DAFTAR PUSTAKA

Akçay, S., Kundegorski, M. E., Devereux, M., & Breckon, T. P. (2016). *Transfer Learning Using Convolutional Neural Networks For Object*. IEEE International Conference on Image Processing (ICIP)., 1057-1061.

Ardinhtc. (2019). *Bunga Matahari Ciri - Ciri dan Habitatnya*. Retrieved June 18, 2020, from ekosistem: https://ekosistem.co.id/bunga-matahari/

Bibitbunga. (2015). *Arti dan Makna Bunga Tulip Berdasarkan Warna*. Retrieved June 18, 2020, from Bibitbunga: https://bibitbunga.com/arti-dan-makna-bunga-tulip-berdasarkan-warna/

Chen, W., Sun, W., & Wang, J. (2018). *A Novel AdaBoost and CNN Base for Vehicle Classification.* IEEE Access, 60445-60455.

Christian. (2015). *Mawar*. Retrieved June 18, 2020, from wikipedia: https://id.wikipedia.org/wiki/Mawar

Data, D. (2018). *Implementasi Deep Learning Sederhana Menggunakan Keras*. Retrieved June 17, 2020, from Medium: https://medium.com/@danau.data/implementasi-deep-learning-sederhana-menggunakan-keras-3f5726f007e7

Dutt, A., & Dutt, A. (2017). *Handwritten digit recognition using deep learning.* Intl. J. of Advanced Research in Computer Engineering & Technology, 990-997.

Florist, M. (2017). *Fakta Menarik Tentang Bunga Daisy*. Retrieved June 18, 2020, from Meme Florist: https://www.memeflorist.com/fakta-menarik-tentang-bunga-daisy/

González-Castejón, M., Visioli, F., & Rodriguez-Casado, A. (2012). *Diverse biological activities of dandelion*. International Life Sciences Institute, 534-547.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning.* London: MIT press.

S., Freifeld, O., Larsen, A. B., Fisher, J., & Hansen, L. (2016). *reaming more data: Class-dependent distributions over diffeomorphisms*

for learned data augmentation. Artificial Intelligence and Statistics, 342-350.

Imam. (2018). *Memahami Epoch Batch Size dan Iteration*. Retrieved July 27, 2020, from Imam Digmi: https://imam.digmi.id/post/memahami-epoch-batch-size-dan-iteration/

Jabbar, H., & Khan, R. Z. (2015). *Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study).* Computer Science, Communication and Instrumentation Devices, 163-172.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). *Gradient-based learning applied to document recognition.* Proceedings of the IEEE, 86(11), 2278-2324.

Louis, M. S., Azad, Z., Delshadtehrani, L., Gupta, S., Warden, P., Reddi, V. J., & Joshi, A. (2019). *Towards Deep Learning using TensorFlow Lite on RISC-V*. Proc. ACM CARRV, 1-6.

Ludwig, J. (2013). *Image Convolution.* portland: Portland State University. Retrieved June 17, 2020, from http://web.pdx.edu/~jduh/courses/Archive/geog481w07/Students/Ludwig_

Paradistia, E. R. (2019). *Pengenalan Convolutional Neural Network*. Retrieved August 4, 2020, from Medium: https://medium.com/@paradistia/pengenalan-convolutional-neural-network-e708b6d29838

Peltarion. (2018). *Categorical Classentropy*. Retrieved July 27, 2020, from Peltarion: https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy

Pertanian, M. (2019). *Pengertian Bunga, Manfaat, Jenis, dan Contohnya*. Retrieved June 18, 2020, from Dosen Pertanian: https://dosenpertanian.com/pengertian-bunga/

Prijono, B. (2018). *IndoMl. Retrieved from Convolutional Neural Network (CNN)* ıtroduction: https://indoml.com/2018/03/07/studen t-notes-convolutional-euralnetworks-cnn-introduction/.

Santosa, B., & Umam, A. (2018). *Data Mining dan Big Data Analytics.* Yogyakarta: Penebar Media Pustaka.

Sena, S. (2018). *Pengenalan Deep Learning Part 8 : Gender Classification using Pre-Trained Network (Transfer Learning)*. Retrieved July 18, 2020, from Medium: https://medium.com/@samuelsena/pengenalan-deep-learning-part-8-gender-classification-using-pre-trained-network-transfer-37ac910500d1

Setiawan, M. (2018). *Klasifikasi Penyakit pada citra daun menggunakan convolutional neural network.* Bogor: IPB.

Sofia, N. (2018). *Convolutional Neural Network*. Retrieved August 4, 2020, from Medium.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). *Dropout: a simple way to prevent neural networks from overfitting.* The journal of machine learning research, 15(1), 1929-1958.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., . . . Rabinovich, A. (2015). *Going deeper with convolutions.* Proceedings of the IEEE conference on computer vision and pattern recognition, 1-12.

Szegedy, C., Vanhoucke, V., Ioffe, S., & Shlens, J. (2016). *Rethinking the Inception Architecture for Computer Vision.* IEEE conference on Computer Vision and Pattern Recognition (CVPR)., 2818-2826.

Xia, X., & Xu, C. (2017). *Inception-v3 for Flower Classification.* International Conference on Image, Vision and Computing, 783-786.

Yegulalp, S. (2019). *What is TensorFlow? The machine learning library explained*. Retrieved June 18, 2020, from InfoWorld: https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html

LAMPIRAN

Lampiran Source Code

inception_v4.py

```
from tensorflow.keras.layers import Input, concatenate, Dropout, Dense, Flatt
en, Activation
from tensorflow.keras.layers import MaxPool2D, Conv2D, AveragePooling2D
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.models import Model

from tensorflow.keras import backend as K
from tensorflow.keras.utils import get_file

"""
Implementation of Inception Network v4 [Inception Network v4 Paper](http://ar
xiv.org/pdf/1602.07261v1.pdf) in Keras.
"""

URL = "https://github.com/titu1994/Inception-
v4/releases/download/v1.2/inception_v4_weights_tf_dim_ordering_tf_kernels.h5"


def conv_block(x, nb_filter, kernel_size, padding='same', strides=(1, 1), use
_bias=False):
    x = Conv2D(nb_filter, kernel_size, strides=strides, padding=padding, use_
bias=use_bias)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    return x


def inception_stem(input):

    # Input Shape is 299 x 299 x 3 (tf) or 3 x 299 x 299 (th)
    x = conv_block(input, 32, (3, 3), strides=(2, 2), padding='valid')
    x = conv_block(x, 32, (3, 3), padding='valid')
    x = conv_block(x, 64, (3, 3))

    x1 = MaxPool2D((3, 3), strides=(2, 2), padding='valid')(x)
    x2 = conv_block(x, 96, (3, 3), strides=(2, 2), padding='valid')

    concatenate([x1, x2])

    conv_block(x, 64, (1, 1))
```

```
    x1 = conv_block(x1, 96, (3, 3), padding='valid')

    x2 = conv_block(x, 64, (1, 1))
    x2 = conv_block(x2, 64, (1, 7))
    x2 = conv_block(x2, 64, (7, 1))
    x2 = conv_block(x2, 96, (3, 3), padding='valid')

    x = concatenate([x1, x2])

    x1 = conv_block(x, 192, (3, 3), strides=(2, 2), padding='valid')
    x2 = MaxPool2D((3, 3), strides=(2, 2), padding='valid')(x)

    x = concatenate([x1, x2])
    return x


def inception_A(input):

    a1 = conv_block(input, 96, (1, 1))

    a2 = conv_block(input, 64, (1, 1))
    a2 = conv_block(a2, 96, (3, 3))

    a3 = conv_block(input, 64, (1, 1))
    a3 = conv_block(a3, 96, (3, 3))
    a3 = conv_block(a3, 96, (3, 3))

    a4 = AveragePooling2D((3, 3), strides=(1, 1), padding='same')(input)
    a4 = conv_block(a4, 96, (1, 1))

    m = concatenate([a1, a2, a3, a4])
    return m


def inception_B(input):
    b1 = conv_block(input, 384, (1, 1))

    b2 = conv_block(input, 192, (1, 1))
    b2 = conv_block(b2, 224, (1, 7))
    b2 = conv_block(b2, 256, (7, 1))

    b3 = conv_block(input, 192, (1, 1))
       conv_block(b3, 192, (7, 1))
       conv_block(b3, 224, (1, 7))
       conv_block(b3, 224, (7, 1))
       conv_block(b3, 256, (1, 7))
```

```
    b4 = AveragePooling2D((3, 3), strides=(1, 1), padding='same')(input)
    b4 = conv_block(b4, 128, (1, 1))

    m = concatenate([b1, b2, b3, b4])
    return m


def inception_C(input):
    c1 = conv_block(input, 256, (1, 1))

    c2 = conv_block(input, 384, (1, 1))
    c2_1 = conv_block(c2, 256, (1, 3))
    c2_2 = conv_block(c2, 256, (3, 1))
    c2 = concatenate([c2_1, c2_2])

    c3 = conv_block(input, 384, (1, 1))
    c3 = conv_block(c3, 448, (3, 1))
    c3 = conv_block(c3, 512, (1, 3))
    c3_1 = conv_block(c3, 256, (1, 3))
    c3_2 = conv_block(c3, 256, (3, 1))
    c3 = concatenate([c3_1, c3_2])

    c4 = AveragePooling2D((3, 3), strides=(1, 1), padding='same')(input)
    c4 = conv_block(c4, 256, (1, 1))

    m = concatenate([c1, c2, c3, c4])
    return m


def reduction_A(input):
    r1 = conv_block(input, 384, (3, 3), strides=(2, 2), padding='valid')

    r2 = conv_block(input, 192, (1, 1))
    r2 = conv_block(r2, 224, (3, 3))
    r2 = conv_block(r2, 256, (3, 3), strides=(2, 2), padding='valid')

    r3 = MaxPool2D((3, 3), strides=(2, 2), padding='valid')(input)

    m = concatenate([r1, r2, r3])
    return m


    ction_B(input):
    .image_data_format() == "th":
    channel_axis = 1
```

```
    else:
        channel_axis = -1

    r1 = conv_block(input, 192, (1, 1))
    r1 = conv_block(r1, 192, (3, 3), strides=(2, 2), padding='valid')

    r2 = conv_block(input, 256, (1, 1))
    r2 = conv_block(r2, 256, (1, 7))
    r2 = conv_block(r2, 320, (7, 1))
    r2 = conv_block(r2, 320, (3, 3), strides=(2, 2), padding='valid')

    r3 = MaxPool2D((3, 3), strides=(2, 2), padding='valid')(input)

    m = concatenate([r1, r2, r3])
    return m


def create_inception_v4(nb_classes=1001, load_weights=True):
    '''
    Creates a inception v4 network

    :param nb_classes: number of classes.txt
    :return: Keras Model with 1 input and 1 output
    '''

    init = Input((299, 299, 3))

    # Input Shape is 299 x 299 x 3 (tf) or 3 x 299 x 299 (th)
    x = inception_stem(init)

    # 4 x Inception A
    for i in range(4):
        x = inception_A(x)

    # Reduction A
    x = reduction_A(x)

    # 7 x Inception B
    for i in range(7):
        x = inception_B(x)

    # Reduction B
        reduction_B(x)

        x Inception C
        i in range(3):
```

```
        x = inception_C(x)

    # Average Pooling
    x = AveragePooling2D((8, 8))(x)

    # Dropout
    x = Dropout(0.2)(x)
    x = Flatten()(x)

    ## Output
    out = Dense(units=nb_classes, activation='softmax')(x)

    model = Model(init, out, name='Inception-v4')

    if load_weights:
        weights = get_file('inception_v4_weights_tf_dim_ordering_tf_kernels.h
5', URL, cache_subdir='models2')

        #model.load_weights(weights)
        print("Model weights loaded.")

    return model


if __name__ == "__main__":
    inception_v4 = create_inception_v4(load_weights=True)
```

main.ipynb

In [1]:

```
import tensorflow as tf
import sys
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import os
import openpyxl
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout, I
nput
sorflow.keras.preprocessing.image import ImageDataGenerator
sorflow.keras.optimizers import RMSprop, SGD, Adam
```

```
from PIL import Image
sys.modules['Image'] = Image

from inception_v4 import create_inception_v4
```

In [2]:

```
BATCH_SIZE=64

datagen = ImageDataGenerator(rescale = 1./255.,
    #featurewise_center=True,
    #featurewise_std_normalization=True,
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.2,
    #vertical_flip=True,
    horizontal_flip=True,
    #preprocessing_function=preprocess_input,
validation_split=0.3)

#testgen = ImageDataGenerator(rescale = 1./255., validation_split=0.3)

train_it = datagen.flow_from_directory(directory='flower_photos/', shuffle=Tr
ue, class_mode="categorical", batch_size=BATCH_SIZE, target_size=(299, 299),
subset='training')
val_it = datagen.flow_from_directory(directory='flower_photos/', shuffle=True
, class_mode="categorical", batch_size=BATCH_SIZE, target_size=(299, 299), su
bset='validation')
```

Out [2]:

```
Found 2917 images belonging to 5 classes.
Found 1247 images belonging to 5 classes.
```

In [3]:

```
base_model = create_inception_v4()
base_model.summary()
```

Out [3]:

```
Output was trimmed for performance reasons.
To see the full output set the setting "python.dataScience.textOutputLimit"
to 0.
…
____________
concatenate_23 (Concatenate)    (None, 8, 8, 512)    0
activation_146[0][0]

activation_147[0][0]
__________________________________________________________________________________
______________________
activation_148 (Activation)     (None, 8, 8, 256)    0
batch_normalization_148[0][0]
__________________________________________________________________________________
______________________
concatenate_24 (Concatenate)    (None, 8, 8, 1536)   0
activation_139[0][0]

concatenate_22[0][0]

concatenate_23[0][0]

activation_148[0][0]
__________________________________________________________________________________
______________________
average_pooling2d_14 (AveragePo (None, 1, 1, 1536)   0
concatenate_24[0][0]
__________________________________________________________________________________
______________________
dropout (Dropout)               (None, 1, 1, 1536)   0
average_pooling2d_14[0][0]
__________________________________________________________________________________
______________________
flatten (Flatten)               (None, 1536)         0
dropout[0][0]
__________________________________________________________________________________
______________________
dense (Dense)                   (None, 1001)         1538537
flatten[0][0]
==================================================================================
====================
Total params: 42,744,521
Trainable params: 42,681,353
Non-trainable params: 63,168
__________________________________________________________________________________
____________________
```

In [4]:

```
# Remove top layer
model = Model(base_model.input, base_model.layers[-2].output)
model.summary()
```

Out [4]:

```
Output was trimmed for performance reasons. To see the full output set the
setting "python.dataScience.textOutputLimit" to 0. ...

concatenate_23 (Concatenate)    (None, 8, 8, 512)    0
activation_146[0][0]

activation_147[0][0]
__________________________________________________________________________________________________
activation_148 (Activation)     (None, 8, 8, 256)    0
batch_normalization_148[0][0]
__________________________________________________________________________________________________
concatenate_24 (Concatenate)    (None, 8, 8, 1536)   0
activation_139[0][0]

concatenate_22[0][0]

concatenate_23[0][0]

activation_148[0][0]
__________________________________________________________________________________________________
average_pooling2d_14 (AveragePo (None, 1, 1, 1536)   0
concatenate_24[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 1, 1, 1536)   0
average_pooling2d_14[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 1536)         0
dropout[0][0]
==================================================================================================
Total params: 41,205,984
Trainable params: 41,142,816
Non-trainable params: 63,168
__________________________________________________________________________________________________
```

In [5]:

```
model.load_weights('someweight')
```

Out [5]:

```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at
0x2276c53fd48>
```

In [6]:

```
r in model.layers:
r.trainable = False
```

In [7]:

```
x = model.output
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
#x = Dense(128, activation='relu')(x)
#x = Dropout(0.4)(x)
x = Dense(64, activation='relu')(x)
x = Dropout(0.2)(x)
#x = Dropout(0.5)(x)
predictions = Dense(5, activation='softmax')(x)

model = Model(inputs=model.input, outputs=predictions)
```

In [8]:

```
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossent
ropy', metrics=['accuracy'])
```

In [9]:

```
checkpoint_path = "checkpoints/cp"
checkpoint_dir = os.path.dirname(checkpoint_path)

# callback
cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                                                 save_weights_only=True,
                                                 save_best_only=True,
                                                 verbose=1)
```

In [10]:

```
EPOCHS = 10
# fit model
history1 = model.fit(train_it, steps_per_epoch=train_it.samples//BATCH_SIZE,
validation_data=val_it, validation_steps=val_it.samples//BATCH_SIZE, epochs=E
POCHS, callbacks=[cp_callback])
```

Out [10]:

```
Output was trimmed for performance reasons. To see the full output set the
setting "python.dataScience.textOutputLimit" to 0. ...

10
============================>.] - ETA: 1s - loss: 0.5062 - accuracy:

005: val_loss improved from 0.50099 to 0.45399, saving model to
nts/cp
```

```
45/45 [==============================] - 84s 2s/step - loss: 0.5026 -
accuracy: 0.8181 - val_loss: 0.4540 - val_accuracy: 0.8438
Epoch 6/10
44/45 [============================>.] - ETA: 1s - loss: 0.4764 - accuracy:
0.8269
Epoch 00006: val_loss did not improve from 0.45399
45/45 [==============================] - 82s 2s/step - loss: 0.4773 -
accuracy: 0.8276 - val_loss: 0.4699 - val_accuracy: 0.8207
Epoch 7/10
44/45 [============================>.] - ETA: 1s - loss: 0.4680 - accuracy:
0.8305
Epoch 00007: val_loss improved from 0.45399 to 0.44528, saving model to
checkpoints/cp
45/45 [==============================] - 84s 2s/step - loss: 0.4685 -
accuracy: 0.8290 - val_loss: 0.4453 - val_accuracy: 0.8322
Epoch 8/10
44/45 [============================>.] - ETA: 1s - loss: 0.4411 - accuracy:
0.8427
Epoch 00008: val_loss improved from 0.44528 to 0.43607, saving model to
checkpoints/cp
45/45 [==============================] - 84s 2s/step - loss: 0.4425 -
accuracy: 0.8423 - val_loss: 0.4361 - val_accuracy: 0.8388
Epoch 9/10
44/45 [============================>.] - ETA: 1s - loss: 0.4375 - accuracy:
0.8391
Epoch 00009: val_loss improved from 0.43607 to 0.41541, saving model to
checkpoints/cp
45/45 [==============================] - 83s 2s/step - loss: 0.4372 -
accuracy: 0.8399 - val_loss: 0.4154 - val_accuracy: 0.8495
Epoch 10/10
44/45 [============================>.] - ETA: 1s - loss: 0.4426 - accuracy:
0.8412
Epoch 00010: val_loss did not improve from 0.41541
45/45 [==============================] - 81s 2s/step - loss: 0.4445 -
accuracy: 0.8406 - val_loss: 0.4479 - val_accuracy: 0.8380
```

In [11]:

```
history = pd.read_excel('not_fine_tuning.xlsx')
history
```

Out [11]:

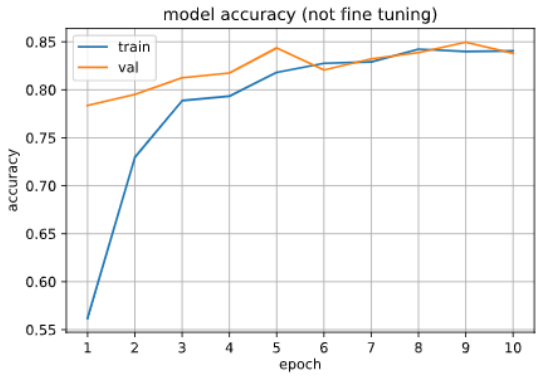
Unnamed: 0	training accuracy	training loss	validation accuracy	validation loss
0	0.561668	1.086226	0.783717	0.621401
1	0.729502	0.710563	0.795230	0.538456
2	0.788718	0.584898	0.812500	0.500990
3	0.793273	0.555414	0.817434	0.507207
4	0.818150	0.500701	0.843750	0.453990
5	0.827610	0.475961	0.820724	0.469853
6	0.829012	0.469172	0.832237	0.445283
7	0.842327	0.443061	0.838816	0.436067
8	0.839874	0.437100	0.849507	0.415413
9	0.840575	0.443652	0.837993	0.447904

In [12]:

```
# accuracy plot
plt.plot(range(1, EPOCHS+1), history['training accuracy'])
plt.plot(range(1, EPOCHS+1), history['validation accuracy'])

plt.title('model accuracy (not fine tuning)')
plt.ylabel('accuracy')
plt.xlabel('epoch', labelpad=2)
plt.xticks(np.arange(1, EPOCHS+1))
plt.legend(['train', 'val'], loc='upper left')
plt.grid(True)
plt.show()
```

Out [12]:

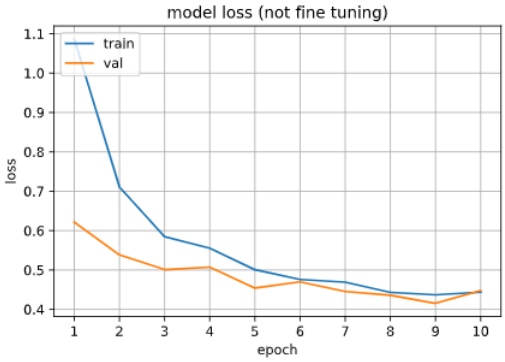


main.ipynb (7 dari 15 halaman)

In [13]:

```
#loss plot
plt.plot(range(1, EPOCHS+1), history['training loss'])
plt.plot(range(1, EPOCHS+1), history['validation loss'])
plt.title('model loss (not fine tuning)')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.xticks(np.arange(1, EPOCHS+1))
plt.legend(['train', 'val'], loc='upper left')
plt.grid(True)
plt.show()
```

Out [13]:



In [14]:

```
for i, layer in enumerate(model.layers):
    print(i, layer.name)
```

main.ipynb (8 dari 15 halaman)

Out [14]:

```
457 conv2d_143
458 batch_normalization_143
459 activation_143
460 conv2d_144
461 batch_normalization_144
462 activation_144
463 conv2d_140
464 conv2d_145
465 batch_normalization_140
466 batch_normalization_145
467 activation_140
468 activation_145
469 conv2d_141
470 conv2d_142
471 conv2d_146
472 conv2d_147
473 average_pooling2d_13
474 conv2d_139
475 batch_normalization_141
476 batch_normalization_142
477 batch_normalization_146
478 batch_normalization_147
479 conv2d_148
480 batch_normalization_139
481 activation_141
482 activation_142
483 activation_146
484 activation_147
485 batch_normalization_148
486 activation_139
487 concatenate_22
        atenate_23
        vation_148
        atenate_24
        age_pooling2d_14
        out
```

```
493 flatten
494 dense_1
495 dropout_1
496 dense_2
497 dropout_2
498 dense_3
```

In [15]:

```
# unfreeze layer 457-498 (3 Inception C + Fully Connected Layer)
for layer in model.layers[:457]:
   layer.trainable = False
for layer in model.layers[457:]:
   layer.trainable = True
```

In [16]:

```
model.compile(optimizer=Adam(learning_rate=0.00147), loss='categorical_crosse
ntropy', metrics=['accuracy'])
```

In [17]:

```
model.summary()
```

main.ipynb (10 dari 15 halaman)

Out [17]:

```
Output was trimmed for performance reasons. To see the full output set the
setting "python.dataScience.textOutputLimit" to 0. ...

concatenate_23 (Concatenate)    (None, 8, 8, 512)    0
activation_146[0][0]

activation_147[0][0]
__________________________________________________________________________
_______________________
activation_148 (Activation)     (None, 8, 8, 256)    0
batch_normalization_148[0][0]
__________________________________________________________________________
_______________________
concatenate_24 (Concatenate)    (None, 8, 8, 1536)   0
activation_139[0][0]

concatenate_22[0][0]

concatenate_23[0][0]

activation_148[0][0]
__________________________________________________________________________
_____________
pooling2d_14 (AveragePo (None, 1, 1, 1536)   0
ate_24[0][0]
__________________________________________________________________________
_____________
```

```
dropout (Dropout)               (None, 1, 1, 1536)   0
average_pooling2d_14[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 1536)         0
dropout[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 256)          393472
flatten[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 256)          0
dense_1[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 64)           16448
dropout_1[0][0]
__________________________________________________________________________________________________
dropout_2 (Dropout)             (None, 64)           0
dense_2[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 5)            325
dropout_2[0][0]
==================================================================================================
Total params: 41,616,229
Trainable params: 4,963,333
Non-trainable params: 36,652,896
__________________________________________________________________________________________________
```

In [18]:

```
model.load_weights('checkpoints/cp')
```

Out [18]:

```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x1b770bb80
48>
```

In [19]:

```
EPOCHS=20
# fit model AGAIN
history2 = model.fit(train_it, steps_per_epoch=train_it.samples//BATCH_SIZE,
validation_data=val_it, validation_steps=val_it.samples//BATCH_SIZE, epochs=E
POCHS, callbacks=[cp_callback])
```

Out [19]:

```
Output was trimmed for performance reasons. To see the full output set the
setting "python.dataScience.textOutputLimit" to 0. ...

Epoch 15/20
44/45 [============================>.] - ETA: 1s - loss: 0.0416 - accuracy:
0.9867
Epoch 00015: val_loss did not improve from 0.41541
45/45 [==============================] - 83s 2s/step - loss: 0.0423 -
accuracy: 0.9863 - val_loss: 0.5802 - val_accuracy: 0.8816
Epoch 16/20
44/45 [============================>.] - ETA: 1s - loss: 0.0411 - accuracy:
0.9882
Epoch 00016: val_loss did not improve from 0.41541
45/45 [==============================] - 83s 2s/step - loss: 0.0412 -
accuracy: 0.9877 - val_loss: 0.5506 - val_accuracy: 0.8816
Epoch 17/20
44/45 [============================>.] - ETA: 1s - loss: 0.0435 - accuracy:
0.9846
Epoch 00017: val_loss did not improve from 0.41541
45/45 [==============================] - 83s 2s/step - loss: 0.0449 -
accuracy: 0.9846 - val_loss: 0.5991 - val_accuracy: 0.8775
Epoch 18/20
44/45 [============================>.] - ETA: 1s - loss: 0.0437 - accuracy:
0.9853
Epoch 00018: val_loss did not improve from 0.41541
45/45 [==============================] - 82s 2s/step - loss: 0.0445 -
accuracy: 0.9853 - val_loss: 0.6229 - val_accuracy: 0.8676
Epoch 19/20
44/45 [============================>.] - ETA: 1s - loss: 0.0368 - accuracy:
0.9885
Epoch 00019: val_loss did not improve from 0.41541
45/45 [==============================] - 81s 2s/step - loss: 0.0361 -
accuracy: 0.9888 - val_loss: 0.6348 - val_accuracy: 0.8750
Epoch 20/20
44/45 [============================>.] - ETA: 1s - loss: 0.0324 - accuracy:
0.9907
Epoch 00020: val_loss did not improve from 0.41541
45/45 [==============================] - 82s 2s/step - loss: 0.0324 -
accuracy: 0.9905 - val_loss: 0.6703 - val_accuracy: 0.8676
```

In [20]:

```
history2 = pd.read_excel('fine_tuning.xlsx')
history2
```

Out [20]:

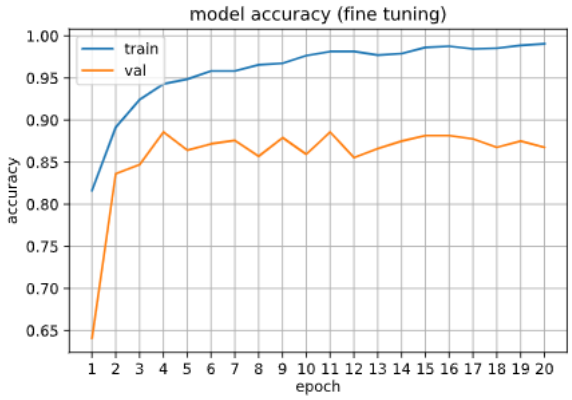
Unnamed: 0	training accuracy	training loss	validation accuracy	validation loss
0	0	0.816398	0.549879	0.641447	2.323072
1	1	0.891731	0.320113	0.836349	0.538896
2	2	0.924317	0.230961	0.847039	0.547969
3	3	0.942887	0.175494	0.885691	0.471739
4	4	0.948493	0.162466	0.864309	0.482623
5	5	0.958304	0.136492	0.871711	0.434055
6	6	0.958304	0.126732	0.875822	0.461649
7	7	0.965662	0.100432	0.856908	0.564352
8	8	0.967414	0.107437	0.879112	0.443001
9	9	0.976524	0.074279	0.859375	0.567514
10	10	0.981430	0.060471	0.885691	0.496638
11	11	0.981430	0.057241	0.855263	0.621717
12	12	0.977225	0.068291	0.865954	0.603964
13	13	0.978977	0.066183	0.875000	0.502842
14	14	0.986335	0.042391	0.881579	0.580151
15	15	0.987737	0.041421	0.881579	0.550600
16	16	0.984583	0.045319	0.877467	0.599068
17	17	0.985284	0.044807	0.867599	0.622883
18	18	0.988788	0.036025	0.875000	0.634792
19	19	0.990540	0.032682	0.867599	0.670286

In [21]:

```
# accuracy plot
plt.plot(range(1, EPOCHS+1), history2['training accuracy'])
plt.plot(range(1, EPOCHS+1), history2['validation accuracy'])

plt.title('model accuracy (fine tuning)')
plt.ylabel('accuracy')
plt.xlabel('epoch', labelpad=2)
plt.xticks(np.arange(1, EPOCHS+1))
plt.legend(['train', 'val'], loc='upper left')
plt.grid(True)
plt.show()
```

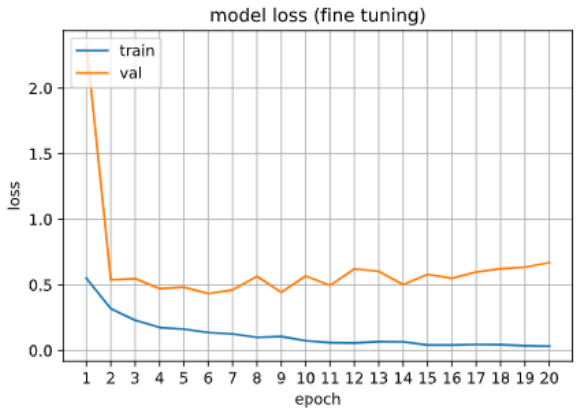
Out [21]:



In [22]:

```
#loss plot
plt.plot(range(1, EPOCHS+1), history2['training loss'])
plt.plot(range(1, EPOCHS+1), history2['validation loss'])
plt.title('model loss (fine tuning)')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.xticks(np.arange(1, EPOCHS+1))
plt.legend(['train', 'val'], loc='upper left')
plt.grid(True)
plt.show()
```

Out [22]:



In [23]:

```
model.save_weights('./checkpoints/finetune')
```

In [24]:

```
pd.DataFrame(np.array([history1.history['accuracy'], history1.history['loss']
, history1.history['val_accuracy'], history1.history['val_loss']]).T , column
s=['training accuracy', 'training loss', 'validation accuracy', 'validation l
oss']).to_excel('not_fine_tuning.xlsx')
```

In [25]:

```
pd.DataFrame(np.array([history2.history['accuracy'], history2.history['loss']
, history2.history['val_accuracy'], history2.history['val_loss']]).T , column
s=['training accuracy', 'training loss', 'validation accuracy', 'validation l
oss']).to_excel('fine_tuning.xlsx')
```

Android

https://github.com/ainunmardiyach/skripsi