

Exploitation of Pretrained Models

Quang-Vinh Dinh
Ph.D. in Computer Science

Outline

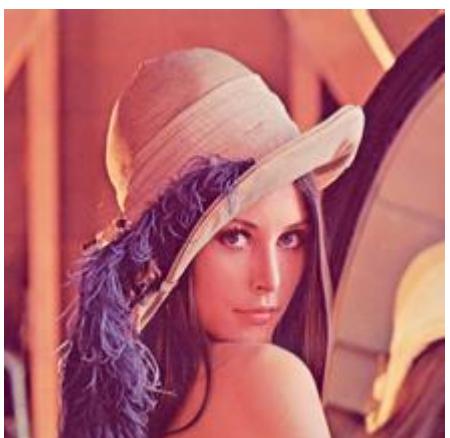
- Review
- Reuse a Pre-trained Model
- Case Study 1
- Case Study 2
- Case Study 3
- Case Study 4
- Case Study 5

Global Pooling

Max pooling: Features are preserved

v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

Data



2x2 max
pooling (

v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

) =

m_1	m_2
m_3	m_4

$$m_1 = \max(v_1, v_2, v_5, v_6)$$

$$m_2 = \max(v_3, v_4, v_7, v_8)$$

$$m_3 = \max(v_9, v_{10}, v_{13}, v_{14})$$

$$m_4 = \max(v_{11}, v_{12}, v_{15}, v_{16})$$

keras.layers.MaxPooling2D(pool_size=2)



Feature map (220x220)

max pooling
(2x2) \rightarrow



Feature map
(110x110)

max pooling
(2x2) \rightarrow



Feature map
(55x55)

Global Pooling

Max pooling

v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

Data

2x2 max
pooling (

v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

) =

m_1	m_2
m_3	m_4

$$m_1 = \max(v_1, v_2, v_5, v_6)$$

$$m_2 = \max(v_3, v_4, v_7, v_8)$$

$$m_3 = \max(v_9, v_{10}, v_{13}, v_{14})$$

$$m_4 = \max(v_{11}, v_{12}, v_{15}, v_{16})$$

Global max pooling

v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

Data

global max
pooling (

v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

) =

m

$$m = \max(v_1, v_2, \dots, v_{16})$$

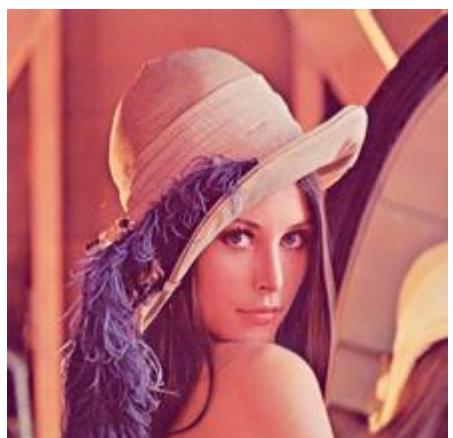
`keras.layers.GlobalMaxPool2D()`

Global Pooling

Average pooling: Features are preserved

v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

Data



average
pooling (

v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

m_1	m_2
m_3	m_4

$$m_1 = \text{mean}(v_1, v_2, v_5, v_6)$$

$$m_2 = \text{mean}(v_3, v_4, v_7, v_8)$$

$$m_3 = \text{mean}(v_9, v_{10}, v_{13}, v_{14})$$

$$m_4 = \text{mean}(v_{11}, v_{12}, v_{15}, v_{16})$$

keras.layers.AveragePooling2D (pool_size=2)



Feature map (220x220)

Average
Pooling (2x2) \rightarrow



Feature map
(110x110)

Average
Pooling (2x2) \rightarrow



Feature map
(55x55)

Global Pooling

Average pooling

v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

Data

average
pooling (

v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

) =

m_1	m_2
m_3	m_4

$$m_1 = \text{mean}(v_1, v_2, v_5, v_6)$$

$$m_2 = \text{mean}(v_3, v_4, v_7, v_8)$$

$$m_3 = \text{mean}(v_9, v_{10}, v_{13}, v_{14})$$

$$m_4 = \text{mean}(v_{11}, v_{12}, v_{15}, v_{16})$$

Global average pooling

v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

Data

global
average
pooling (

v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

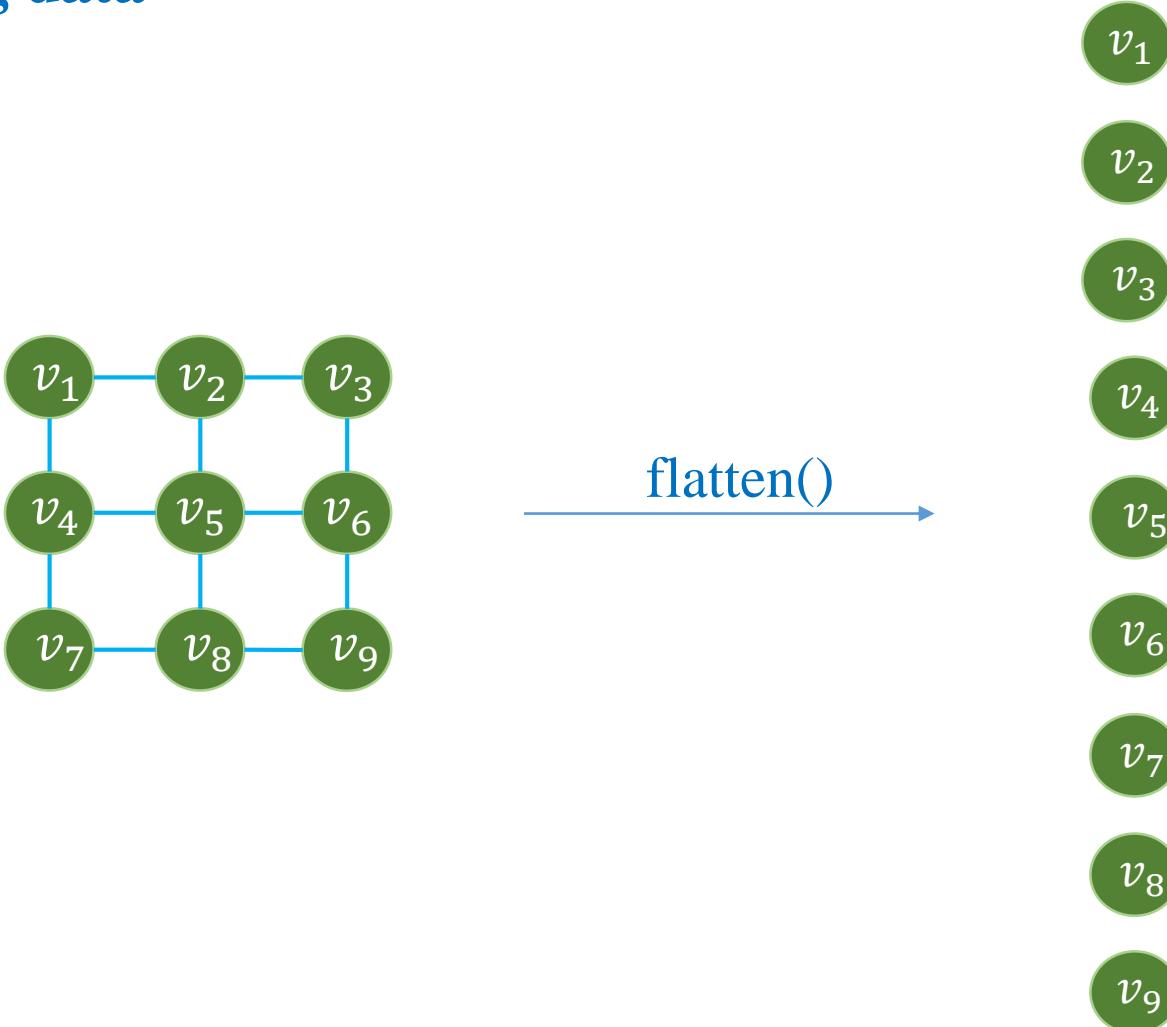
) =

$$m = \text{average}(v_1, v_2, \dots, v_{16})$$

keras.layers.GlobalAveragePooling2D()

Flattening

❖ Flattening data



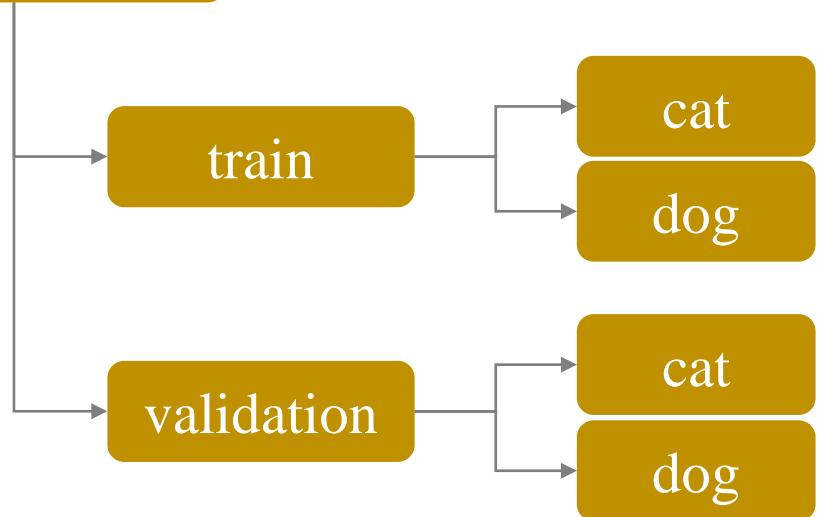
Outline

- Review
- Reuse a Pre-trained Model
- Case Study 1
- Case Study 2
- Case Study 3
- Case Study 4
- Case Study 5

Data Processing

❖ Cat-Dog dataset

cats_and_dogs

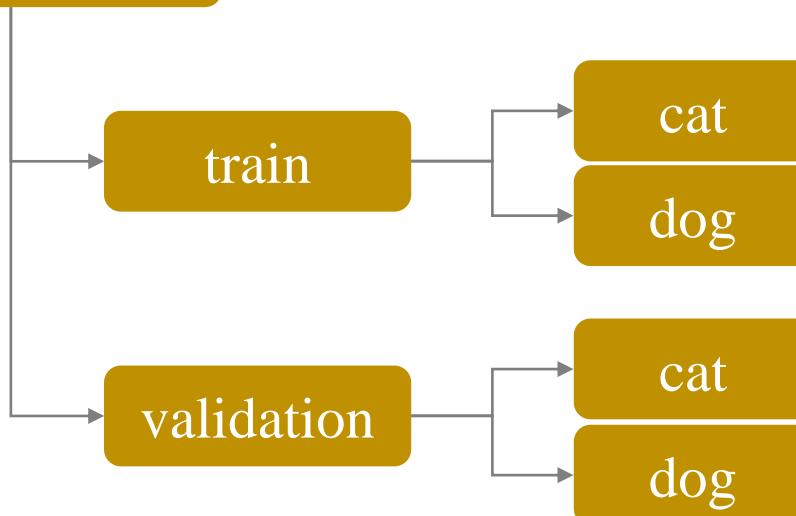


Data Processing

❖ In keras

```
tf.keras.preprocessing.image_dataset_from_directory (  
    directory,  
    batch_size=32,  
    image_size=(256, 256),  
    )
```

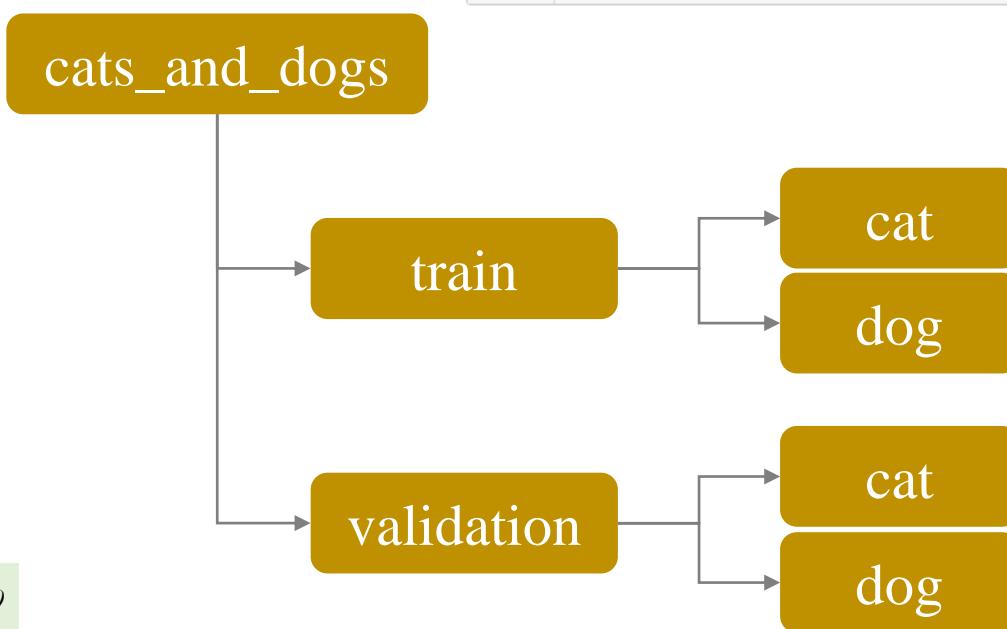
cats_and_dogs



Data Processing

❖ In keras

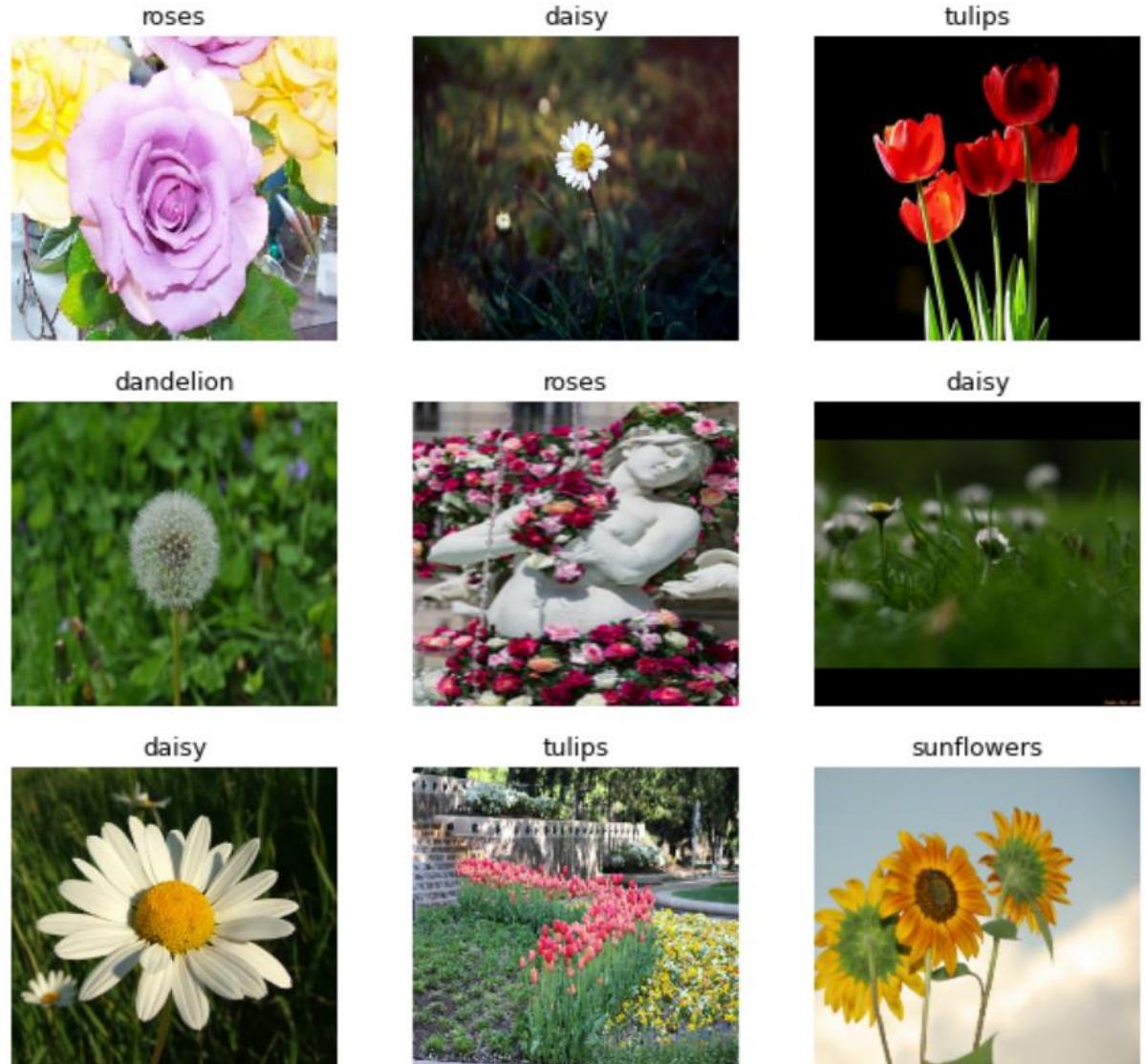
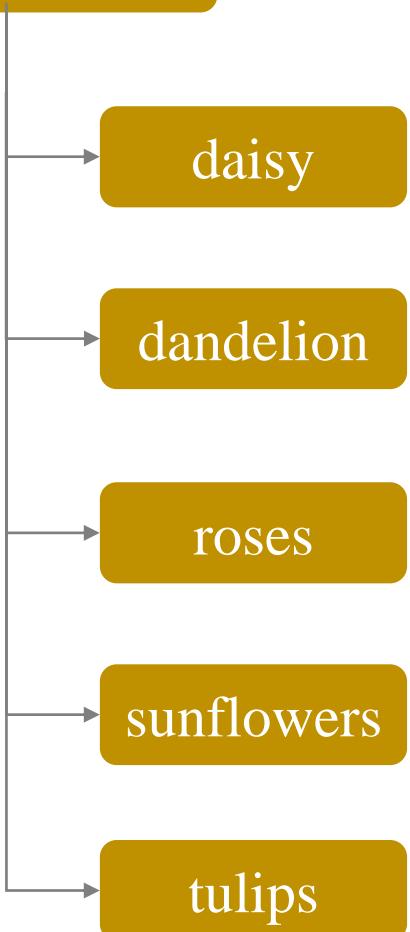
```
1 # training data
2 train_dataset = image_dataset_from_directory('cats_and_dogs/train',
3                                              shuffle=True,
4                                              batch_size=256,
5                                              image_size=(160, 160))
6
7 # validation data
8 validation_dataset = image_dataset_from_directory('cats_and_dogs/validation',
9                                              shuffle=True,
10                                             batch_size=256,
11                                             image_size=(160, 160))
```



Data Processing

❖ In keras

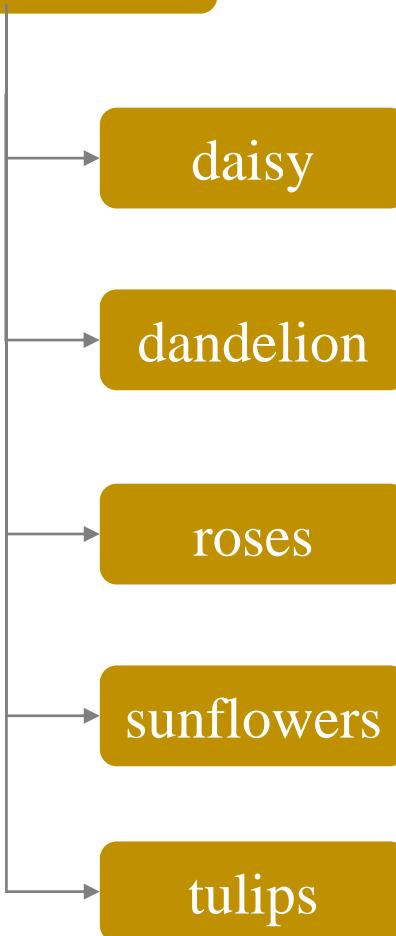
flower_photos



Data Processing

❖ In keras

flower_photos

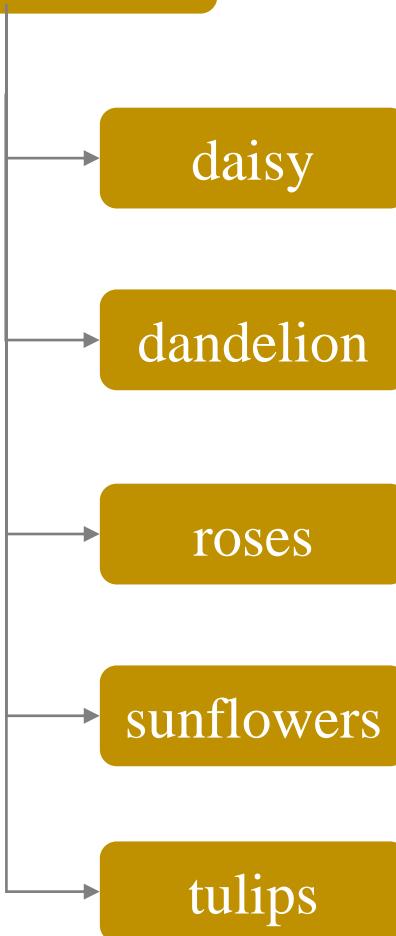


```
tf.keras.preprocessing.image_dataset_from_directory (  
    directory,  
    batch_size=32,  
    image_size=(256, 256),  
    validation_split=None,  
    subset=None  
)
```

Data Processing

❖ In keras

flower_photos



```
1 # training data
2 train_dataset = tf.keras.preprocessing.image_dataset_from_directory(
3
4
5
6
7
8
9
10 # validation data
11 validation_dataset = tf.keras.preprocessing.image_dataset_from_directory(
12
13
14
15
16
17
```

Outline

- Review
- Reuse a Pre-trained Model
- Case Study 1
- Case Study 2
- Case Study 3
- Case Study 4
- Case Study 5

Data Processing Layer

❖ In Keras

`tf.keras.layers.experimental.preprocessing`

`preprocessing.CenterCrop`

`preprocessing.Normalization`

`preprocessing.RandomFlip`

`preprocessing.RandomRotation`

`preprocessing.RandomTranslation`

`preprocessing.Rescaling`

Data Processing Layer

❖ In Keras

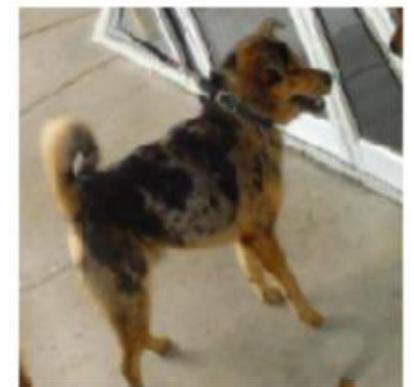
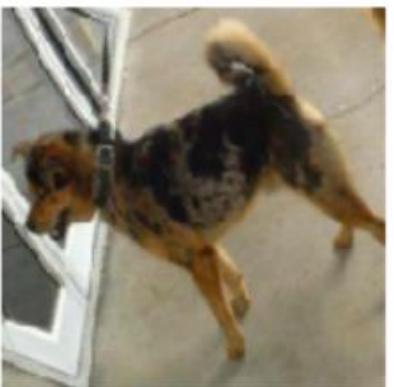
```
1 # data augmentation
2 data_augmentation = tf.keras.Sequential([
3     tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal')
4 ])
5
6 # training
7 for images, _ in train_dataset:
8     augmented_image = data_augmentation(images)
```



Data Processing Layer

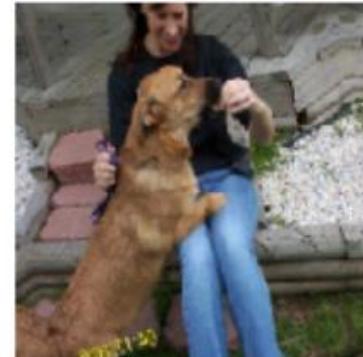
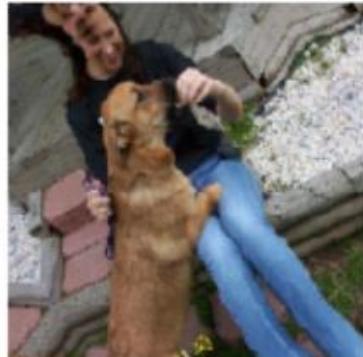
❖ In Keras

```
1 # data augmentation
2 data_augmentation = tf.keras.Sequential([
3     tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
4     tf.keras.layers.experimental.preprocessing.RandomRotation(0.2)
5 ])
6
7 # training
8 for images, _ in train_dataset:
9     augmented_image = data_augmentation(images)
```



Data Processing Layer

```
1 # data augmentation
2 data_augmentation = tf.keras.Sequential([
3     tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
4     tf.keras.layers.experimental.preprocessing.RandomRotation(0.2)
5 ])
6
7 # training
8 for images, _ in train_dataset.take(1):
9     # show images
10    plt.figure(figsize=(18, 15))
11    for i in range(6):
12        # data augmentation
13        augmented_image = data_augmentation(images[0:1]/255.)
14
15        ax = plt.subplot(1, 6, i+1)
16        plt.imshow(augmented_image[0])
17        plt.axis('off')
```



Data Processing Layer

```
1 # data augmentation
2 data_augmentation = tf.keras.Sequential([
3     tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
4     tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
5     tf.keras.layers.experimental.preprocessing.Rescaling(1./255., offset=0.)
6 ])
7
8 # training
9 for images, _ in train_dataset.take(1):
10     # show images
11     plt.figure(figsize=(18, 15))
12     for i in range(6):
13         # data augmentation
14         augmented_image = data_augmentation(images[0:1])
15
16         ax = plt.subplot(1, 6, i+1)
17         plt.imshow(augmented_image[0])
18         plt.axis('off')
```



Outline

- Review
- Reuse a Pre-trained Model
- Case Study 1
- Case Study 2
- Case Study 3
- Case Study 4
- Case Study 5

Network Manipulation

❖ Network 1

```
1 # LeNet-like
2 import tensorflow as tf
3
4 # model architecture
5 model = tf.keras.Sequential()
6 # input shape (28,28,1)
7 model.add(tf.keras.Input(shape=(28, 28, 1)))
8
9 # convolution 1
10 model.add(tf.keras.layers.Conv2D(6, (3,3), padding='same', activation='relu'))
11 # max pooling 1
12 model.add(tf.keras.layers.MaxPooling2D(pool_size=2))
13
14 # convolution 2
15 model.add(tf.keras.layers.Conv2D(filters=16, kernel_size=5, activation='relu'))
16 # max pooling 2
17 model.add(tf.keras.layers.MaxPooling2D(pool_size=2))
18
19 # Flatten
20 model.add(tf.keras.layers.Flatten())
21
22 # fully connected
23 model.add(tf.keras.layers.Dense(120, activation='relu'))
24 model.add(tf.keras.layers.Dense(84, activation='relu'))
25 model.add(tf.keras.layers.Dense(10, activation='softmax'))
26
27 # model summary
28 model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
<hr/>		
conv2d (Conv2D)	(None, 28, 28, 6)	60
<hr/>		
max_pooling2d (MaxPooling2D)	(None, 14, 14, 6)	0
<hr/>		
conv2d_1 (Conv2D)	(None, 10, 10, 16)	2416
<hr/>		
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 16)	0
<hr/>		
flatten (Flatten)	(None, 400)	0
<hr/>		
dense (Dense)	(None, 120)	48120
<hr/>		
dense_1 (Dense)	(None, 84)	10164
<hr/>		
dense_2 (Dense)	(None, 10)	850
<hr/>		
Total params: 61,610		
Trainable params: 61,610		
Non-trainable params: 0		

Network Manipulation

❖ Network 1

```
1 # LeNet-like
2 import tensorflow as tf
3
4 # model architecture
5 model = tf.keras.Sequential()
6 # input shape (28,28,1)
7 model.add(tf.keras.Input(shape=(28, 28, 1)))
8
9 # convolution 1
10 model.add(tf.keras.layers.Conv2D(6, (3,3), padding='same', activation='relu'))
11 # max pooling 1
12 model.add(tf.keras.layers.MaxPooling2D(pool_size=2))
13
14 # convolution 2
15 model.add(tf.keras.layers.Conv2D(filters=16, kernel_size=5, activation='relu'))
16 # max pooling 2
17 model.add(tf.keras.layers.MaxPooling2D(pool_size=2))
18
19 # Flatten
20 model.add(tf.keras.layers.Flatten())
21
22 # fully connected
23 model.add(tf.keras.layers.Dense(120, activation='relu'))
24 model.add(tf.keras.layers.Dense(84, activation='relu'))
25 model.add(tf.keras.layers.Dense(10, activation='softmax'))
26
27 # model summary
28 model.summary()
```

```
1 print(type(model.layers))
2 print(len(model.layers))

<class 'list'>
8

1 for layer in model.layers:
2     print(layer.name, '-', layer.trainable)

conv2d_5 - True
max_pooling2d_4 - True
conv2d_6 - True
max_pooling2d_5 - True
flatten_2 - True
dense_6 - True
dense_7 - True
dense_8 - True
```

Network Manipulation

❖ Network 1

```
1 # LeNet-like
2 import tensorflow as tf
3
4 # model architecture
5 model = tf.keras.Sequential()
6 # input shape (28,28,1)
7 model.add(tf.keras.Input(shape=(28, 28, 1)))
8
9 # convolution 1
10 model.add(tf.keras.layers.Conv2D(6, (3,3), padding='same', activation='relu'))
11 # max pooling 1
12 model.add(tf.keras.layers.MaxPooling2D(pool_size=2))
13
14 # convolution 2
15 model.add(tf.keras.layers.Conv2D(filters=16, kernel_size=5, activation='relu'))
16 # max pooling 2
17 model.add(tf.keras.layers.MaxPooling2D(pool_size=2))
18
19 # Flatten
20 model.add(tf.keras.layers.Flatten())
21
22 # fully connected
23 model.add(tf.keras.layers.Dense(120, activation='relu'))
24 model.add(tf.keras.layers.Dense(84, activation='relu'))
25 model.add(tf.keras.layers.Dense(10, activation='softmax'))
26
27 # model summary
28 model.summary()
```

```
1 model.trainable = False
2 for layer in model.layers:
3     print(layer.name, '-', layer.trainable)
```

```
conv2d_5 - False
max_pooling2d_4 - False
conv2d_6 - False
max_pooling2d_5 - False
flatten_2 - False
dense_6 - False
dense_7 - False
dense_8 - False
```

Network Manipulation

❖ Network 1

```
1 # LeNet-like
2 import tensorflow as tf
3
4 # model architecture
5 model = tf.keras.Sequential()
6 # input shape (28,28,1)
7 model.add(tf.keras.Input(shape=(28, 28, 1)))
8
9 # convolution 1
10 model.add(tf.keras.layers.Conv2D(6, (3,3), padding='same', activation='relu'))
11 # max pooling 1
12 model.add(tf.keras.layers.MaxPooling2D(pool_size=2))
13
14 # convolution 2
15 model.add(tf.keras.layers.Conv2D(filters=16, kernel_size=5, activation='relu'))
16 # max pooling 2
17 model.add(tf.keras.layers.MaxPooling2D(pool_size=2))
18
19 # Flatten
20 model.add(tf.keras.layers.Flatten())
21
22 # fully connected
23 model.add(tf.keras.layers.Dense(120, activation='relu'))
24 model.add(tf.keras.layers.Dense(84, activation='relu'))
25 model.add(tf.keras.layers.Dense(10, activation='softmax'))
26
27 # model summary
28 model.summary()
```

```
1 # number of layers
2 print('Number of layers is ', len(model.layers), '\n')
3
4 # freeze some layers
5 model.layers[0].trainable = False
6 model.layers[5].trainable = False
7
8 for layer in model.layers:
9     print(layer.name, '-', layer.trainable)
```

Number of layers is 8

conv2d - False
max_pooling2d - True
conv2d_1 - True
max_pooling2d_1 - True
flatten - True
dense - False
dense_1 - True
dense_2 - True

Network Manipulation

❖ Network 1

```
1 # LeNet-like
2 import tensorflow as tf
3
4 # model architecture
5 model = tf.keras.Sequential()
6 # input shape (28,28,1)
7 model.add(tf.keras.Input(shape=(28, 28, 1)))
8
9 # convolution 1
10 model.add(tf.keras.layers.Conv2D(6, (3,3), padding='same', activation='relu'))
11 # max pooling 1
12 model.add(tf.keras.layers.MaxPooling2D(pool_size=2))
13
14 # convolution 2
15 model.add(tf.keras.layers.Conv2D(filters=16, kernel_size=5, activation='relu'))
16 # max pooling 2
17 model.add(tf.keras.layers.MaxPooling2D(pool_size=2))
18
19 # Flatten
20 model.add(tf.keras.layers.Flatten())
21
22 # fully connected
23 model.add(tf.keras.layers.Dense(120, activation='relu'))
24 model.add(tf.keras.layers.Dense(84, activation='relu'))
25 model.add(tf.keras.layers.Dense(10, activation='softmax'))
26
27 # model summary
28 model.summary()
```

```
1 # number of layers
2 print('Number of layers is ', len(model.layers), '\n')
3
4 # freeze some last layers
5 for layer in model.layers[5:]:
6     layer.trainable = False
7
8 for layer in model.layers:
9     print(layer.name, '-', layer.trainable)
```

Number of layers is 8

conv2d - True
max_pooling2d - True
conv2d_1 - True
max_pooling2d_1 - True
flatten - True
dense - False
dense_1 - False
dense_2 - False

Network Manipulation

❖ Network 1

```
1 print('Number of tensors is',
2       len(model.trainable_variables))
3
4 for v in model.trainable_variables:
5     print(v.shape)
```

```
Number of tensors is 6
(3, 3, 1, 1)
(1,)
(3, 3, 1, 2)
(2,)
(2, 2)
(2,)
```

```
1 # A network
2 import tensorflow as tf
3
4 # model architecture
5 model = tf.keras.Sequential()
6 model.add(tf.keras.layers.Input(shape=(5, 5, 1)))
7 model.add(tf.keras.layers.Conv2D(1, (3,3), activation='relu'))
8 model.add(tf.keras.layers.Conv2D(2, (3,3), activation='relu'))
9 model.add(tf.keras.layers.Flatten())
10 model.add(tf.keras.layers.Dense(2, activation='softmax'))
11
12 # model summary
13 model.summary()
```

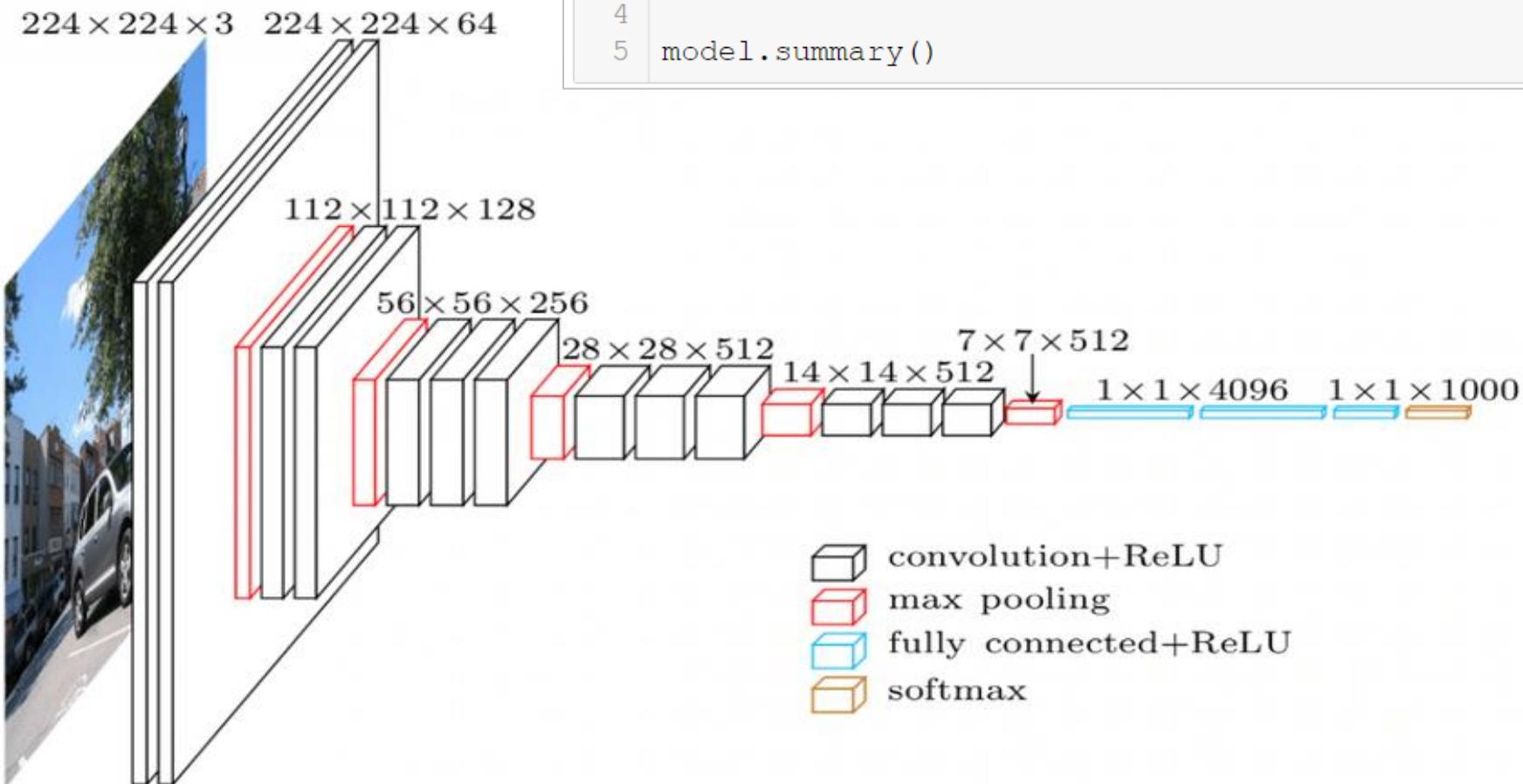
Model: "sequential"

Layer (type)	Output Shape	Param #
<hr/>		
conv2d (Conv2D)	(None, 3, 3, 1)	10
<hr/>		
conv2d_1 (Conv2D)	(None, 1, 1, 2)	20
<hr/>		
flatten (Flatten)	(None, 2)	0
<hr/>		
dense (Dense)	(None, 2)	6
<hr/>		

Total params: 36
Trainable params: 36
Non-trainable params: 0

Network Manipulation

❖ VGG16 Model



```
1 import tensorflow as tf
2 model = tf.keras.applications.VGG16(input_shape=(224, 224, 3),
3                                     include_top=True,
4                                     weights=None)
5 model.summary()
```

Network Manipulation

❖ VGG16 Model

```
1 base_model = tf.keras.applications.VGG16(input_shape=(160,160,3),  
2                                         include_top=False,  
3                                         weights=None)  
4 base_model.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
<hr/>		
input_1 (InputLayer)	[None, 160, 160, 3]	0
block1_conv1 (Conv2D)	(None, 160, 160, 64)	1792
block1_conv2 (Conv2D)	(None, 160, 160, 64)	36928
block1_pool (MaxPooling2D)	(None, 80, 80, 64)	0
block2_conv1 (Conv2D)	(None, 80, 80, 128)	73856
block2_conv2 (Conv2D)	(None, 80, 80, 128)	147584
block2_pool (MaxPooling2D)	(None, 40, 40, 128)	0
block3_conv1 (Conv2D)	(None, 40, 40, 256)	295168
block3_conv2 (Conv2D)	(None, 40, 40, 256)	590080
block3_conv3 (Conv2D)	(None, 40, 40, 256)	590080
block3_pool (MaxPooling2D)	(None, 20, 20, 256)	0

block4_conv1 (Conv2D)	(None, 20, 20, 512)	1180160
block4_conv2 (Conv2D)	(None, 20, 20, 512)	2359808
block4_conv3 (Conv2D)	(None, 20, 20, 512)	2359808
block4_pool (MaxPooling2D)	(None, 10, 10, 512)	0
block5_conv1 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv2 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv3 (Conv2D)	(None, 10, 10, 512)	2359808
block5_pool (MaxPooling2D)	(None, 5, 5, 512)	0
flatten (Flatten)	(None, 12800)	0
fc1 (Dense)	(None, 4096)	52432896
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000
<hr/>		
Total params: 88,025,896		
Trainable params: 88,025,896		
Non-trainable params: 0		

Network Manipulation

❖ VGG16 Model

```
1 model = tf.keras.applications.VGG16(input_shape=(160,160,3),  
2                                     include_top=True,  
3                                     weights=None)  
4 print(len(model.trainable_variables))
```

32

Model: "vgg16"

Layer (type)	Output Shape	Param #
<hr/>		
input_1 (InputLayer)	[None, 160, 160, 3]	0
block1_conv1 (Conv2D)	(None, 160, 160, 64)	1792
block1_conv2 (Conv2D)	(None, 160, 160, 64)	36928
block1_pool (MaxPooling2D)	(None, 80, 80, 64)	0
block2_conv1 (Conv2D)	(None, 80, 80, 128)	73856
block2_conv2 (Conv2D)	(None, 80, 80, 128)	147584
block2_pool (MaxPooling2D)	(None, 40, 40, 128)	0
block3_conv1 (Conv2D)	(None, 40, 40, 256)	295168
block3_conv2 (Conv2D)	(None, 40, 40, 256)	590080
block3_conv3 (Conv2D)	(None, 40, 40, 256)	590080
block3_pool (MaxPooling2D)	(None, 20, 20, 256)	0

block4_conv1 (Conv2D)	(None, 20, 20, 512)	1180160
block4_conv2 (Conv2D)	(None, 20, 20, 512)	2359808
block4_conv3 (Conv2D)	(None, 20, 20, 512)	2359808
block4_pool (MaxPooling2D)	(None, 10, 10, 512)	0
block5_conv1 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv2 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv3 (Conv2D)	(None, 10, 10, 512)	2359808
block5_pool (MaxPooling2D)	(None, 5, 5, 512)	0
flatten (Flatten)	(None, 12800)	0
fc1 (Dense)	(None, 4096)	52432896
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000
<hr/>		
Total params: 88,025,896		
Trainable params: 88,025,896		
Non-trainable params: 0		

Network Manipulation

❖ VGG16 Model

```
1 model = tf.keras.applications.VGG16(input_shape=(160, 160, 3),  
2                                     include_top=True,  
3                                     weights=None)  
4  
5 for layer in model.layers[:14]:  
6     layer.trainable = False  
7  
8 print(len(model.trainable_variables))
```

12

Model: "vgg16"

Layer (type)	Output Shape	Param #
<hr/>		
input_1 (InputLayer)	[None, 160, 160, 3]	0
block1_conv1 (Conv2D)	(None, 160, 160, 64)	1792
block1_conv2 (Conv2D)	(None, 160, 160, 64)	36928
block1_pool (MaxPooling2D)	(None, 80, 80, 64)	0
block2_conv1 (Conv2D)	(None, 80, 80, 128)	73856
block2_conv2 (Conv2D)	(None, 80, 80, 128)	147584
block2_pool (MaxPooling2D)	(None, 40, 40, 128)	0
block3_conv1 (Conv2D)	(None, 40, 40, 256)	295168
block3_conv2 (Conv2D)	(None, 40, 40, 256)	590080
block3_conv3 (Conv2D)	(None, 40, 40, 256)	590080
block3_pool (MaxPooling2D)	(None, 20, 20, 256)	0

block4_conv1 (Conv2D)	(None, 20, 20, 512)	1180160
block4_conv2 (Conv2D)	(None, 20, 20, 512)	2359808
block4_conv3 (Conv2D)	(None, 20, 20, 512)	2359808
block4_pool (MaxPooling2D)	(None, 10, 10, 512)	0
block5_conv1 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv2 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv3 (Conv2D)	(None, 10, 10, 512)	2359808
block5_pool (MaxPooling2D)	(None, 5, 5, 512)	0
flatten (Flatten)	(None, 12800)	0
fc1 (Dense)	(None, 4096)	52432896
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000
<hr/>		
Total params: 88,025,896		
Trainable params: 88,025,896		
Non-trainable params: 0		

Network Manipulation

❖ Network 2

```
1 # top=False
2 model = tf.keras.applications.VGG16(input_shape=(160,160,3),
3                                     include_top=False,
4                                     weights=None)
5 model.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
<hr/>		
input_1 (InputLayer)	[None, 160, 160, 3]	0
block1_conv1 (Conv2D)	(None, 160, 160, 64)	1792
block1_conv2 (Conv2D)	(None, 160, 160, 64)	36928
block1_pool (MaxPooling2D)	(None, 80, 80, 64)	0
block2_conv1 (Conv2D)	(None, 80, 80, 128)	73856
block2_conv2 (Conv2D)	(None, 80, 80, 128)	147584
block2_pool (MaxPooling2D)	(None, 40, 40, 128)	0
block3_conv1 (Conv2D)	(None, 40, 40, 256)	295168
block3_conv2 (Conv2D)	(None, 40, 40, 256)	590080
block3_conv3 (Conv2D)	(None, 40, 40, 256)	590080
block3_pool (MaxPooling2D)	(None, 20, 20, 256)	0

block4_conv1 (Conv2D)	(None, 20, 20, 512)	1180160
block4_conv2 (Conv2D)	(None, 20, 20, 512)	2359808
block4_conv3 (Conv2D)	(None, 20, 20, 512)	2359808
block4_pool (MaxPooling2D)	(None, 10, 10, 512)	0
block5_conv1 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv2 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv3 (Conv2D)	(None, 10, 10, 512)	2359808
block5_pool (MaxPooling2D)	(None, 5, 5, 512)	0
<hr/>		
Total params: 14,714,688		
Trainable params: 14,714,688		
Non-trainable params: 0		

Network Manipulation

❖ Network 2

```
1 import tensorflow as tf
2
3 # get VGG16
4 model = tf.keras.applications.VGG16(input_shape=(160,160,3),
5                                     include_top=False,
6                                     weights=None)
7
8 # construct a new network
9 inputs = tf.keras.Input(shape=(160, 160, 3))
10 x = model(inputs)
11 x = tf.keras.layers.Flatten()(x)
12 x = tf.keras.layers.Dense(2)(x)
13 new_model = tf.keras.Model(inputs, x)
14
15 # summary
16 new_model.summary()
```

Model: "model"

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 160, 160, 3)]	0
vgg16 (Functional)	(None, 5, 5, 512)	14714688
flatten (Flatten)	(None, 12800)	0
dense (Dense)	(None, 2)	25602

Total params: 14,740,290
Trainable params: 14,740,290
Non-trainable params: 0

Network Manipulation

❖ Network 2

❖ Fattening

```
1 import tensorflow as tf
2
3 # get VGG16
4 model = tf.keras.applications.VGG16(input_shape=(160, 160, 3),
5                                     include_top=False,
6                                     weights=None)
7
8 # construct a new network
9 inputs = tf.keras.Input(shape=(160, 160, 3))
10 x = model(inputs)
11 x = tf.keras.layers.Flatten()(x)
12 x = tf.keras.layers.Dense(2)(x)
13 new_model = tf.keras.Model(inputs, x)
14
15 # summary
16 new_model.summary()
```

Model: "model"

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 160, 160, 3)]	0
vgg16 (Functional)	(None, 5, 5, 512)	14714688
flatten (Flatten)	(None, 12800)	0
dense (Dense)	(None, 2)	25602

Total params: 14,740,290
Trainable params: 14,740,290
Non-trainable params: 0

Network Manipulation

❖ Network 2

❖ Max pooling

```
1 import tensorflow as tf
2
3 # get VGG16
4 model = tf.keras.applications.VGG16(input_shape=(160,160,3),
5                                     include_top=False,
6                                     weights=None)
7
8 # construct a new network
9 inputs = tf.keras.Input(shape=(160, 160, 3))
10 x = model(inputs)
11 x = tf.keras.layers.MaxPool2D(pool_size=(5, 5))(x)
12 x = tf.keras.layers.Reshape((-1,))(x)
13 x = tf.keras.layers.Dense(2)(x)
14 new_model = tf.keras.Model(inputs, x)
15
16 # summary
17 new_model.summary()
```

```
Model: "model"
=====
Layer (type)          Output Shape         Param #
=====
input_2 (InputLayer)  [None, 160, 160, 3]   0
vgg16 (Functional)    (None, 5, 5, 512)     14714688
max_pooling2d (MaxPooling2D) (None, 1, 1, 512) 0
reshape (Reshape)      (None, 512)           0
dense (Dense)          (None, 2)             1026
=====
Total params: 14,715,714
Trainable params: 14,715,714
Non-trainable params: 0
```

Network Manipulation

❖ Network 2

❖ Global max pooling

```
1 import tensorflow as tf
2
3 # get VGG16
4 model = tf.keras.applications.VGG16(input_shape=(160,160,3),
5                                     include_top=False,
6                                     weights=None)
7
8 # construct a new network
9 inputs = tf.keras.Input(shape=(160, 160, 3))
10 x = model(inputs)
11 x = tf.keras.layers.GlobalMaxPool2D()(x)
12 x = tf.keras.layers.Dense(2)(x)
13 new_model = tf.keras.Model(inputs, x)
14
15 # summary
16 new_model.summary()
```

```
Model: "model_1"
Layer (type)          Output Shape         Param #
=====
input_4 (InputLayer)   [ (None, 160, 160, 3) ]   0
vgg16 (Functional)    (None, 5, 5, 512)        14714688
global_max_pooling2d (Global (None, 512)        0
dense_1 (Dense)       (None, 2)                 1026
=====
Total params: 14,715,714
Trainable params: 14,715,714
Non-trainable params: 0
```

Network Manipulation

❖ Network 2

❖ Global average pooling

```
1 import tensorflow as tf
2
3 # get VGG16
4 model = tf.keras.applications.VGG16(input_shape=(160,160,3),
5                                     include_top=False,
6                                     weights=None)
7
8 # construct a new network
9 inputs = tf.keras.Input(shape=(160, 160, 3))
10 x = model(inputs)
11 x = tf.keras.layers.GlobalAveragePooling2D()(x)
12 x = tf.keras.layers.Dense(1)(x)
13 new_model = tf.keras.Model(inputs, x)
14
15 # summary
16 new_model.summary()
```

Model: "model_2"

Layer (type)	Output Shape	Param #
input_6 (InputLayer)	[(None, 160, 160, 3)]	0
vgg16 (Functional)	(None, 5, 5, 512)	14714688
global_average_pooling2d (G1	(None, 512)	0
dense_2 (Dense)	(None, 1)	513
Total params:	14,715,201	
Trainable params:	14,715,201	
Non-trainable params:	0	

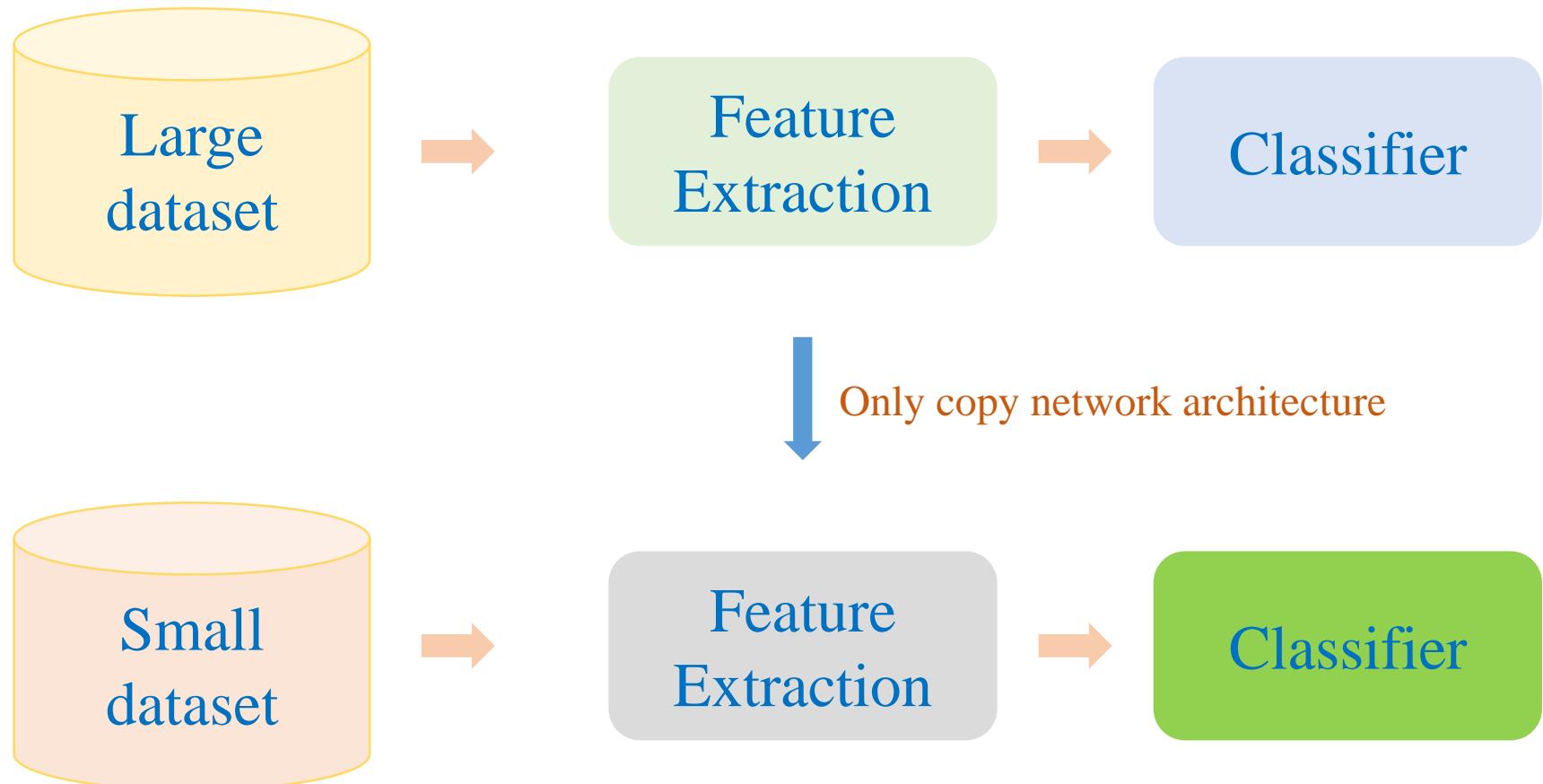
Outline

- Review
- Reuse a Pre-trained Model
- Case Study 1
- Case Study 2
- Case Study 3
- Case Study 4
- Case Study 5

Exploitation of Pretrained Models

❖ Train from scratch

ImageNet dataset
(1.2 million images
of 1000 categories)



Exploitation of Pretrained Models

❖ Transfer Learning

ImageNet dataset
(1.2 million images
of 1000 categories)



Feature Extraction

Classifier

Copy architecture and
pretrained weights



Feature Extraction

Classifier

Trained with the small dataset

Exploitation of Pretrained Models

❖ Fine Tuning

ImageNet dataset
(1.2 million images
of 1000 categories)



Feature
Extraction



Classifier

Copy architecture and
pretrained weights



Feature
Extraction

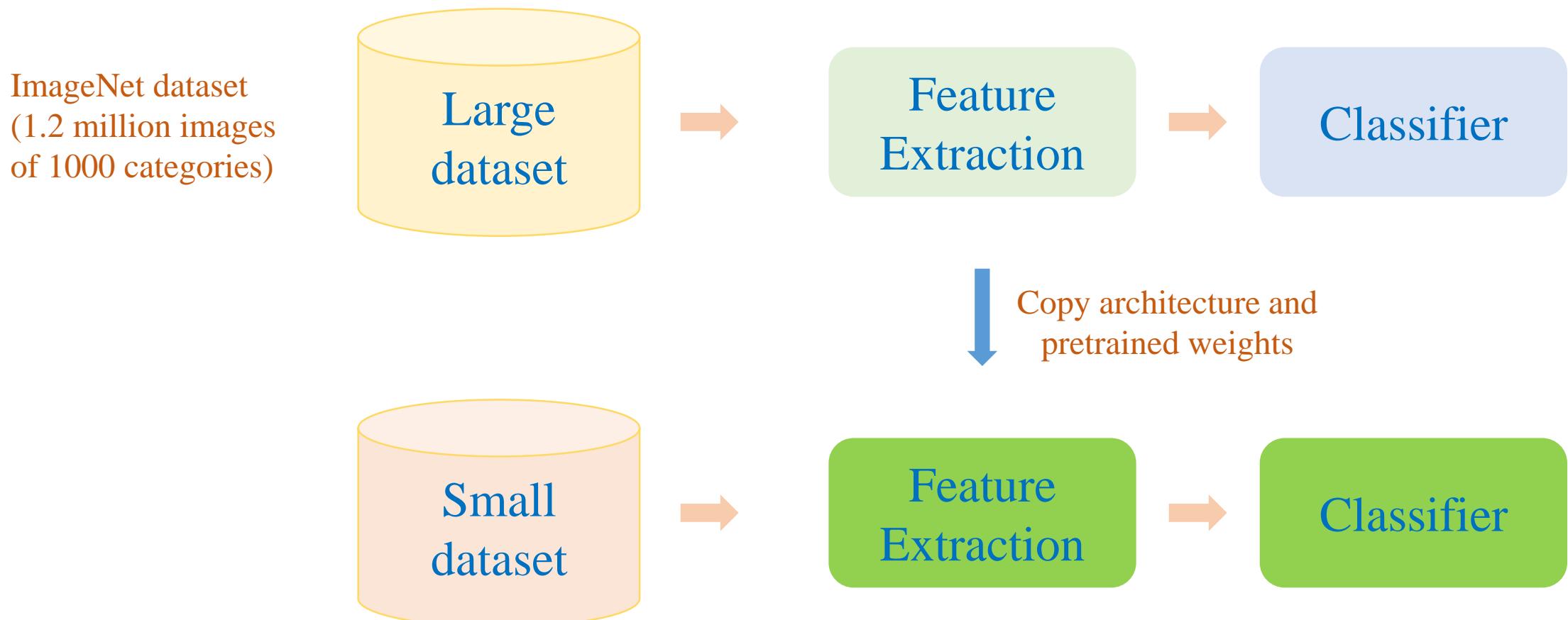


Classifier

Trained with the small dataset

Exploitation of Pretrained Models

❖ As Initialization



Outline

- Review
- Reuse a Pre-trained Model
- Case Study 1
- Case Study 2
- Case Study 3
- Case Study 4
- Case Study 5

Case Study 1

❖ Cat-Dog dataset

❖ Train from scratch

```
1 PATH = 'cats_and_dogs_small/'  
2  
3 train_dir = os.path.join(PATH, 'train')  
validation_dir = os.path.join(PATH, 'validation')  
4  
5 BATCH_SIZE = 256  
6 IMG_SIZE = (160, 160)  
7 BUFFER_SIZE = BATCH_SIZE*5  
8  
9 train_dataset = image_dataset_from_directory(train_dir,  
10                                         shuffle=True,  
11                                         batch_size=BATCH_SIZE,  
12                                         image_size=IMG_SIZE)
```

Found 2000 files belonging to 2 classes.

```
1 validation_dataset = image_dataset_from_directory(validation_dir,  
2                                         shuffle=True,  
3                                         batch_size=BATCH_SIZE,  
4                                         image_size=IMG_SIZE)
```

Found 1000 files belonging to 2 classes.

Case Study 1

- ❖ Cat-Dog dataset
- ❖ Train from scratch

Model: "vgg16"

Layer (type)	Output Shape	Param #
<hr/>		
input_1 (InputLayer)	[(None, 160, 160, 3)]	0
block1_conv1 (Conv2D)	(None, 160, 160, 64)	1792
block1_conv2 (Conv2D)	(None, 160, 160, 64)	36928
block1_pool (MaxPooling2D)	(None, 80, 80, 64)	0
block2_conv1 (Conv2D)	(None, 80, 80, 128)	73856
block2_conv2 (Conv2D)	(None, 80, 80, 128)	147584
block2_pool (MaxPooling2D)	(None, 40, 40, 128)	0
block3_conv1 (Conv2D)	(None, 40, 40, 256)	295168
block3_conv2 (Conv2D)	(None, 40, 40, 256)	590080
block3_conv3 (Conv2D)	(None, 40, 40, 256)	590080
block3_pool (MaxPooling2D)	(None, 20, 20, 256)	0

```
1 # top=False
2 model = tf.keras.applications.VGG16(input_shape=(160,160,3),
3                                     include_top=False,
4                                     weights=None)
5 model.summary()
```

block4_conv1 (Conv2D)	(None, 20, 20, 512)	1180160
block4_conv2 (Conv2D)	(None, 20, 20, 512)	2359808
block4_conv3 (Conv2D)	(None, 20, 20, 512)	2359808
block4_pool (MaxPooling2D)	(None, 10, 10, 512)	0
block5_conv1 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv2 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv3 (Conv2D)	(None, 10, 10, 512)	2359808
block5_pool (MaxPooling2D)	(None, 5, 5, 512)	0
<hr/>		
Total params: 14,714,688		
Trainable params: 14,714,688		
Non-trainable params: 0		

Case Study 1

❖ Cat-Dog dataset

❖ Train from scratch

```
1 # process data
2 data_augmentation = tf.keras.Sequential([
3     tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
4     tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
5     tf.keras.layers.experimental.preprocessing.Rescaling(1./127.5, offset=-1)
6 ])
7
8 # flattening
9 flatten = tf.keras.layers.Flatten()
10
11 # final layer
12 prediction_layer = tf.keras.layers.Dense(1)
13
14 # construct a new network
15 inputs = tf.keras.Input(shape=(160, 160, 3))
16 x = data_augmentation(inputs)
17 x = base_model(x)
18 x = flatten(x)
19 outputs = prediction_layer(x)
20 model = tf.keras.Model(inputs, outputs)
```

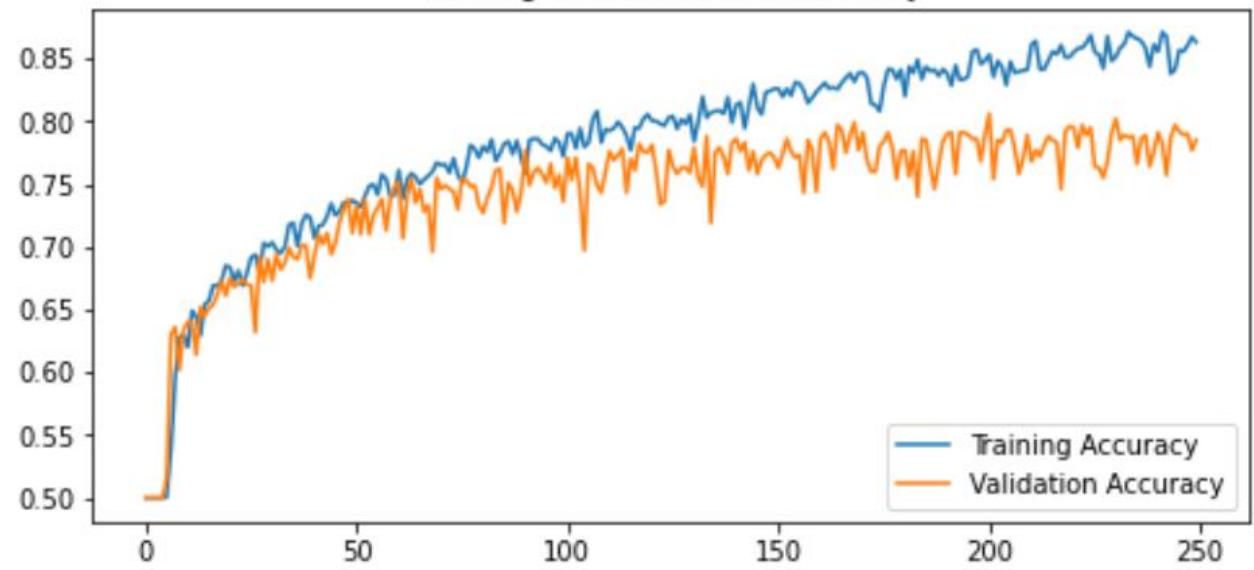
Case Study 1

❖ Cat-Dog dataset

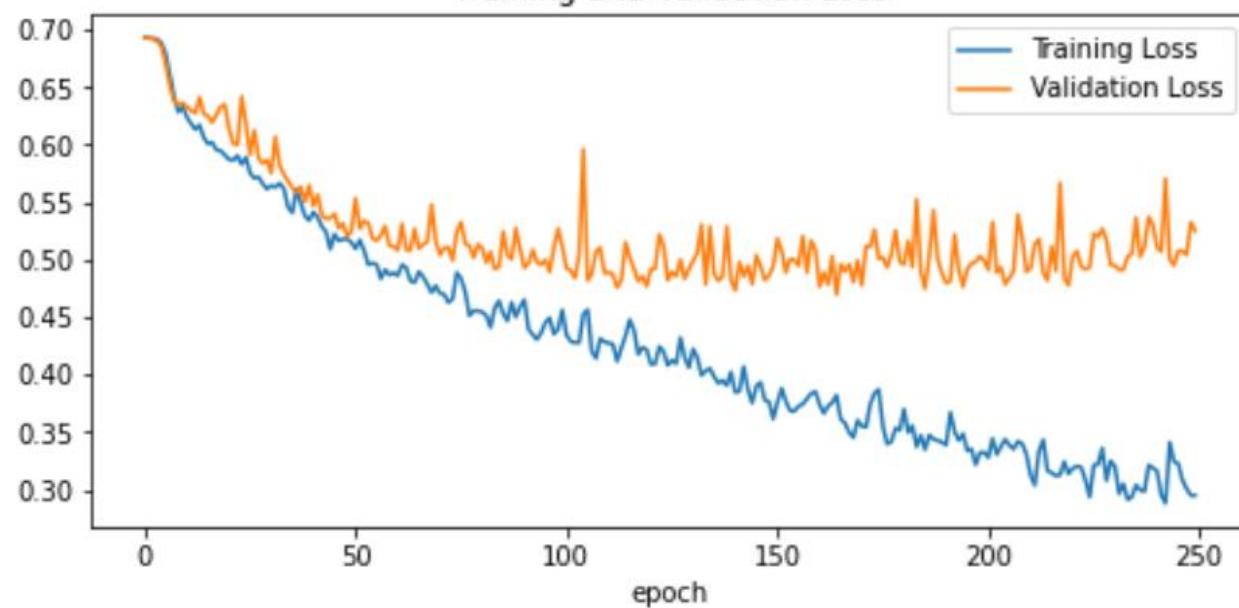
❖ Train from scratch

```
1 base_learning_rate =
2 model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
3                 optimizer = tf.keras.optimizers.Adam(lr=0.00001),
4                 metrics=['accuracy'])
5
6 history_fine = model.fit(train_dataset,
7                           epochs=250,
8                           validation_data=validation_dataset)
```

Training and Validation Accuracy



Training and Validation Loss



Case Study 1

❖ Transfer learning

```
1 base_model = tf.keras.applications.VGG16(input_shape=(160,160,3),  
2 include_top=False,  
3 weights='imagenet')  
4 base_model.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
=====		
input_1 (InputLayer)	[(None, 160, 160, 3)]	0
block1_conv1 (Conv2D)	(None, 160, 160, 64)	1792
block1_conv2 (Conv2D)	(None, 160, 160, 64)	36928
block1_pool (MaxPooling2D)	(None, 80, 80, 64)	0
block2_conv1 (Conv2D)	(None, 80, 80, 128)	73856
block2_conv2 (Conv2D)	(None, 80, 80, 128)	147584
block2_pool (MaxPooling2D)	(None, 40, 40, 128)	0
block3_conv1 (Conv2D)	(None, 40, 40, 256)	295168
block3_conv2 (Conv2D)	(None, 40, 40, 256)	590080
block3_conv3 (Conv2D)	(None, 40, 40, 256)	590080
block3_pool (MaxPooling2D)	(None, 20, 20, 256)	0

block4_conv1 (Conv2D)	(None, 20, 20, 512)	1180160
block4_conv2 (Conv2D)	(None, 20, 20, 512)	2359808
block4_conv3 (Conv2D)	(None, 20, 20, 512)	2359808
block4_pool (MaxPooling2D)	(None, 10, 10, 512)	0
block5_conv1 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv2 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv3 (Conv2D)	(None, 10, 10, 512)	2359808
block5_pool (MaxPooling2D)	(None, 5, 5, 512)	0
=====		
Total params: 14,714,688		
Trainable params: 14,714,688		
Non-trainable params: 0		

Case Study 1

❖ Transfer learning

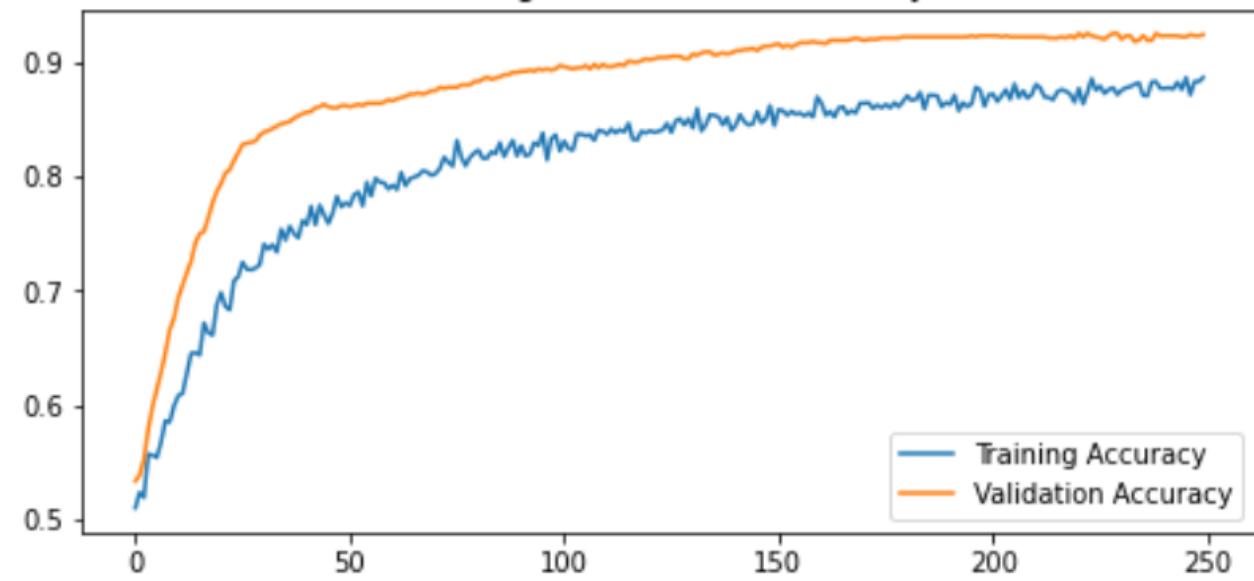
```
1 # freeze the base model
2 base_model.trainable = False
3
4 # process data
5 data_augmentation = tf.keras.Sequential([
6     tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
7     tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
8     tf.keras.layers.experimental.preprocessing.Rescaling(1./127.5, offset= -1)
9 ])
10
11 # flattening
12 flatten = tf.keras.layers.Flatten()
13
14 # final layer
15 prediction_layer = tf.keras.layers.Dense(1)
16
17 # construct a new network
18 inputs = tf.keras.Input(shape=(160, 160, 3))
19 x = data_augmentation(inputs)
20 x = base_model(x)
21 x = flatten(x)
22 outputs = prediction_layer(x)
23 model = tf.keras.Model(inputs, outputs)
```

Case Study 1

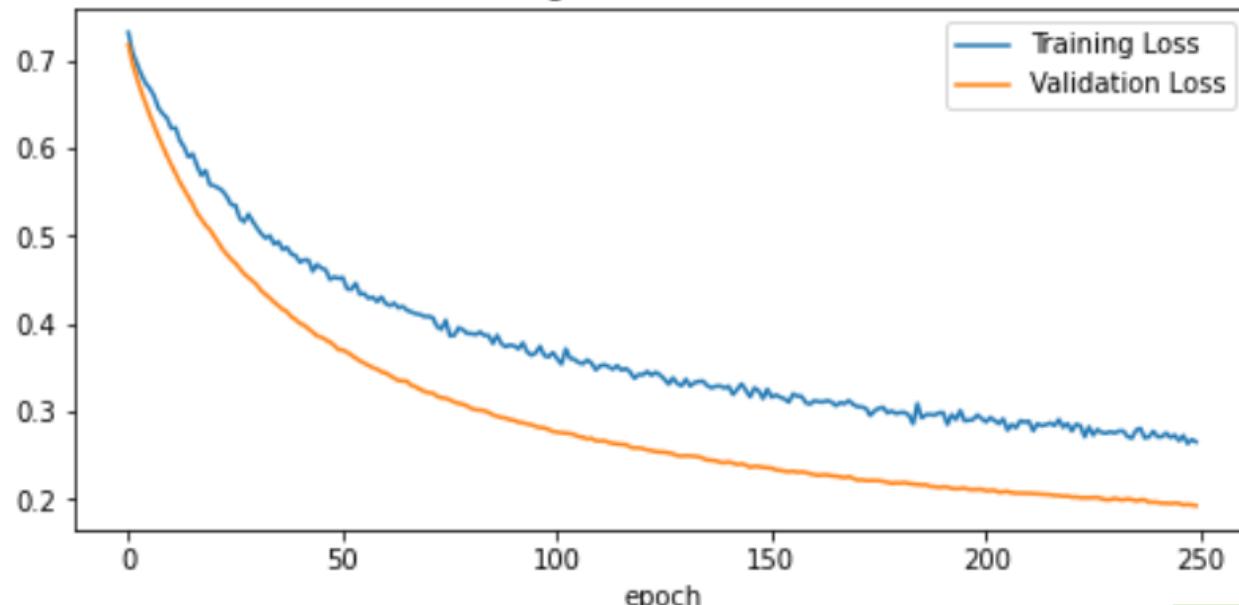
❖ Transfer learning

```
1 base_learning_rate =
2 model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
3                 optimizer = tf.keras.optimizers.Adam(lr=0.00001),
4                 metrics=['accuracy'])
5
6 history_fine = model.fit(train_dataset,
7                           epochs=250,
8                           validation_data=validation_dataset)
```

Training and Validation Accuracy



Training and Validation Loss



Case Study 1

❖ Fine tuning

```
1 base_model = tf.keras.applications.VGG16(input_shape=(160,160,3),  
2 include_top=False,  
3 weights='imagenet')  
4 base_model.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
=====		
input_1 (InputLayer)	[(None, 160, 160, 3)]	0
block1_conv1 (Conv2D)	(None, 160, 160, 64)	1792
block1_conv2 (Conv2D)	(None, 160, 160, 64)	36928
block1_pool (MaxPooling2D)	(None, 80, 80, 64)	0
block2_conv1 (Conv2D)	(None, 80, 80, 128)	73856
block2_conv2 (Conv2D)	(None, 80, 80, 128)	147584
block2_pool (MaxPooling2D)	(None, 40, 40, 128)	0
block3_conv1 (Conv2D)	(None, 40, 40, 256)	295168
block3_conv2 (Conv2D)	(None, 40, 40, 256)	590080
block3_conv3 (Conv2D)	(None, 40, 40, 256)	590080
block3_pool (MaxPooling2D)	(None, 20, 20, 256)	0

block4_conv1 (Conv2D)	(None, 20, 20, 512)	1180160
block4_conv2 (Conv2D)	(None, 20, 20, 512)	2359808
block4_conv3 (Conv2D)	(None, 20, 20, 512)	2359808
block4_pool (MaxPooling2D)	(None, 10, 10, 512)	0
block5_conv1 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv2 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv3 (Conv2D)	(None, 10, 10, 512)	2359808
block5_pool (MaxPooling2D)	(None, 5, 5, 512)	0
=====		
Total params: 14,714,688		
Trainable params: 14,714,688		
Non-trainable params: 0		

Case Study 1

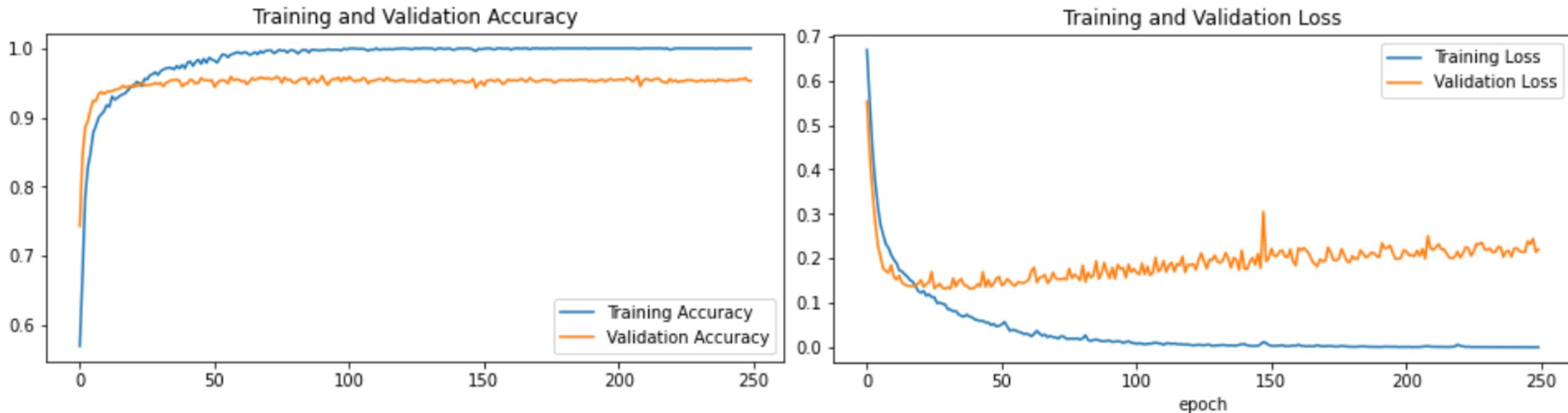
❖ Fine tuning

```
1 # Freeze some first the layers
2 fine_tune_at = 14
3 for layer in base_model.layers[:fine_tune_at]:
4     layer.trainable = False
5
6 # process data
7 data_augmentation = tf.keras.Sequential([
8     tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
9     tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
10    tf.keras.layers.experimental.preprocessing.Rescaling(1./127.5, offset=-1)
11])
12
13 # flattening
14 flatten = tf.keras.layers.Flatten()
15
16 # final layer
17 prediction_layer = tf.keras.layers.Dense(1)
18
19 # construct a new network
20 inputs = tf.keras.Input(shape=(160, 160, 3))
21 x = data_augmentation(inputs)
22 x = base_model(x)
23 x = flatten(x)
24 outputs = prediction_layer(x)
25 model = tf.keras.Model(inputs, outputs)
```

Case Study 1

❖ Fine tuning

```
1 base_learning_rate =
2 model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
3                 optimizer = tf.keras.optimizers.Adam(lr=0.00001),
4                 metrics=['accuracy'])
5
6 history_fine = model.fit(train_dataset,
7                           epochs=250,
8                           validation_data=validation_dataset)
```



Case Study 1

❖ Use pretrained weights as init weights

```
1 base_model = tf.keras.applications.VGG16(input_shape=(160,160,3),  
2                                         include_top=False,  
3                                         weights='imagenet')  
4 base_model.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
=====		
input_1 (InputLayer)	[(None, 160, 160, 3)]	0
block1_conv1 (Conv2D)	(None, 160, 160, 64)	1792
block1_conv2 (Conv2D)	(None, 160, 160, 64)	36928
block1_pool (MaxPooling2D)	(None, 80, 80, 64)	0
block2_conv1 (Conv2D)	(None, 80, 80, 128)	73856
block2_conv2 (Conv2D)	(None, 80, 80, 128)	147584
block2_pool (MaxPooling2D)	(None, 40, 40, 128)	0
block3_conv1 (Conv2D)	(None, 40, 40, 256)	295168
block3_conv2 (Conv2D)	(None, 40, 40, 256)	590080
block3_conv3 (Conv2D)	(None, 40, 40, 256)	590080
block3_pool (MaxPooling2D)	(None, 20, 20, 256)	0

block4_conv1 (Conv2D)	(None, 20, 20, 512)	1180160
block4_conv2 (Conv2D)	(None, 20, 20, 512)	2359808
block4_conv3 (Conv2D)	(None, 20, 20, 512)	2359808
block4_pool (MaxPooling2D)	(None, 10, 10, 512)	0
block5_conv1 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv2 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv3 (Conv2D)	(None, 10, 10, 512)	2359808
block5_pool (MaxPooling2D)	(None, 5, 5, 512)	0
=====		
Total params: 14,714,688		
Trainable params: 14,714,688		
Non-trainable params: 0		

Case Study 1

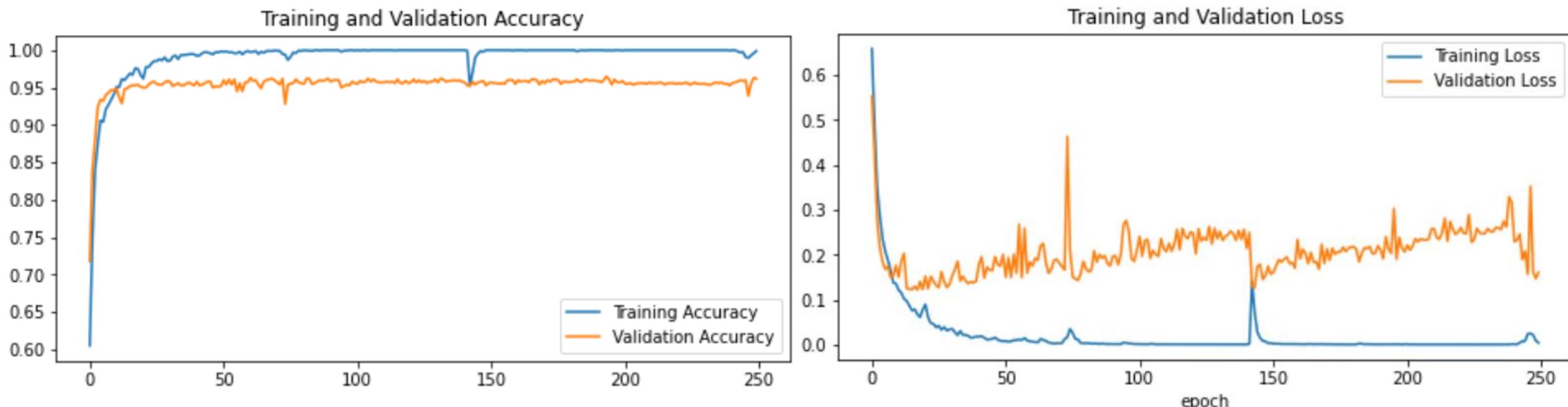
❖ Use pretrained weights as init weights

```
1 # process data
2 data_augmentation = tf.keras.Sequential([
3     tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
4     tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
5     tf.keras.layers.experimental.preprocessing.Rescaling(1./127.5, offset= -1)
6 ])
7
8 # flattening
9 flatten = tf.keras.layers.Flatten()
10
11 # final layer
12 prediction_layer = tf.keras.layers.Dense(1)
13
14 # construct a new network
15 inputs = tf.keras.Input(shape=(160, 160, 3))
16 x = data_augmentation(inputs)
17 x = base_model(x)
18 x = flatten(x)
19 outputs = prediction_layer(x)
20 model = tf.keras.Model(inputs, outputs)
```

Case Study 1

❖ Use pretrained weights
init weights

```
1 base_learning_rate =
2 model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
3                 optimizer = tf.keras.optimizers.Adam(lr=0.00001),
4                 metrics=['accuracy'])
5
6 history_fine = model.fit(train_dataset,
7                           epochs=250,
8                           validation_data=validation_dataset)
```



Case Study 1

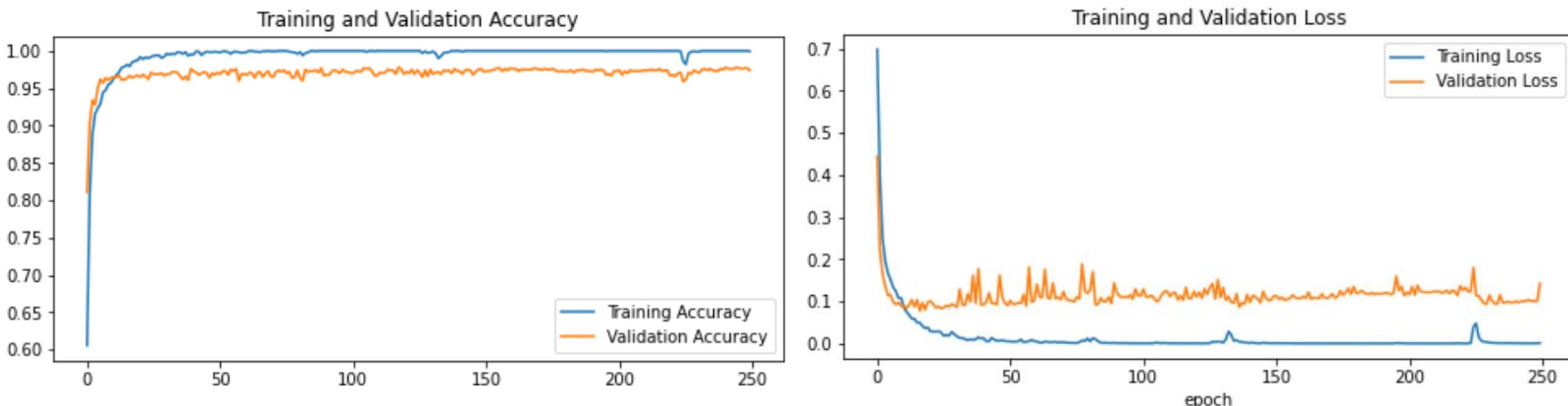
- ❖ Use pretrained weights as init weights
- ❖ Global max pooling

```
1 # process data
2 data_augmentation = tf.keras.Sequential([
3     tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
4     tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
5     tf.keras.layers.experimental.preprocessing.Rescaling(1./127.5, offset= -1)
6 ])
7
8 # flattening
9 global_max = tf.keras.layers.GlobalMaxPool2D()
10
11 # final layer
12 prediction_layer = tf.keras.layers.Dense(1)
13
14 # construct a new network
15 inputs = tf.keras.Input(shape=(160, 160, 3))
16 x = data_augmentation(inputs)
17 x = base_model(x)
18 x = global_max(x)
19 outputs = prediction_layer(x)
20 model = tf.keras.Model(inputs, outputs)
```

Case Study 1

- ❖ Use pretrained weights init weights
- ❖ Global max pooling

```
1 base_learning_rate =
2 model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
3                 optimizer = tf.keras.optimizers.Adam(lr=0.00001),
4                 metrics=['accuracy'])
5
6 history_fine = model.fit(train_dataset,
7                           epochs=250,
8                           validation_data=validation_dataset)
```



Case Study 1

- ❖ Use pretrained weights as init weights
- ❖ Global average pooling

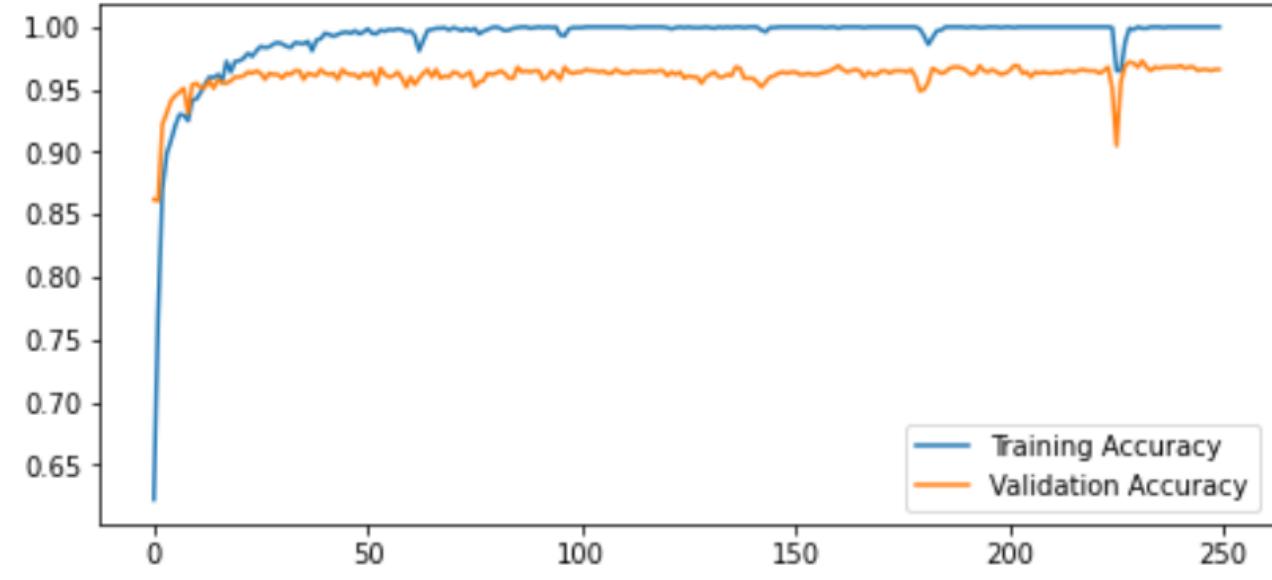
```
1 # process data
2 data_augmentation = tf.keras.Sequential([
3     tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
4     tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
5     tf.keras.layers.experimental.preprocessing.Rescaling(1./127.5, offset= -1)
6 ])
7
8 # flattening
9 global_average = tf.keras.layers.GlobalAveragePooling2D()
10
11 # final layer
12 prediction_layer = tf.keras.layers.Dense(1)
13
14 # construct a new network
15 inputs = tf.keras.Input(shape=(160, 160, 3))
16 x = data_augmentation(inputs)
17 x = base_model(x)
18 x = global_average(x)
19 outputs = prediction_layer(x)
20 model = tf.keras.Model(inputs, outputs)
```

Case Study 1

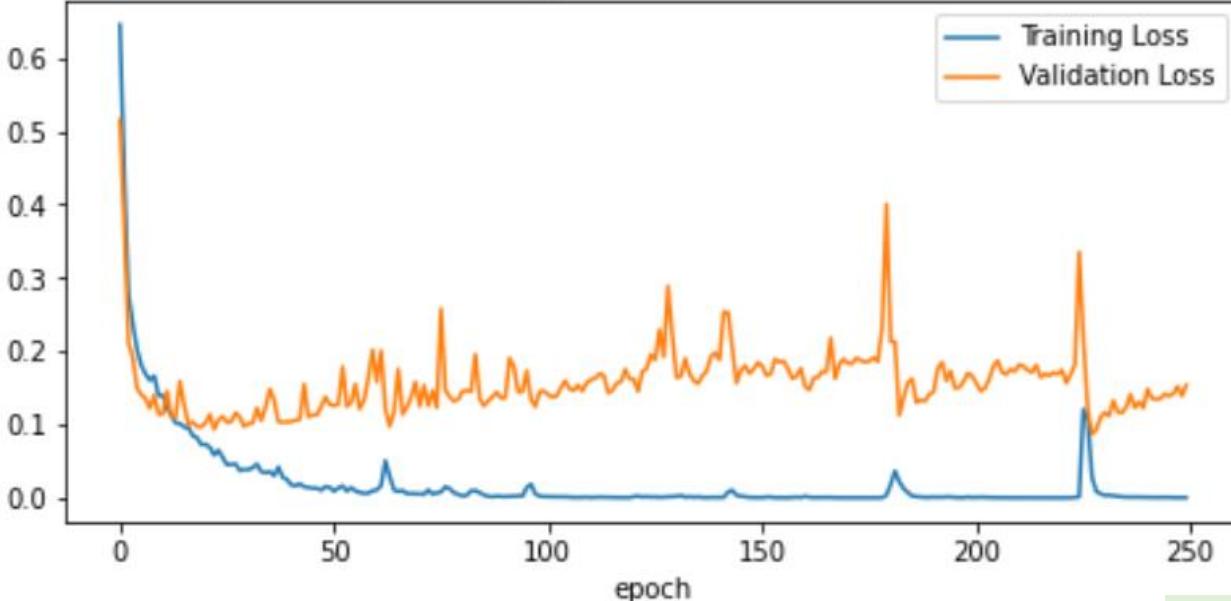
- ❖ Use pretrained weights init weights
- ❖ Global average pooling

```
1 base_learning_rate =
2 model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
3                 optimizer = tf.keras.optimizers.Adam(lr=0.00001),
4                 metrics=['accuracy'])
5
6 history_fine = model.fit(train_dataset,
7                           epochs=250,
8                           validation_data=validation_dataset)
```

Training and Validation Accuracy



Training and Validation Loss



Outline

- Review
- Reuse a Pre-trained Model
- Case Study 1
- Case Study 2
- Case Study 3
- Case Study 4
- Case Study 5

Case Study 2

❖ Cat-Dog dataset

❖ Full data

```
1 # training data
2 train_dataset = image_dataset_from_directory('cats_and_dog_large/train',
3                                              shuffle=True,
4                                              batch_size=128,
5                                              image_size=(160, 160))
6
7 # validation data
8 validation_dataset = image_dataset_from_directory('cats_and_dog_large/validation',
9                                              shuffle=True,
10                                             batch_size=128,
11                                             image_size=(160, 160))
```

Found 22000 files belonging to 2 classes.

Found 3000 files belonging to 2 classes.

Case Study 2

- ❖ Cat-Dog dataset
- ❖ Train from scratch

Model: "vgg16"

Layer (type)	Output Shape	Param #
<hr/>		
input_1 (InputLayer)	[None, 160, 160, 3]	0
block1_conv1 (Conv2D)	(None, 160, 160, 64)	1792
block1_conv2 (Conv2D)	(None, 160, 160, 64)	36928
block1_pool (MaxPooling2D)	(None, 80, 80, 64)	0
block2_conv1 (Conv2D)	(None, 80, 80, 128)	73856
block2_conv2 (Conv2D)	(None, 80, 80, 128)	147584
block2_pool (MaxPooling2D)	(None, 40, 40, 128)	0
block3_conv1 (Conv2D)	(None, 40, 40, 256)	295168
block3_conv2 (Conv2D)	(None, 40, 40, 256)	590080
block3_conv3 (Conv2D)	(None, 40, 40, 256)	590080
block3_pool (MaxPooling2D)	(None, 20, 20, 256)	0

```
1 # top=False
2 model = tf.keras.applications.VGG16(input_shape=(160,160,3),
3                                     include_top=False,
4                                     weights=None)
5 model.summary()
```

block4_conv1 (Conv2D)	(None, 20, 20, 512)	1180160
block4_conv2 (Conv2D)	(None, 20, 20, 512)	2359808
block4_conv3 (Conv2D)	(None, 20, 20, 512)	2359808
block4_pool (MaxPooling2D)	(None, 10, 10, 512)	0
block5_conv1 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv2 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv3 (Conv2D)	(None, 10, 10, 512)	2359808
block5_pool (MaxPooling2D)	(None, 5, 5, 512)	0
<hr/>		
Total params: 14,714,688		
Trainable params: 14,714,688		
Non-trainable params: 0		

Case Study 2

❖ Cat-Dog dataset

❖ Train from scratch

```
1 # process data
2 data_augmentation = tf.keras.Sequential([
3     tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
4     tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
5     tf.keras.layers.experimental.preprocessing.Rescaling(1./127.5, offset=-1)
6 ])
7
8 # flattening
9 flatten = tf.keras.layers.Flatten()
10
11 # final layer
12 prediction_layer = tf.keras.layers.Dense(1)
13
14 # construct a new network
15 inputs = tf.keras.Input(shape=(160, 160, 3))
16 x = data_augmentation(inputs)
17 x = base_model(x)
18 x = flatten(x)
19 outputs = prediction_layer(x)
20 model = tf.keras.Model(inputs, outputs)
```

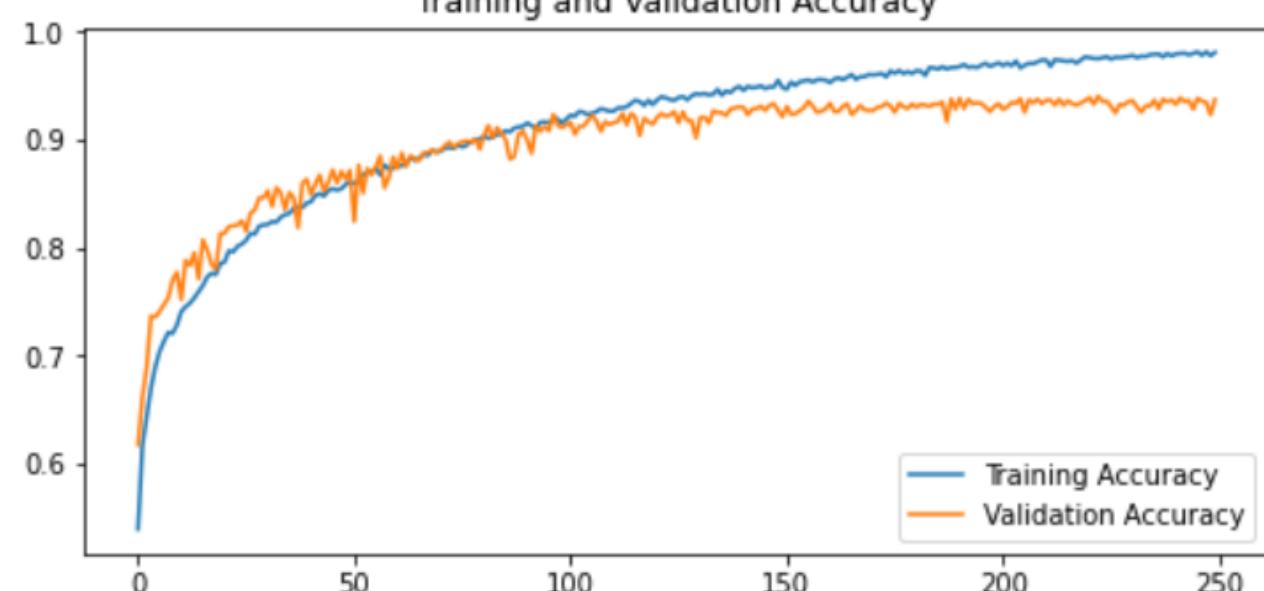
Case Study 2

❖ Cat-Dog dataset

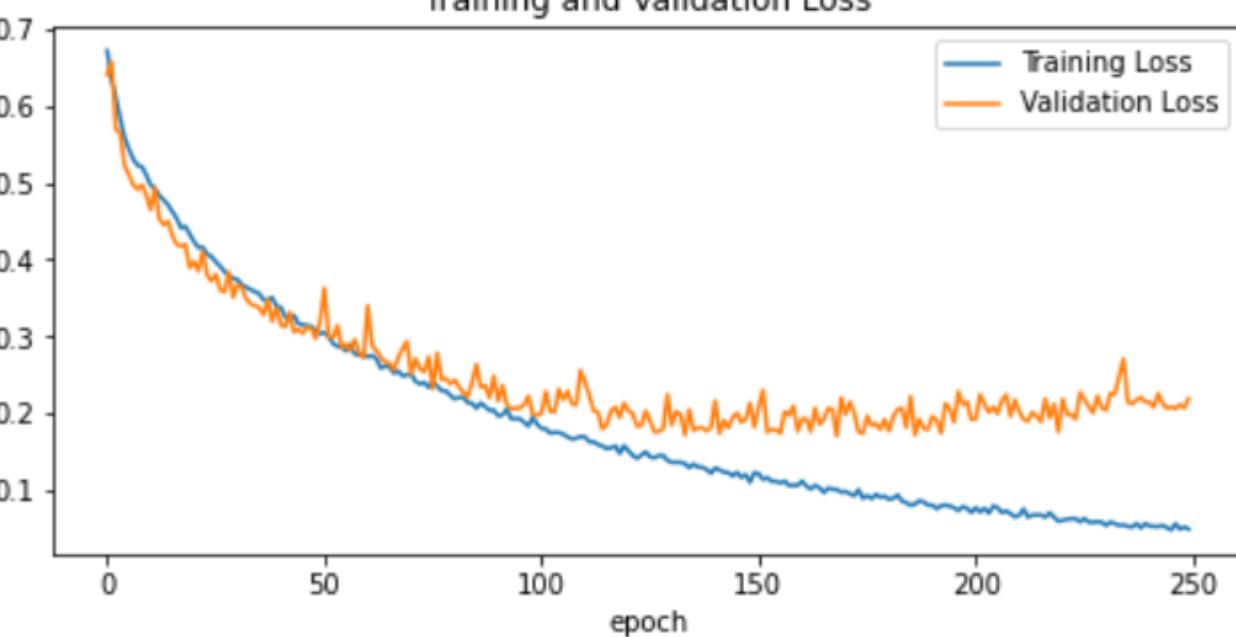
❖ Train from scratch

```
2 model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),  
3                 optimizer = tf.keras.optimizers.Adam(lr=0.00001),  
4                 metrics=['accuracy'])  
5  
6 history_fine = model.fit(train_dataset,  
7                           epochs=250,  
8                           validation_data=validation_dataset)
```

Training and Validation Accuracy



Training and Validation Loss



Case Study 2

❖ Transfer learning

```
1 base_model = tf.keras.applications.VGG16(input_shape=(160,160,3),  
2 include_top=False,  
3 weights='imagenet')  
4 base_model.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
=====		
input_1 (InputLayer)	[(None, 160, 160, 3)]	0
block1_conv1 (Conv2D)	(None, 160, 160, 64)	1792
block1_conv2 (Conv2D)	(None, 160, 160, 64)	36928
block1_pool (MaxPooling2D)	(None, 80, 80, 64)	0
block2_conv1 (Conv2D)	(None, 80, 80, 128)	73856
block2_conv2 (Conv2D)	(None, 80, 80, 128)	147584
block2_pool (MaxPooling2D)	(None, 40, 40, 128)	0
block3_conv1 (Conv2D)	(None, 40, 40, 256)	295168
block3_conv2 (Conv2D)	(None, 40, 40, 256)	590080
block3_conv3 (Conv2D)	(None, 40, 40, 256)	590080
block3_pool (MaxPooling2D)	(None, 20, 20, 256)	0

block4_conv1 (Conv2D)	(None, 20, 20, 512)	1180160
block4_conv2 (Conv2D)	(None, 20, 20, 512)	2359808
block4_conv3 (Conv2D)	(None, 20, 20, 512)	2359808
block4_pool (MaxPooling2D)	(None, 10, 10, 512)	0
block5_conv1 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv2 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv3 (Conv2D)	(None, 10, 10, 512)	2359808
block5_pool (MaxPooling2D)	(None, 5, 5, 512)	0
=====		
Total params: 14,714,688		
Trainable params: 14,714,688		
Non-trainable params: 0		

Case Study 2

❖ Transfer learning

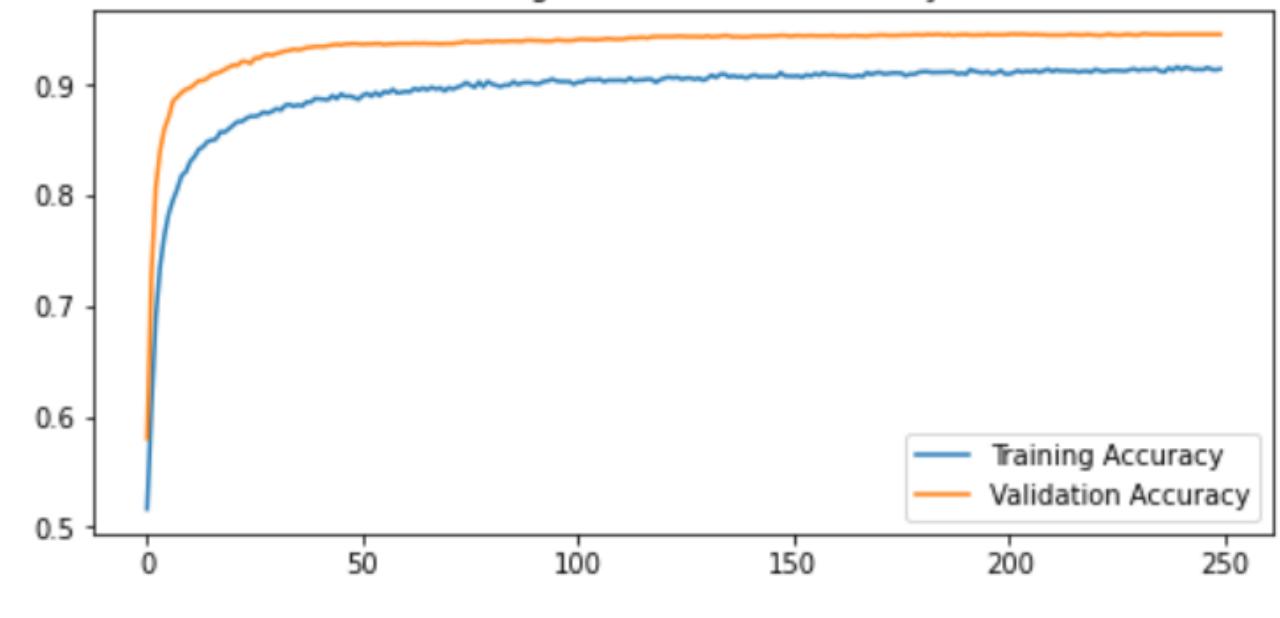
```
1 # freeze the base model
2 base_model.trainable = False
3
4 # process data
5 data_augmentation = tf.keras.Sequential([
6     tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
7     tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
8     tf.keras.layers.experimental.preprocessing.Rescaling(1./127.5, offset= -1)
9 ])
10
11 # flattening
12 flatten = tf.keras.layers.Flatten()
13
14 # final layer
15 prediction_layer = tf.keras.layers.Dense(1)
16
17 # construct a new network
18 inputs = tf.keras.Input(shape=(160, 160, 3))
19 x = data_augmentation(inputs)
20 x = base_model(x)
21 x = flatten(x)
22 outputs = prediction_layer(x)
23 model = tf.keras.Model(inputs, outputs)
```

Case Study 2

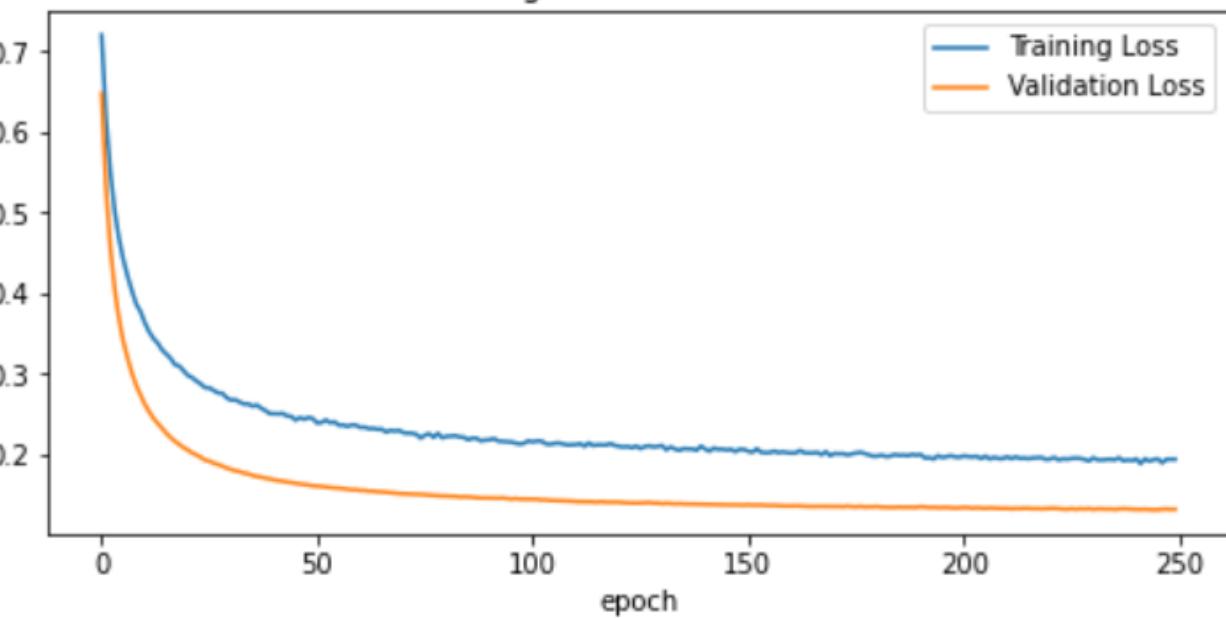
❖ Transfer learning

```
2 model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),  
3                 optimizer = tf.keras.optimizers.Adam(lr=0.00001),  
4                 metrics=['accuracy'])  
5  
6 history_fine = model.fit(train_dataset,  
7                           epochs=250,  
8                           validation_data=validation_dataset)
```

Training and Validation Accuracy



Training and Validation Loss



Case Study 2

❖ Fine tuning

```
1 base_model = tf.keras.applications.VGG16(input_shape=(160,160,3),  
2     include_top=False,  
3     weights='imagenet')  
4 base_model.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
=====		
input_1 (InputLayer)	[(None, 160, 160, 3)]	0
block1_conv1 (Conv2D)	(None, 160, 160, 64)	1792
block1_conv2 (Conv2D)	(None, 160, 160, 64)	36928
block1_pool (MaxPooling2D)	(None, 80, 80, 64)	0
block2_conv1 (Conv2D)	(None, 80, 80, 128)	73856
block2_conv2 (Conv2D)	(None, 80, 80, 128)	147584
block2_pool (MaxPooling2D)	(None, 40, 40, 128)	0
block3_conv1 (Conv2D)	(None, 40, 40, 256)	295168
block3_conv2 (Conv2D)	(None, 40, 40, 256)	590080
block3_conv3 (Conv2D)	(None, 40, 40, 256)	590080
block3_pool (MaxPooling2D)	(None, 20, 20, 256)	0

block4_conv1 (Conv2D)	(None, 20, 20, 512)	1180160
block4_conv2 (Conv2D)	(None, 20, 20, 512)	2359808
block4_conv3 (Conv2D)	(None, 20, 20, 512)	2359808
block4_pool (MaxPooling2D)	(None, 10, 10, 512)	0
block5_conv1 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv2 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv3 (Conv2D)	(None, 10, 10, 512)	2359808
block5_pool (MaxPooling2D)	(None, 5, 5, 512)	0
=====		
Total params: 14,714,688		
Trainable params: 14,714,688		
Non-trainable params: 0		

Case Study 2

❖ Fine tuning

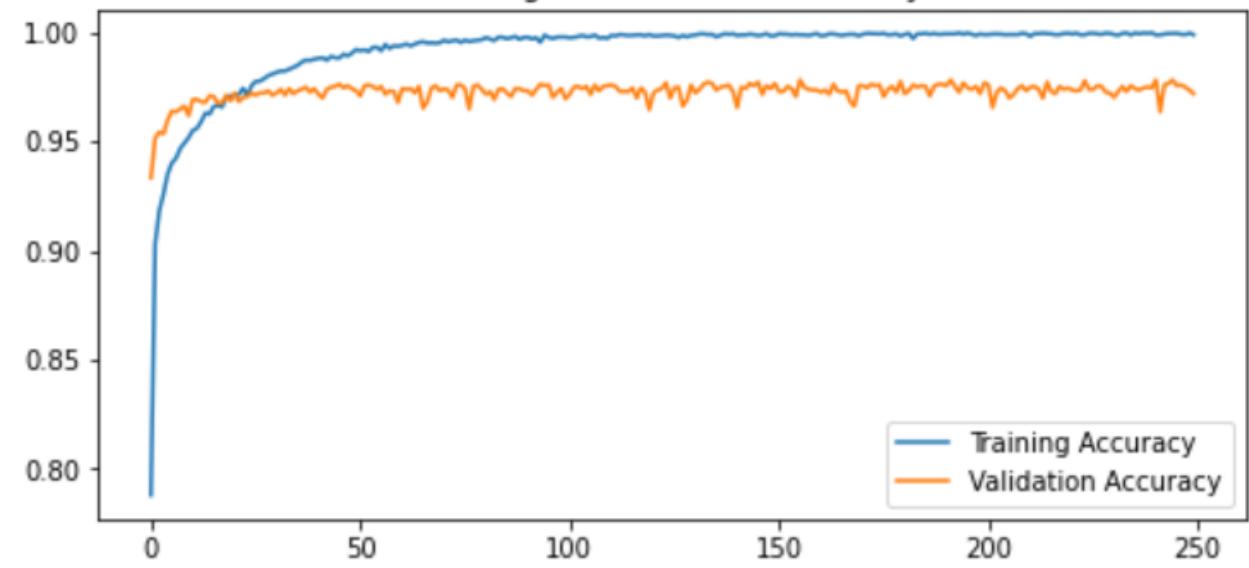
```
1 # Freeze some first the layers
2 fine_tune_at = 14
3 for layer in base_model.layers[:fine_tune_at]:
4     layer.trainable = False
5
6 # process data
7 data_augmentation = tf.keras.Sequential([
8     tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
9     tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
10    tf.keras.layers.experimental.preprocessing.Rescaling(1./127.5, offset=-1)
11])
12
13 # flattening
14 flatten = tf.keras.layers.Flatten()
15
16 # final layer
17 prediction_layer = tf.keras.layers.Dense(1)
18
19 # construct a new network
20 inputs = tf.keras.Input(shape=(160, 160, 3))
21 x = data_augmentation(inputs)
22 x = base_model(x)
23 x = flatten(x)
24 outputs = prediction_layer(x)
25 model = tf.keras.Model(inputs, outputs)
```

Case Study 2

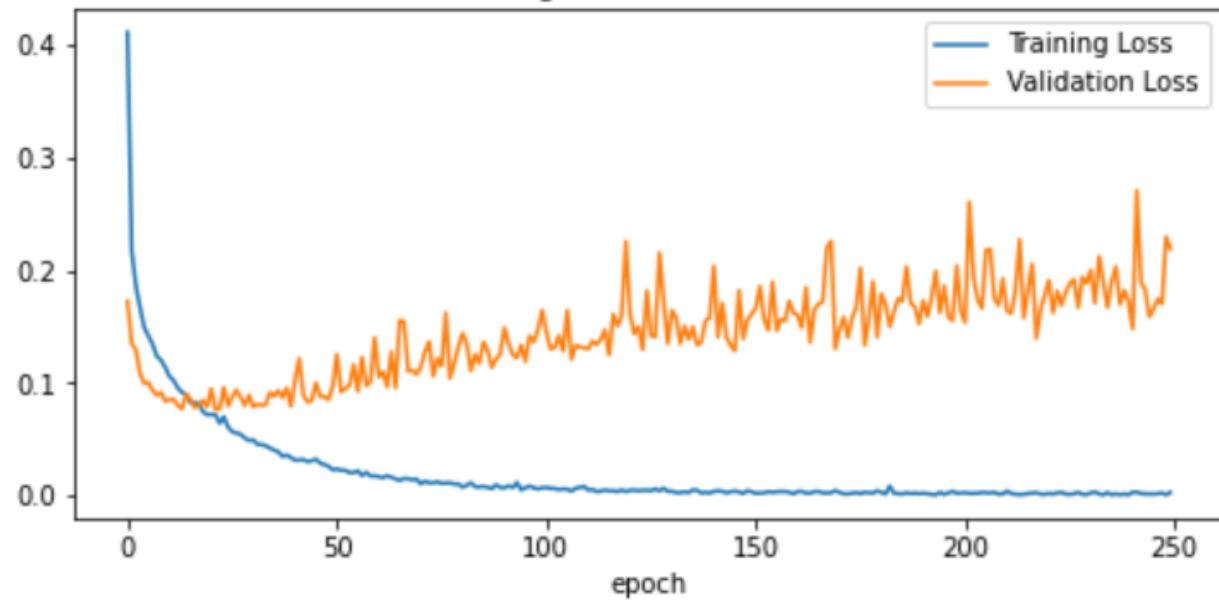
❖ Fine tuning

```
2 model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),  
3                 optimizer = tf.keras.optimizers.Adam(lr=0.00001),  
4                 metrics=['accuracy'])  
5  
6 history_fine = model.fit(train_dataset,  
7                           epochs=250,  
8                           validation_data=validation_dataset)
```

Training and Validation Accuracy



Training and Validation Loss



Case Study 2

❖ Use pretrained weights as init weights

```

1 base_model = tf.keras.applications.VGG16(input_shape=(160,160,3),
2                                         include_top=False,
3                                         weights='imagenet')
4 base_model.summary()

```

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 160, 160, 3)]	0
block1_conv1 (Conv2D)	(None, 160, 160, 64)	1792
block1_conv2 (Conv2D)	(None, 160, 160, 64)	36928
block1_pool (MaxPooling2D)	(None, 80, 80, 64)	0
block2_conv1 (Conv2D)	(None, 80, 80, 128)	73856
block2_conv2 (Conv2D)	(None, 80, 80, 128)	147584
block2_pool (MaxPooling2D)	(None, 40, 40, 128)	0
block3_conv1 (Conv2D)	(None, 40, 40, 256)	295168
block3_conv2 (Conv2D)	(None, 40, 40, 256)	590080
block3_conv3 (Conv2D)	(None, 40, 40, 256)	590080
block3_pool (MaxPooling2D)	(None, 20, 20, 256)	0

block4_conv1 (Conv2D)	(None, 20, 20, 512)	1180160
block4_conv2 (Conv2D)	(None, 20, 20, 512)	2359808
block4_conv3 (Conv2D)	(None, 20, 20, 512)	2359808
block4_pool (MaxPooling2D)	(None, 10, 10, 512)	0
block5_conv1 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv2 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv3 (Conv2D)	(None, 10, 10, 512)	2359808
block5_pool (MaxPooling2D)	(None, 5, 5, 512)	0
<hr/>		
Total params: 14,714,688		
Trainable params: 14,714,688		
Non-trainable params: 0		

Case Study 2

❖ Use pretrained weights as init weights

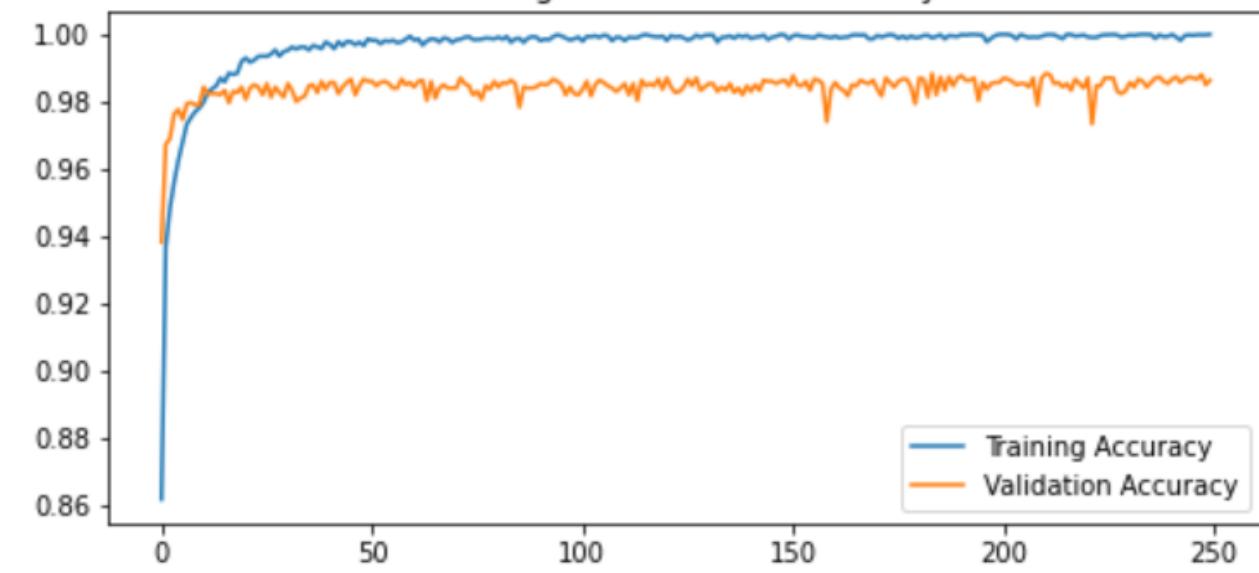
```
1 # process data
2 data_augmentation = tf.keras.Sequential([
3     tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
4     tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
5     tf.keras.layers.experimental.preprocessing.Rescaling(1./127.5, offset= -1)
6 ])
7
8 # flattening
9 flatten = tf.keras.layers.Flatten()
10
11 # final layer
12 prediction_layer = tf.keras.layers.Dense(1)
13
14 # construct a new network
15 inputs = tf.keras.Input(shape=(160, 160, 3))
16 x = data_augmentation(inputs)
17 x = base_model(x)
18 x = flatten(x)
19 outputs = prediction_layer(x)
20 model = tf.keras.Model(inputs, outputs)
```

Case Study 2

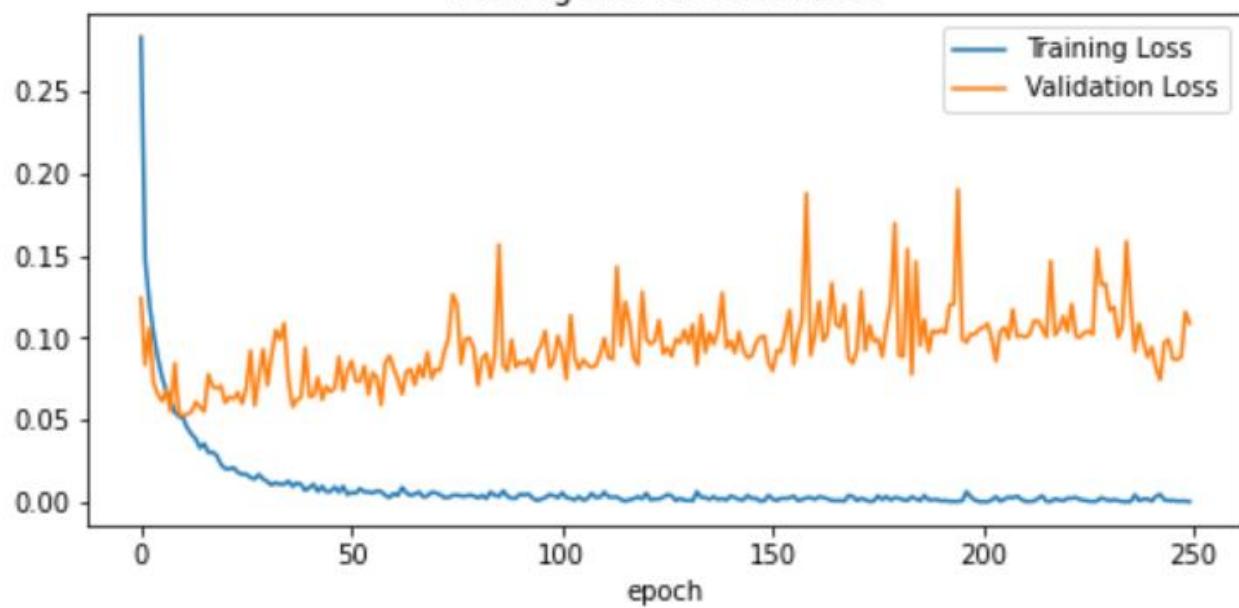
❖ Use pretrained weights init weights

```
2 model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),  
3                 optimizer = tf.keras.optimizers.Adam(lr=0.00001),  
4                 metrics=['accuracy'])  
5  
6 history_fine = model.fit(train_dataset,  
7                           epochs=250,  
8                           validation_data=validation_dataset)
```

Training and Validation Accuracy



Training and Validation Loss



Case Study 2

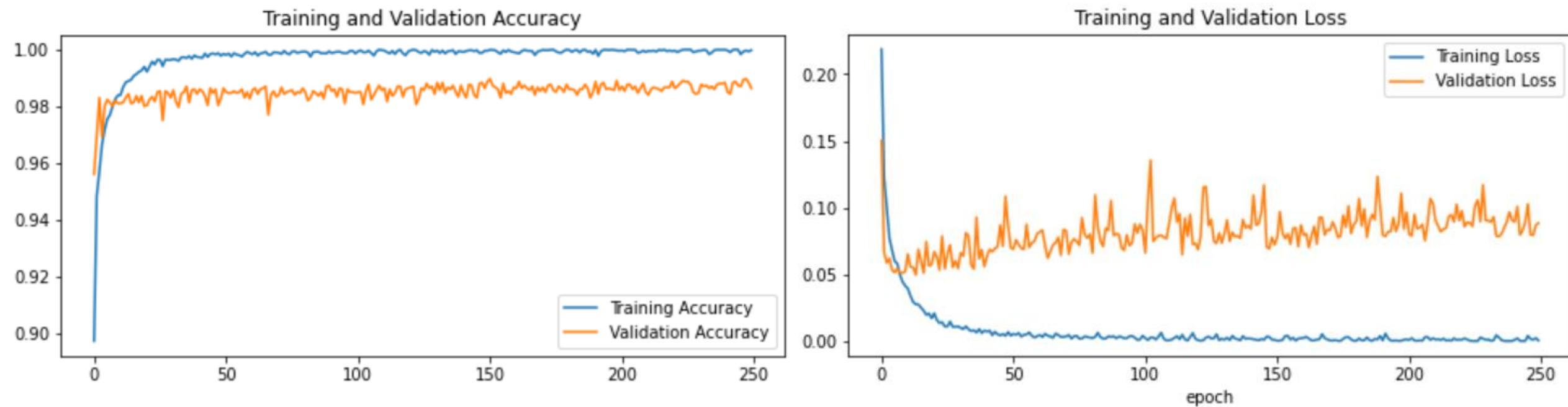
- ❖ Use pretrained weights as init weights
- ❖ Global max pooling

```
1 # process data
2 data_augmentation = tf.keras.Sequential([
3     tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
4     tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
5     tf.keras.layers.experimental.preprocessing.Rescaling(1./127.5, offset= -1)
6 ])
7
8 # flattening
9 global_max = tf.keras.layers.GlobalMaxPool2D()
10
11 # final layer
12 prediction_layer = tf.keras.layers.Dense(1)
13
14 # construct a new network
15 inputs = tf.keras.Input(shape=(160, 160, 3))
16 x = data_augmentation(inputs)
17 x = base_model(x)
18 x = global_max(x)
19 outputs = prediction_layer(x)
20 model = tf.keras.Model(inputs, outputs)
```

Case Study 2

- ❖ Use pretrained weights init weights
- ❖ Global max pooling

```
2 model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),  
3                 optimizer = tf.keras.optimizers.Adam(lr=0.00001),  
4                 metrics=['accuracy'])  
5  
6 history_fine = model.fit(train_dataset,  
7                           epochs=250,  
8                           validation_data=validation_dataset)
```



Case Study 2

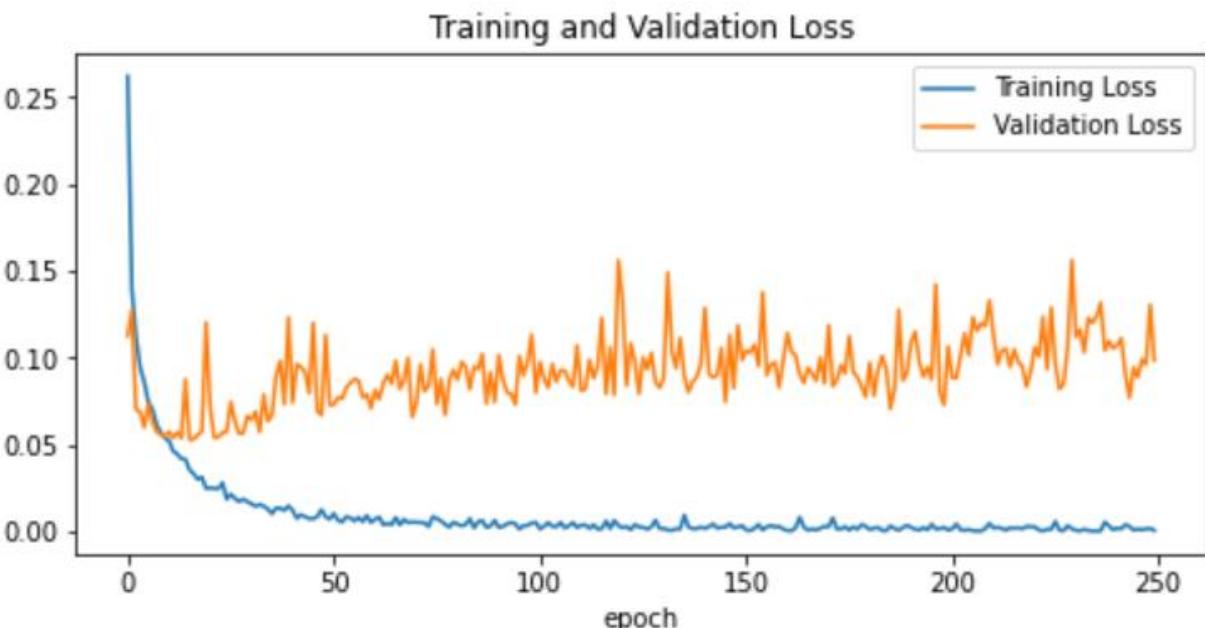
- ❖ Use pretrained weights as init weights
- ❖ Global average pooling

```
1 # process data
2 data_augmentation = tf.keras.Sequential([
3     tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
4     tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
5     tf.keras.layers.experimental.preprocessing.Rescaling(1./127.5, offset= -1)
6 ])
7
8 # flattening
9 global_average = tf.keras.layers.GlobalAveragePooling2D()
10
11 # final layer
12 prediction_layer = tf.keras.layers.Dense(1)
13
14 # construct a new network
15 inputs = tf.keras.Input(shape=(160, 160, 3))
16 x = data_augmentation(inputs)
17 x = base_model(x)
18 x = global_average(x)
19 outputs = prediction_layer(x)
20 model = tf.keras.Model(inputs, outputs)
```

Case Study 2

- ❖ Use pretrained weights init weights
- ❖ Global average pooling

```
2 model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),  
3                 optimizer = tf.keras.optimizers.Adam(lr=0.00001),  
4                 metrics=['accuracy'])  
5  
6 history_fine = model.fit(train_dataset,  
7                           epochs=250,  
8                           validation_data=validation_dataset)
```



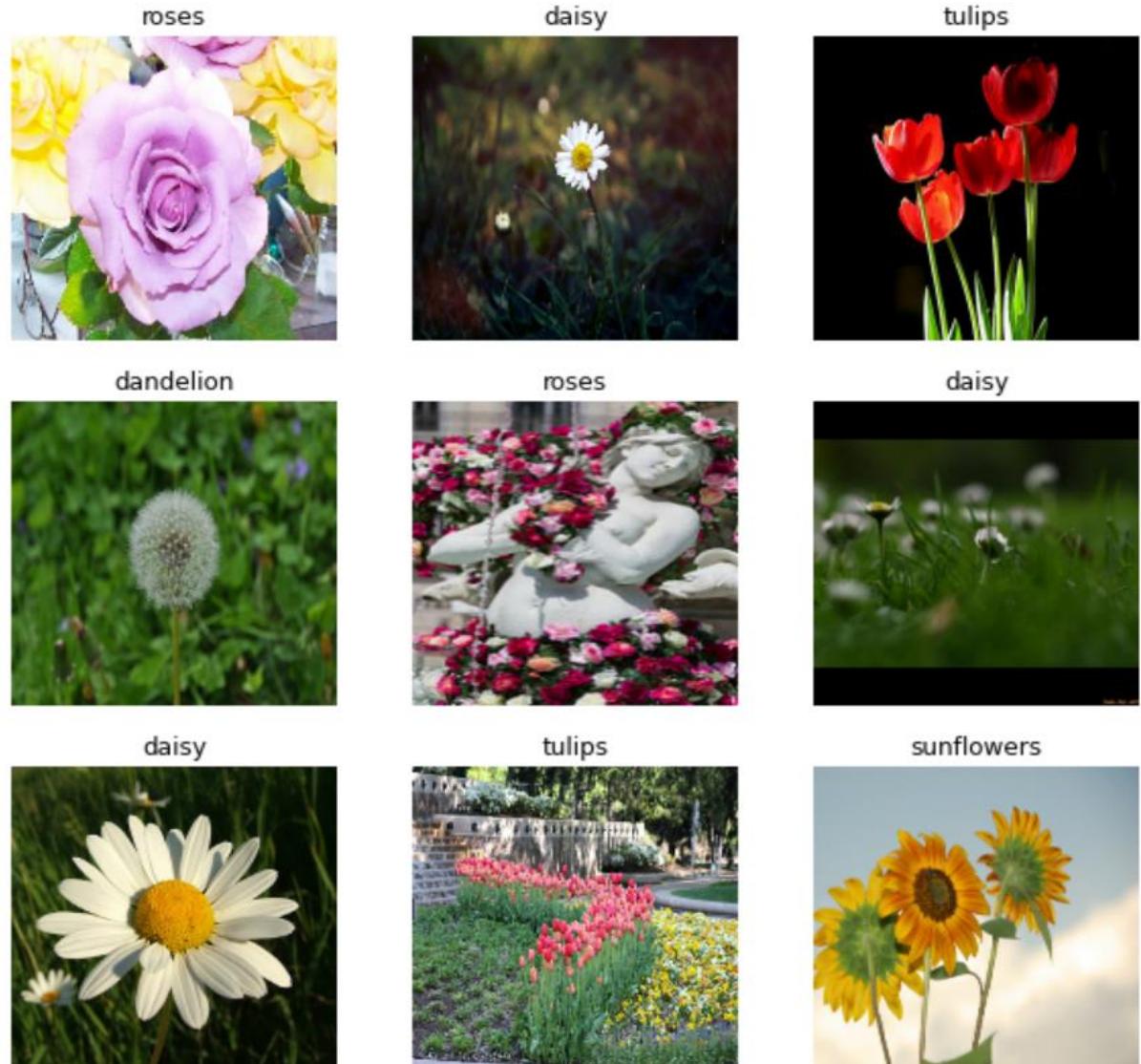
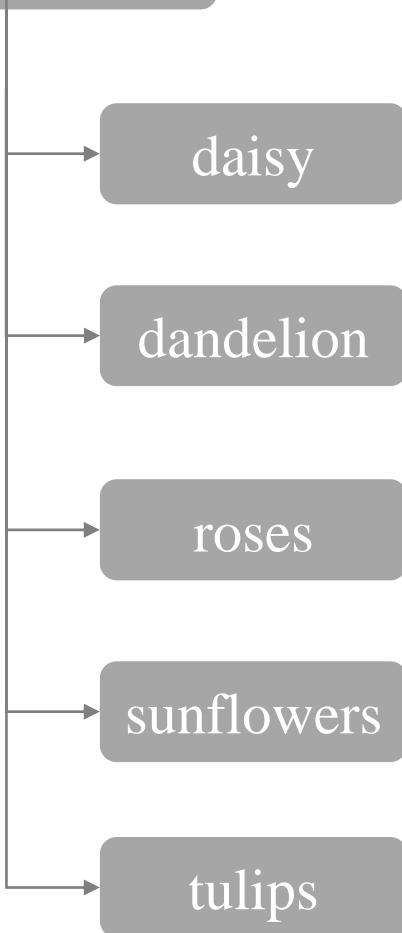
Outline

- Review
- Reuse a Pre-trained Model
- Case Study 1
- Case Study 2
- Case Study 3
- Case Study 4
- Case Study 5

Case Study 3

❖ In keras

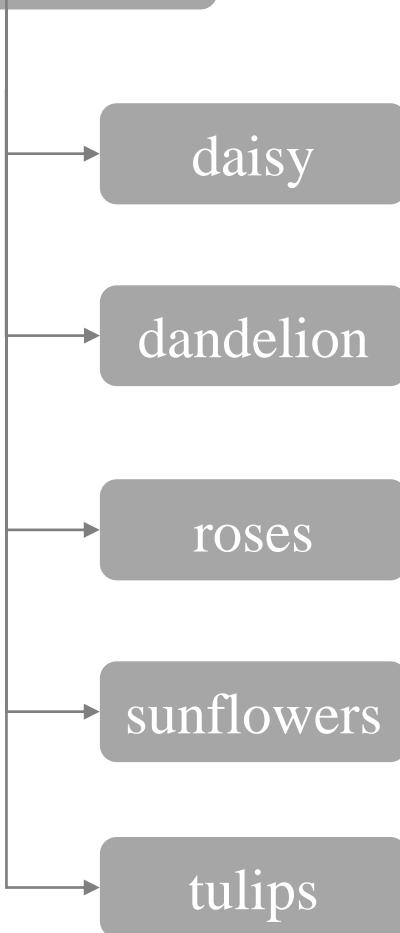
flower_photos



Case Study 3

❖ In keras

flower_photos



```
1 PATH = 'flower_photos/'  
2  
3 train_dir = os.path.join(PATH, 'train')  
4 validation_dir = os.path.join(PATH, 'validation')  
5  
6 BATCH_SIZE = 256  
7 IMG_SIZE = (160, 160)  
8 BUFFER_SIZE = BATCH_SIZE*5  
9  
10 train_dataset = tf.keras.preprocessing.image_dataset_from_directory(  
11     PATH,  
12     validation_split=0.2,  
13     subset="training",  
14     seed=123,  
15     image_size=IMG_SIZE,  
16     batch_size=BATCH_SIZE)
```

Found 3670 files belonging to 5 classes.
Using 2936 files for training.

```
1 validation_dataset = tf.keras.preprocessing.image_dataset_from_directory(  
2     PATH,  
3     validation_split=0.2,  
4     subset="validation",  
5     seed=123,  
6     image_size=IMG_SIZE,  
7     batch_size=BATCH_SIZE)
```

Found 3670 files belonging to 5 classes.
Using 734 files for validation.

Case Study 3

❖ Flower dataset

❖ Train from scratch

Model: "vgg16"

Layer (type)	Output Shape	Param #
<hr/>		
input_1 (InputLayer)	[None, 160, 160, 3]	0
block1_conv1 (Conv2D)	(None, 160, 160, 64)	1792
block1_conv2 (Conv2D)	(None, 160, 160, 64)	36928
block1_pool (MaxPooling2D)	(None, 80, 80, 64)	0
block2_conv1 (Conv2D)	(None, 80, 80, 128)	73856
block2_conv2 (Conv2D)	(None, 80, 80, 128)	147584
block2_pool (MaxPooling2D)	(None, 40, 40, 128)	0
block3_conv1 (Conv2D)	(None, 40, 40, 256)	295168
block3_conv2 (Conv2D)	(None, 40, 40, 256)	590080
block3_conv3 (Conv2D)	(None, 40, 40, 256)	590080
block3_pool (MaxPooling2D)	(None, 20, 20, 256)	0

```
1 # top=False
2 model = tf.keras.applications.VGG16(input_shape=(160,160,3),
3                                     include_top=False,
4                                     weights=None)
5 model.summary()
```

block4_conv1 (Conv2D)	(None, 20, 20, 512)	1180160
block4_conv2 (Conv2D)	(None, 20, 20, 512)	2359808
block4_conv3 (Conv2D)	(None, 20, 20, 512)	2359808
block4_pool (MaxPooling2D)	(None, 10, 10, 512)	0
block5_conv1 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv2 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv3 (Conv2D)	(None, 10, 10, 512)	2359808
block5_pool (MaxPooling2D)	(None, 5, 5, 512)	0
<hr/>		
Total params: 14,714,688		
Trainable params: 14,714,688		
Non-trainable params: 0		

Case Study 3

❖ Flower dataset

❖ Train from scratch

```
# process data
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
    tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
    tf.keras.layers.experimental.preprocessing.Rescaling(1./127.5, offset=-1)
])

# flattening
flatten = tf.keras.layers.Flatten()

# final layer
prediction_layer = tf.keras.layers.Dense(5)

# construct a new network
inputs = tf.keras.Input(shape=(160, 160, 3))
x = data_augmentation(inputs)
x = base_model(x)
x = flatten(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)
```

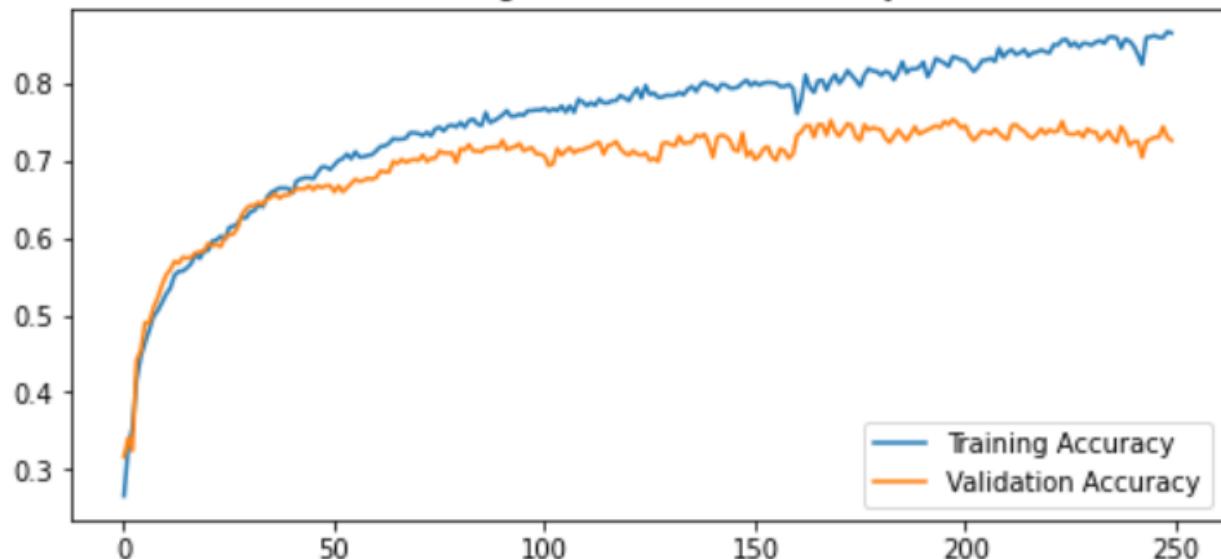
Case Study 3

❖ Flower dataset

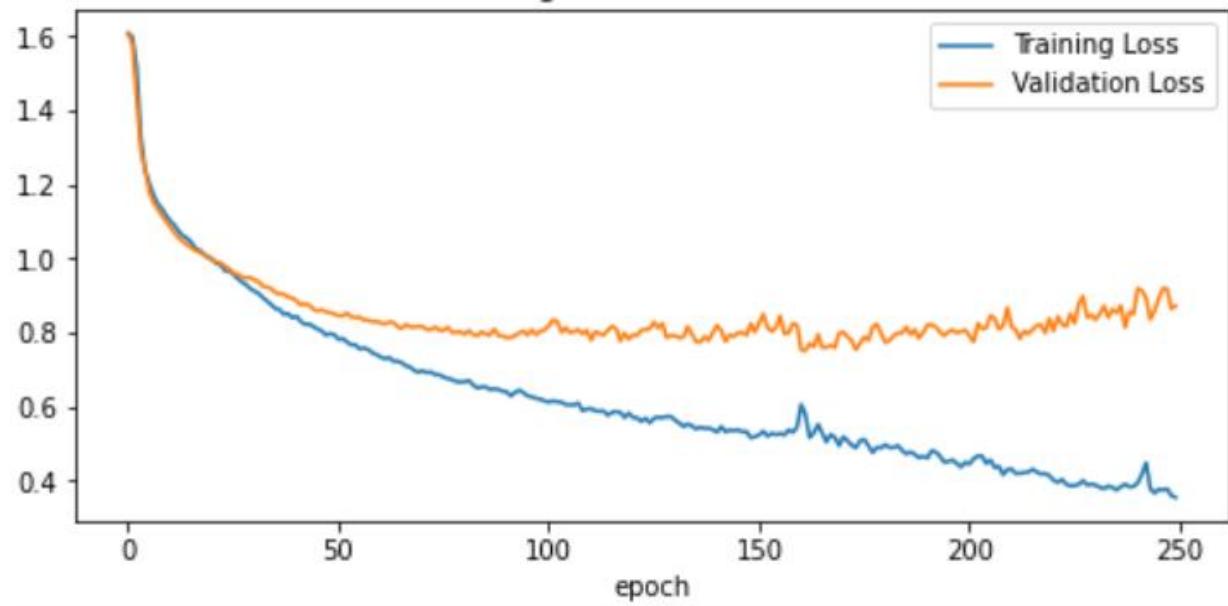
❖ Train from scratch

```
model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),  
              optimizer = tf.keras.optimizers.Adam(lr=0.00001),  
              metrics=['accuracy'])  
  
history_fine = model.fit(train_dataset,  
                         epochs=250,  
                         validation_data=validation_dataset)
```

Training and Validation Accuracy



Training and Validation Loss



Case Study 3

❖ Transfer learning

```
1 base_model = tf.keras.applications.VGG16(input_shape=(160,160,3),  
2 include_top=False,  
3 weights='imagenet')  
4 base_model.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
=====		
input_1 (InputLayer)	[(None, 160, 160, 3)]	0
block1_conv1 (Conv2D)	(None, 160, 160, 64)	1792
block1_conv2 (Conv2D)	(None, 160, 160, 64)	36928
block1_pool (MaxPooling2D)	(None, 80, 80, 64)	0
block2_conv1 (Conv2D)	(None, 80, 80, 128)	73856
block2_conv2 (Conv2D)	(None, 80, 80, 128)	147584
block2_pool (MaxPooling2D)	(None, 40, 40, 128)	0
block3_conv1 (Conv2D)	(None, 40, 40, 256)	295168
block3_conv2 (Conv2D)	(None, 40, 40, 256)	590080
block3_conv3 (Conv2D)	(None, 40, 40, 256)	590080
block3_pool (MaxPooling2D)	(None, 20, 20, 256)	0

block4_conv1 (Conv2D)	(None, 20, 20, 512)	1180160
block4_conv2 (Conv2D)	(None, 20, 20, 512)	2359808
block4_conv3 (Conv2D)	(None, 20, 20, 512)	2359808
block4_pool (MaxPooling2D)	(None, 10, 10, 512)	0
block5_conv1 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv2 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv3 (Conv2D)	(None, 10, 10, 512)	2359808
block5_pool (MaxPooling2D)	(None, 5, 5, 512)	0
=====		
Total params: 14,714,688		
Trainable params: 14,714,688		
Non-trainable params: 0		

Case Study 3

❖ Transfer learning

```
# freeze the base model
base_model.trainable = False

# process data
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
    tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
    tf.keras.layers.experimental.preprocessing.Rescaling(1./127.5, offset=-1)
])

# flattening
flatten = tf.keras.layers.Flatten()

# final layer
prediction_layer = tf.keras.layers.Dense(5)

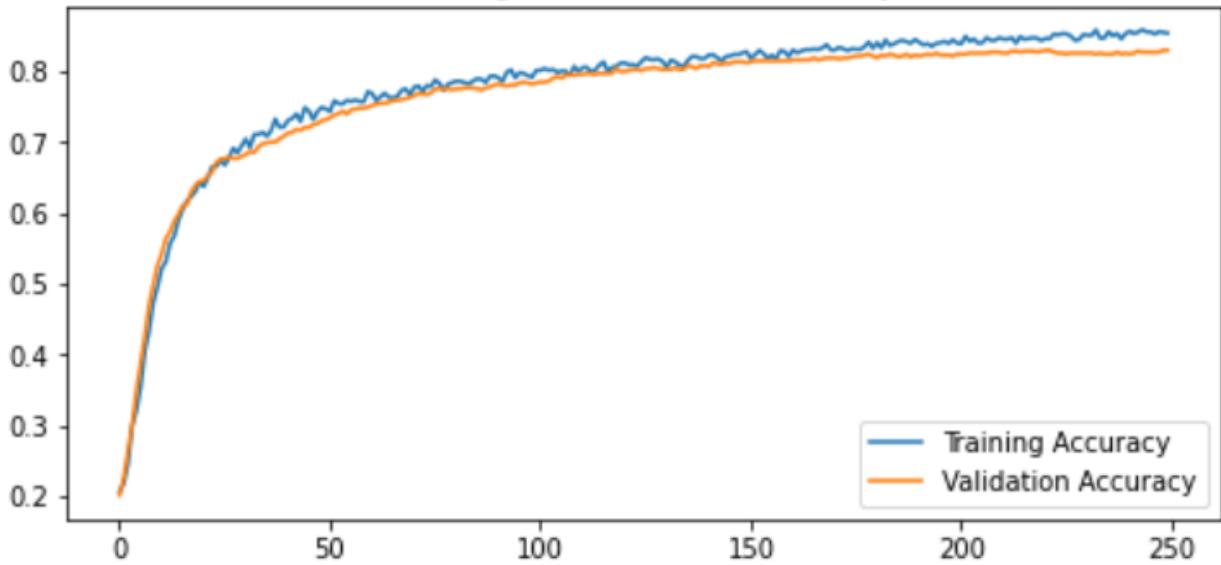
# construct a new network
inputs = tf.keras.Input(shape=(160, 160, 3))
x = data_augmentation(inputs)
x = base_model(x)
x = flatten(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)
```

Case Study 3

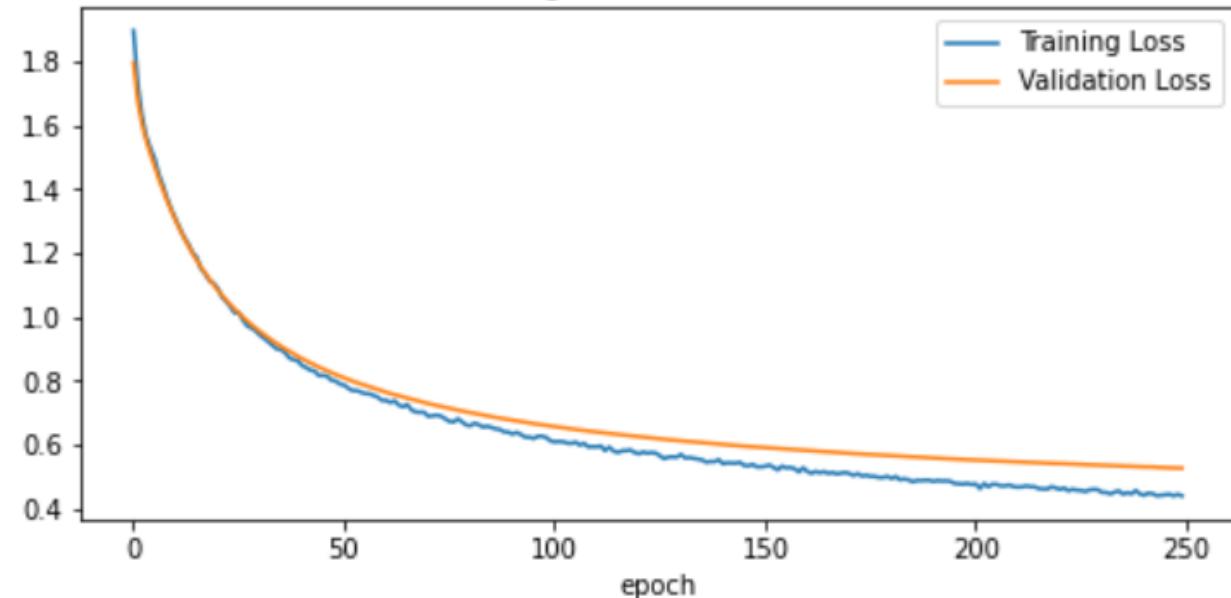
❖ Transfer learning

```
model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),  
              optimizer = tf.keras.optimizers.Adam(lr=0.00001),  
              metrics=['accuracy'])  
  
history_fine = model.fit(train_dataset,  
                         epochs=250,  
                         validation_data=validation_dataset)
```

Training and Validation Accuracy



Training and Validation Loss



Case Study 3

❖ Fine tuning

```
1 base_model = tf.keras.applications.VGG16(input_shape=(160,160,3),  
2     include_top=False,  
3     weights='imagenet')  
4 base_model.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
=====		
input_1 (InputLayer)	[(None, 160, 160, 3)]	0
block1_conv1 (Conv2D)	(None, 160, 160, 64)	1792
block1_conv2 (Conv2D)	(None, 160, 160, 64)	36928
block1_pool (MaxPooling2D)	(None, 80, 80, 64)	0
block2_conv1 (Conv2D)	(None, 80, 80, 128)	73856
block2_conv2 (Conv2D)	(None, 80, 80, 128)	147584
block2_pool (MaxPooling2D)	(None, 40, 40, 128)	0
block3_conv1 (Conv2D)	(None, 40, 40, 256)	295168
block3_conv2 (Conv2D)	(None, 40, 40, 256)	590080
block3_conv3 (Conv2D)	(None, 40, 40, 256)	590080
block3_pool (MaxPooling2D)	(None, 20, 20, 256)	0

block4_conv1 (Conv2D)	(None, 20, 20, 512)	1180160
block4_conv2 (Conv2D)	(None, 20, 20, 512)	2359808
block4_conv3 (Conv2D)	(None, 20, 20, 512)	2359808
block4_pool (MaxPooling2D)	(None, 10, 10, 512)	0
block5_conv1 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv2 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv3 (Conv2D)	(None, 10, 10, 512)	2359808
block5_pool (MaxPooling2D)	(None, 5, 5, 512)	0
=====		
Total params: 14,714,688		
Trainable params: 14,714,688		
Non-trainable params: 0		

Case Study 3

❖ Fine tuning

```
# Freeze some first the layers
fine_tune_at = 14
for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False

# process data
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
    tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
    tf.keras.layers.experimental.preprocessing.Rescaling(1./127.5, offset=-1)
])

# flattening
flatten = tf.keras.layers.Flatten()

# final layer
prediction_layer = tf.keras.layers.Dense(5)

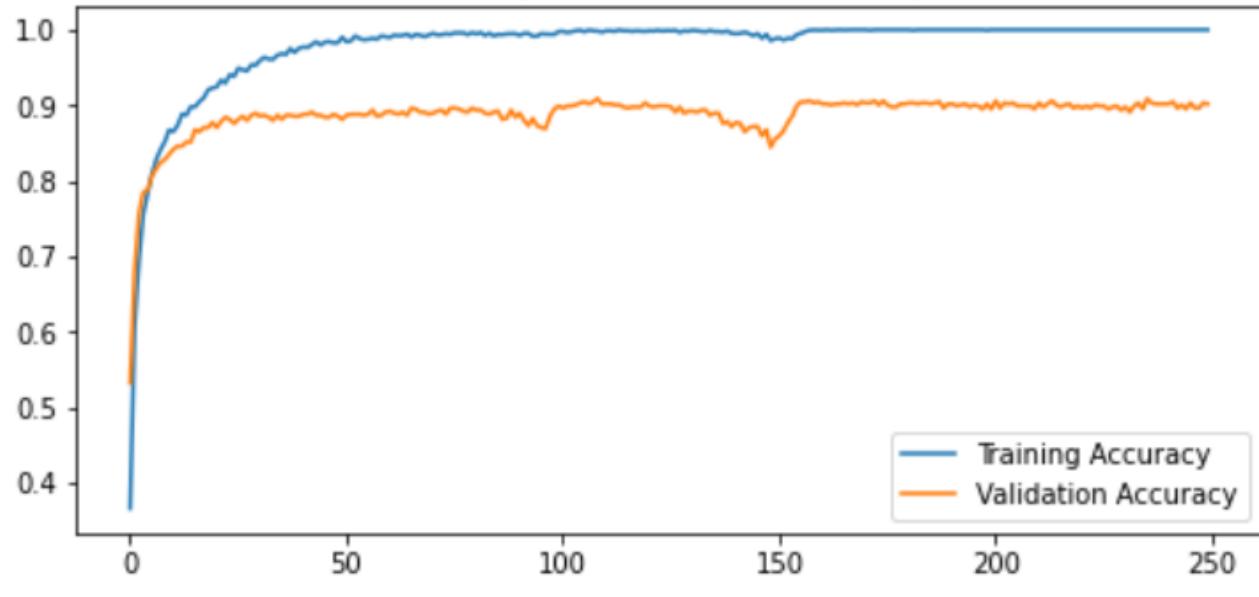
# construct a new network
inputs = tf.keras.Input(shape=(160, 160, 3))
x = data_augmentation(inputs)
x = base_model(x)
x = flatten(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)
```

Case Study 3

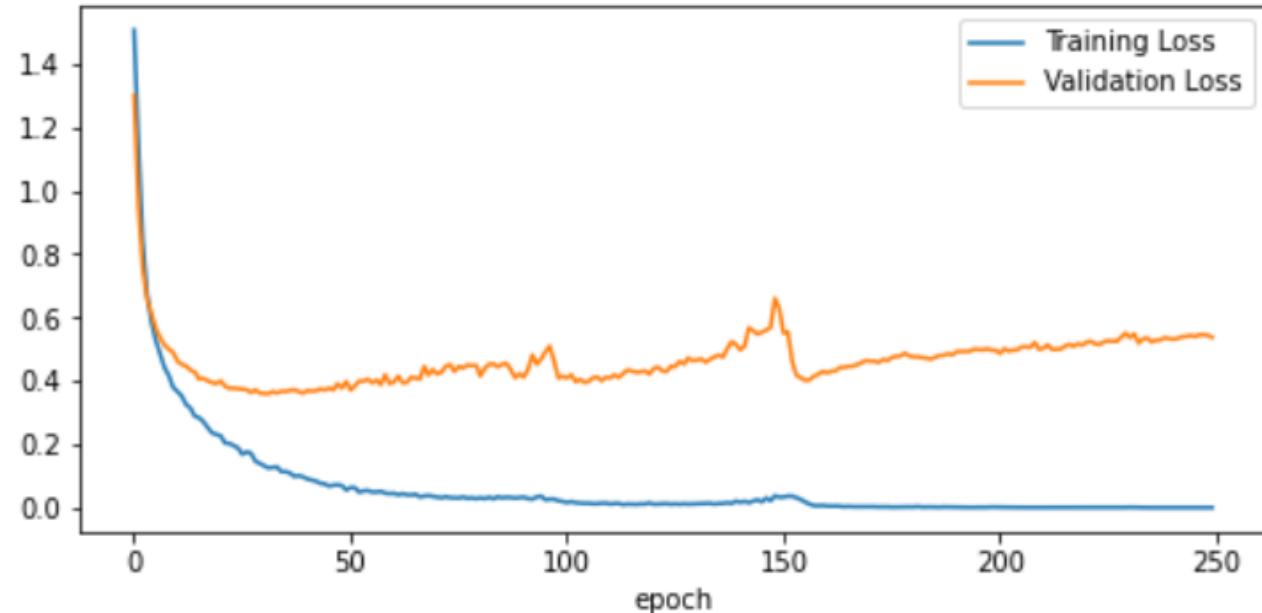
❖ Fine tuning

```
model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),  
              optimizer = tf.keras.optimizers.Adam(lr=0.00001),  
              metrics=['accuracy'])  
  
history_fine = model.fit(train_dataset,  
                         epochs=250,  
                         validation_data=validation_dataset)
```

Training and Validation Accuracy



Training and Validation Loss



Case Study 3

❖ Use pretrained weights as init weights

```
1 base_model = tf.keras.applications.VGG16(input_shape=(160,160,3),  
2                                         include_top=False,  
3                                         weights='imagenet')  
4 base_model.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
=====		
input_1 (InputLayer)	[(None, 160, 160, 3)]	0
block1_conv1 (Conv2D)	(None, 160, 160, 64)	1792
block1_conv2 (Conv2D)	(None, 160, 160, 64)	36928
block1_pool (MaxPooling2D)	(None, 80, 80, 64)	0
block2_conv1 (Conv2D)	(None, 80, 80, 128)	73856
block2_conv2 (Conv2D)	(None, 80, 80, 128)	147584
block2_pool (MaxPooling2D)	(None, 40, 40, 128)	0
block3_conv1 (Conv2D)	(None, 40, 40, 256)	295168
block3_conv2 (Conv2D)	(None, 40, 40, 256)	590080
block3_conv3 (Conv2D)	(None, 40, 40, 256)	590080
block3_pool (MaxPooling2D)	(None, 20, 20, 256)	0

block4_conv1 (Conv2D)	(None, 20, 20, 512)	1180160
block4_conv2 (Conv2D)	(None, 20, 20, 512)	2359808
block4_conv3 (Conv2D)	(None, 20, 20, 512)	2359808
block4_pool (MaxPooling2D)	(None, 10, 10, 512)	0
block5_conv1 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv2 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv3 (Conv2D)	(None, 10, 10, 512)	2359808
block5_pool (MaxPooling2D)	(None, 5, 5, 512)	0
=====		
Total params: 14,714,688		
Trainable params: 14,714,688		
Non-trainable params: 0		

Case Study 3

❖ Use pretrained weights as init weights

```
# process data
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
    tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
    tf.keras.layers.experimental.preprocessing.Rescaling(1./127.5, offset= -1)
])

# flattening
flatten = tf.keras.layers.Flatten()

# final layer
prediction_layer = tf.keras.layers.Dense(5)

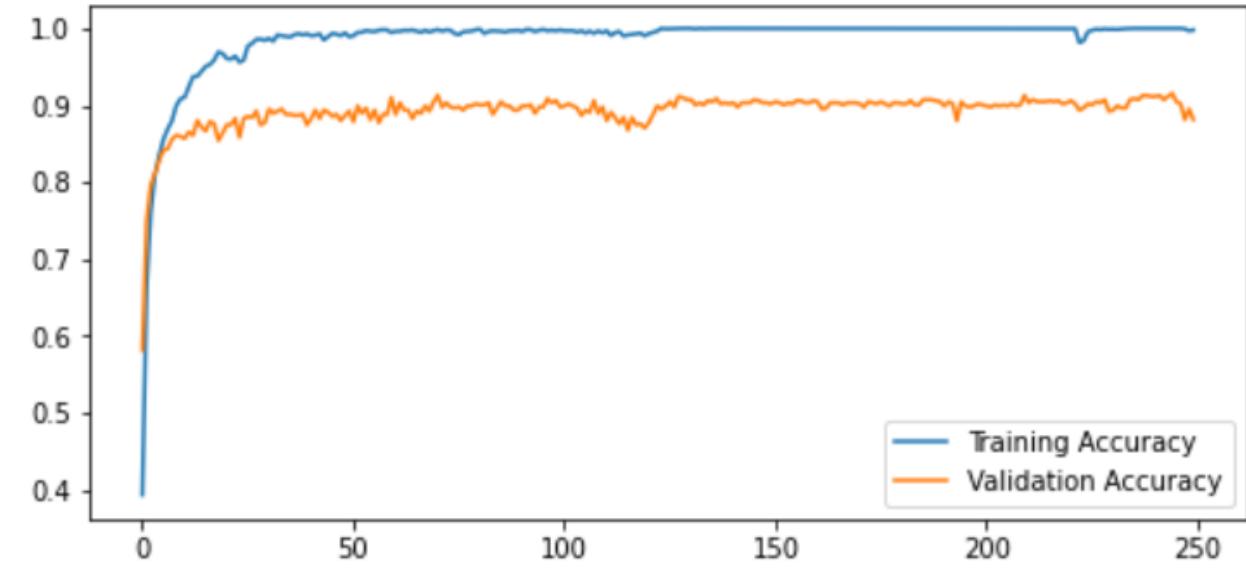
# construct a new network
inputs = tf.keras.Input(shape=(160, 160, 3))
x = data_augmentation(inputs)
x = base_model(x)
x = flatten(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)
```

Case Study 3

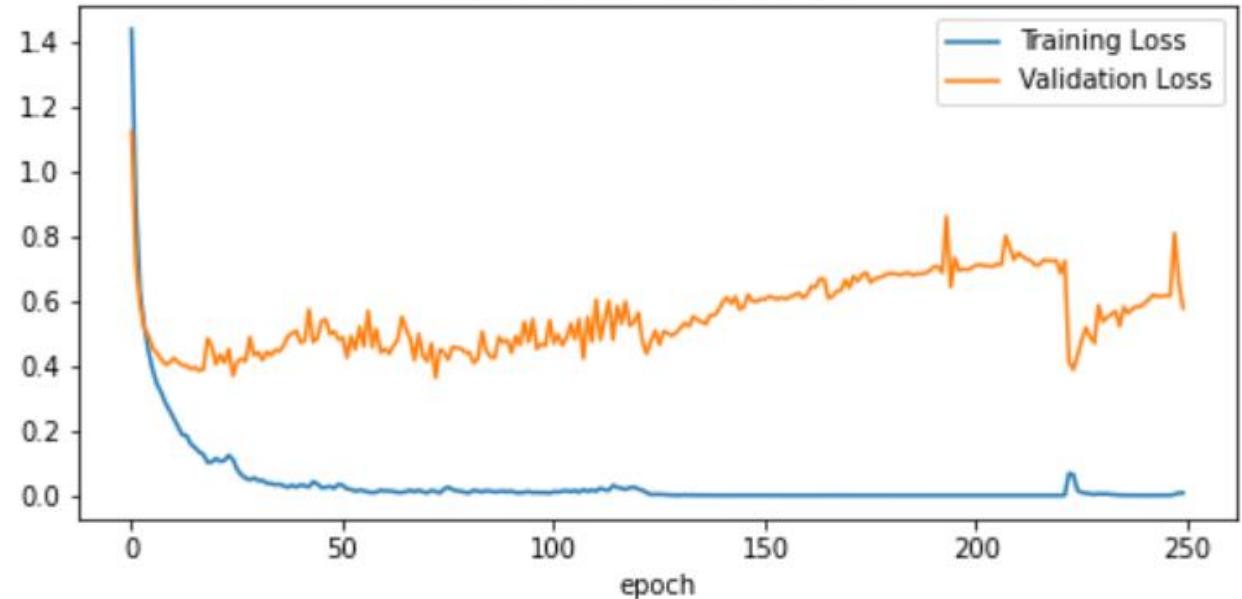
❖ Use pretrained weights as init weights

```
model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),  
              optimizer = tf.keras.optimizers.Adam(lr=0.00001),  
              metrics=['accuracy'])  
  
history_fine = model.fit(train_dataset,  
                         epochs=250,  
                         validation_data=validation_dataset)
```

Training and Validation Accuracy



Training and Validation Loss



Case Study 3

- ❖ Use pretrained weights as init weights
- ❖ Global max pooling

```
# process data
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
    tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
    tf.keras.layers.experimental.preprocessing.Rescaling(1./127.5, offset= -1)
])

# flattening
global_max = tf.keras.layers.GlobalMaxPool2D()

# final layer
prediction_layer = tf.keras.layers.Dense(5)

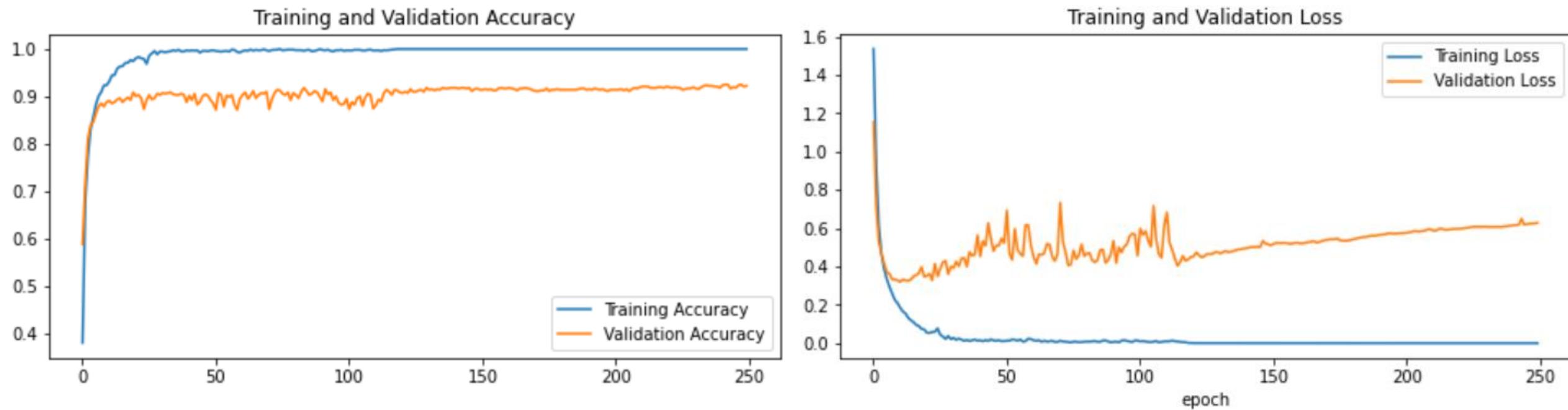
# construct a new network
inputs = tf.keras.Input(shape=(160, 160, 3))
x = data_augmentation(inputs)
x = base_model(x)
x = global_max(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)
```

Case Study 3

- ❖ Use pretrained weights as init weights

- ❖ Global max pooling

```
model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),  
              optimizer = tf.keras.optimizers.Adam(lr=0.00001),  
              metrics=['accuracy'])  
  
history_fine = model.fit(train_dataset,  
                         epochs=250,  
                         validation_data=validation_dataset)
```



Case Study 3

- ❖ Use pretrained weights as init weights
- ❖ Global average pooling

```
# process data
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
    tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
    tf.keras.layers.experimental.preprocessing.Rescaling(1./127.5, offset= -1)
])

# flattening
global_average = tf.keras.layers.GlobalAveragePooling2D()

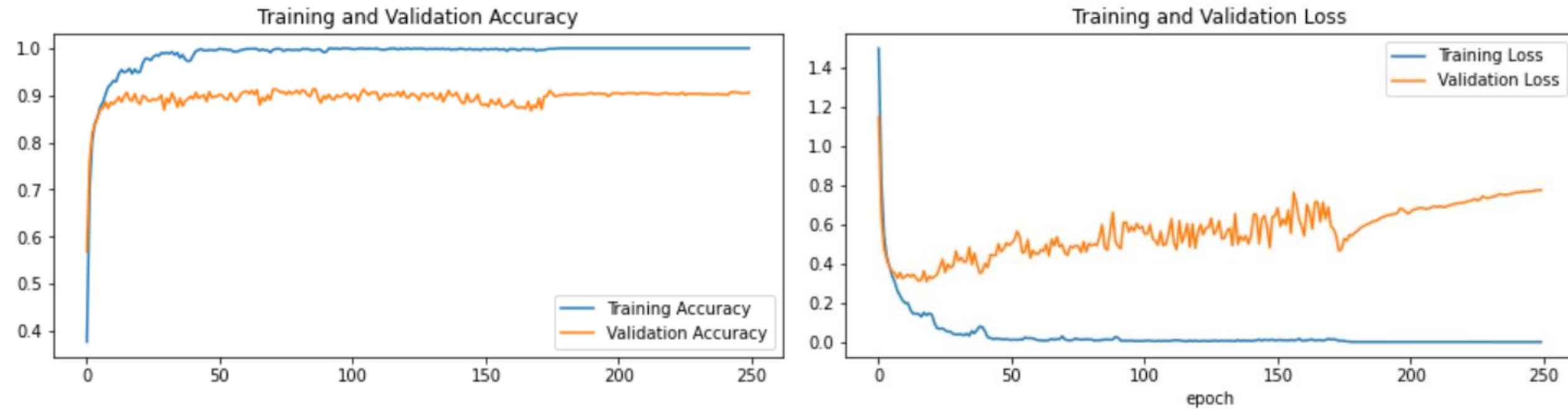
# final layer
prediction_layer = tf.keras.layers.Dense(5)

# construct a new network
inputs = tf.keras.Input(shape=(160, 160, 3))
x = data_augmentation(inputs)
x = base_model(x)
x = global_average(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)
```

Case Study 3

- ❖ Use pretrained weights as init weights
- ❖ Global average pooling

```
model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),  
              optimizer = tf.keras.optimizers.Adam(lr=0.00001),  
              metrics=['accuracy'])  
  
history_fine = model.fit(train_dataset,  
                         epochs=250,  
                         validation_data=validation_dataset)
```



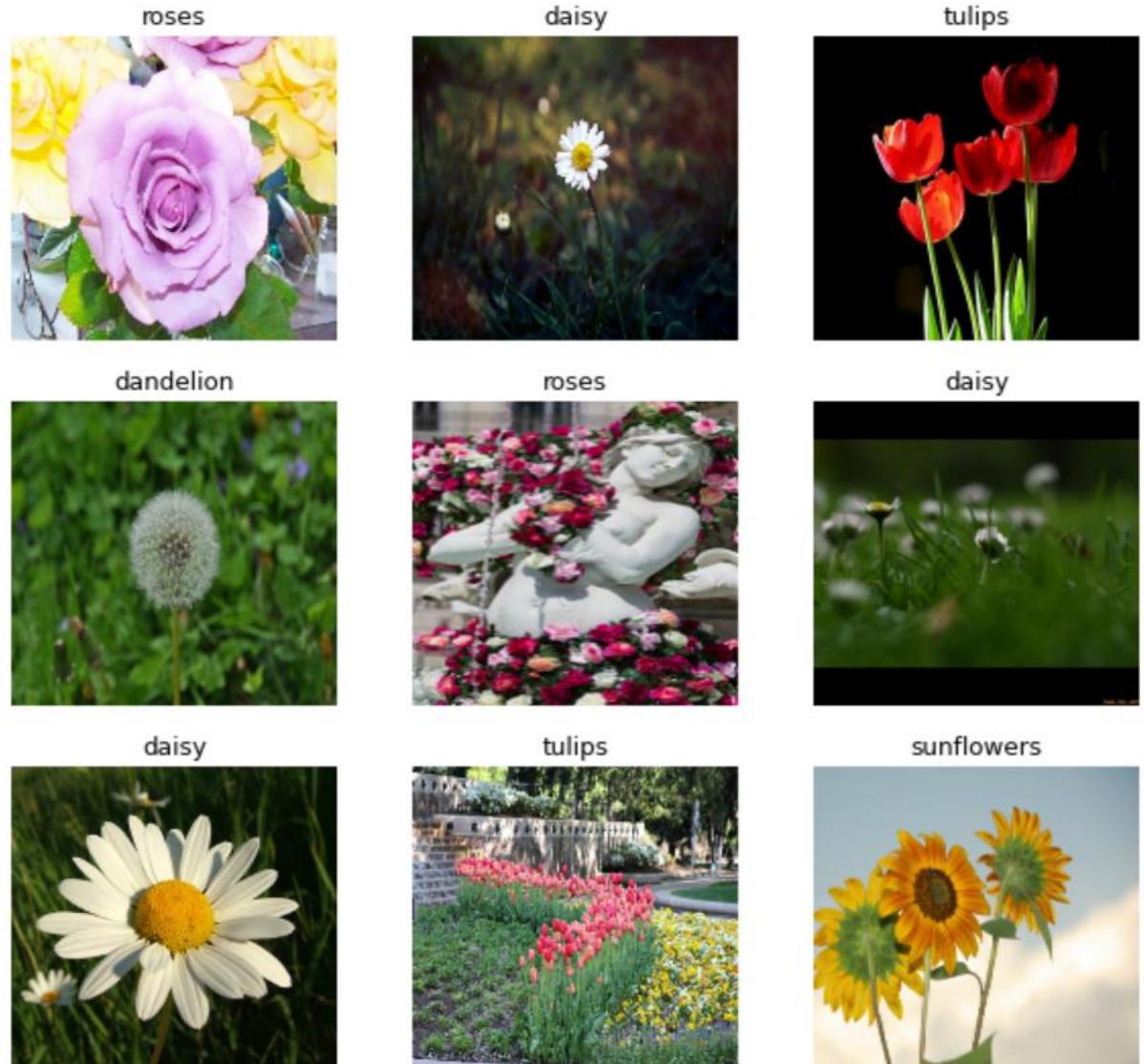
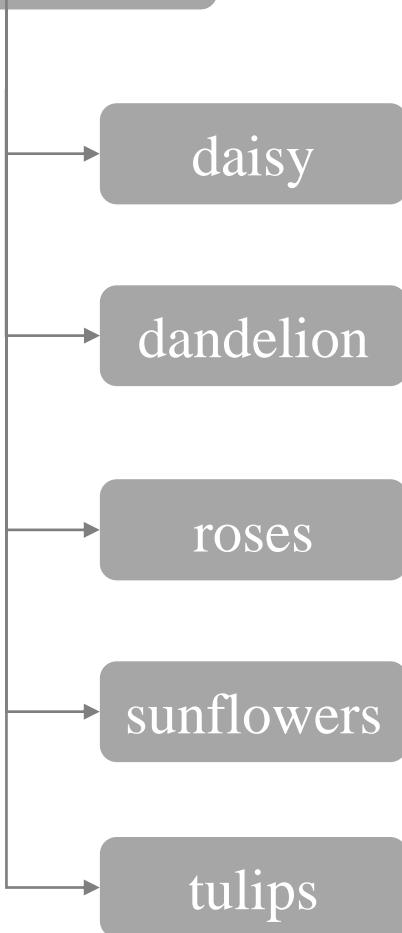
Outline

- Review
- Reuse a Pre-trained Model
- Case Study 1
- Case Study 2
- Case Study 3
- Case Study 4
- Case Study 5

Case Study 4

❖ In keras

flower_photos



Case Study 4

❖ In keras

flower_photos

daisy

dandelion

roses

sunflowers

tulips

```
1 PATH = 'flower_photos/'  
2  
3 train_dir = os.path.join(PATH, 'train')  
validation_dir = os.path.join(PATH, 'validation')  
4  
5 BATCH_SIZE = 256  
IMG_SIZE = (160, 160)  
BUFFER_SIZE = BATCH_SIZE*5  
6  
7 train_dataset = tf.keras.preprocessing.image_dataset_from_directory(  
8     PATH,  
9     validation_split=0.2,  
10    subset="training",  
11    seed=123,  
12    image_size=IMG_SIZE,  
13    batch_size=BATCH_SIZE)
```

Found 3670 files belonging to 5 classes.
Using 2936 files for training.

```
1 validation_dataset = tf.keras.preprocessing.image_dataset_from_directory(  
2     PATH,  
3     validation_split=0.2,  
4     subset="validation",  
5     seed=123,  
6     image_size=IMG_SIZE,  
7     batch_size=BATCH_SIZE)
```

Found 3670 files belonging to 5 classes.
Using 734 files for validation.

Case Study 4

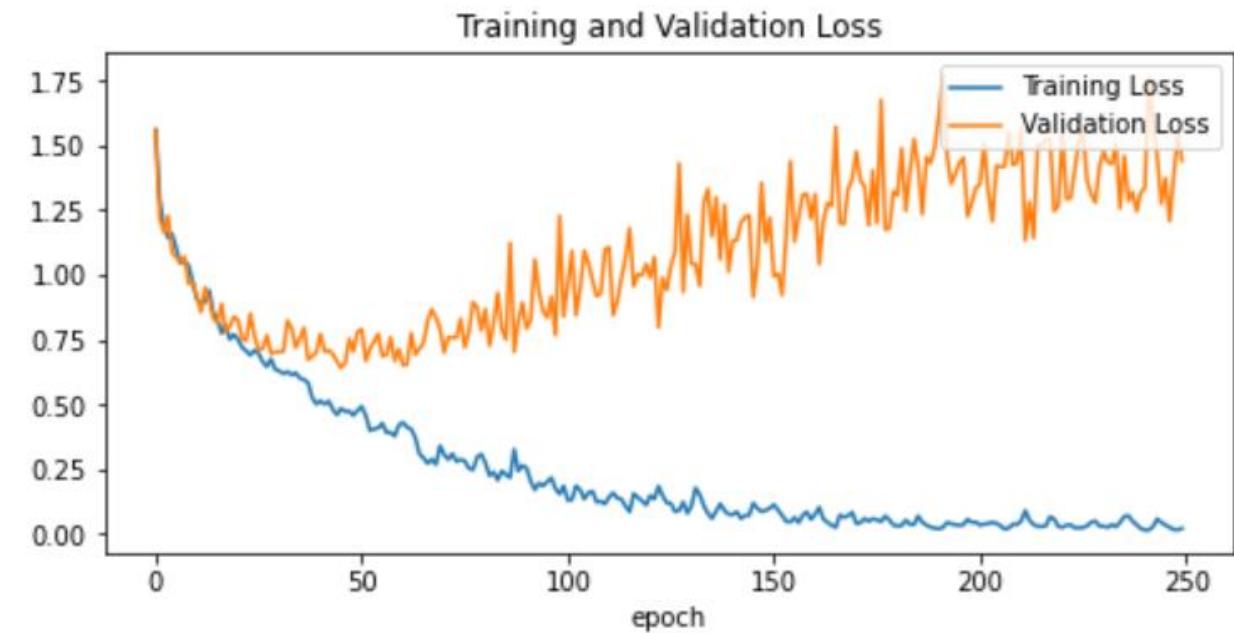
❖ Model

❖ Dropout 20%

```
1 # process data
2 data_augmentation = tf.keras.Sequential([
3     tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
4     tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
5     tf.keras.layers.experimental.preprocessing.Rescaling(1./127.5, offset=-1)
6 ])
7
8 # flattening
9 flatten = tf.keras.layers.Flatten()
10
11 # final layer
12 prediction_layer = tf.keras.layers.Dense(5)
13
14 # construct a new network
15 inputs = tf.keras.Input(shape=(160, 160, 3))
16 x = data_augmentation(inputs)
17 x = base_model(x)
18 x = flatten(x)
19 x = tf.keras.layers.Dropout(0.2)(x)
20 outputs = prediction_layer(x)
21 model = tf.keras.Model(inputs, outputs)
```

Case Study 4

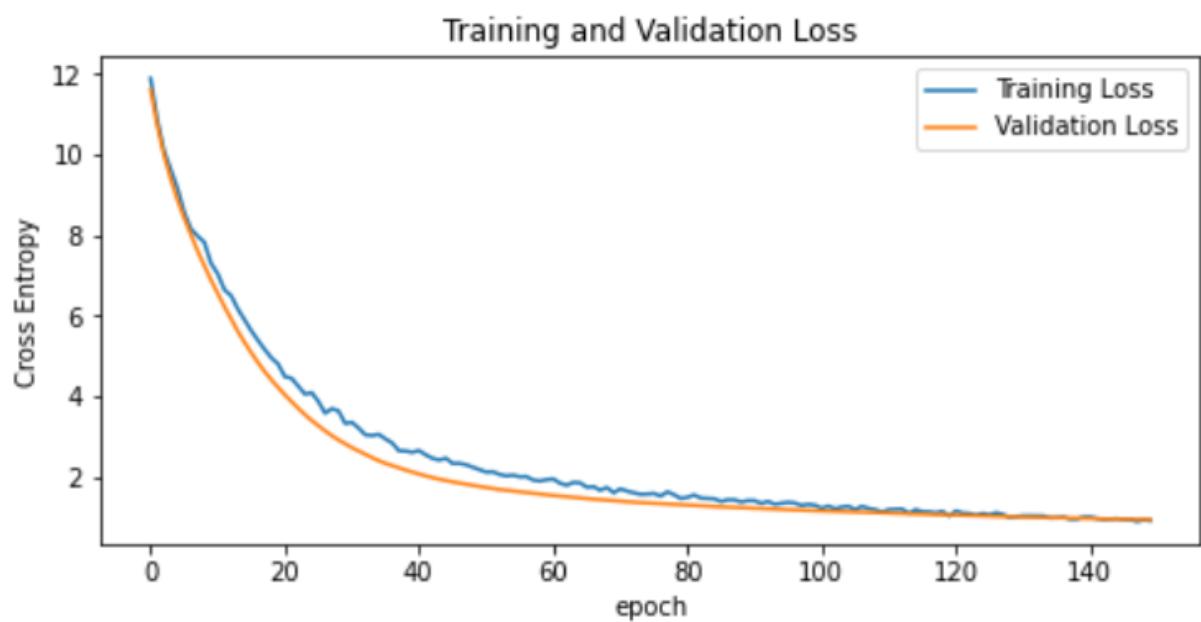
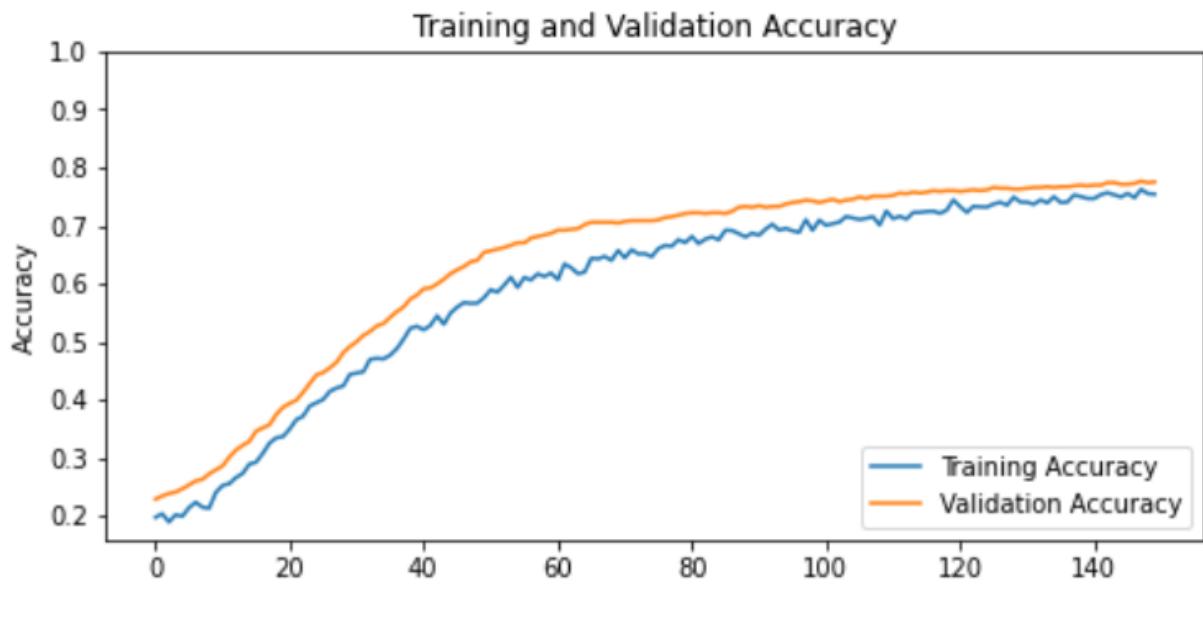
❖ Train from scratch



Use dropout 20% before prediction layer

Case Study 4

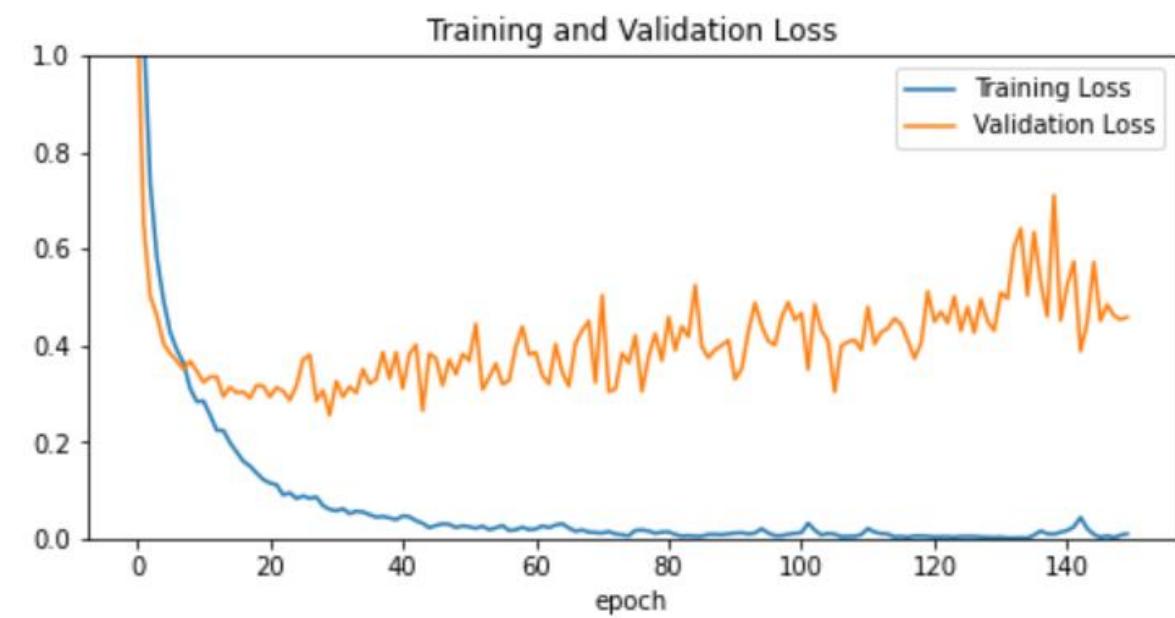
❖ Transfer learning



Use dropout 20% before prediction layer

Case Study 4

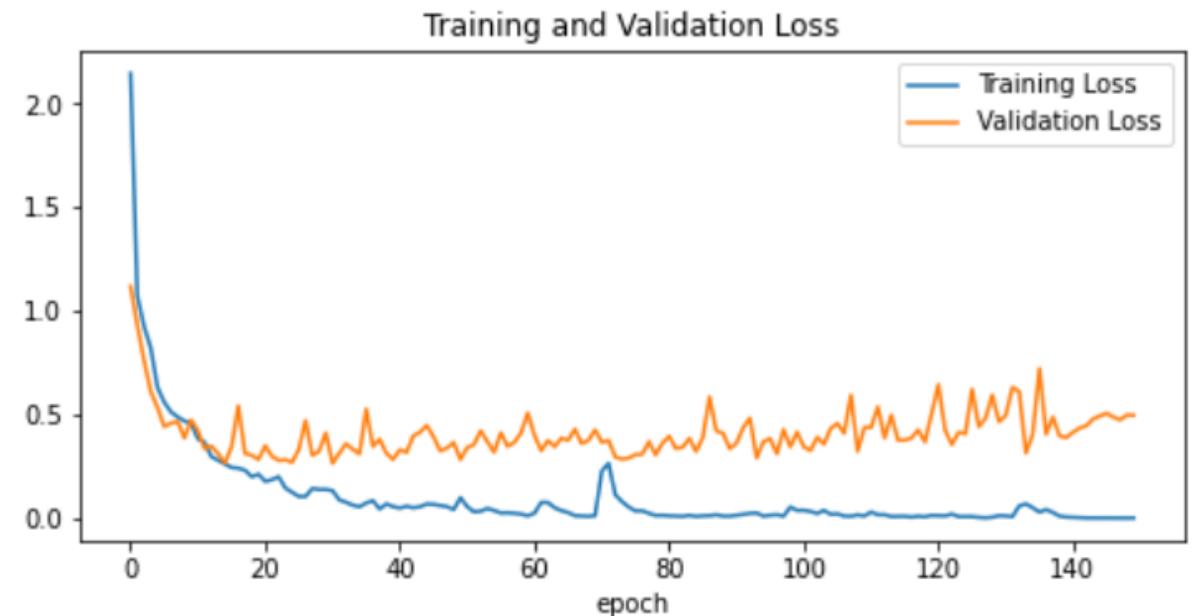
❖ Fine tuning



Use dropout 20% before prediction layer

Case Study 4

❖ For initialization



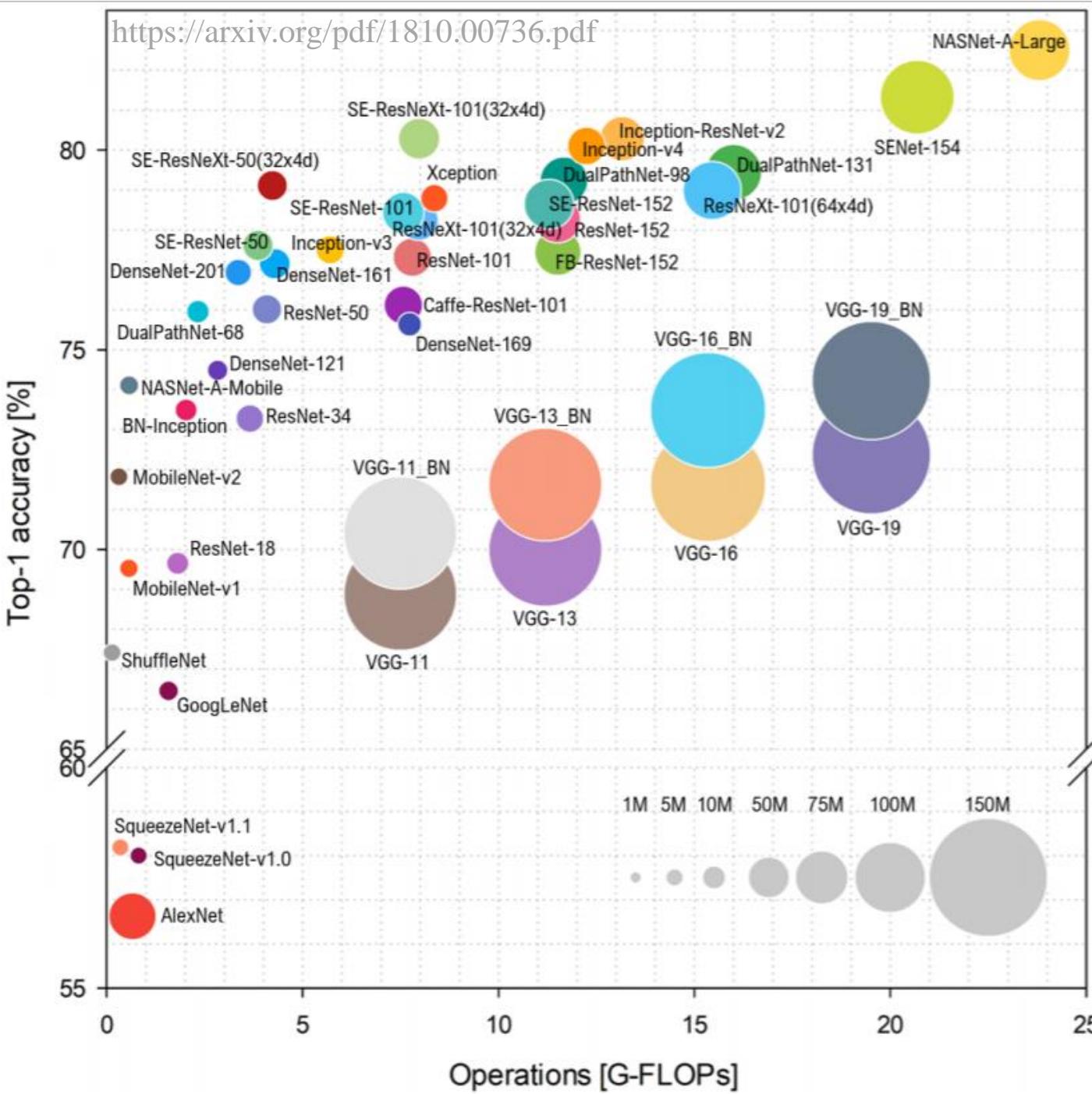
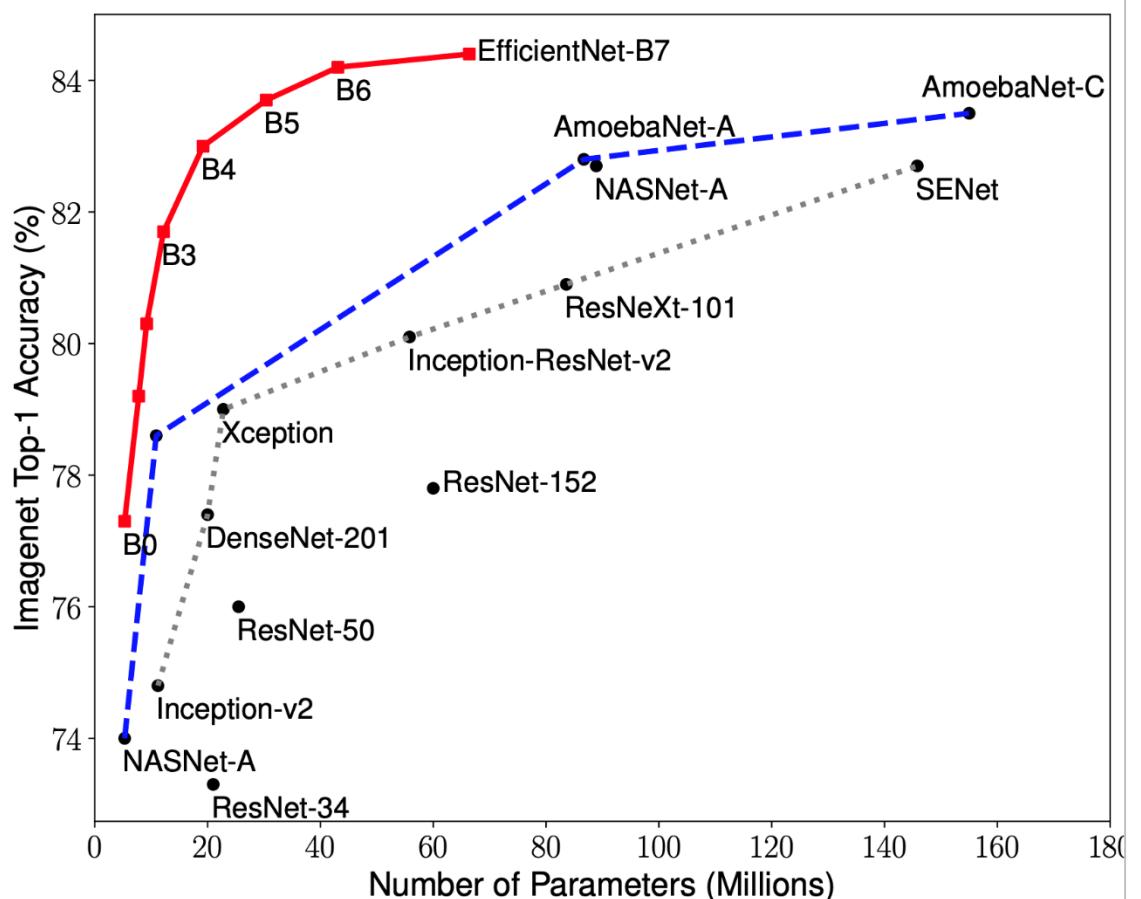
Use dropout 20% before prediction layer

Outline

- Review
- Reuse a Pre-trained Model
- Case Study 1
- Case Study 2
- Case Study 3
- Case Study 4
- Case Study 5

Different Networks

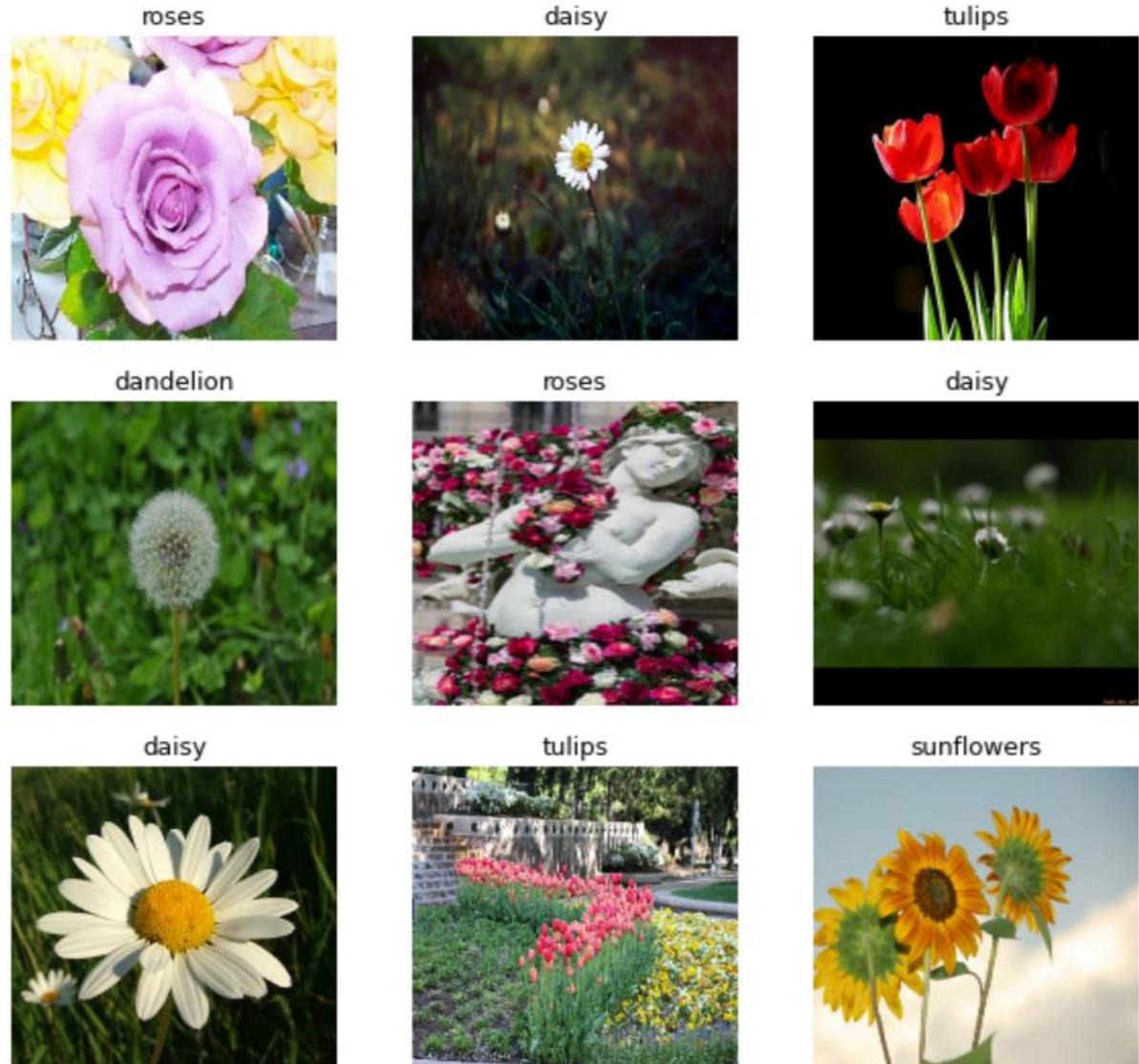
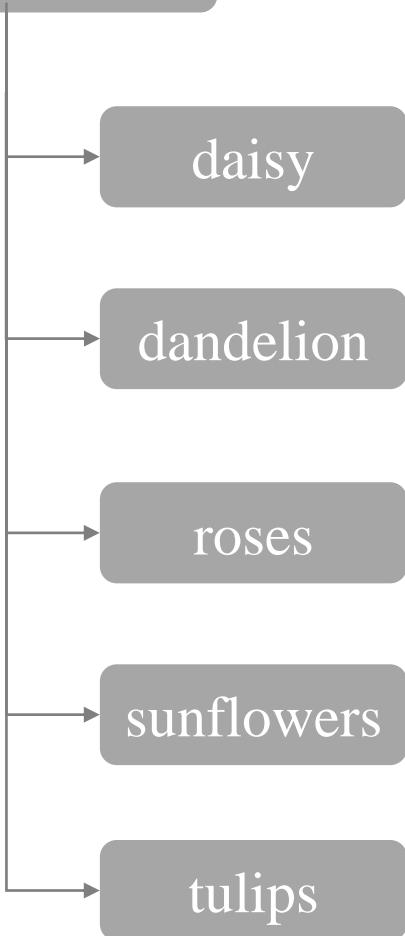
❖ Comparison



Case Study 5

❖ In keras

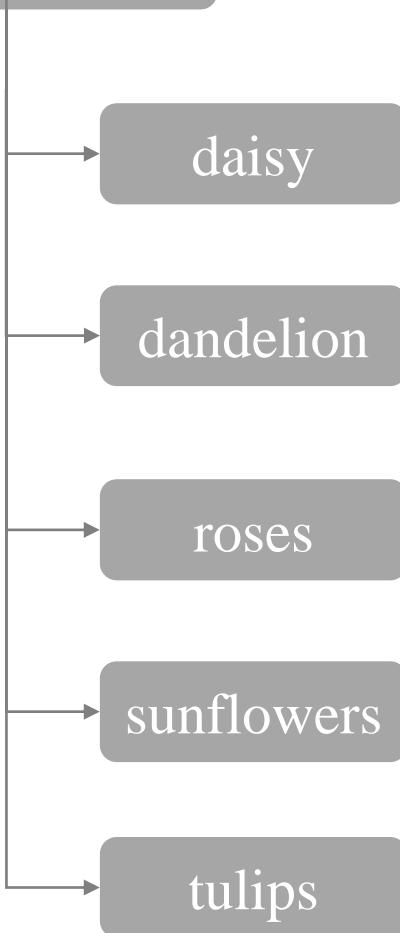
flower_photos



Case Study 5

❖ In keras

flower_photos



```
1 PATH = 'flower_photos/'  
2  
3 train_dir = os.path.join(PATH, 'train')  
4 validation_dir = os.path.join(PATH, 'validation')  
5  
6 BATCH_SIZE = 256  
7 IMG_SIZE = (160, 160)  
8 BUFFER_SIZE = BATCH_SIZE*5  
9  
10 train_dataset = tf.keras.preprocessing.image_dataset_from_directory(  
11     PATH,  
12     validation_split=0.2,  
13     subset="training",  
14     seed=123,  
15     image_size=IMG_SIZE,  
16     batch_size=BATCH_SIZE)
```

Found 3670 files belonging to 5 classes.
Using 2936 files for training.

```
1 validation_dataset = tf.keras.preprocessing.image_dataset_from_directory(  
2     PATH,  
3     validation_split=0.2,  
4     subset="validation",  
5     seed=123,  
6     image_size=IMG_SIZE,  
7     batch_size=BATCH_SIZE)
```

Found 3670 files belonging to 5 classes.
Using 734 files for validation.

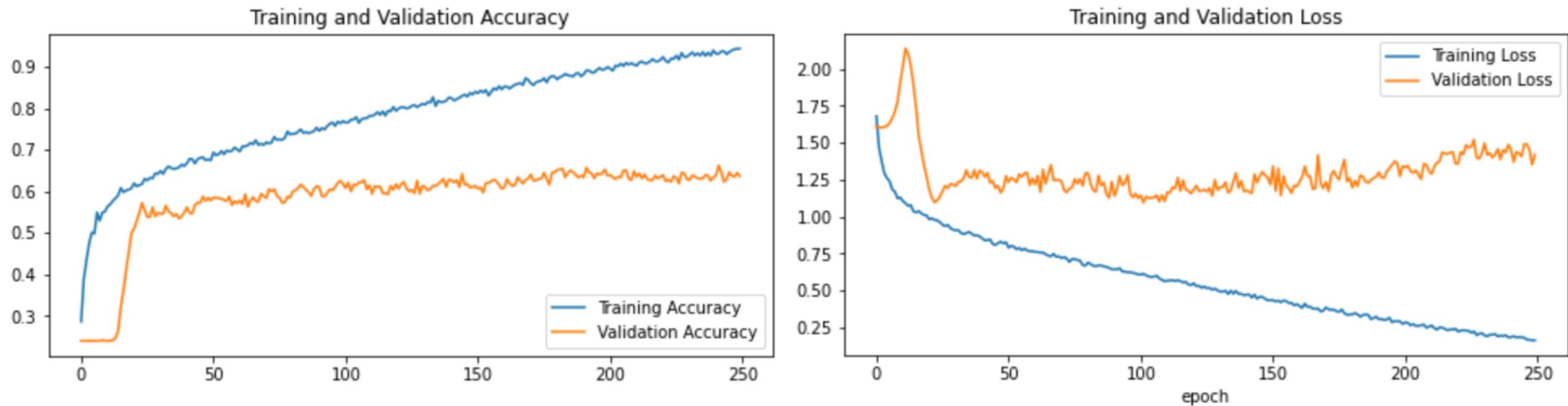
Case Study 5

❖ Model

```
1  ### from scratch
2
3  # Create the base model
4  base_model = tf.keras.applications.InceptionV3(input_shape=(160,160,3),
5                                              include_top=False,
6                                              weights=None)
7  # process data
8  data_augmentation = tf.keras.Sequential([
9      tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
10     tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
11     tf.keras.layers.experimental.preprocessing.Rescaling(1./127.5, offset=-1)
12 ])
13
14 # flattening
15 flatten = tf.keras.layers.Flatten()
16
17 # final layer
18 prediction_layer = tf.keras.layers.Dense(5)
19
20 # construct a new network
21 inputs = tf.keras.Input(shape=(160, 160, 3))
22 x = data_augmentation(inputs)
23 x = base_model(x)
24 x = flatten(x)
25 outputs = prediction_layer(x)
26 model = tf.keras.Model(inputs, outputs)
27
28 # train
29 model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
30                 optimizer = tf.keras.optimizers.Adam(lr=0.00001),
31                 metrics=['accuracy'])
32 history_fine = model.fit(train_dataset, epochs=250,
33                           validation_data=validation_dataset)
```

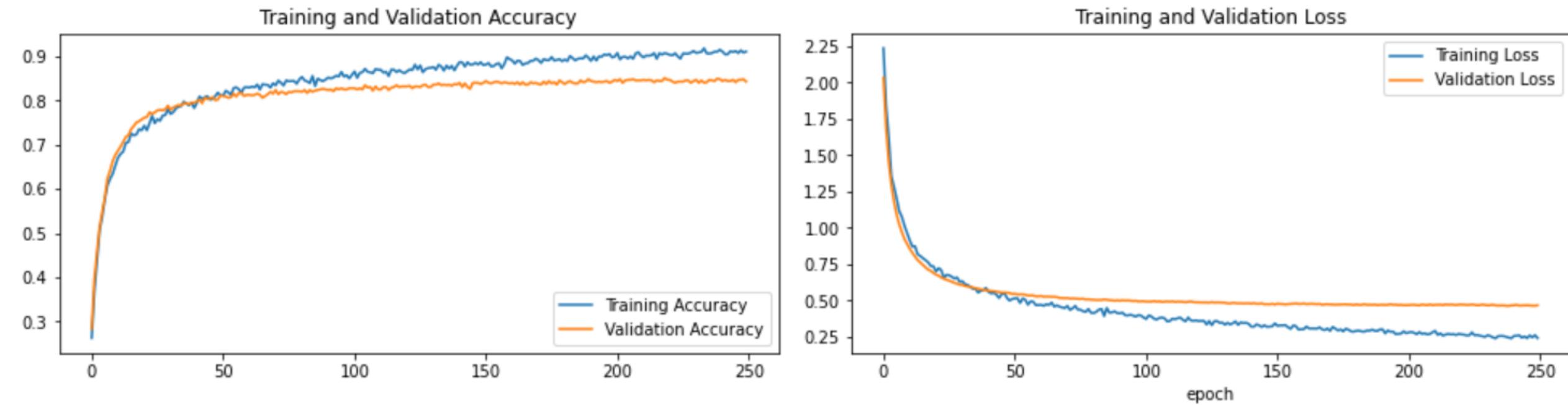
Case Study 5

❖ Train from scratch



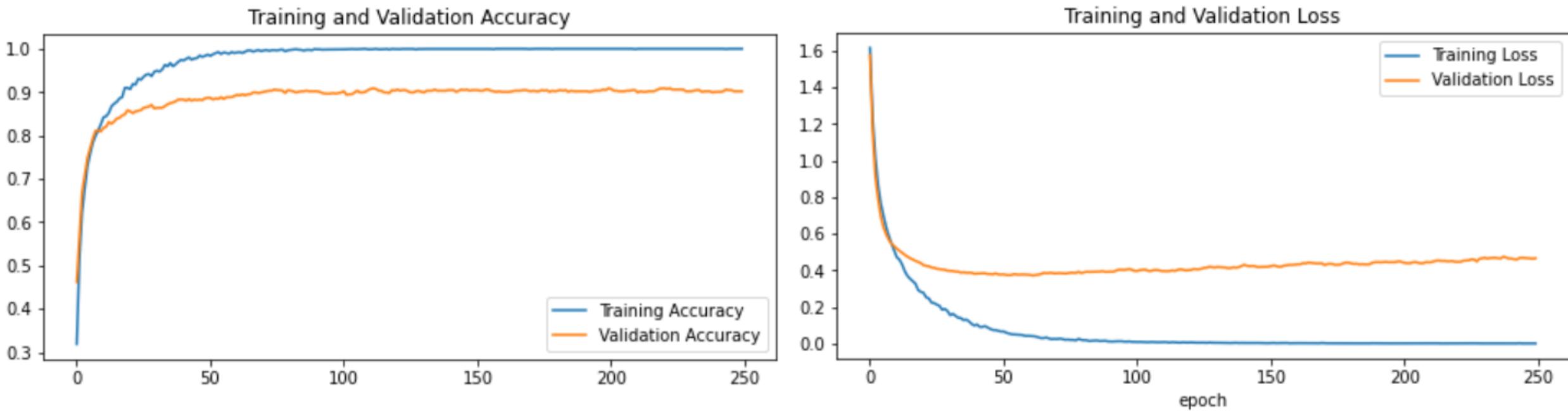
Case Study 5

❖ Transfer learning



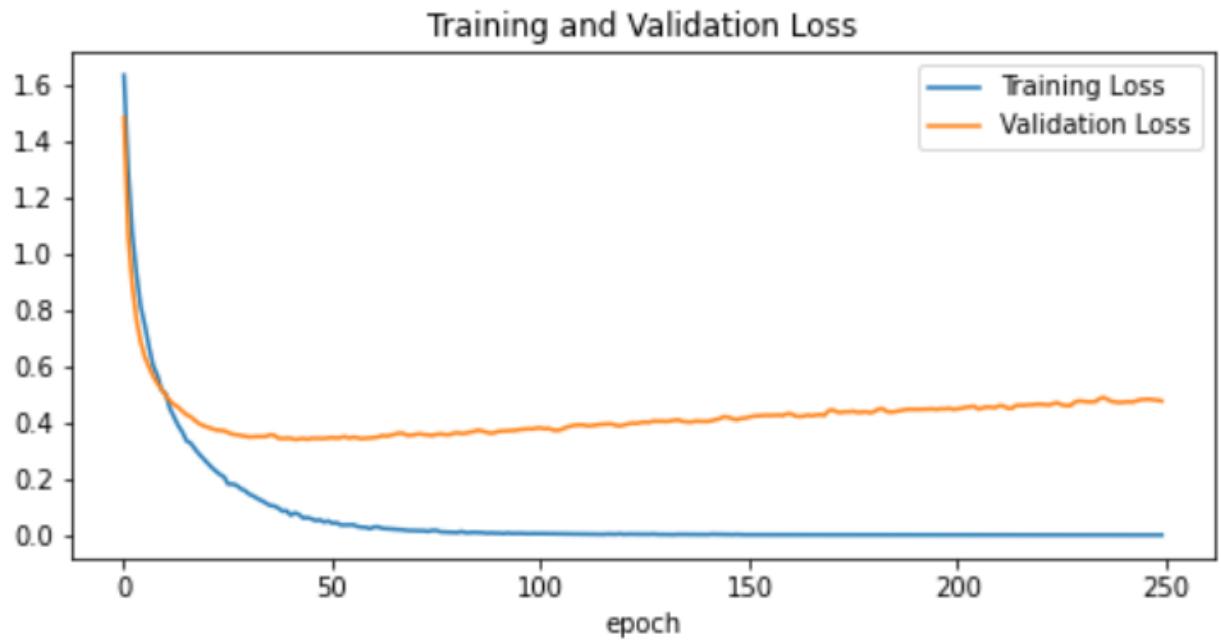
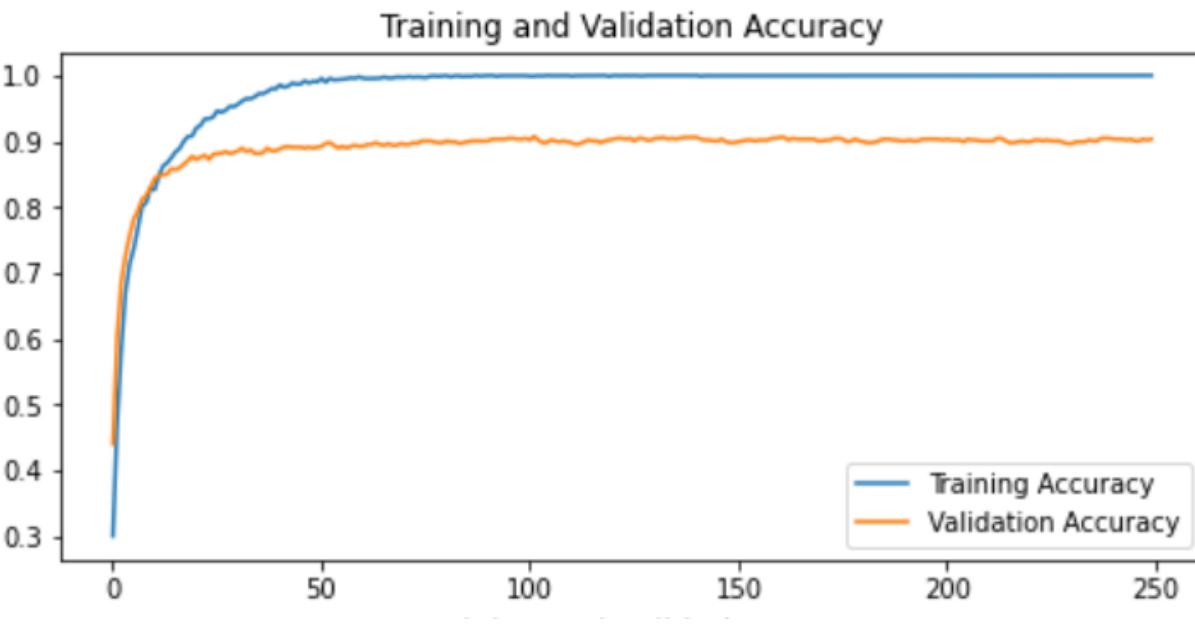
Case Study 5

❖ Fine tuning



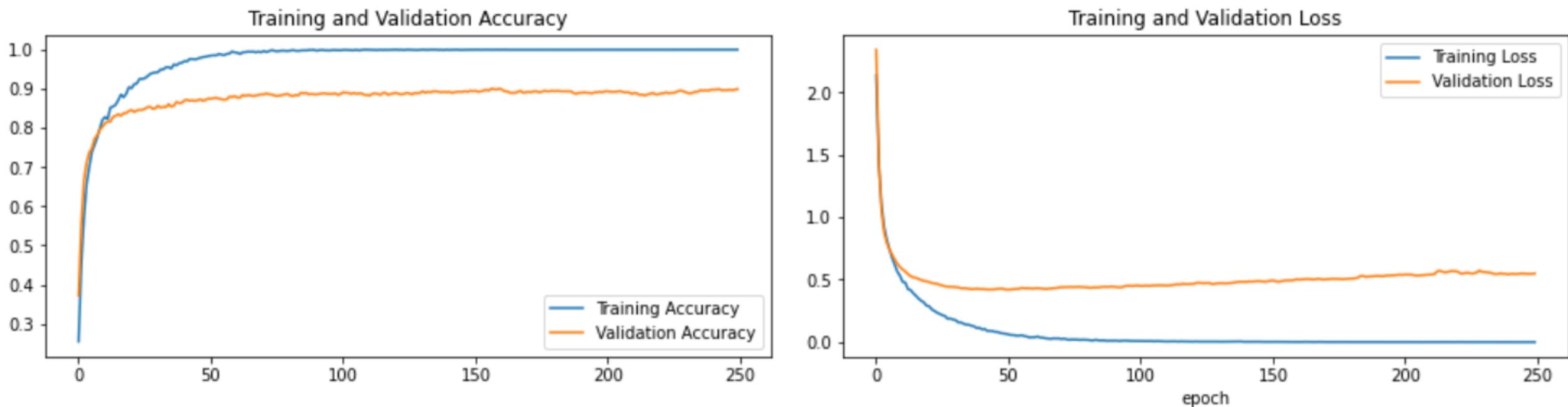
Case Study 5

- ❖ Use pretrained weights as init weights



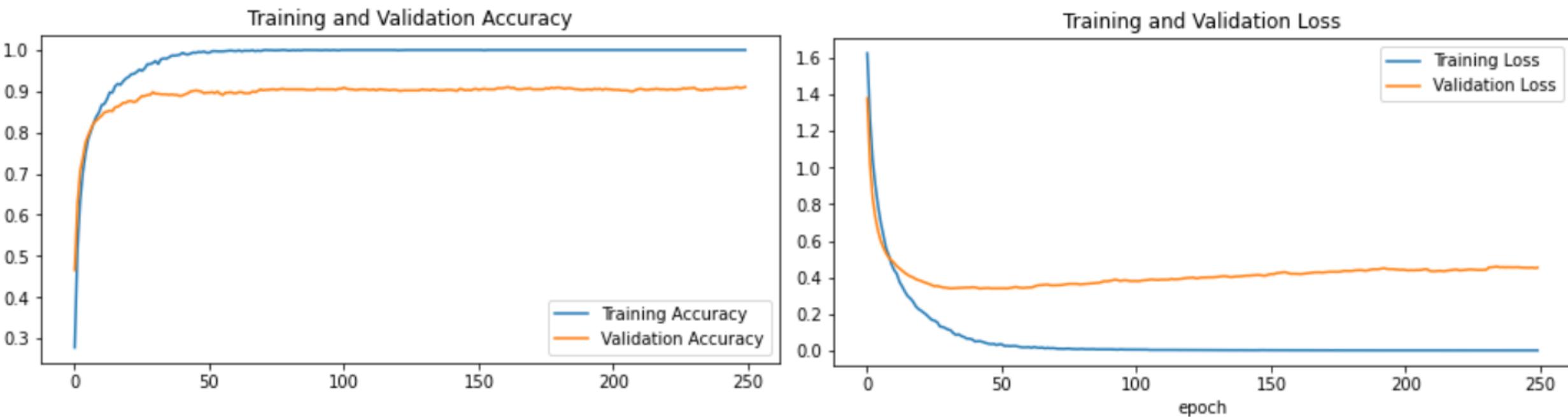
Case Study 5

- ❖ Use pretrained weights as init weights
- ❖ Global max pooling



Case Study 5

- ❖ Use pretrained weights as init weights
 - ❖ Global average pooling



Thank you!