## PERCOBAAN 1

def train\_val\_generators(TRAINING\_DIR, VALIDATION\_DIR):

# Instantiate the ImageDataGenerator class (don't forget to set the rescale argument)

train\_datagen = ImageDataGenerator(rescale = 1.0/255.,

fill\_mode='nearest',

rotation\_range=45,

width\_shift\_range=0.1,

height\_shift\_range=0.1,

shear\_range=0.1,

zoom\_range=0.1,

horizontal\_flip=True,

)

# Pass in the appropriate arguments to the flow\_from\_directory method

train\_generator = train\_datagen.flow\_from\_directory(directory=TRAINING\_DIR,

batch\_size=128,

class\_mode='categorical',

target\_size=(100, 100))

# Instantiate the ImageDataGenerator class (don't forget to set the rescale argument)

validation\_datagen = ImageDataGenerator(rescale = 1.0 / 255. )

# Pass in the appropriate arguments to the flow\_from\_directory method

validation\_generator = validation\_datagen.flow\_from\_directory(directory=VALIDATION\_DIR,

batch\_size=128,

class\_mode='categorical',

target\_size=(100, 100))

### END CODE HERE

return train\_generator, validation\_generator

IMAGE\_SIZE = [100, 100]

inception = InceptionV3(input\_shape=IMAGE\_SIZE + [3], weights='imagenet', include\_top=False)

for layer in inception.layers:

layer.trainable = False

model = tf.keras.models.Sequential([

inception,

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(512, activation='relu'),

tf.keras.layers.Dropout(rate=0.2),

tf.keras.layers.Dense(4, activation='softmax')

])

Optimizer Adam

### **Hasil 1**

Epoch 1/20

C:\Users\ASUS\anaconda3\Lib\site-packages\keras\src\trainers\data\_adapters\py\_dataset\_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().\_\_init\_\_(\*\*kwargs)` in its constructor. `\*\*kwargs` can include `workers`, `use\_multiprocessing`, `max\_queue\_size`. Do not pass these arguments to `fit()`, as they will be ignored.  
 self.\_warn\_if\_super\_not\_called()

**3/25** ━━━━━━━━━━━━━━━━━━━━ **42s** 2s/step - accuracy: 0.2251 - loss: 3.7621

C:\Users\ASUS\anaconda3\Lib\contextlib.py:155: UserWarning: Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least `steps\_per\_epoch \* epochs` batches. You may need to use the `.repeat()` function when building your dataset.  
 self.gen.throw(typ, value, traceback)

**25/25** ━━━━━━━━━━━━━━━━━━━━ **46s** 606ms/step - accuracy: 0.2252 - loss: 4.0979 - val\_accuracy: 0.2750 - val\_loss: 5.4240  
Epoch 2/20  
**25/25** ━━━━━━━━━━━━━━━━━━━━ **16s** 311ms/step - accuracy: 0.3772 - loss: 3.2237 - val\_accuracy: 0.2750 - val\_loss: 3.3467  
Epoch 3/20  
**25/25** ━━━━━━━━━━━━━━━━━━━━ **15s** 253ms/step - accuracy: 0.3997 - loss: 1.8244 - val\_accuracy: 0.4375 - val\_loss: 1.1787  
Epoch 4/20  
**25/25** ━━━━━━━━━━━━━━━━━━━━ **14s** 250ms/step - accuracy: 0.3360 - loss: 1.5317 - val\_accuracy: 0.5750 - val\_loss: 1.1013  
Epoch 5/20  
**25/25** ━━━━━━━━━━━━━━━━━━━━ **14s** 250ms/step - accuracy: 0.4604 - loss: 1.2375 - val\_accuracy: 0.4250 - val\_loss: 1.2734  
Epoch 6/20  
**25/25** ━━━━━━━━━━━━━━━━━━━━ **16s** 289ms/step - accuracy: 0.4761 - loss: 1.1750 - val\_accuracy: 0.4750 - val\_loss: 1.2300  
Epoch 7/20  
**25/25** ━━━━━━━━━━━━━━━━━━━━ **15s** 265ms/step - accuracy: 0.4983 - loss: 1.0570 - val\_accuracy: 0.4875 - val\_loss: 1.1205  
Epoch 8/20  
**25/25** ━━━━━━━━━━━━━━━━━━━━ **16s** 271ms/step - accuracy: 0.5500 - loss: 1.0505 - val\_accuracy: 0.6125 - val\_loss: 0.9834  
Epoch 9/20  
**25/25** ━━━━━━━━━━━━━━━━━━━━ **14s** 249ms/step - accuracy: 0.5924 - loss: 0.9811 - val\_accuracy: 0.5750 - val\_loss: 0.9374  
Epoch 10/20  
**25/25** ━━━━━━━━━━━━━━━━━━━━ **14s** 247ms/step - accuracy: 0.5998 - loss: 0.9717 - val\_accuracy: 0.6625 - val\_loss: 0.8357  
Epoch 11/20  
**25/25** ━━━━━━━━━━━━━━━━━━━━ **14s** 254ms/step - accuracy: 0.6212 - loss: 0.9110 - val\_accuracy: 0.7125 - val\_loss: 0.7825  
Epoch 12/20  
**25/25** ━━━━━━━━━━━━━━━━━━━━ **14s** 248ms/step - accuracy: 0.6243 - loss: 0.9058 - val\_accuracy: 0.6625 - val\_loss: 0.7863  
Epoch 13/20  
**25/25** ━━━━━━━━━━━━━━━━━━━━ **14s** 255ms/step - accuracy: 0.6584 - loss: 0.8641 - val\_accuracy: 0.7250 - val\_loss: 0.7689  
Epoch 14/20  
**25/25** ━━━━━━━━━━━━━━━━━━━━ **14s** 250ms/step - accuracy: 0.6870 - loss: 0.8012 - val\_accuracy: 0.7375 - val\_loss: 0.6781  
Epoch 15/20  
**25/25** ━━━━━━━━━━━━━━━━━━━━ **20s** 346ms/step - accuracy: 0.6741 - loss: 0.7646 - val\_accuracy: 0.8000 - val\_loss: 0.5925  
Epoch 16/20  
**25/25** ━━━━━━━━━━━━━━━━━━━━ **21s** 341ms/step - accuracy: 0.6896 - loss: 0.8240 - val\_accuracy: 0.7500 - val\_loss: 0.6360  
Epoch 17/20  
**25/25** ━━━━━━━━━━━━━━━━━━━━ **21s** 327ms/step - accuracy: 0.7062 - loss: 0.7601 - val\_accuracy: 0.7875 - val\_loss: 0.5979  
Epoch 18/20  
**25/25** ━━━━━━━━━━━━━━━━━━━━ **16s** 281ms/step - accuracy: 0.7575 - loss: 0.6874 - val\_accuracy: 0.8125 - val\_loss: 0.5147  
Epoch 19/20  
**25/25** ━━━━━━━━━━━━━━━━━━━━ **17s** 351ms/step - accuracy: 0.7529 - loss: 0.7020 - val\_accuracy: 0.8125 - val\_loss: 0.5370  
Epoch 20/20  
**25/25** ━━━━━━━━━━━━━━━━━━━━ **31s** 757ms/step - accuracy: 0.7396 - loss: 0.6997 - val\_accuracy: 0.8250 - val\_loss: 0.5328

(syntax plot)

loss = history.history['loss']

val\_loss = history.history['val\_loss']

accuracy = history.history['accuracy']

val\_accuracy = history.history['val\_accuracy']

epochs = range(1, len(loss) + 1)

# Plot loss

plt.plot(epochs, loss, label='Training Loss')

plt.plot(epochs, val\_loss, label='Validation Loss')

plt.title('Training and Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

# Plot accuracy

plt.plot(epochs, accuracy, label='Training Accuracy')

plt.plot(epochs, val\_accuracy, label='Validation Accuracy')

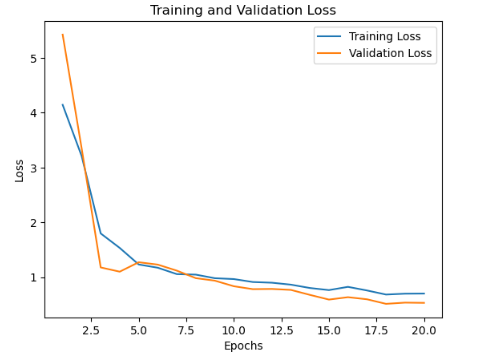
plt.title('Training and Validation Accuracy')

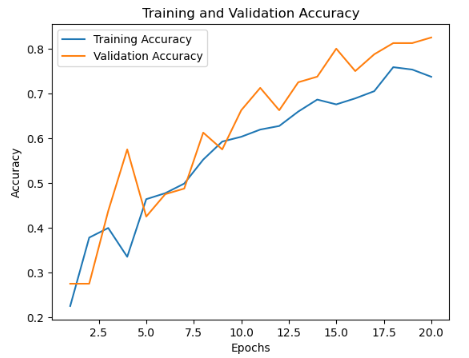
plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()





## **PERCOBAAN 2**

def train\_val\_generators(TRAINING\_DIR, VALIDATION\_DIR):

# Instantiate the ImageDataGenerator class (don't forget to set the rescale argument)

train\_datagen = ImageDataGenerator(rescale = 1.0/255.,

fill\_mode='nearest',

rotation\_range=45,

width\_shift\_range=0.1,

height\_shift\_range=0.1,

shear\_range=0.1,

zoom\_range=0.1,

horizontal\_flip=True,

)

# Pass in the appropriate arguments to the flow\_from\_directory method

train\_generator = train\_datagen.flow\_from\_directory(directory=TRAINING\_DIR,

batch\_size=128,

class\_mode='categorical',

target\_size=(100, 100))

# Instantiate the ImageDataGenerator class (don't forget to set the rescale argument)

validation\_datagen = ImageDataGenerator(rescale = 1.0 / 255. )

# Pass in the appropriate arguments to the flow\_from\_directory method

validation\_generator = validation\_datagen.flow\_from\_directory(directory=VALIDATION\_DIR,

batch\_size=128,

class\_mode='categorical',

target\_size=(100, 100))

### END CODE HERE

return train\_generator, validation\_generator

IMAGE\_SIZE = [100, 100]

inception = InceptionV3(input\_shape=IMAGE\_SIZE + [3], weights='imagenet', include\_top=False)

for layer in inception.layers:

layer.trainable = False

model = tf.keras.models.Sequential([

inception,

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(1028, activation='relu'),

tf.keras.layers.Dropout(rate=0.3),

tf.keras.layers.Dense(4, activation='softmax')

])

Optimizer Adam

### **Hasil 2**

Epoch 1/20

**25/25** ━━━━━━━━━━━━━━━━━━━━ **57s** 759ms/step - accuracy: 0.3470 - loss: 6.5031 - val\_accuracy: 0.4000 - val\_loss: 4.4431

Epoch 2/20

**25/25** ━━━━━━━━━━━━━━━━━━━━ **16s** 286ms/step - accuracy: 0.3035 - loss: 3.0845 - val\_accuracy: 0.2750 - val\_loss: 2.8332

Epoch 3/20

**25/25** ━━━━━━━━━━━━━━━━━━━━ **16s** 293ms/step - accuracy: 0.3469 - loss: 2.4373 - val\_accuracy: 0.3000 - val\_loss: 2.7974

Epoch 4/20

**25/25** ━━━━━━━━━━━━━━━━━━━━ **16s** 293ms/step - accuracy: 0.4093 - loss: 1.7070 - val\_accuracy: 0.5625 - val\_loss: 0.9937

Epoch 5/20

**25/25** ━━━━━━━━━━━━━━━━━━━━ **16s** 284ms/step - accuracy: 0.4194 - loss: 1.3554 - val\_accuracy: 0.3750 - val\_loss: 1.2242

Epoch 6/20

**25/25** ━━━━━━━━━━━━━━━━━━━━ **16s** 281ms/step - accuracy: 0.4800 - loss: 1.2325 - val\_accuracy: 0.6625 - val\_loss: 0.9151

Epoch 7/20

**25/25** ━━━━━━━━━━━━━━━━━━━━ **16s** 273ms/step - accuracy: 0.5369 - loss: 1.0569 - val\_accuracy: 0.5375 - val\_loss: 0.9821

Epoch 8/20

**25/25** ━━━━━━━━━━━━━━━━━━━━ **16s** 273ms/step - accuracy: 0.5209 - loss: 1.0975 - val\_accuracy: 0.5500 - val\_loss: 0.9595

Epoch 9/20

**25/25** ━━━━━━━━━━━━━━━━━━━━ **16s** 274ms/step - accuracy: 0.5591 - loss: 1.0191 - val\_accuracy: 0.6375 - val\_loss: 0.8547

Epoch 10/20

**25/25** ━━━━━━━━━━━━━━━━━━━━ **16s** 286ms/step - accuracy: 0.6327 - loss: 0.9268 - val\_accuracy: 0.7625 - val\_loss: 0.7837

Epoch 11/20

**25/25** ━━━━━━━━━━━━━━━━━━━━ **16s** 260ms/step - accuracy: 0.6200 - loss: 0.8821 - val\_accuracy: 0.7750 - val\_loss: 0.7153

Epoch 12/20

**25/25** ━━━━━━━━━━━━━━━━━━━━ **15s** 269ms/step - accuracy: 0.6520 - loss: 0.8578 - val\_accuracy: 0.8250 - val\_loss: 0.6531

Epoch 13/20

**25/25** ━━━━━━━━━━━━━━━━━━━━ **16s** 294ms/step - accuracy: 0.6731 - loss: 0.7937 - val\_accuracy: 0.7875 - val\_loss: 0.6595

Epoch 14/20

**25/25** ━━━━━━━━━━━━━━━━━━━━ **16s** 323ms/step - accuracy: 0.6872 - loss: 0.8056 - val\_accuracy: 0.7875 - val\_loss: 0.6083

Epoch 15/20

**25/25** ━━━━━━━━━━━━━━━━━━━━ **21s** 451ms/step - accuracy: 0.6751 - loss: 0.8022 - val\_accuracy: 0.8250 - val\_loss: 0.5475

Epoch 16/20

**25/25** ━━━━━━━━━━━━━━━━━━━━ **35s** 803ms/step - accuracy: 0.6986 - loss: 0.7436 - val\_accuracy: 0.8000 - val\_loss: 0.5348

Epoch 17/20

**25/25** ━━━━━━━━━━━━━━━━━━━━ **21s** 324ms/step - accuracy: 0.6948 - loss: 0.7845 - val\_accuracy: 0.8250 - val\_loss: 0.5309

Epoch 18/20

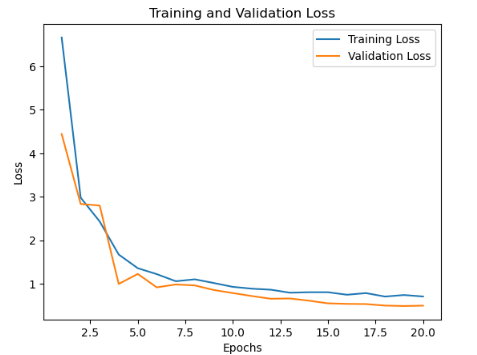
**25/25** ━━━━━━━━━━━━━━━━━━━━ **33s** 639ms/step - accuracy: 0.7113 - loss: 0.7009 - val\_accuracy: 0.8125 - val\_loss: 0.4960

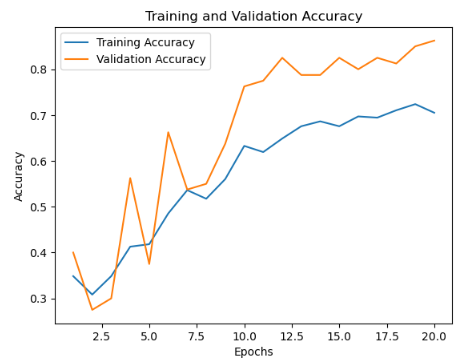
Epoch 19/20

**25/25** ━━━━━━━━━━━━━━━━━━━━ **23s** 259ms/step - accuracy: 0.7224 - loss: 0.7392 - val\_accuracy: 0.8500 - val\_loss: 0.4862

Epoch 20/20

**25/25** ━━━━━━━━━━━━━━━━━━━━ **15s** 268ms/step - accuracy: 0.7042 - loss: 0.7101 - val\_accuracy: 0.8625 - val\_loss: 0.4942





## **PERCOBAAN 3**

def train\_val\_generators(TRAINING\_DIR, VALIDATION\_DIR):

# Instantiate the ImageDataGenerator class (don't forget to set the rescale argument)

train\_datagen = ImageDataGenerator(rescale = 1.0/255.,

fill\_mode='nearest',

rotation\_range=45,

width\_shift\_range=0.1,

height\_shift\_range=0.1,

shear\_range=0.1,

zoom\_range=0.1,

horizontal\_flip=True,

)

# Pass in the appropriate arguments to the flow\_from\_directory method

train\_generator = train\_datagen.flow\_from\_directory(directory=TRAINING\_DIR,

batch\_size=128,

class\_mode='categorical',

target\_size=(100, 100))

# Instantiate the ImageDataGenerator class (don't forget to set the rescale argument)

validation\_datagen = ImageDataGenerator(rescale = 1.0 / 255. )

# Pass in the appropriate arguments to the flow\_from\_directory method

validation\_generator = validation\_datagen.flow\_from\_directory(directory=VALIDATION\_DIR,

batch\_size=128,

class\_mode='categorical',

target\_size=(100, 100))

### END CODE HERE

return train\_generator, validation\_generator

IMAGE\_SIZE = [100, 100]

inception = InceptionV3(input\_shape=IMAGE\_SIZE + [3], weights='imagenet', include\_top=False)

for layer in inception.layers:

layer.trainable = False

model = tf.keras.models.Sequential([

inception,

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(1028, activation='relu'),

tf.keras.layers.Dropout(rate=0.3),

tf.keras.layers.Dense(4, activation='softmax')

])

Optimizer rmsprop (ga define learning rate)

### **Hasil 3**

Epoch 1/25

**25/25** ━━━━━━━━━━━━━━━━━━━━ **31s** 510ms/step - accuracy: 0.6007 - loss: 3.5662 - val\_accuracy: 0.4000 - val\_loss: 5.2730

Epoch 2/25

**25/25** ━━━━━━━━━━━━━━━━━━━━ **18s** 320ms/step - accuracy: 0.5194 - loss: 2.5783 - val\_accuracy: 0.8250 - val\_loss: 0.5064

Epoch 3/25

**25/25** ━━━━━━━━━━━━━━━━━━━━ **16s** 276ms/step - accuracy: 0.7922 - loss: 0.5406 - val\_accuracy: 0.8500 - val\_loss: 0.4179

Epoch 4/25

**25/25** ━━━━━━━━━━━━━━━━━━━━ **27s** 511ms/step - accuracy: 0.8461 - loss: 0.4223 - val\_accuracy: 0.8625 - val\_loss: 0.3990

Epoch 5/25

**25/25** ━━━━━━━━━━━━━━━━━━━━ **29s** 458ms/step - accuracy: 0.8612 - loss: 0.3873 - val\_accuracy: 0.7750 - val\_loss: 0.5114

Epoch 6/25

**25/25** ━━━━━━━━━━━━━━━━━━━━ **28s** 627ms/step - accuracy: 0.7595 - loss: 0.6612 - val\_accuracy: 0.7750 - val\_loss: 0.5681

Epoch 7/25

**25/25** ━━━━━━━━━━━━━━━━━━━━ **26s** 636ms/step - accuracy: 0.6625 - loss: 0.8462 - val\_accuracy: 0.8750 - val\_loss: 0.3460

Epoch 8/25

**25/25** ━━━━━━━━━━━━━━━━━━━━ **17s** 350ms/step - accuracy: 0.8500 - loss: 0.3913 - val\_accuracy: 0.9250 - val\_loss: 0.2477

Epoch 9/25

**25/25** ━━━━━━━━━━━━━━━━━━━━ **16s** 328ms/step - accuracy: 0.8667 - loss: 0.3392 - val\_accuracy: 0.9125 - val\_loss: 0.2730

Epoch 10/25

**25/25** ━━━━━━━━━━━━━━━━━━━━ **16s** 399ms/step - accuracy: 0.8724 - loss: 0.3473 - val\_accuracy: 0.9250 - val\_loss: 0.2295

Epoch 11/25

**25/25** ━━━━━━━━━━━━━━━━━━━━ **20s** 432ms/step - accuracy: 0.7732 - loss: 0.5687 - val\_accuracy: 0.7375 - val\_loss: 0.6099

Epoch 12/25

**25/25** ━━━━━━━━━━━━━━━━━━━━ **20s** 448ms/step - accuracy: 0.7399 - loss: 0.6739 - val\_accuracy: 0.9250 - val\_loss: 0.2449

Epoch 13/25

**25/25** ━━━━━━━━━━━━━━━━━━━━ **15s** 319ms/step - accuracy: 0.8561 - loss: 0.4057 - val\_accuracy: 0.8500 - val\_loss: 0.3549

Epoch 14/25

**25/25** ━━━━━━━━━━━━━━━━━━━━ **15s** 332ms/step - accuracy: 0.8202 - loss: 0.4761 - val\_accuracy: 0.8875 - val\_loss: 0.2484

Epoch 15/25

**25/25** ━━━━━━━━━━━━━━━━━━━━ **13s** 233ms/step - accuracy: 0.8474 - loss: 0.3844 - val\_accuracy: 0.9125 - val\_loss: 0.2408

Epoch 16/25

**25/25** ━━━━━━━━━━━━━━━━━━━━ **19s** 376ms/step - accuracy: 0.8848 - loss: 0.3212 - val\_accuracy: 0.8875 - val\_loss: 0.2815

Epoch 17/25

**25/25** ━━━━━━━━━━━━━━━━━━━━ **13s** 297ms/step - accuracy: 0.8593 - loss: 0.3807 - val\_accuracy: 0.8250 - val\_loss: 0.5027

Epoch 18/25

**25/25** ━━━━━━━━━━━━━━━━━━━━ **19s** 386ms/step - accuracy: 0.7781 - loss: 0.5820 - val\_accuracy: 0.6625 - val\_loss: 1.1982

Epoch 19/25

**25/25** ━━━━━━━━━━━━━━━━━━━━ **16s** 362ms/step - accuracy: 0.7176 - loss: 0.9638 - val\_accuracy: 0.9250 - val\_loss: 0.2416

Epoch 20/25

**25/25** ━━━━━━━━━━━━━━━━━━━━ **17s** 359ms/step - accuracy: 0.8927 - loss: 0.3375 - val\_accuracy: 0.9250 - val\_loss: 0.2287

Epoch 21/25

**25/25** ━━━━━━━━━━━━━━━━━━━━ **14s** 235ms/step - accuracy: 0.8383 - loss: 0.3837 - val\_accuracy: 0.9125 - val\_loss: 0.2562

Epoch 22/25

**25/25** ━━━━━━━━━━━━━━━━━━━━ **20s** 403ms/step - accuracy: 0.8737 - loss: 0.3446 - val\_accuracy: 0.8375 - val\_loss: 0.4727

Epoch 23/25

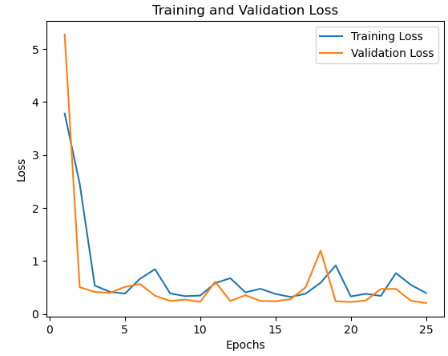
**25/25** ━━━━━━━━━━━━━━━━━━━━ **24s** 634ms/step - accuracy: 0.7250 - loss: 0.7645 - val\_accuracy: 0.7875 - val\_loss: 0.4752

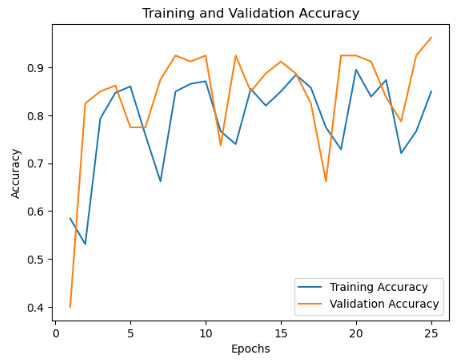
Epoch 24/25

**25/25** ━━━━━━━━━━━━━━━━━━━━ **17s** 334ms/step - accuracy: 0.7642 - loss: 0.5561 - val\_accuracy: 0.9250 - val\_loss: 0.2502

Epoch 25/25

**25/25** ━━━━━━━━━━━━━━━━━━━━ **19s** 400ms/step - accuracy: 0.8499 - loss: 0.3969 - val\_accuracy: 0.9625 - val\_loss: 0.2087





## **PERCOBAAN 4**

def train\_val\_generators(TRAINING\_DIR, VALIDATION\_DIR):

# Instantiate the ImageDataGenerator class (don't forget to set the rescale argument)

train\_datagen = ImageDataGenerator(rescale = 1.0/255.,

fill\_mode='nearest',

rotation\_range=45,

width\_shift\_range=0.1,

height\_shift\_range=0.1,

shear\_range=0.1,

zoom\_range=0.1,

horizontal\_flip=True,

)

# Pass in the appropriate arguments to the flow\_from\_directory method

train\_generator = train\_datagen.flow\_from\_directory(directory=TRAINING\_DIR,

batch\_size=128,

class\_mode='categorical',

target\_size=(100, 100))

# Instantiate the ImageDataGenerator class (don't forget to set the rescale argument)

validation\_datagen = ImageDataGenerator(rescale = 1.0 / 255. )

# Pass in the appropriate arguments to the flow\_from\_directory method

validation\_generator = validation\_datagen.flow\_from\_directory(directory=VALIDATION\_DIR,

batch\_size=128,

class\_mode='categorical',

target\_size=(100, 100))

### END CODE HERE

return train\_generator, validation\_generator

IMAGE\_SIZE = [100, 100]

inception = InceptionV3(input\_shape=IMAGE\_SIZE + [3], weights='imagenet', include\_top=False)

for layer in inception.layers:

layer.trainable = False

model = tf.keras.models.Sequential([

inception,

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(1028, activation='relu'),

tf.keras.layers.Dropout(rate=0.3),

tf.keras.layers.Dense(5, activation='softmax')

])

model.compile(optimizer=tf.keras.optimizers.RMSprop(learning\_rate=0.001),

loss='categorical\_crossentropy',

metrics=['accuracy'])

### **Hasil 4**

Epoch 1/50

C:\Users\Latifatul Khumairoh\anaconda3\Lib\site-packages\keras\src\trainers\data\_adapters\py\_dataset\_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().\_\_init\_\_(\*\*kwargs)` in its constructor. `\*\*kwargs` can include `workers`, `use\_multiprocessing`, `max\_queue\_size`. Do not pass these arguments to `fit()`, as they will be ignored.

self.\_warn\_if\_super\_not\_called()

**5/50** ━━━━━━━━━━━━━━━━━━━━ **31s** 695ms/step - accuracy: 0.2619 - loss: 11.2347

C:\Users\Latifatul Khumairoh\anaconda3\Lib\contextlib.py:155: UserWarning: Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least `steps\_per\_epoch \* epochs` batches. You may need to use the `.repeat()` function when building your dataset.

self.gen.throw(typ, value, traceback)

**50/50** ━━━━━━━━━━━━━━━━━━━━ **26s** 191ms/step - accuracy: 0.2953 - loss: 12.4698 - val\_accuracy: 0.1961 - val\_loss: 10.2455

Epoch 2/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **10s** 90ms/step - accuracy: 0.3876 - loss: 3.4967 - val\_accuracy: 0.2843 - val\_loss: 1.5876

Epoch 3/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **10s** 86ms/step - accuracy: 0.4849 - loss: 1.3733 - val\_accuracy: 0.3725 - val\_loss: 1.4744

Epoch 4/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **10s** 82ms/step - accuracy: 0.4881 - loss: 1.4103 - val\_accuracy: 0.5392 - val\_loss: 1.2708

Epoch 5/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **10s** 84ms/step - accuracy: 0.5149 - loss: 1.3129 - val\_accuracy: 0.4902 - val\_loss: 1.3710

Epoch 6/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **10s** 87ms/step - accuracy: 0.4868 - loss: 1.3019 - val\_accuracy: 0.3431 - val\_loss: 1.8407

Epoch 7/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **10s** 83ms/step - accuracy: 0.5432 - loss: 1.1967 - val\_accuracy: 0.4412 - val\_loss: 1.3479

Epoch 8/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **9s** 82ms/step - accuracy: 0.5598 - loss: 1.1739 - val\_accuracy: 0.4020 - val\_loss: 1.4905

Epoch 9/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **9s** 81ms/step - accuracy: 0.5649 - loss: 1.1507 - val\_accuracy: 0.3824 - val\_loss: 1.3729

Epoch 10/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **9s** 82ms/step - accuracy: 0.5640 - loss: 1.1114 - val\_accuracy: 0.5784 - val\_loss: 1.0842

Epoch 11/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **9s** 81ms/step - accuracy: 0.5995 - loss: 1.0668 - val\_accuracy: 0.3529 - val\_loss: 1.7292

Epoch 12/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **9s** 82ms/step - accuracy: 0.5579 - loss: 1.2235 - val\_accuracy: 0.5980 - val\_loss: 1.0744

Epoch 13/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **9s** 82ms/step - accuracy: 0.6503 - loss: 0.9628 - val\_accuracy: 0.4804 - val\_loss: 1.3760

Epoch 14/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **9s** 81ms/step - accuracy: 0.5327 - loss: 1.2438 - val\_accuracy: 0.4412 - val\_loss: 1.2522

Epoch 15/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **9s** 82ms/step - accuracy: 0.6603 - loss: 0.8932 - val\_accuracy: 0.5000 - val\_loss: 1.3659

Epoch 16/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **10s** 85ms/step - accuracy: 0.6150 - loss: 1.0746 - val\_accuracy: 0.6176 - val\_loss: 0.9867

Epoch 17/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **10s** 83ms/step - accuracy: 0.6771 - loss: 0.8519 - val\_accuracy: 0.5882 - val\_loss: 1.0733

Epoch 18/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **10s** 84ms/step - accuracy: 0.6465 - loss: 0.9211 - val\_accuracy: 0.5490 - val\_loss: 1.1577

Epoch 19/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **10s** 91ms/step - accuracy: 0.5992 - loss: 1.0674 - val\_accuracy: 0.4902 - val\_loss: 1.8383

Epoch 20/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **10s** 85ms/step - accuracy: 0.6376 - loss: 1.0120 - val\_accuracy: 0.6667 - val\_loss: 0.8884

Epoch 21/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **10s** 82ms/step - accuracy: 0.7030 - loss: 0.8231 - val\_accuracy: 0.6961 - val\_loss: 0.8043

Epoch 22/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **10s** 85ms/step - accuracy: 0.6834 - loss: 0.9233 - val\_accuracy: 0.6569 - val\_loss: 1.0166

Epoch 23/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **10s** 96ms/step - accuracy: 0.7051 - loss: 0.8254 - val\_accuracy: 0.5000 - val\_loss: 1.1898

Epoch 24/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **10s** 82ms/step - accuracy: 0.7128 - loss: 0.7456 - val\_accuracy: 0.6569 - val\_loss: 0.9000

Epoch 25/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **9s** 82ms/step - accuracy: 0.6986 - loss: 0.8021 - val\_accuracy: 0.5490 - val\_loss: 1.2751

Epoch 26/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **9s** 84ms/step - accuracy: 0.6679 - loss: 0.9029 - val\_accuracy: 0.6961 - val\_loss: 0.9219

Epoch 27/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **10s** 85ms/step - accuracy: 0.7224 - loss: 0.7646 - val\_accuracy: 0.6569 - val\_loss: 0.8402

Epoch 28/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **10s** 88ms/step - accuracy: 0.7096 - loss: 0.7553 - val\_accuracy: 0.5980 - val\_loss: 0.8707

Epoch 29/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **10s** 85ms/step - accuracy: 0.7364 - loss: 0.7122 - val\_accuracy: 0.5588 - val\_loss: 1.0960

Epoch 30/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **10s** 88ms/step - accuracy: 0.7159 - loss: 0.7228 - val\_accuracy: 0.7059 - val\_loss: 0.8253

Epoch 31/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **10s** 86ms/step - accuracy: 0.7283 - loss: 0.6917 - val\_accuracy: 0.7255 - val\_loss: 0.6985

Epoch 32/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **9s** 82ms/step - accuracy: 0.7597 - loss: 0.6121 - val\_accuracy: 0.6569 - val\_loss: 0.9252

Epoch 33/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **10s** 88ms/step - accuracy: 0.7100 - loss: 0.7731 - val\_accuracy: 0.7353 - val\_loss: 0.6879

Epoch 34/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **9s** 84ms/step - accuracy: 0.7591 - loss: 0.6270 - val\_accuracy: 0.7647 - val\_loss: 0.6256

Epoch 35/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **9s** 82ms/step - accuracy: 0.7530 - loss: 0.6523 - val\_accuracy: 0.7255 - val\_loss: 0.7099

Epoch 36/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **9s** 85ms/step - accuracy: 0.7563 - loss: 0.6739 - val\_accuracy: 0.7059 - val\_loss: 0.7624

Epoch 37/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **10s** 85ms/step - accuracy: 0.7614 - loss: 0.6429 - val\_accuracy: 0.7647 - val\_loss: 0.6528

Epoch 38/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **9s** 84ms/step - accuracy: 0.8137 - loss: 0.4921 - val\_accuracy: 0.6667 - val\_loss: 0.9156

Epoch 39/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **11s** 109ms/step - accuracy: 0.6914 - loss: 0.8952 - val\_accuracy: 0.6373 - val\_loss: 0.9650

Epoch 40/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **10s** 84ms/step - accuracy: 0.7748 - loss: 0.6002 - val\_accuracy: 0.7843 - val\_loss: 0.5898

Epoch 41/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **10s** 93ms/step - accuracy: 0.7946 - loss: 0.5761 - val\_accuracy: 0.7157 - val\_loss: 0.6958

Epoch 42/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **12s** 94ms/step - accuracy: 0.7575 - loss: 0.6709 - val\_accuracy: 0.7549 - val\_loss: 0.6147

Epoch 43/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **10s** 90ms/step - accuracy: 0.7933 - loss: 0.6199 - val\_accuracy: 0.6471 - val\_loss: 1.0403

Epoch 44/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **10s** 88ms/step - accuracy: 0.7966 - loss: 0.5350 - val\_accuracy: 0.7647 - val\_loss: 0.6677

Epoch 45/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **10s** 86ms/step - accuracy: 0.7674 - loss: 0.6163 - val\_accuracy: 0.7745 - val\_loss: 0.5836

Epoch 46/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **13s** 124ms/step - accuracy: 0.8302 - loss: 0.4846 - val\_accuracy: 0.8431 - val\_loss: 0.5174

Epoch 47/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **11s** 99ms/step - accuracy: 0.7819 - loss: 0.5949 - val\_accuracy: 0.7941 - val\_loss: 0.6535

Epoch 48/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **15s** 102ms/step - accuracy: 0.7586 - loss: 0.6777 - val\_accuracy: 0.7353 - val\_loss: 0.7566

Epoch 49/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **11s** 92ms/step - accuracy: 0.8326 - loss: 0.4720 - val\_accuracy: 0.7941 - val\_loss: 0.6045

Epoch 50/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **10s** 89ms/step - accuracy: 0.8413 - loss: 0.4173 - val\_accuracy: 0.7353 - val\_loss: 0.7662

A graph with blue and orange lines

Description automatically generated

A graph of a graph

Description automatically generated