AI VIETNAM
AI Insight Course

Exploitation of Pretrained Models

Quang-Vinh Dinh
Ph.D. in Computer Science

Year 2020

Outline

- Global Pooling
- Data Processing
- Data Processing Layer
- Network Manipulation
- Reuse a Pre-trained Model
- Case Study 1

Global Pooling

Max pooling: Features are preserved

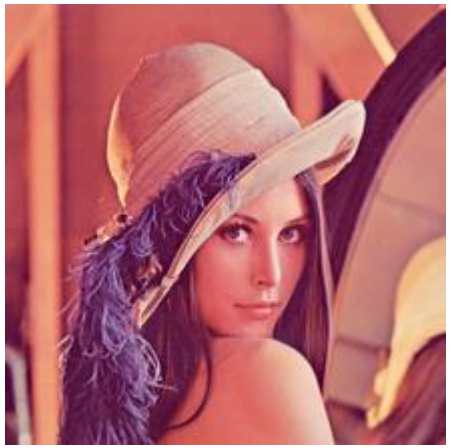
v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

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) →

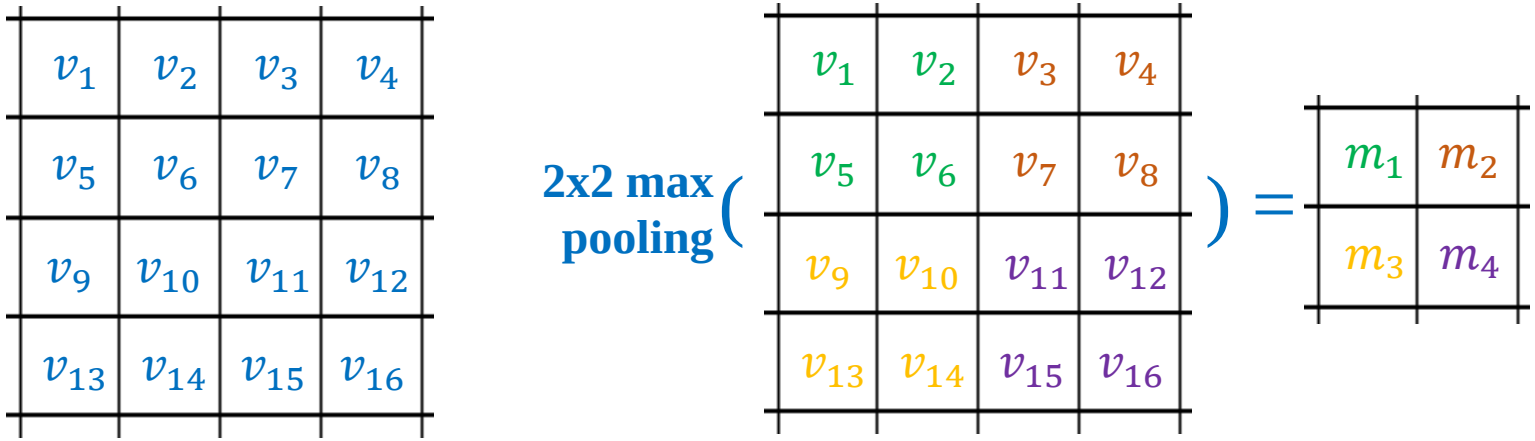
Feature map (110x110)

max pooling (2x2) →

Feature map (55x55)

Global Pooling

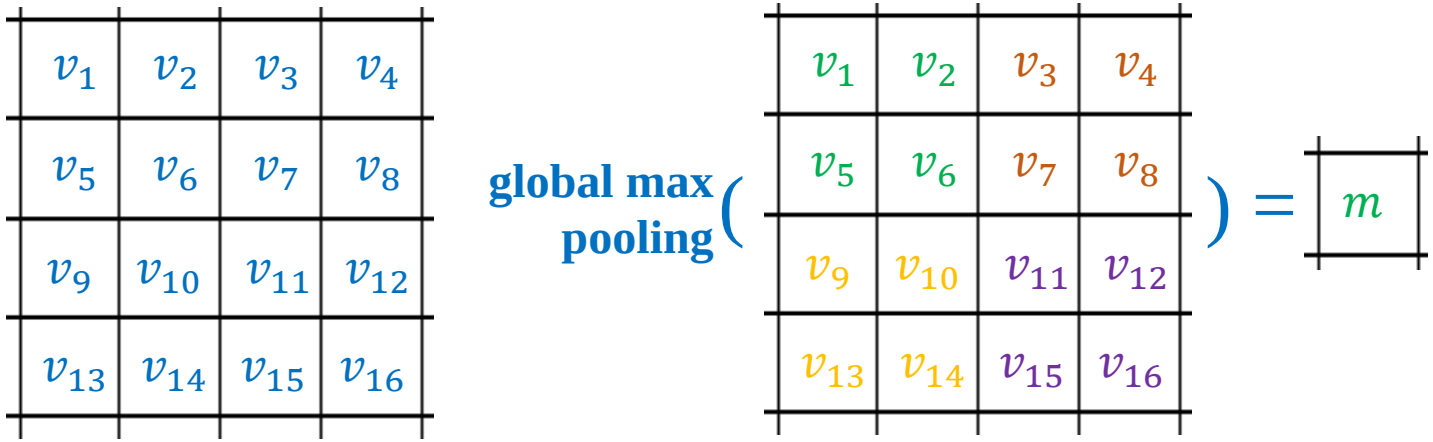
Max pooling



Data

$$m_1 = \max(v_1, v_2, v_5, v_6)$$
$$m_2 = \max(v_3, v_4, v_7, v_8)$$
$$m_3 = \max(v_9, v_{10}, v_{13}, v_{14})$$
$$m_4 = \max(v_{11}, v_{12}, v_{15}, v_{16})$$

Global max pooling



Data

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

keras.layers.GlobalMaxPool2D()

Global Pooling

Average pooling: Features are preserved

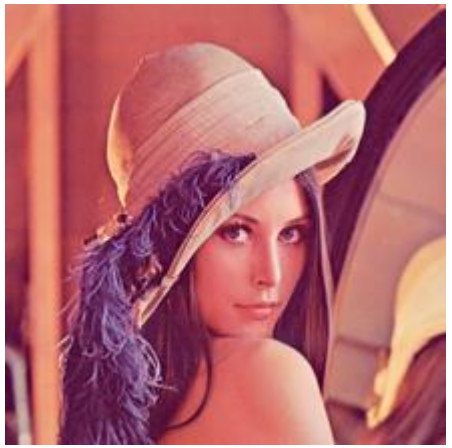
v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

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) →

Feature map (110x110)

Average Pooling (2x2) →

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

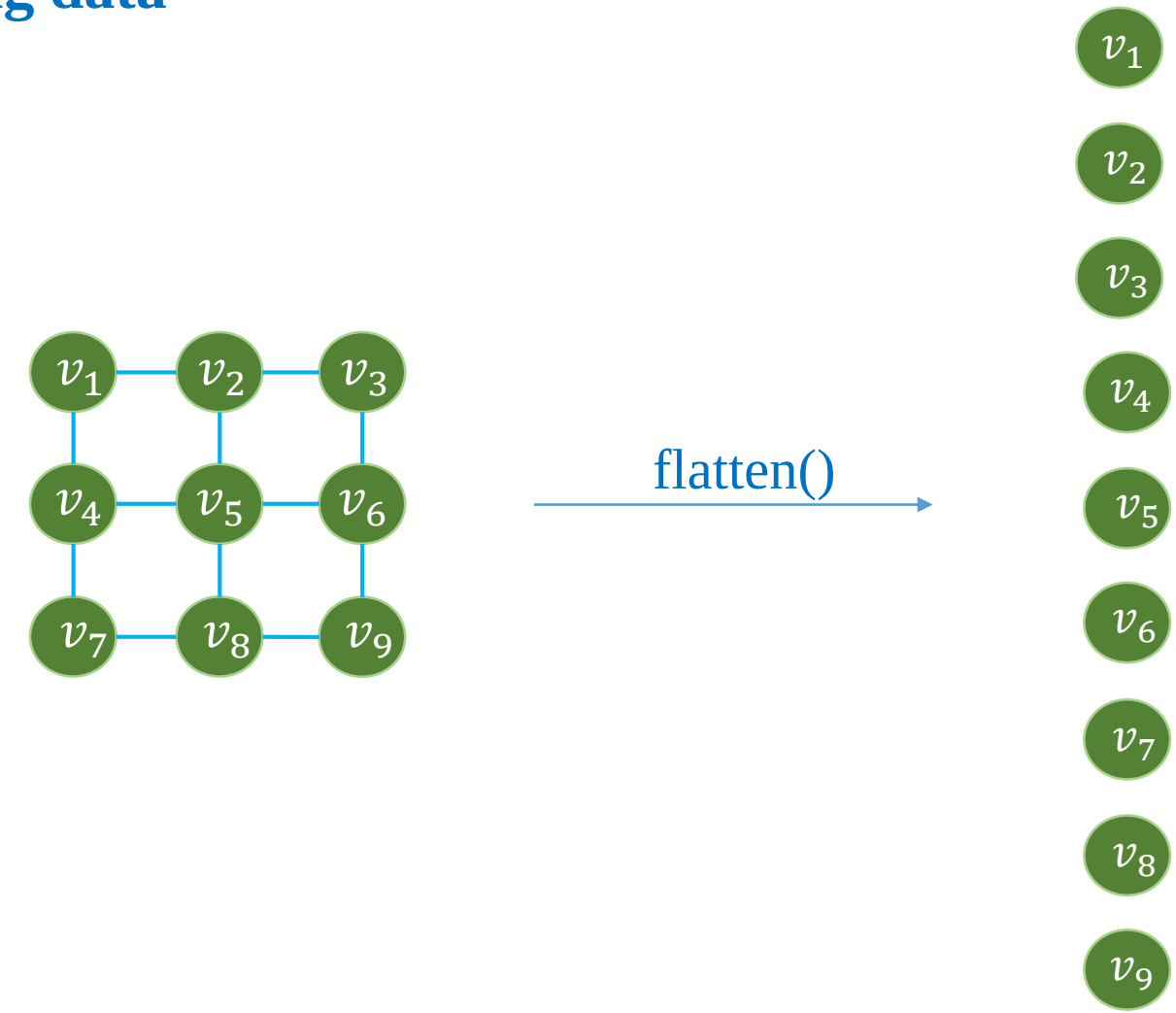
global average pooling (| v_1 | v_2 | v_3 | v_4 | | v_5 | v_6 | v_7 | v_8 | | v_9 | v_{10} | v_{11} | v_{12} | | v_{13} | v_{14} | v_{15} | v_{16} |) = | m |

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

keras.layers.GlobalAveragePooling2D()

Flattening

❖ Flattening data

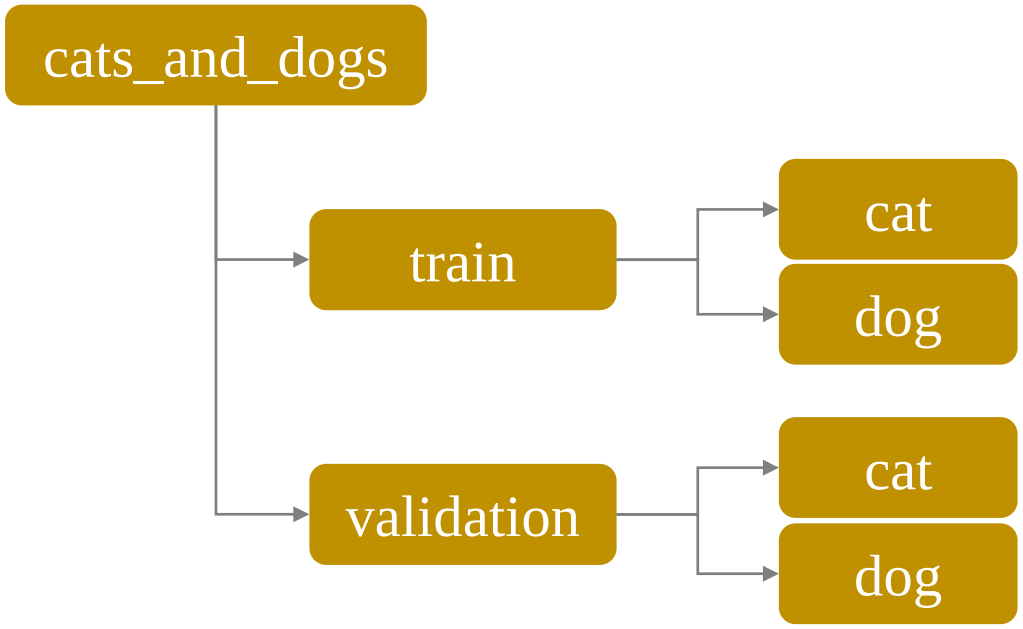


Outline

- Global Pooling
- Data Processing
- Data Processing Layer
- Network Manipulation
- Reuse a Pre-trained Model
- Case Study 1

Data Processing

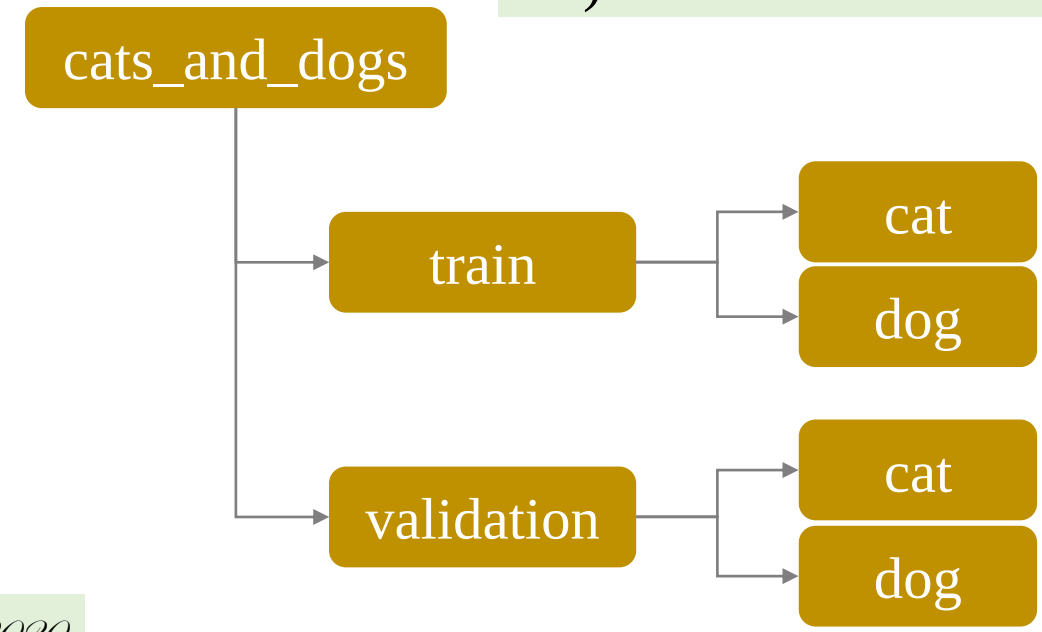
❖ Cat-Dog dataset



Data Processing

❖ In keras

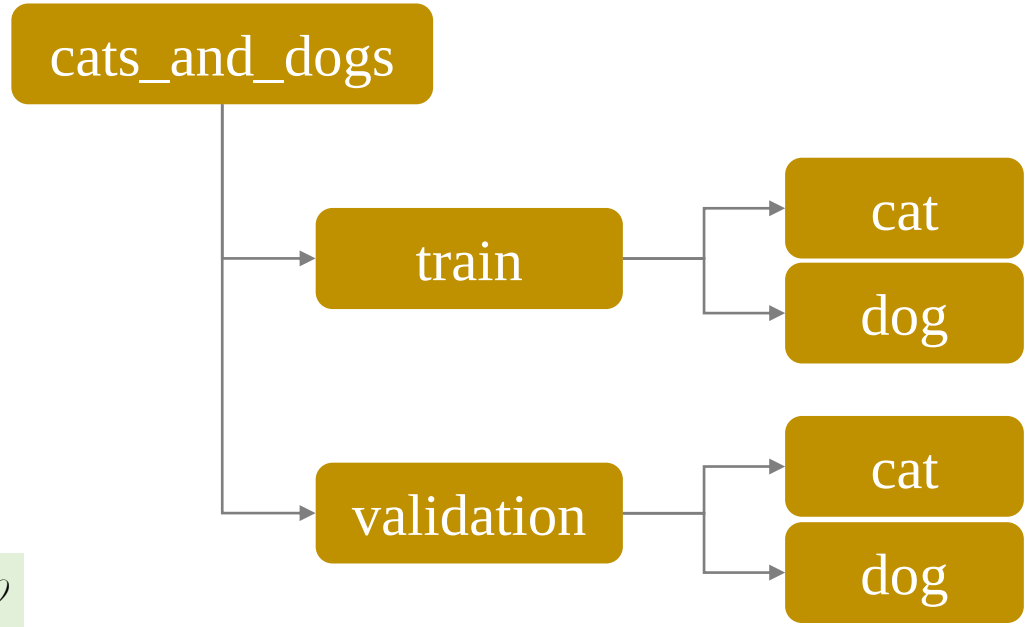
```
tf.keras.preprocessing.image_dataset_from_directory (
    directory,
    batch_size=32,
    image_size=(256, 256),
)
```



Data Processing

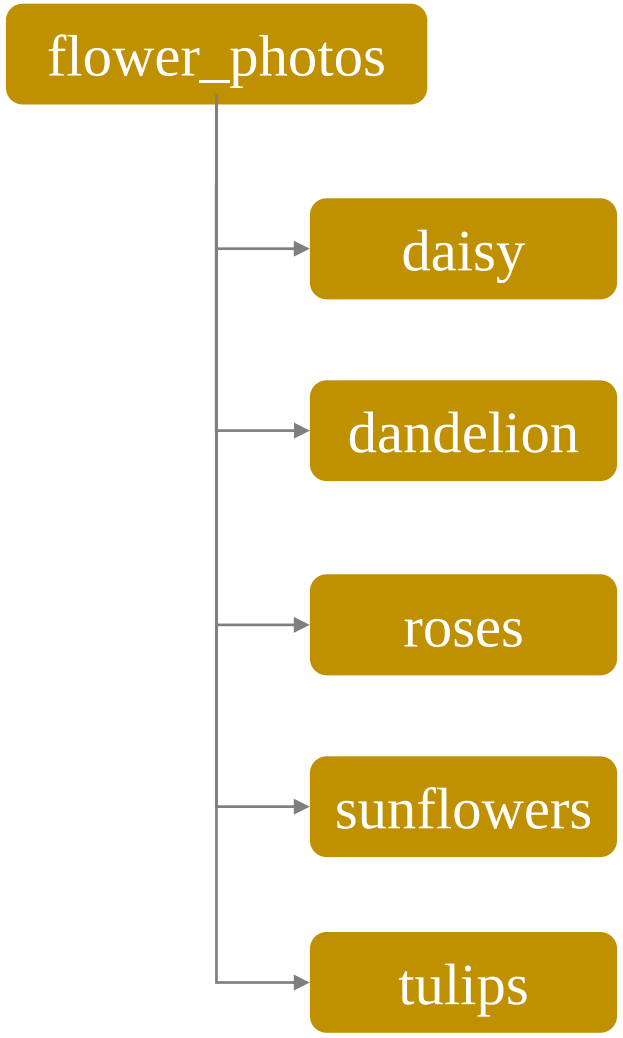
❖ In keras

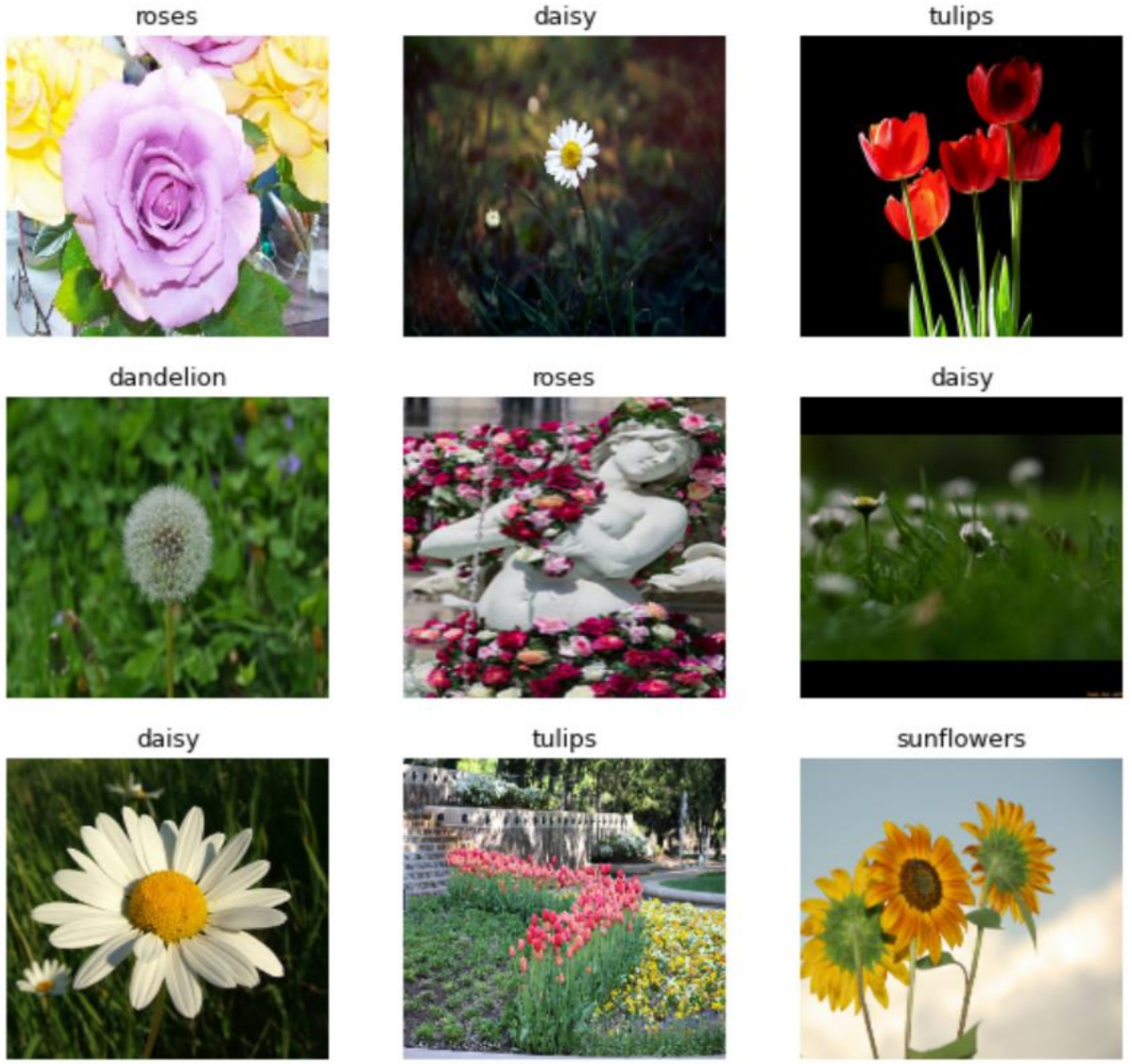
```python
# training data
train_dataset = image_dataset_from_directory('cats_and_dogs/train',
                                             shuffle=True,
                                             batch_size=256,
                                             image_size=(160, 160))

# validation data
validation_dataset = image_dataset_from_directory('cats_and_dogs/validation',
                                                  shuffle=True,
                                                  batch_size=256,
                                                  image_size=(160, 160))
```



Data Processing

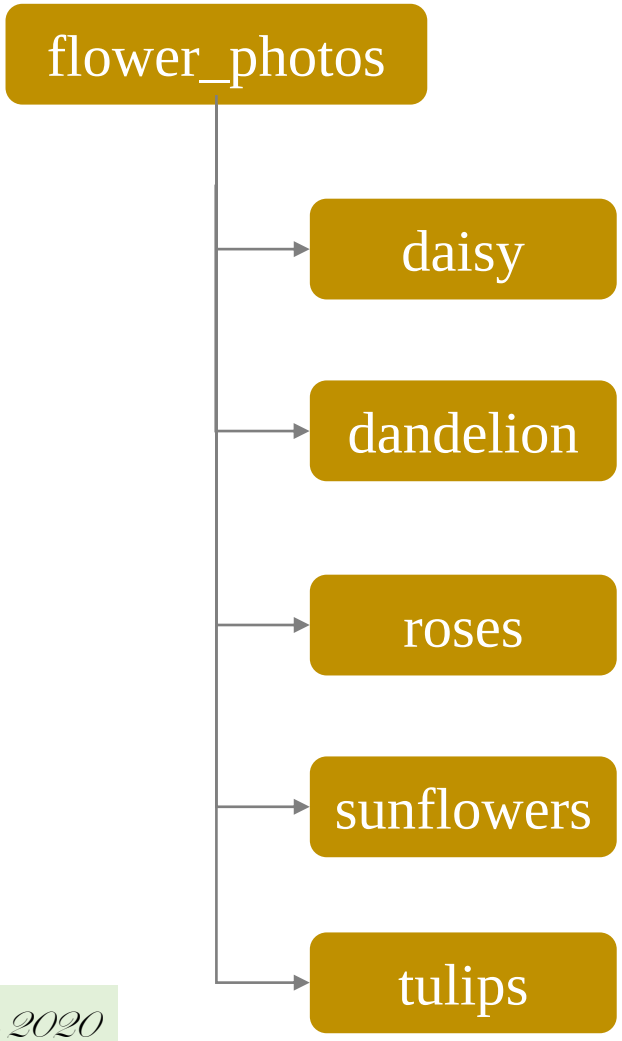
❖ In keras





Data Processing

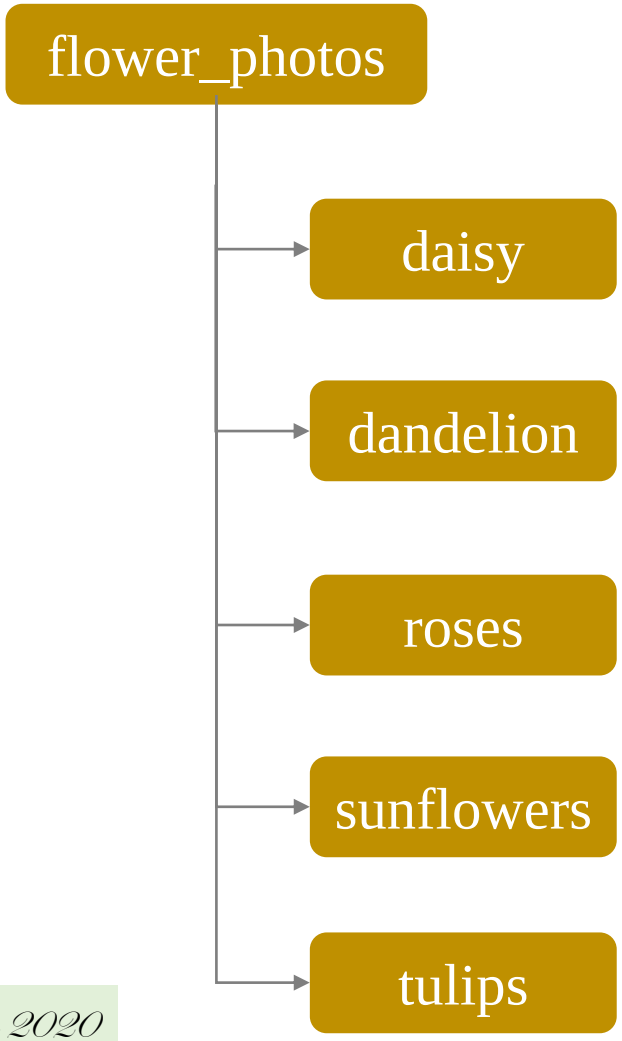
❖ In keras




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

Data Processing

❖ In keras




```python
# training data
train_dataset = tf.keras.preprocessing.image_dataset_from_directory(
                                                'flower_photos/',
                                                validation_split=0.2,
                                                subset="training",
                                                seed=123,
                                                image_size=(160, 160),
                                                batch_size=256)

# validation data
validation_dataset = tf.keras.preprocessing.image_dataset_from_directory(
                                                'flower_photos/',
                                                validation_split=0.2,
                                                subset="validation",
                                                seed=123,
                                                image_size=(160, 160),
                                                batch_size=256)
```

Outline

- Global Pooling
- Data Processing
- Data Processing Layer
- Network Manipulation
- Reuse a Pre-trained Model
- Case Study 1

Data Processing Layer

❖ In Keras

tf.keras.layers.experimental.preprocessing

preprocessing.CenterCrop	preprocessing.RandomRotation
preprocessing.Normalization	preprocessing.RandomTranslation
preprocessing.RandomFlip	preprocessing.Rescaling

Data Processing Layer

❖ In Keras

```
# data augmentation
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal')
])

# training
for images, _ in train_dataset:
    augmented_image = data_augmentation(images)
```

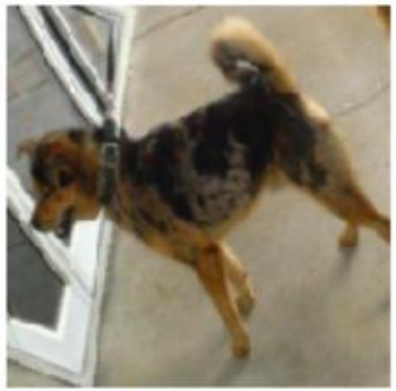




Data Processing Layer

❖ In Keras

```
# data augmentation
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
    tf.keras.layers.experimental.preprocessing.RandomRotation(0.2)
])

# training
for images, _ in train_dataset:
    augmented_image = data_augmentation(images)
```





AI VIETNAM
AI Insight Course

Data Processing Layer

```python
# data augmentation
data_augmentation = tf.keras.Sequential([
  tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
  tf.keras.layers.experimental.preprocessing.RandomRotation(0.2)
])

# training
for images, _ in train_dataset.take(1):
    # show images
    plt.figure(figsize=(18, 15))
    for i in range(6):
        # data augmentation
        augmented_image = data_augmentation(images[0:1]/255.)

        ax = plt.subplot(1, 6, i+1)
        plt.imshow(augmented_image[0])
        plt.axis('off')
```

AI VIETNAM
AI Insight Course

Data Processing Layer

```python
# data augmentation
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./255., offset=0.)
])

# training
for images, _ in train_dataset.take(1):
    # show images
    plt.figure(figsize=(18, 15))
    for i in range(6):
        # data augmentation
        augmented_image = data_augmentation(images[0:1])

        ax = plt.subplot(1, 6, i+1)
        plt.imshow(augmented_image[0])
        plt.axis('off')
```

Outline

- Global Pooling
- Data Processing
- Data Processing Layer
- Network Manipulation
- Reuse a Pre-trained Model
- Case Study 1

Network Manipulation

❖ Network 1

```python
# LeNet-like
import tensorflow as tf

# model architecture
model = tf.keras.Sequential()
# input shape (28,28,1)
model.add(tf.keras.Input(shape=(28, 28, 1)))

# convolution 1
model.add(tf.keras.layers.Conv2D(6, (3,3), padding='same', activation='relu'))
# max pooling 1
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))

# convolution 2
model.add(tf.keras.layers.Conv2D(filters=16, kernel_size=5, activation='relu'))
# max pooling 2
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))

# Flatten
model.add(tf.keras.layers.Flatten())

# fully connected
model.add(tf.keras.layers.Dense(120, activation='relu'))
model.add(tf.keras.layers.Dense(84, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='softmax'))

# model summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 6)         60
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 16)          0
_________________________________________________________________
flatten (Flatten)            (None, 400)               0
_________________________________________________________________
dense (Dense)                (None, 120)               48120
_________________________________________________________________
dense_1 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_2 (Dense)              (None, 10)                850
=================================================================
Total params: 61,610
Trainable params: 61,610
Non-trainable params: 0
_________________________________________________________________
```

Network Manipulation

❖ Network 1

```python
# LeNet-like
import tensorflow as tf

# model architecture
model = tf.keras.Sequential()
# input shape (28,28,1)
model.add(tf.keras.Input(shape=(28, 28, 1)))

# convolution 1
model.add(tf.keras.layers.Conv2D(6, (3,3), padding='same', activation='relu'))
# max pooling 1
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))

# convolution 2
model.add(tf.keras.layers.Conv2D(filters=16, kernel_size=5, activation='relu'))
# max pooling 2
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))

# Flatten
model.add(tf.keras.layers.Flatten())

# fully connected
model.add(tf.keras.layers.Dense(120, activation='relu'))
model.add(tf.keras.layers.Dense(84, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='softmax'))

# model summary
model.summary()
```

```python
print(type(model.layers))
print(len(model.layers))
```

```
<class 'list'>
8
```

```python
for layer in model.layers:
    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

```python
# LeNet-like
import tensorflow as tf

# model architecture
model = tf.keras.Sequential()
# input shape (28,28,1)
model.add(tf.keras.Input(shape=(28, 28, 1)))

# convolution 1
model.add(tf.keras.layers.Conv2D(6, (3,3), padding='same', activation='relu'))
# max pooling 1
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))

# convolution 2
model.add(tf.keras.layers.Conv2D(filters=16, kernel_size=5, activation='relu'))
# max pooling 2
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))

# Flatten
model.add(tf.keras.layers.Flatten())

# fully connected
model.add(tf.keras.layers.Dense(120, activation='relu'))
model.add(tf.keras.layers.Dense(84, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='softmax'))

# model summary
model.summary()
```

```python
model.trainable = False
for layer in model.layers:
    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

```python
# LeNet-like
import tensorflow as tf

# model architecture
model = tf.keras.Sequential()
# input shape (28,28,1)
model.add(tf.keras.Input(shape=(28, 28, 1)))

# convolution 1
model.add(tf.keras.layers.Conv2D(6, (3,3), padding='same', activation='relu'))
# max pooling 1
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))

# convolution 2
model.add(tf.keras.layers.Conv2D(filters=16, kernel_size=5, activation='relu'))
# max pooling 2
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))

# Flatten
model.add(tf.keras.layers.Flatten())

# fully connected
model.add(tf.keras.layers.Dense(120, activation='relu'))
model.add(tf.keras.layers.Dense(84, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='softmax'))

# model summary
model.summary()
```

```python
# number of layers
print('Number of layers is ', len(model.layers), '\n')

# freeze some layers
model.layers[0].trainable = False
model.layers[5].trainable = False

for layer in model.layers:
    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

```python
# LeNet-like
import tensorflow as tf

# model architecture
model = tf.keras.Sequential()
# input shape (28,28,1)
model.add(tf.keras.Input(shape=(28, 28, 1)))

# convolution 1
model.add(tf.keras.layers.Conv2D(6, (3,3), padding='same', activation='relu'))
# max pooling 1
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))

# convolution 2
model.add(tf.keras.layers.Conv2D(filters=16, kernel_size=5, activation='relu'))
# max pooling 2
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))

# Flatten
model.add(tf.keras.layers.Flatten())

# fully connected
model.add(tf.keras.layers.Dense(120, activation='relu'))
model.add(tf.keras.layers.Dense(84, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='softmax'))

# model summary
model.summary()
```

```python
# number of layers
print('Number of layers is ', len(model.layers), '\n')

# freeze some last layers
for layer in model.layers[5:]:
    layer.trainable = False

for layer in model.layers:
    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

```
print('Number of tensors is',
      len(model.trainable_variables))

for v in model.trainable_variables:
    print(v.shape)
```

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

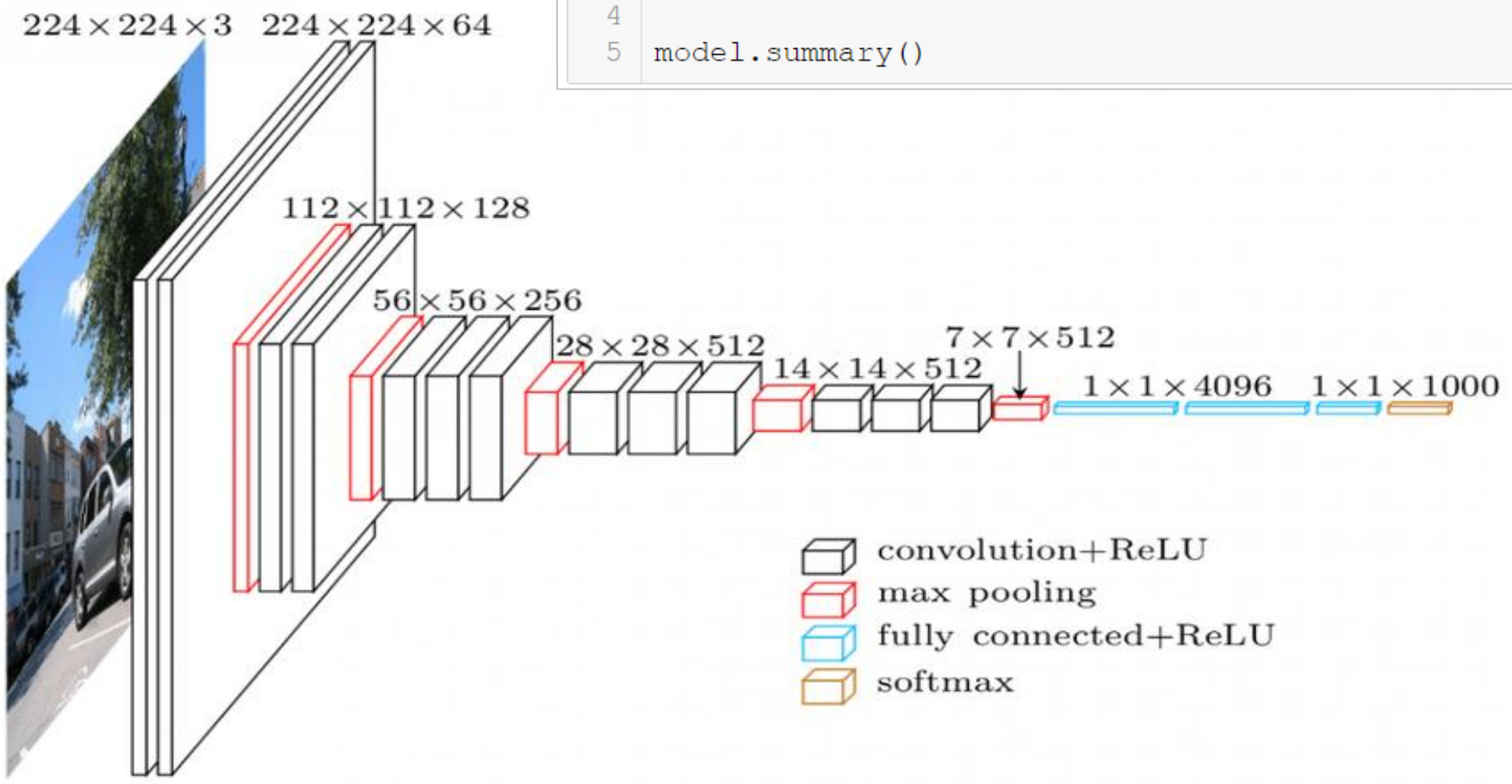
```
# A network
import tensorflow as tf

# model architecture
model = tf.keras.Sequential()
model.add(tf.keras.Input(shape=(5, 5, 1)))
model.add(tf.keras.layers.Conv2D(1, (3,3), activation='relu'))
model.add(tf.keras.layers.Conv2D(2, (3,3), activation='relu'))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(2, activation='softmax'))

# model summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 3, 3, 1)           10
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 1, 1, 2)           20
_________________________________________________________________
flatten (Flatten)            (None, 2)                 0
_________________________________________________________________
dense (Dense)                (None, 2)                 6
=================================================================
Total params: 36
Trainable params: 36
Non-trainable params: 0
_________________________________________________________________
```

Network Manipulation

❖ VGG16 Model

```
import tensorflow as tf
model = tf.keras.applications.VGG16(input_shape=(244,244,3),
                                    include_top=True,
                                    weights=None)
model.summary()
```



Network Manipulation

❖ VGG16 Model

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

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
_________________________________________________________________
flatten (Flatten)            (None, 12800)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              52432896
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 88,025,896
Trainable params: 88,025,896
Non-trainable params: 0
_________________________________________________________________
```

Network Manipulation

❖ VGG16 Model

```
model = tf.keras.applications.VGG16(input_shape=(160,160,3),
                                    include_top=True,
                                    weights=None)

print(len(model.trainable_variables))
```

```
32
```

```
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
_________________________________________________________________
flatten (Flatten)            (None, 12800)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              52432896
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 88,025,896
Trainable params: 88,025,896
Non-trainable params: 0
_________________________________________________________________
```

Network Manipulation

❖ VGG16 Model

```
model = tf.keras.applications.VGG16(input_shape=(160,160,3),
                                    include_top=True,
                                    weights=None)


for layer in model.layers[:14]:
    layer.trainable = False

print(len(model.trainable_variables))
```

```
12
```

```
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
_________________________________________________________________
flatten (Flatten)            (None, 12800)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              52432896
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 88,025,896
Trainable params: 88,025,896
Non-trainable params: 0
_________________________________________________________________
```

Network Manipulation

❖ Network 2

```python
# top=False
model = tf.keras.applications.VGG16(input_shape=(160,160,3),
                                    include_top=False,
                                    weights=None)

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
_________________________________________________________________
```

Network Manipulation

❖ Network 2

```python
import tensorflow as tf

# get VGG16
model = tf.keras.applications.VGG16(input_shape=(160,160,3),
                                    include_top=False,
                                    weights=None)

# construct a new network
inputs = tf.keras.Input(shape=(160, 160, 3))
x = model(inputs)
x = tf.keras.layers.Flatten()(x)
x = tf.keras.layers.Dense(2)(x)
new_model = tf.keras.Model(inputs, x)

# summary
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

```python
import tensorflow as tf

# get VGG16
model = tf.keras.applications.VGG16(input_shape=(160,160,3),
                                    include_top=False,
                                    weights=None)

# construct a new network
inputs = tf.keras.Input(shape=(160, 160, 3))
x = model(inputs)
x = tf.keras.layers.Flatten()(x)
x = tf.keras.layers.Dense(2)(x)
new_model = tf.keras.Model(inputs, x)

# summary
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

```python
import tensorflow as tf

# get VGG16
model = tf.keras.applications.VGG16(input_shape=(160,160,3),
                                    include_top=False,
                                    weights=None)

# construct a new network
inputs = tf.keras.Input(shape=(160, 160, 3))
x = model(inputs)
x = tf.keras.layers.MaxPool2D(pool_size=(5, 5))(x)
x = tf.keras.layers.Reshape((-1,))(x)
x = tf.keras.layers.Dense(2)(x)
new_model = tf.keras.Model(inputs, x)

# summary
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

```python
import tensorflow as tf

# get VGG16
model = tf.keras.applications.VGG16(input_shape=(160,160,3),
                                    include_top=False,
                                    weights=None)

# construct a new network
inputs = tf.keras.Input(shape=(160, 160, 3))
x = model(inputs)
x = tf.keras.layers.GlobalMaxPool2D()(x)
x = tf.keras.layers.Dense(2)(x)
new_model = tf.keras.Model(inputs, x)

# summary
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

```python
import tensorflow as tf

# get VGG16
model = tf.keras.applications.VGG16(input_shape=(160,160,3),
                                    include_top=False,
                                    weights=None)

# construct a new network
inputs = tf.keras.Input(shape=(160, 160, 3))
x = model(inputs)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Dense(1)(x)
new_model = tf.keras.Model(inputs, x)

# summary
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 (Gl (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 513
=================================================================
Total params: 14,715,201
Trainable params: 14,715,201
Non-trainable params: 0
_________________________________________________________________
```

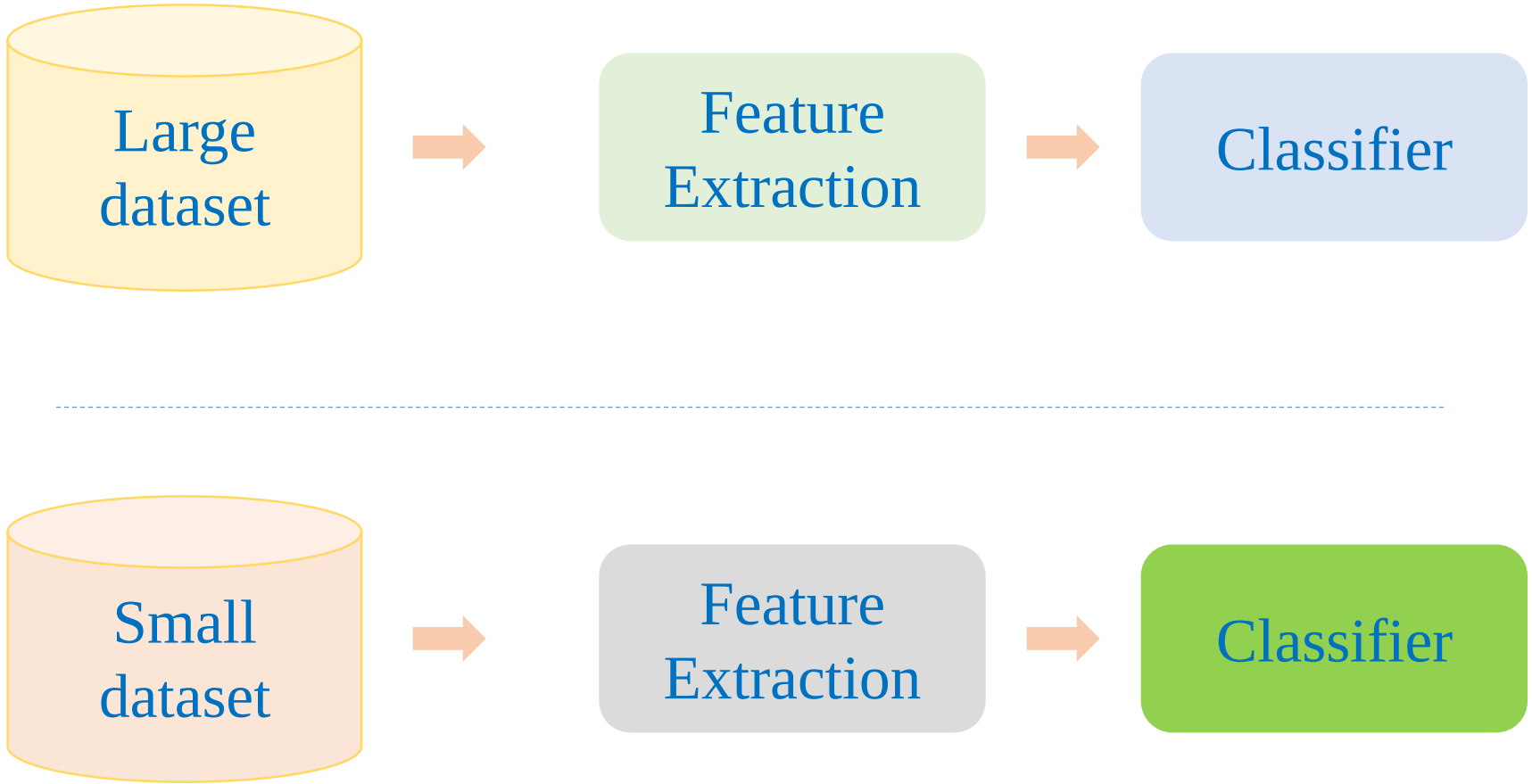
Outline

- Global Pooling
- Data Processing
- Data Processing Layer
- Network Manipulation
- Reuse a Pre-trained Model
- Case Study 1

Transfer Learning

❖ Transfer Learning

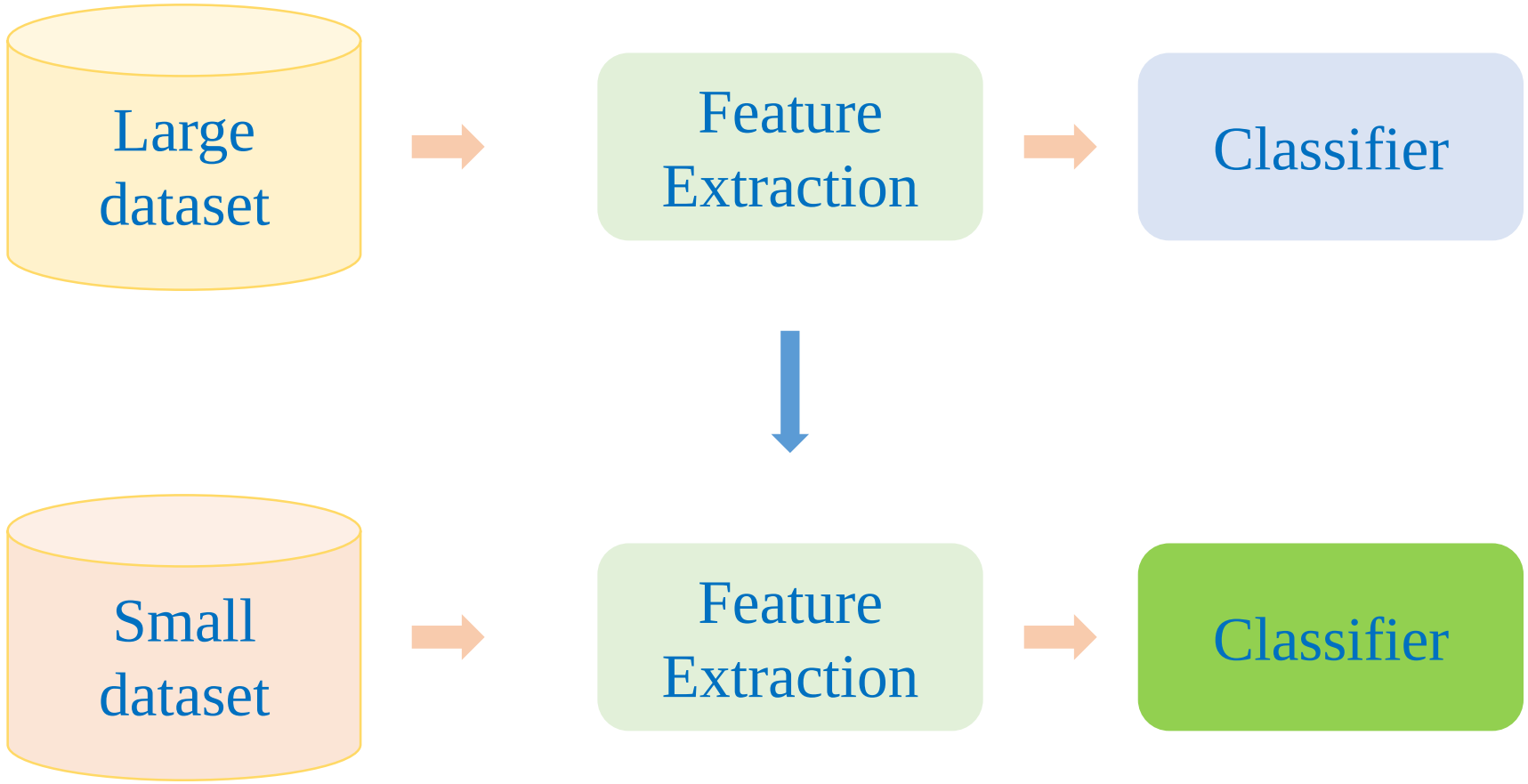
ImageNet dataset
(1.2 million images
of 1000 categories)



Transfer Learning

❖ Transfer Learning

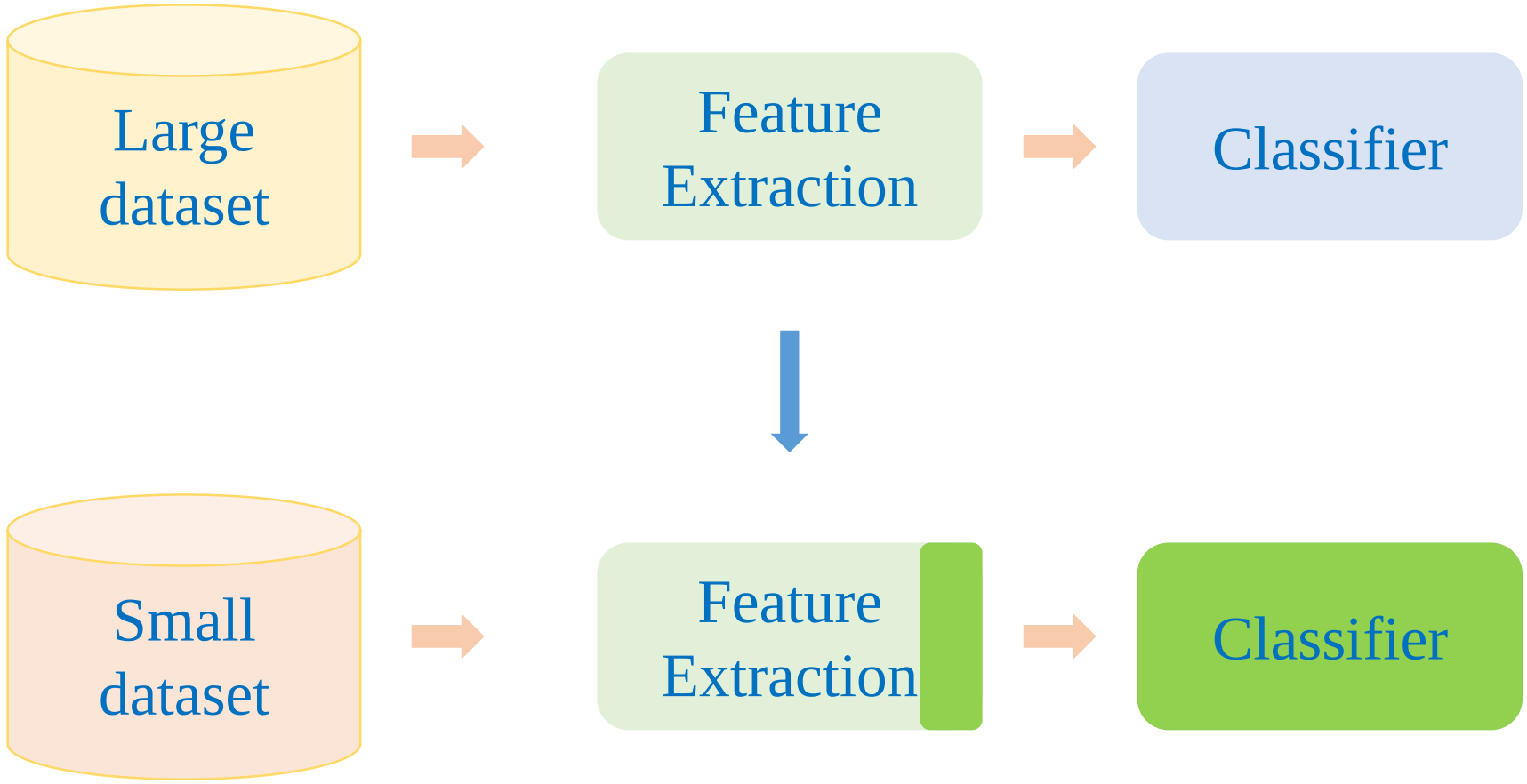
ImageNet dataset
(1.2 million images
of 1000 categories)



Transfer Learning

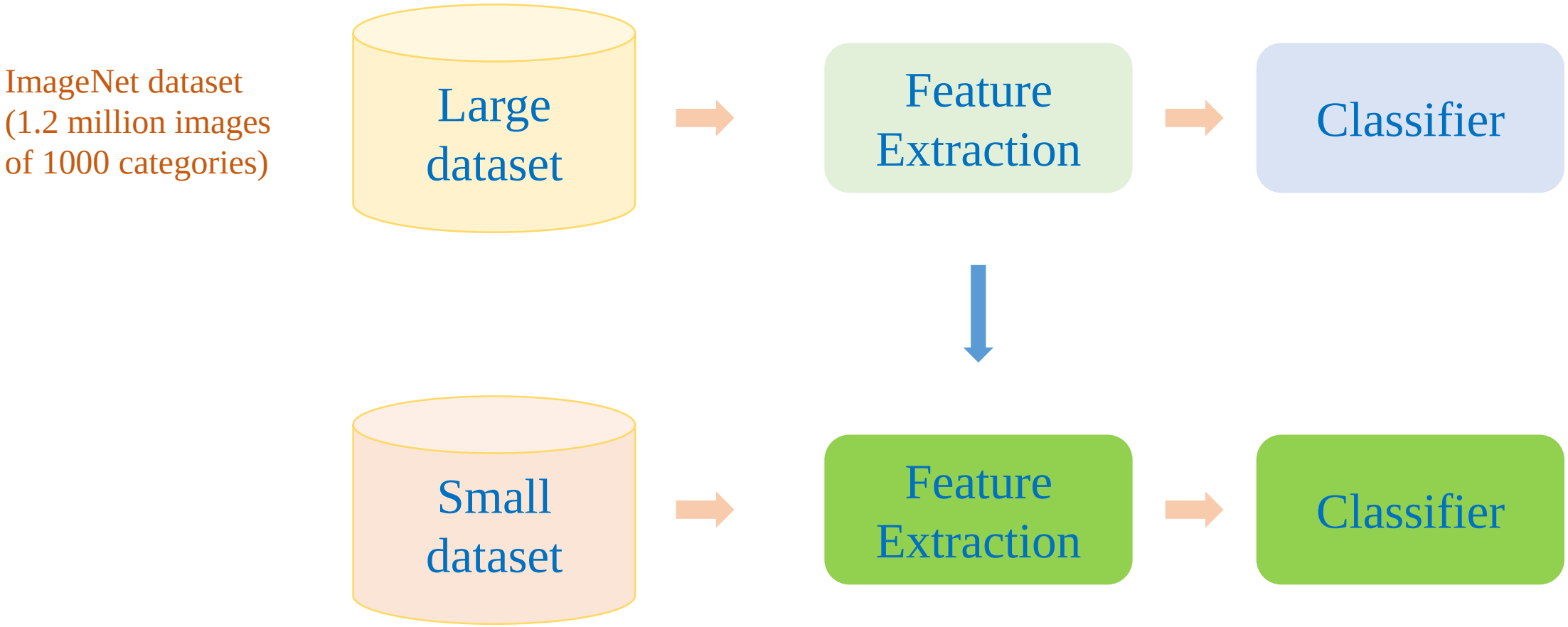
❖ Fine Tuning

ImageNet dataset
(1.2 million images
of 1000 categories)



Transfer Learning

❖ As Initialization



Outline

- Global Pooling
- Data Processing
- Data Processing Layer
- Network Manipulation
- Reuse a Pre-trained Model
- Case Study 1

Case Study 1

❖ Cat-Dog dataset

❖ Train from scratch

```
PATH = 'cats_and_dogs_small/'

train_dir = os.path.join(PATH, 'train')
validation_dir = os.path.join(PATH, 'validation')

BATCH_SIZE = 256
IMG_SIZE = (160, 160)
BUFFER_SIZE = BATCH_SIZE*5

train_dataset = image_dataset_from_directory(train_dir,
                                             shuffle=True,
                                             batch_size=BATCH_SIZE,
                                             image_size=IMG_SIZE)
```

```
Found 2000 files belonging to 2 classes.
```

```
validation_dataset = image_dataset_from_directory(validation_dir,
                                                  shuffle=True,
                                                  batch_size=BATCH_SIZE,
                                                  image_size=IMG_SIZE)
```

```
Found 1000 files belonging to 2 classes.
```

Case Study 1

- Cat-Dog dataset
  - Train from scratch

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

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

❖ Cat-Dog dataset

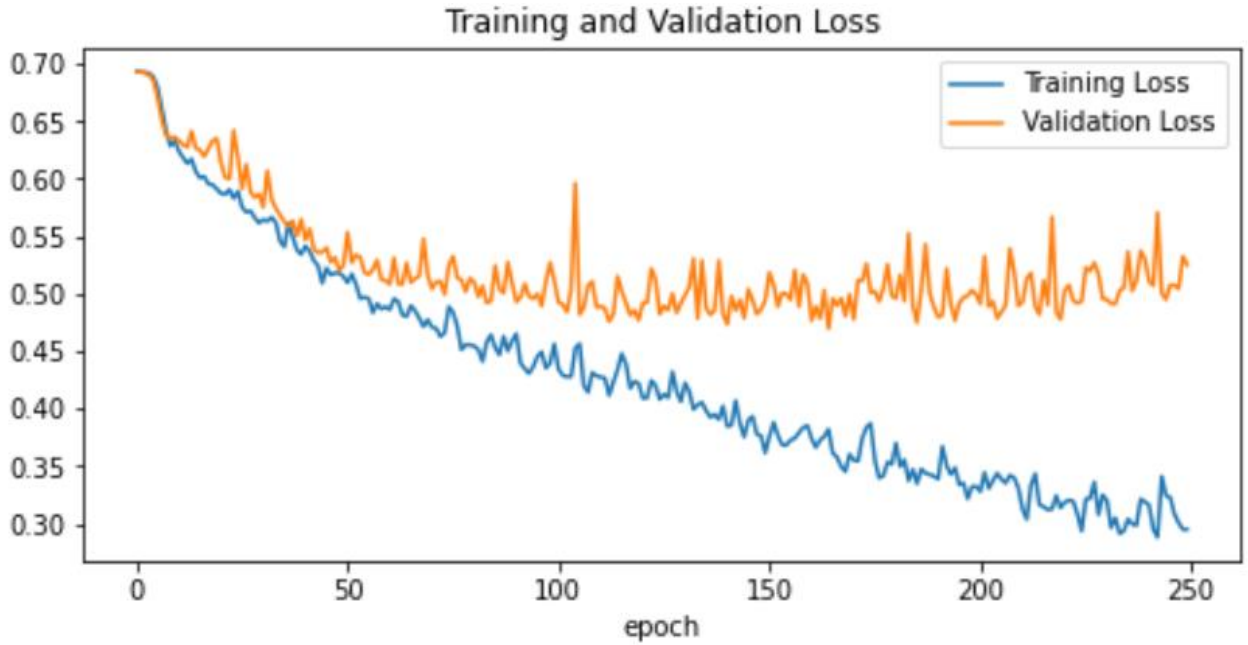
❖ Train from scratch

```
# 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(1)

# 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 1

- Cat-Dog dataset
  - Train from scratch

```
base_learning_rate =
model.compile(loss=tf.keras.losses.BinaryCrossentropy(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)
```



0.791

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

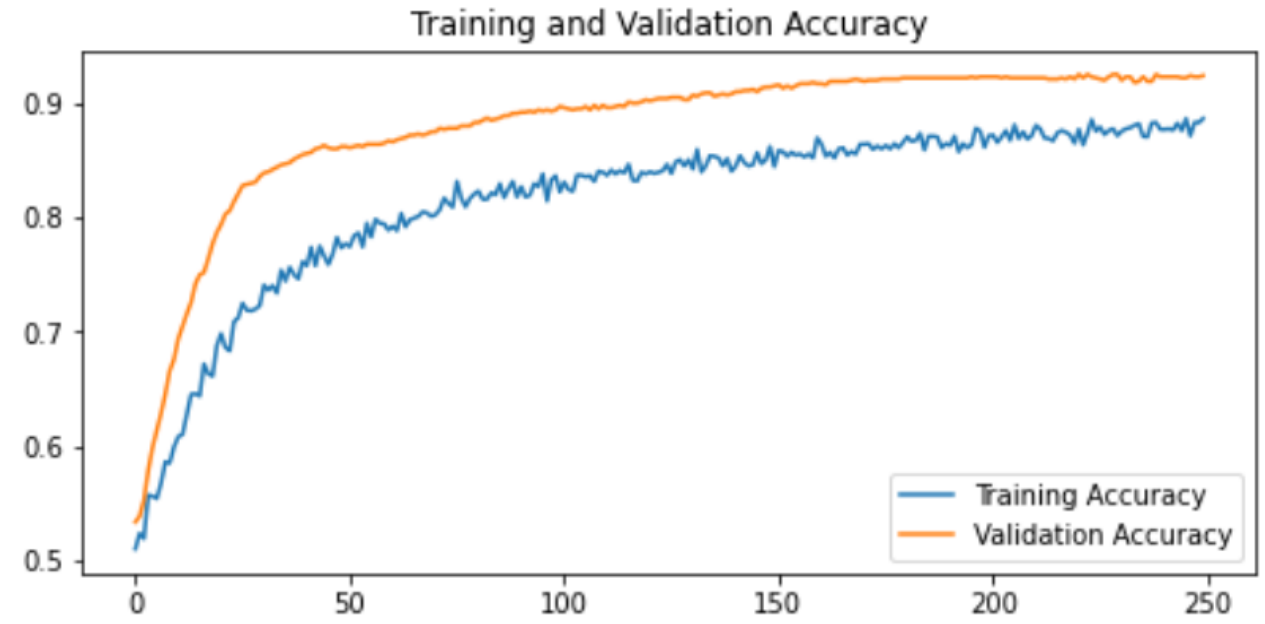
❖ Transfer learning

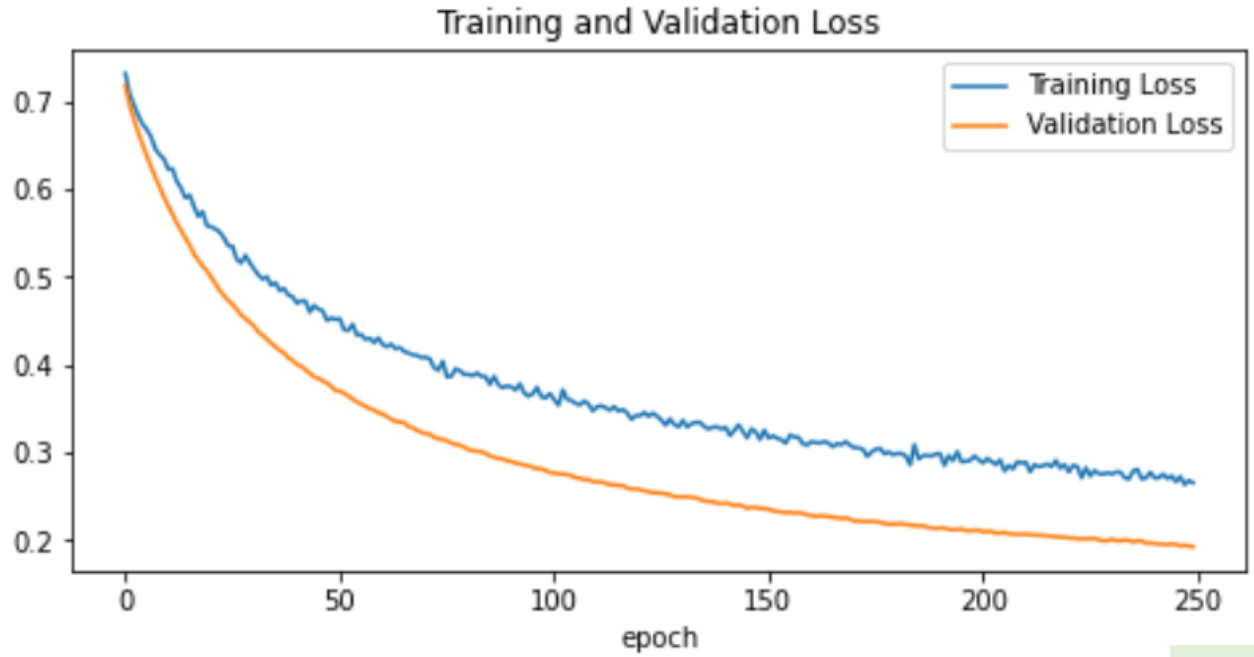
```python
# freeze the base model
base_model.trainable = False

# 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(1)

# 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 1

❖ **Transfer learning**

```
base_learning_rate =
model.compile(loss=tf.keras.losses.BinaryCrossentropy(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)
```





Year 2020 0.925

Case Study 1

❖ Fine tuning

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

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

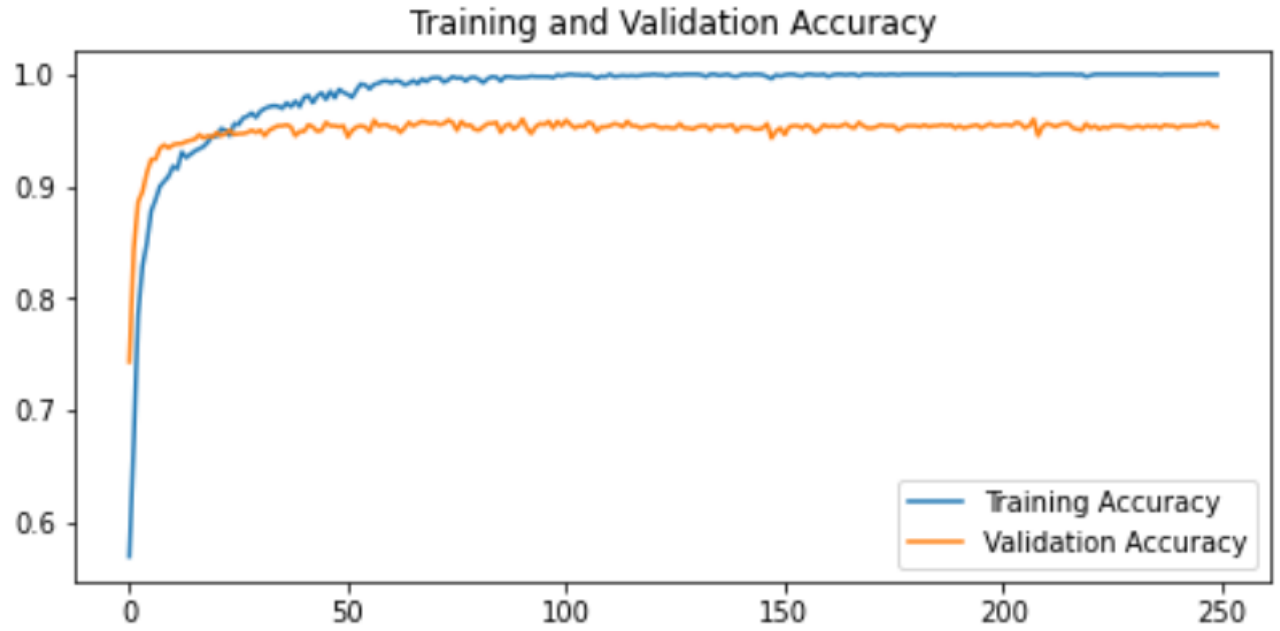
❖ Fine tuning

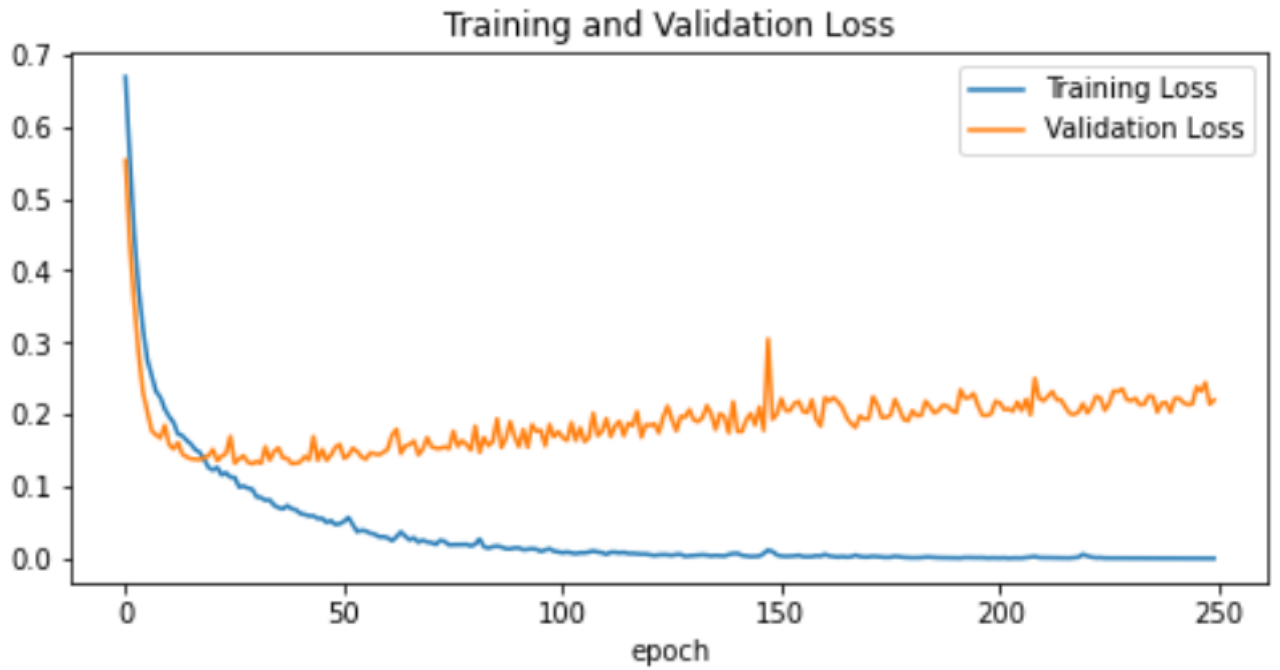
```
# Freeze some first the layers
fine_tune_at = 14
for layer in base_model.layers[:fine_tune_at]:
    layer.trainable =  False

# 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(1)

# 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 1

❖ **Fine tuning**

```
base_learning_rate =
model.compile(loss=tf.keras.losses.BinaryCrossentropy(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)
```





Year 2020 0.953

Case Study 1

❖ **Use pretrained weights as init weights**

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

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

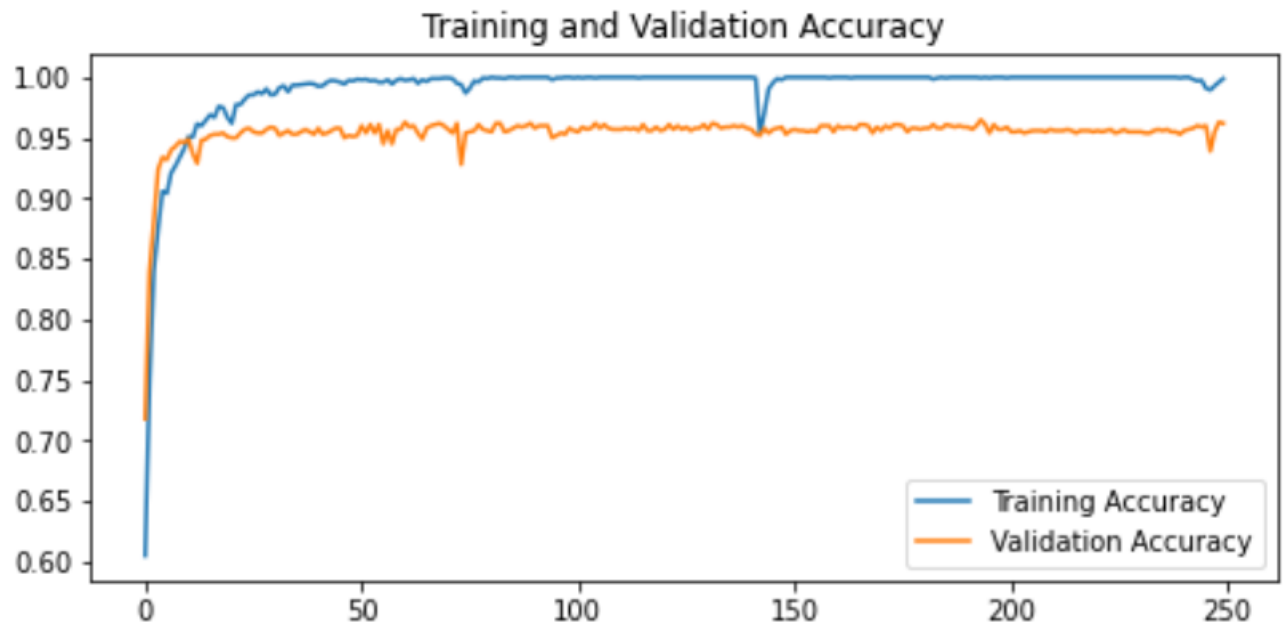
❖ Use pretrained weights as init weights

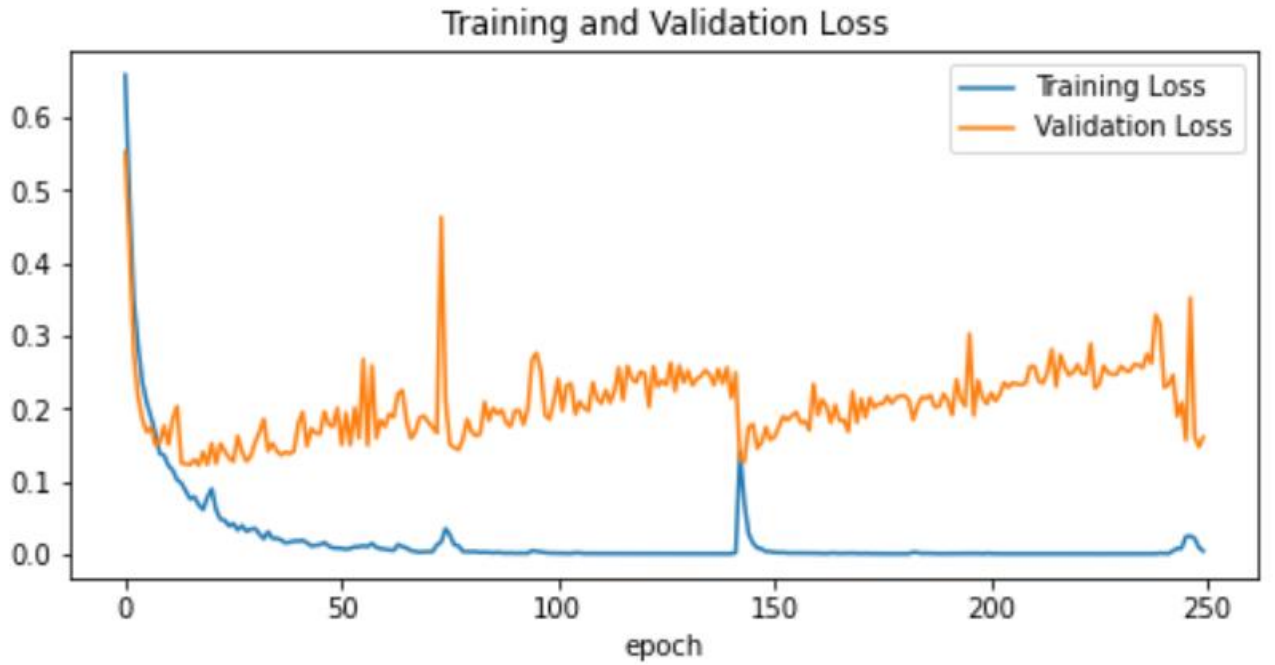
```
# process data
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
    tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
    tf.keras.layers.experimental.preprocessing.Rescaling(1./127.5, offset= -1)
])

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

# final layer
prediction_layer = tf.keras.layers.Dense(1)

# 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 1

- **Use pretrained weights init weights**

```
base_learning_rate =
model.compile(loss=tf.keras.losses.BinaryCrossentropy(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 1

❖ Use pretrained weights as init weights

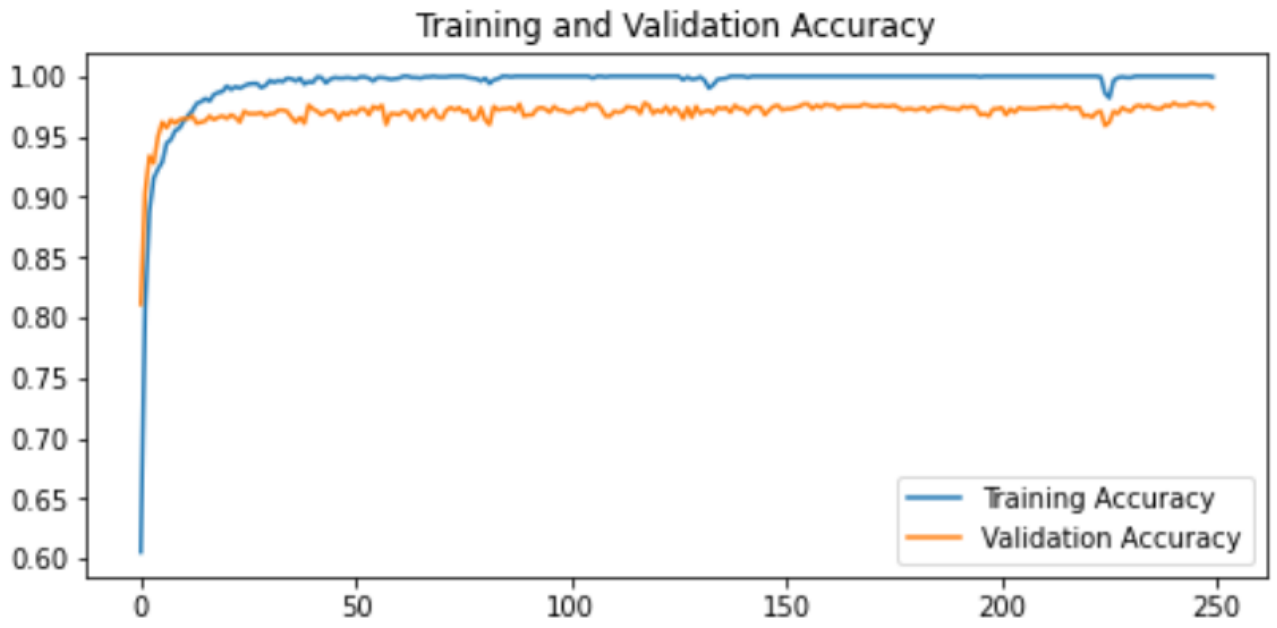
❖ Global max pooling

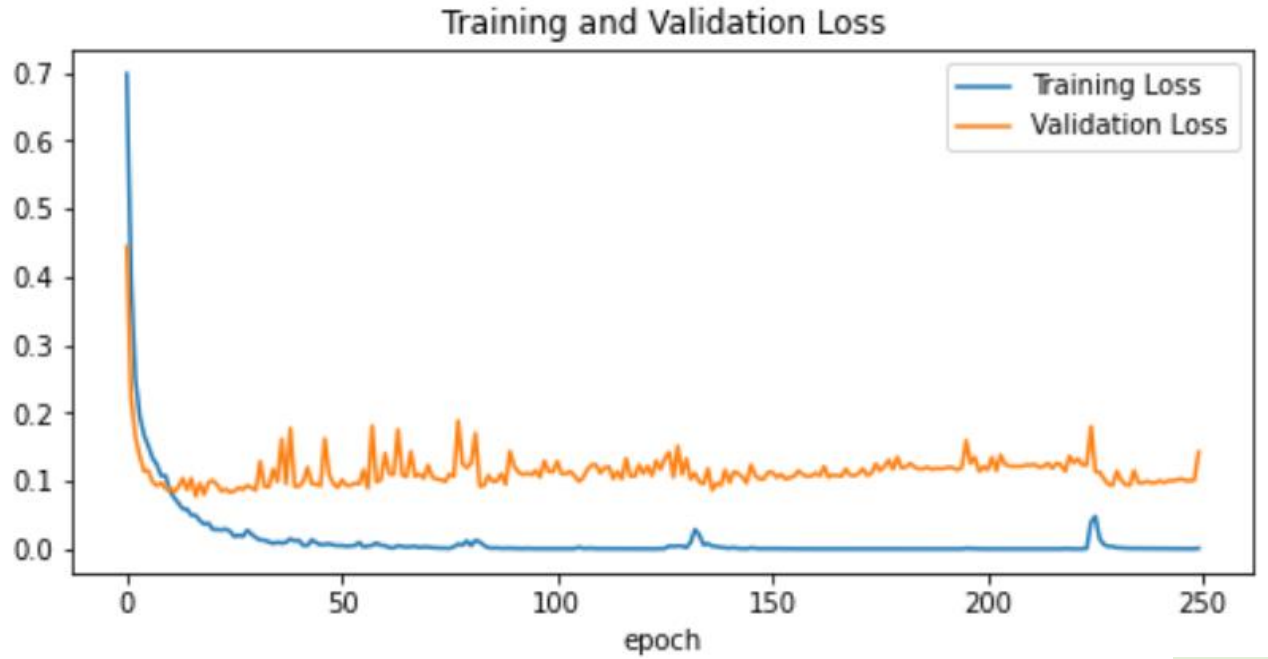
```
# process data
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
    tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
    tf.keras.layers.experimental.preprocessing.Rescaling(1./127.5, offset= -1)
])

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

# final layer
prediction_layer = tf.keras.layers.Dense(1)

# 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 1

- Use pretrained weights init weights
- Global max pooling

```
base_learning_rate =
model.compile(loss=tf.keras.losses.BinaryCrossentropy(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)
```





Year 2020 0.978

Case Study 1

❖ Use pretrained weights as init weights

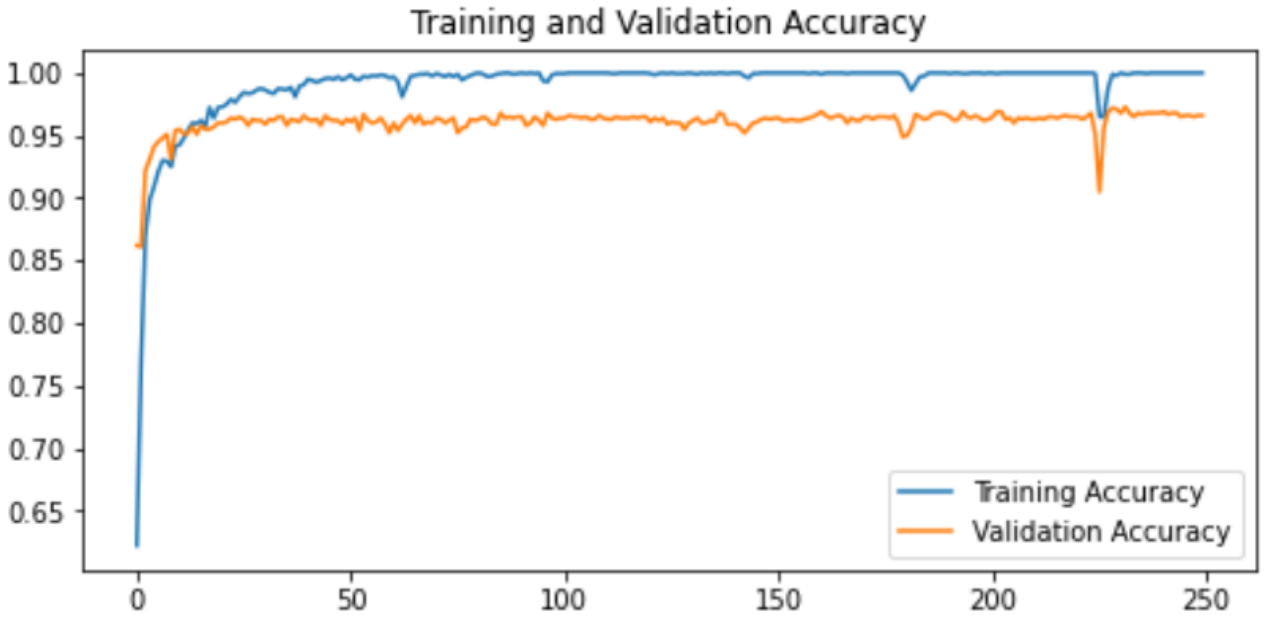
❖ Global average pooling

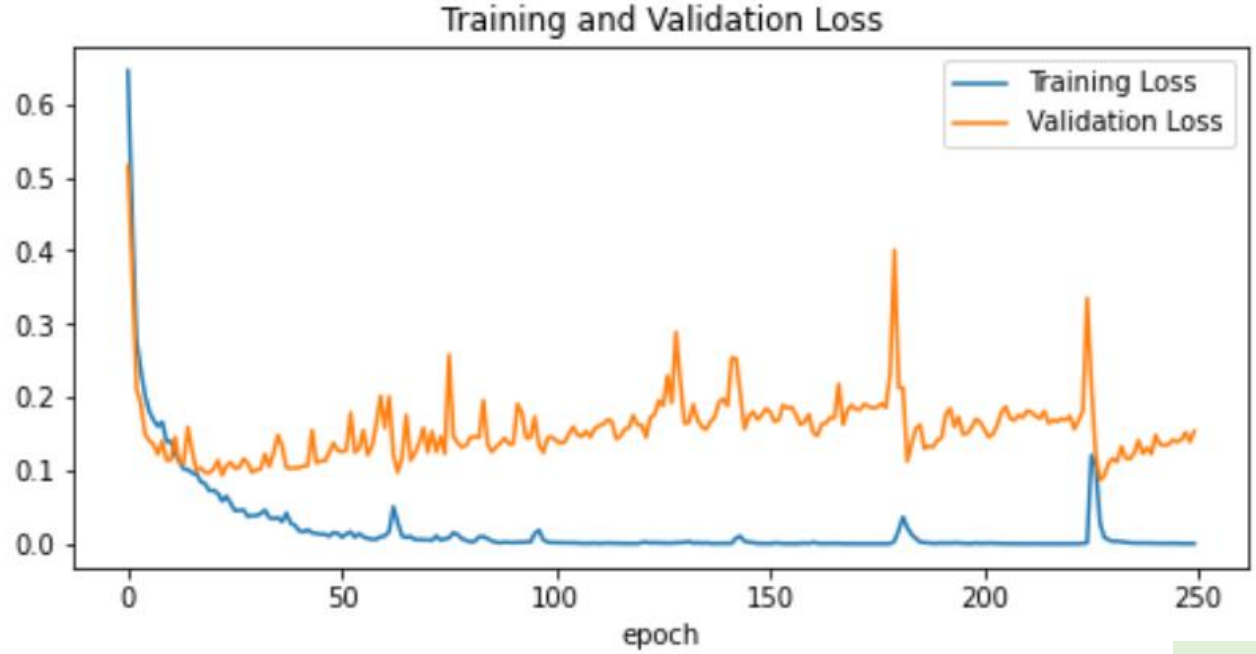
```
# process data
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
    tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
    tf.keras.layers.experimental.preprocessing.Rescaling(1./127.5, offset= -1)
])

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

# final layer
prediction_layer = tf.keras.layers.Dense(1)

# 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 1

- Use pretrained weights init weights
- Global average pooling

```
base_learning_rate =
model.compile(loss=tf.keras.losses.BinaryCrossentropy(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)
```





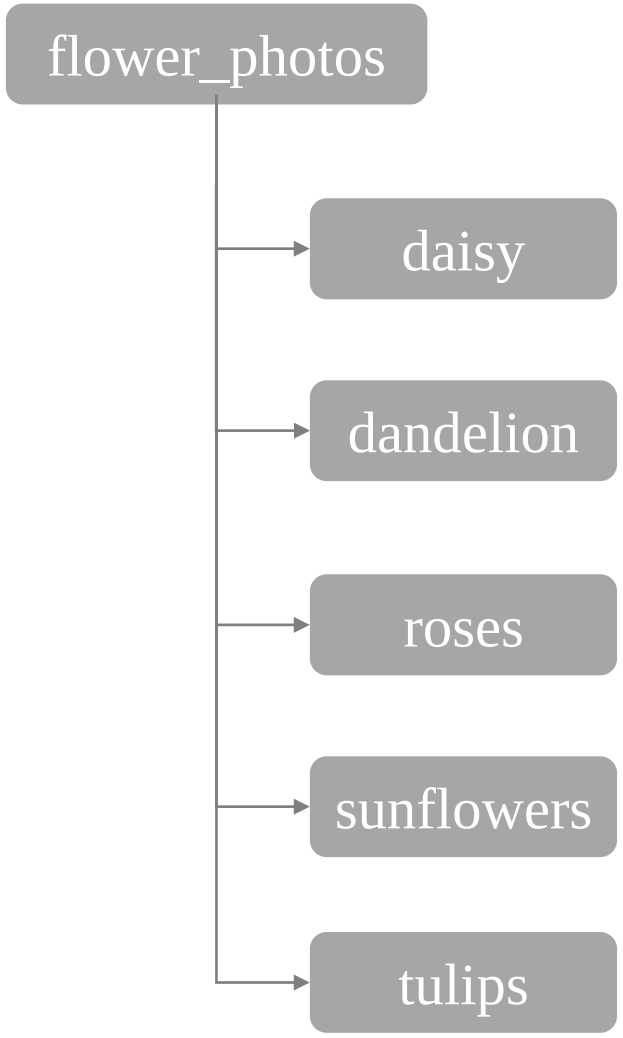
Year 2020 0.969

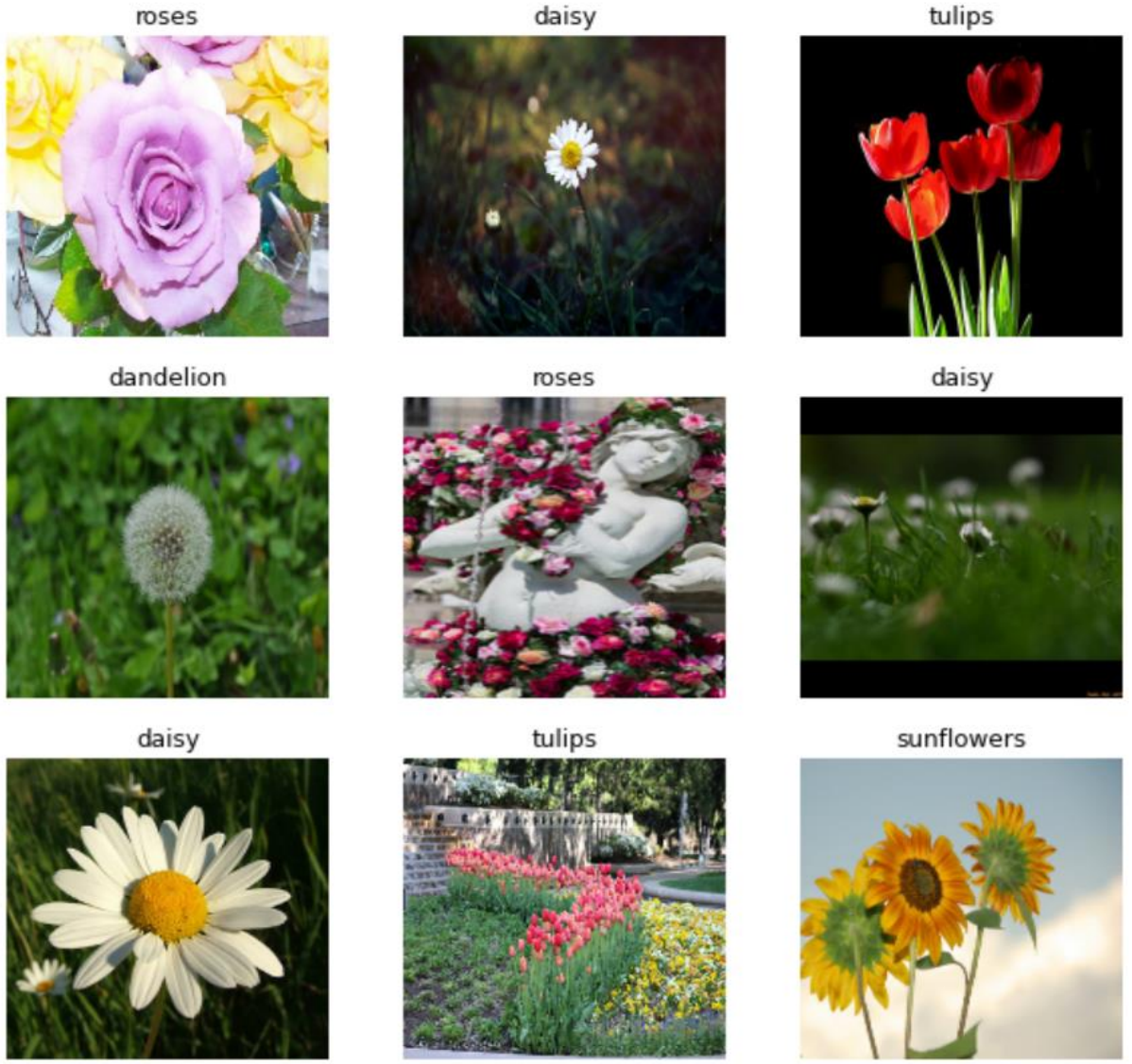
Outline

- Global Pooling
- Data Processing
- Data Processing Layer
- Network Manipulation
- Reuse a Pre-trained Model
- Case Study 1

Data Processing

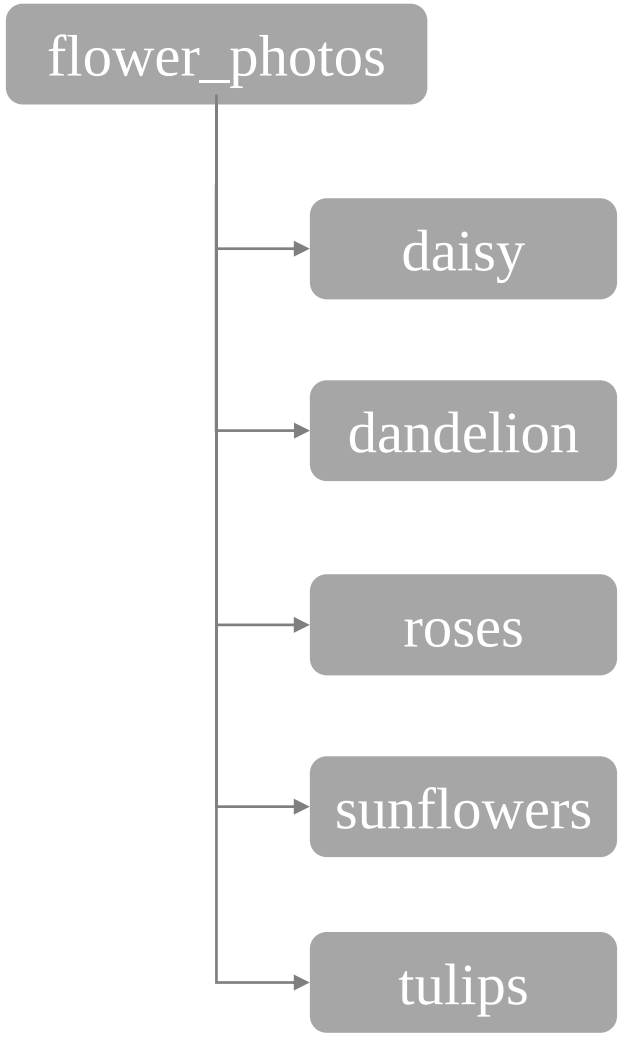
❖ In keras





Data Processing

❖ In keras




```python
PATH = 'flower_photos/'

train_dir = os.path.join(PATH, 'train')
validation_dir = os.path.join(PATH, 'validation')

BATCH_SIZE = 256
IMG_SIZE = (160, 160)
BUFFER_SIZE = BATCH_SIZE*5

train_dataset = tf.keras.preprocessing.image_dataset_from_directory(
                                                    PATH,
                                                    validation_split=0.2,
                                                    subset="training",
                                                    seed=123,
                                                    image_size=IMG_SIZE,
                                                    batch_size=BATCH_SIZE)
```

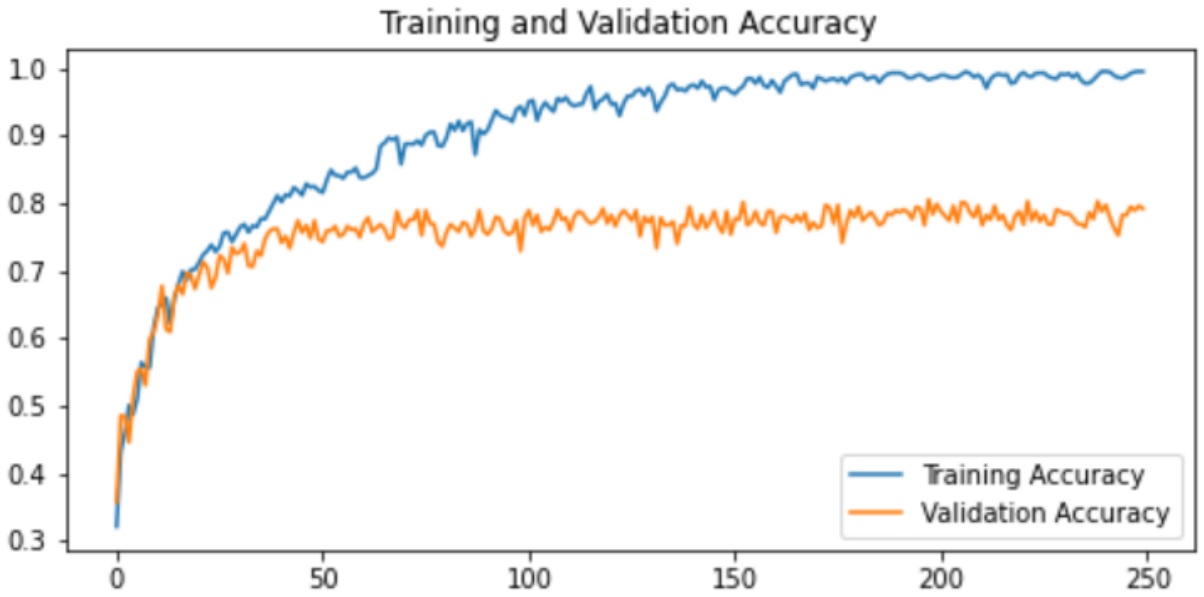
```
Found 3670 files belonging to 5 classes.
Using 2936 files for training.
```

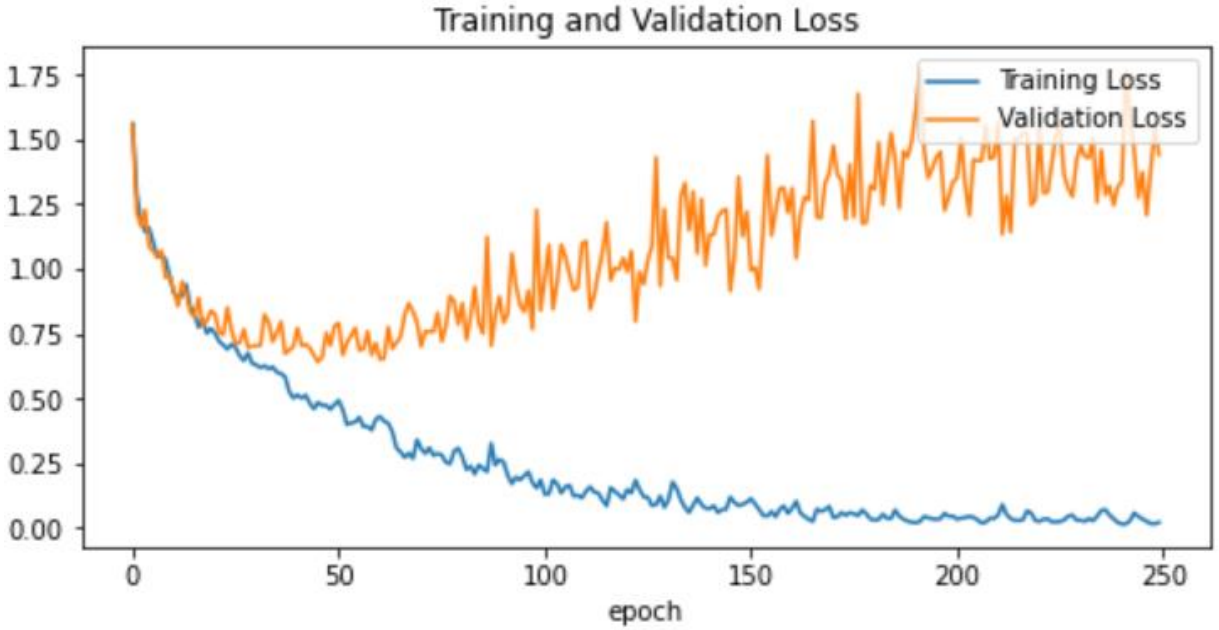
```python
validation_dataset = tf.keras.preprocessing.image_dataset_from_directory(
                                                    PATH,
                                                    validation_split=0.2,
                                                    subset="validation",
                                                    seed=123,
                                                    image_size=IMG_SIZE,
                                                    batch_size=BATCH_SIZE)
```

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

Case Study 2

❖ Train from scratch

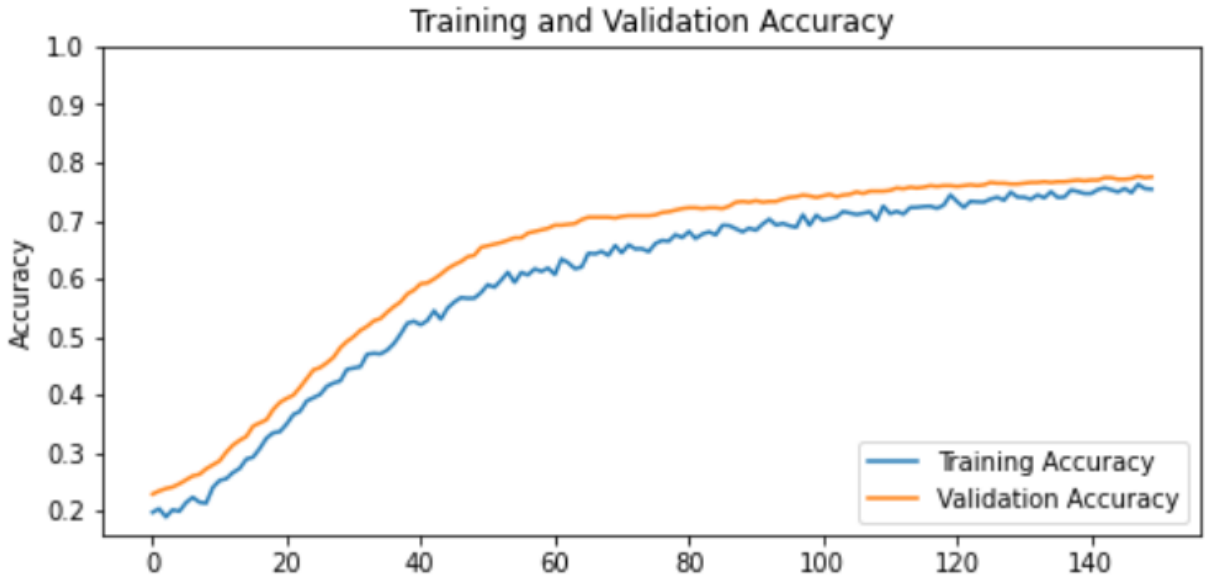


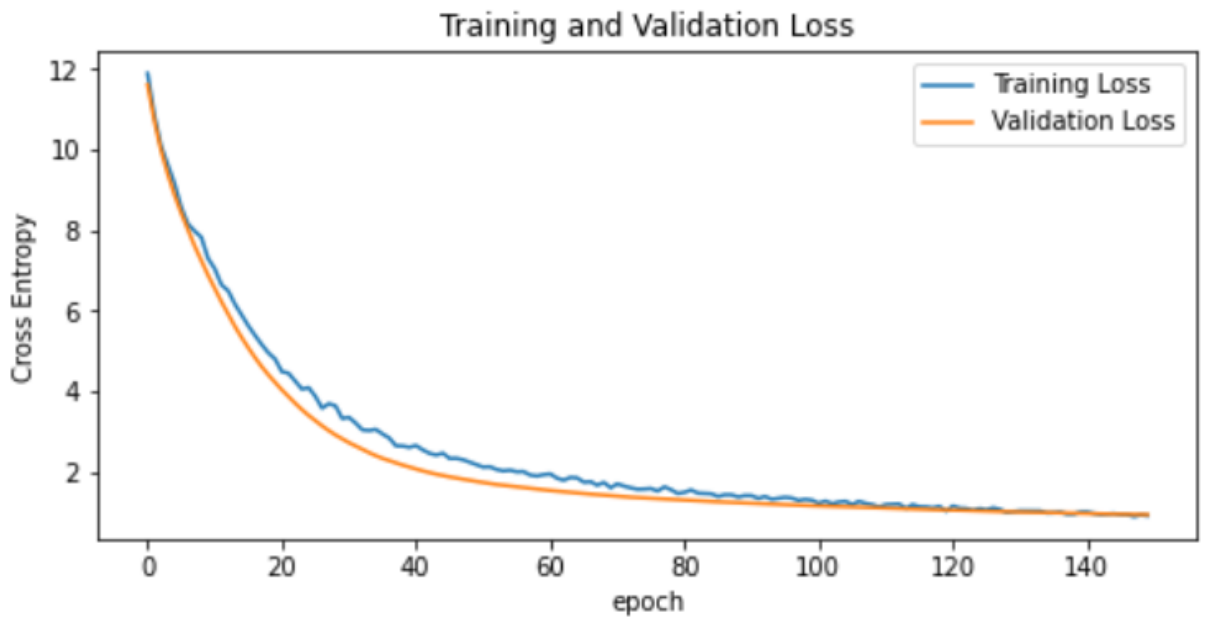


Use dropout 20% before prediction layer

Case Study 2

❖ Transfer learning

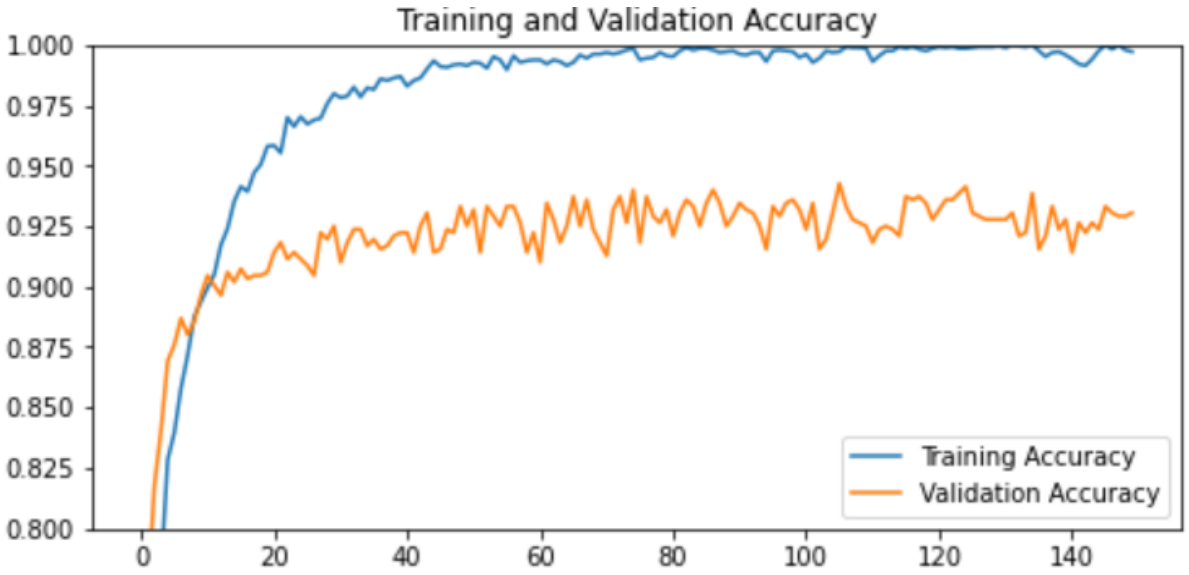




Use dropout 20% before prediction layer

Case Study 2

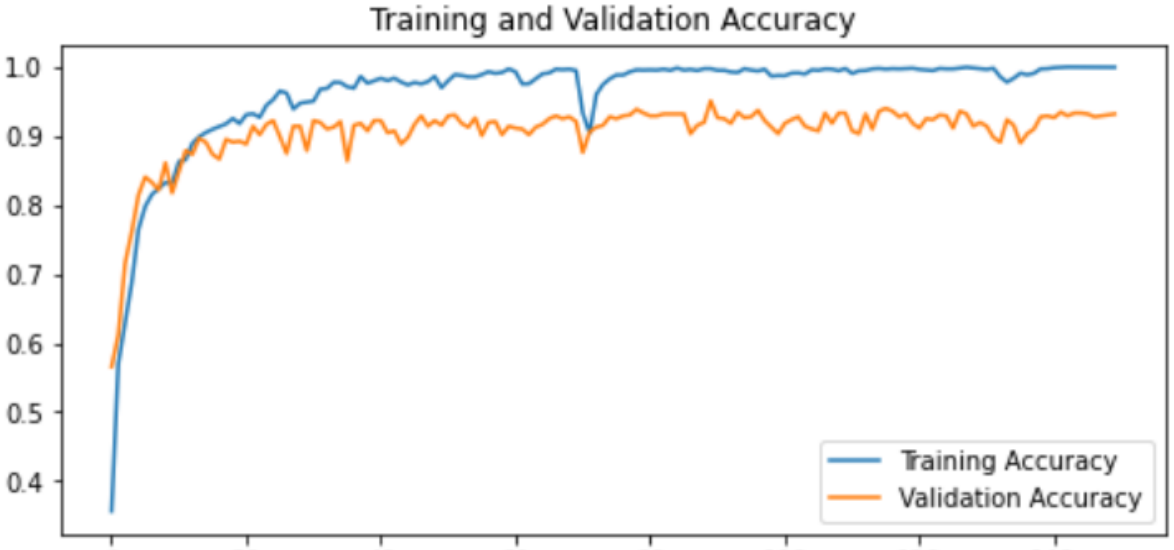
❖ Fine tuning



Use dropout 20% before prediction layer

Case Study 2

❖ For initialization



Use dropout 20% before prediction layer


Thank you!