학습된다

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
_________________________________________________________________
```

모델 컴파일

- model.compile(loss='binary_crossentropy', optimizer='rmsprop', 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))

```
Accuracy of each epoch: [0.507 0.566 0.636 0.674 0.688 0.775 0.741 0.738 0.776 0.728 0.763 0.816
 0.748 0.762 0.784 0.777 0.811 0.8   0.816 0.818 0.812 0.804 0.825 0.804
 0.829 0.814 0.818 0.826 0.848 0.817 0.85  0.82  0.828 0.836 0.829 0.85
 0.817 0.84  0.828 0.834 0.829 0.855 0.834 0.849 0.835 0.825 0.849 0.82
 0.84  0.851 0.826 0.86  0.852 0.858 0.831 0.839 0.85  0.85  0.859 0.861
 0.834 0.848 0.862 0.818 0.845 0.854 0.864 0.845 0.855 0.865 0.862 0.862
 0.861 0.861 0.859 0.864 0.858 0.893 0.866 0.853 0.875 0.855 0.868 0.869
 0.868 0.882 0.87  0.85  0.864 0.873 0.861 0.88  0.88  0.874 0.883 0.866
 0.858 0.868 0.864 0.869]

Validation Accuracy of each epoch: [0.546 0.794 0.671 0.717 0.849 0.725 0.852 0.894 0.739 0.841 0.879 0.778
 0.832 0.904 0.82  0.895 0.844 0.896 0.791 0.906 0.813 0.825 0.809 0.813
 0.902 0.792 0.911 0.909 0.858 0.904 0.859 0.902 0.889 0.915 0.895 0.825
 0.858 0.89  0.908 0.883 0.866 0.9   0.897 0.862 0.835 0.871 0.845 0.886
 0.868 0.896 0.894 0.916 0.912 0.91  0.912 0.905 0.912 0.91  0.907 0.911
 0.884 0.904 0.918 0.914 0.896 0.878 0.913 0.886 0.914 0.872 0.86  0.848
 0.89  0.879 0.912 0.865 0.908 0.918 0.854 0.915 0.917 0.91  0.891 0.917
 0.888 0.887 0.888 0.915 0.918 0.888 0.92  0.918 0.889 0.889 0.901 0.898
 0.913 0.91  0.919 0.916]
```

손실값 확인

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

```
Loss of each epoch: [1.5   0.871 0.787 0.642 0.67  0.512 0.616 0.622 0.515 0.678 0.549 0.423
 0.604 0.566 0.518 0.51  0.44  0.492 0.467 0.435 0.467 0.496 0.434 0.503
 0.397 0.473 0.462 0.444 0.363 0.438 0.361 0.446 0.433 0.387 0.449 0.358
 0.461 0.413 0.418 0.401 0.449 0.379 0.419 0.38  0.413 0.464 0.365 0.436
 0.406 0.383 0.44  0.316 0.411 0.337 0.441 0.424 0.367 0.376 0.375 0.363
 0.429 0.399 0.365 0.442 0.374 0.351 0.314 0.408 0.355 0.334 0.335 0.336
 0.347 0.353 0.349 0.36  0.389 0.277 0.337 0.384 0.321 0.385 0.326 0.327
 0.365 0.311 0.348 0.403 0.331 0.336 0.358 0.309 0.326 0.333 0.286 0.355
 0.376 0.353 0.351 0.328]

Validation Loss of each epoch: [0.782 0.439 0.687 0.608 0.378 0.69  0.358 0.274 0.623 0.349 0.273 0.548
 0.389 0.257 0.468 0.283 0.395 0.297 0.546 0.263 0.506 0.455 0.518 0.515
 0.281 0.564 0.239 0.256 0.397 0.227 0.395 0.224 0.3   0.233 0.298 0.503
 0.417 0.296 0.271 0.335 0.362 0.235 0.263 0.374 0.485 0.356 0.449 0.303
 0.375 0.309 0.304 0.215 0.217 0.218 0.258 0.245 0.253 0.215 0.23  0.245
 0.311 0.265 0.228 0.249 0.261 0.354 0.253 0.34  0.209 0.379 0.406 0.5
 0.3   0.334 0.259 0.404 0.257 0.226 0.452 0.217 0.216 0.263 0.293 0.206
 0.316 0.317 0.332 0.207 0.221 0.335 0.207 0.218 0.337 0.344 0.267 0.283
 0.228 0.281 0.233 0.234]
```

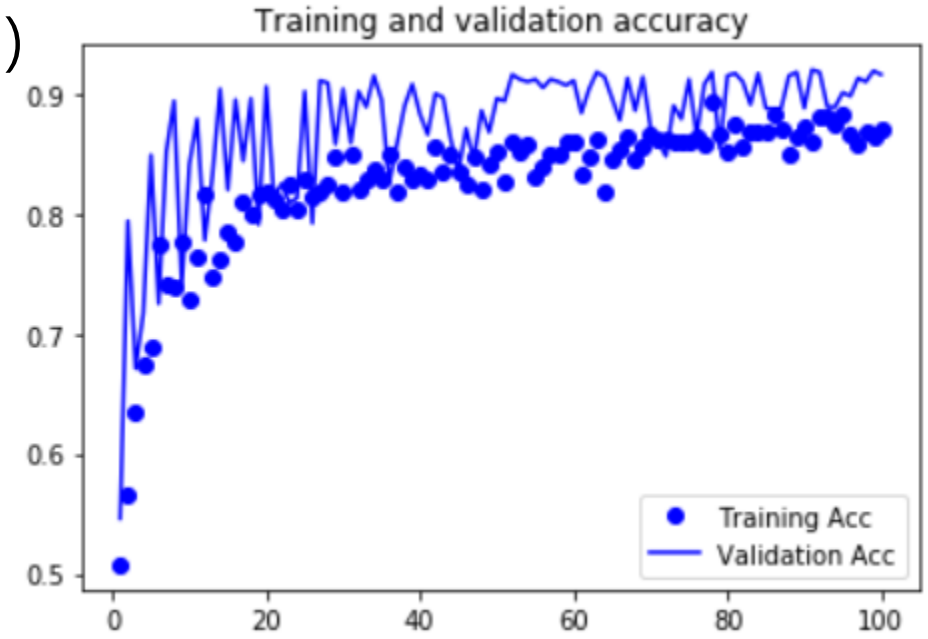
정확도 그래프 확인

- 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()

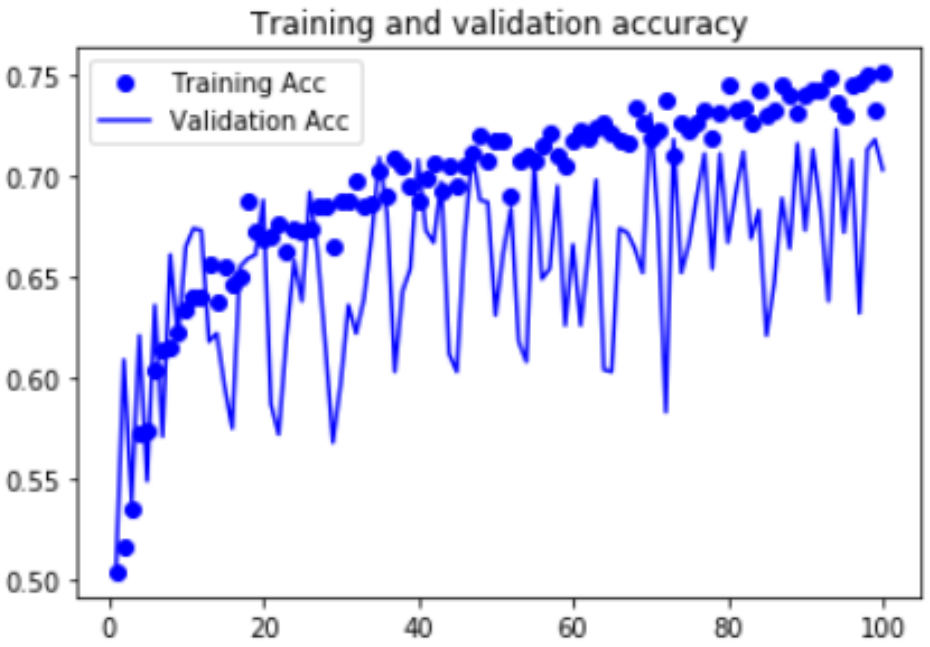
- plt.show()



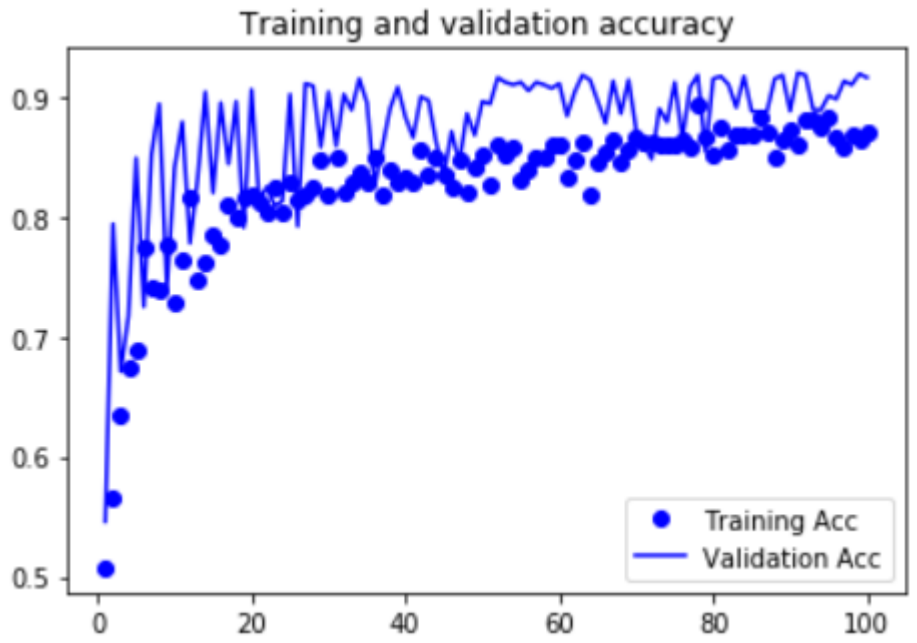
정확도 그래프 비교

- 사전학습 모델 사용 전과 후 비교

일반 CNN 모델



사전학습 모델



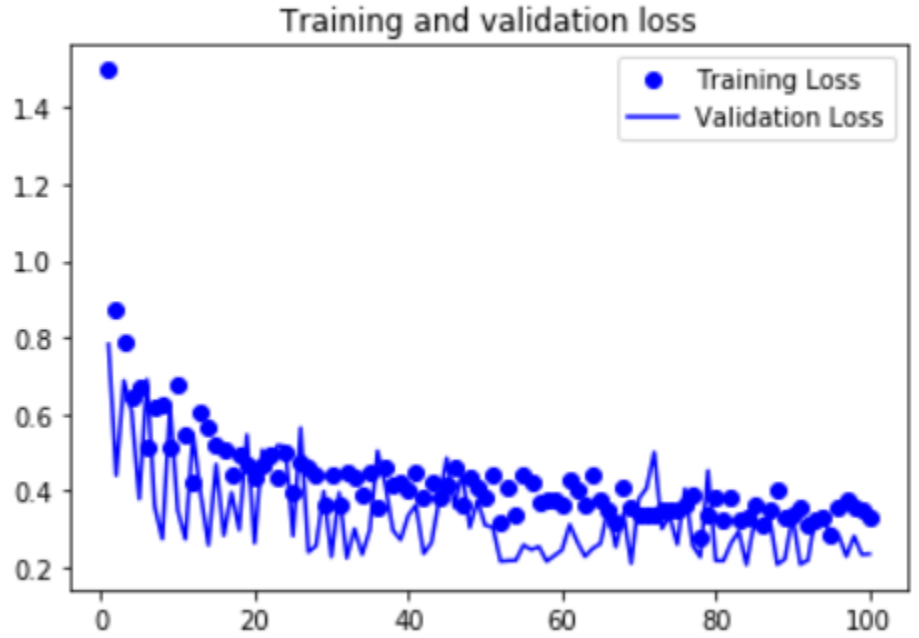
validation 데이터 기준으로 그래프가 더 안정된 것은 물론,
성능이 더 높아졌다

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()

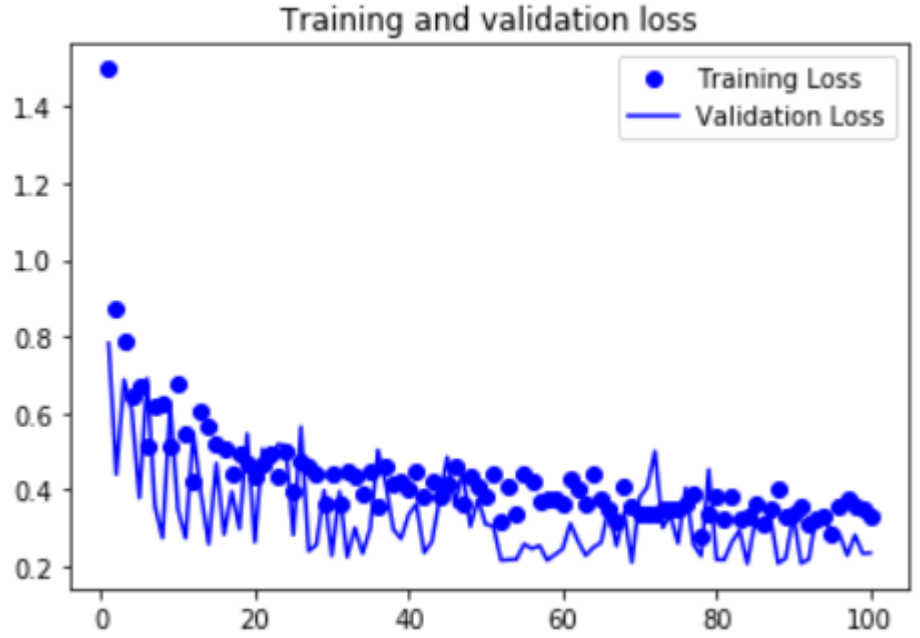


손실값 그래프 비교

- 사전학습 모델 사용 전과 후 비교

일반 CNN 모델

사전학습 모델



validation 데이터 기준으로 그래프가 더 안정된 것은 물론, 성능이 더 높아졌다

테스트 데이터 평가 및 예측

- model.evaluate(test_generator)

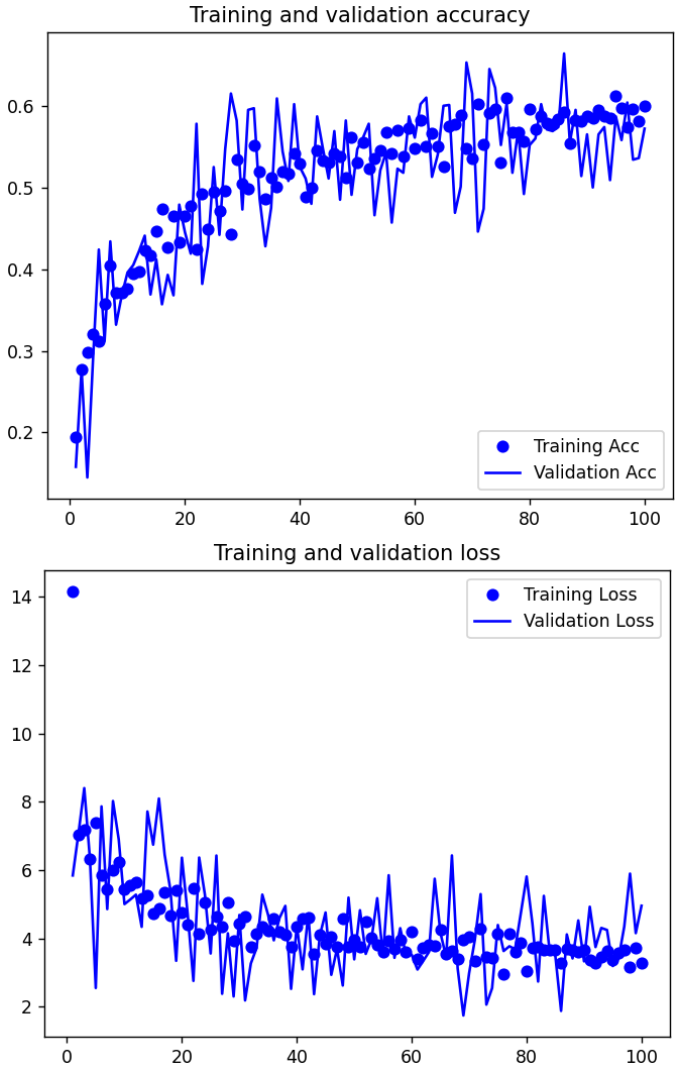
```
[0.23717114329338074, 0.9079999923706055]
```

- model.predict(test_generator)

```
array([[9.87694502e-01],
       [1.00000000e+00],
       [2.06154755e-06],
       [9.99999762e-01],
       [2.05078311e-02],
       [5.67057577e-04],
       [9.99999762e-01],
       [7.71910744e-03],
       [3.51411989e-04],
       [9.99968052e-01],
       [9.99994755e-01],
       [9.95376229e-01],
       [2.26524435e-06],
       [9.99191225e-01],
       [5.14927983e-01]
```

연습문제

- ResNet50을 사용하여 Intel_Image_Classification을 분류
  - 문서를 보고 입력 사이즈 확인
  - 다중 분류이므로 이에 맞게 전이학습 모델 구성
  - ImageDataGenerator.flow_from_directory() 사용
  - 데이터 증강은 하지 않음
  - 파일이 많으므로 Local PC에서 진행
  - 단, pytest_img > Intel_image_Classification 폴더에 소량의 데이터가 있음
  - 앞에서 만든 CNN 모델과 성능을 비교해보시오



패키지 임포트는 `from tensorflow.keras.applications.vgg19 import VGG19` 와 같이 VGG19가 아닌 소문자 `vgg19`로 임포트해야 한다