ASSIGNMENT3 SVM_1301213434_Ichwan Rizky Wahyudin

April 5, 2024

1 [ASSIGNMENT-3] Klasifikasi cats vs dogs menggunakan supervised learning

Nama : Ichwan Rizky Wahyudin
NIM : 1301213434
Kelas : IF-45-09
Metode : Support Vector Machine(SVM)
Link Source Code : https://github.com/irizkyw/ComputerVision/tree/ASSIGNMENTS/W5_Assignment3

```
[ ]: import os
     import pandas as pd
     import numpy as np
     import tensorflow as tf
     import matplotlib.pyplot as plt
```

1.0.1 Image Augmentation

Penambahan noise pada data training agar tidak terjadinya overfitting

```
[ ]: from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
[ ]: def add_noise(image, mean=0, var=0.1):
         """
         Adds Gaussian noise to an image.

         Args:
             image (numpy.ndarray): Input image.
             mean (float, optional): Mean of the Gaussian distribution (default is␣
      ↪0).
             var (float, optional): Variance of the Gaussian distribution (default␣
      ↪is 0.1).

         Returns:
             numpy.ndarray: Noisy image.
         """
         row, col, ch = image.shape
         sigma = var ** 0.5
         gauss = np.random.normal(mean, sigma, (row, col, ch))
```

```
    noisy_image = image + gauss
    noisy_image = np.clip(noisy_image, 0, 255)
    return noisy_image.astype(np.float32)
```

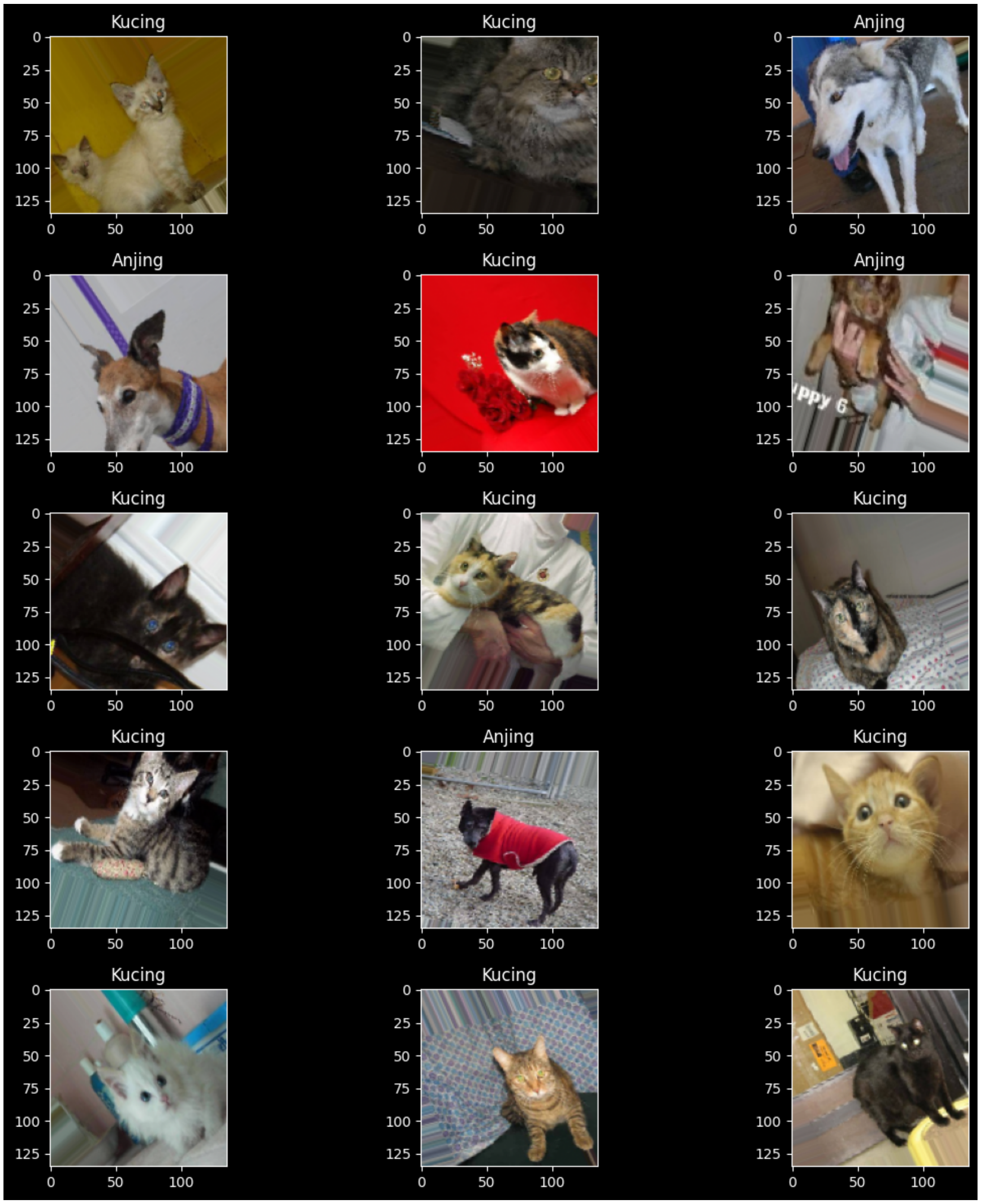
[ ]:
```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    preprocessing_function=add_noise
)

validation_datagen = ImageDataGenerator(rescale=1./255)
```

[ ]:
```
image_size = (135, 135)

train_generator = train_datagen.flow_from_directory(
    'data/train',
    target_size=(image_size[0], image_size[1]),
    batch_size=32,
    class_mode='binary'
)

valid_generator = validation_datagen.flow_from_directory(
    'data/validation',
    target_size=(image_size[0], image_size[1]),
    batch_size=32,
    class_mode='binary'
)
```

```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
```

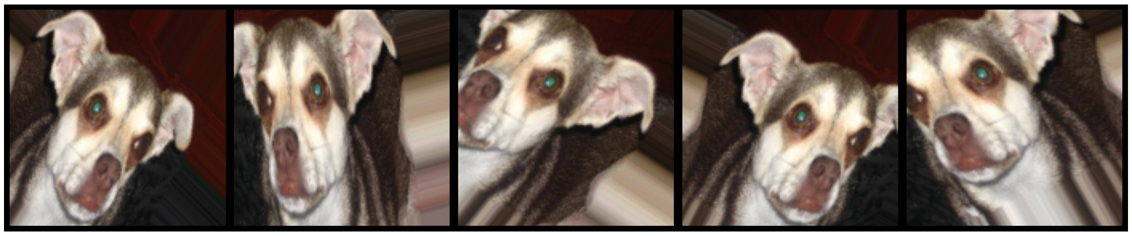
[ ]:
```
plt.figure(figsize=(12, 12))
for i in range(0, 15):
    plt.subplot(5, 3, i+1)
    for X_batch, Y_batch in train_generator:
        image = X_batch[0]
        plt.imshow(image)
        if Y_batch[0] == 0:
            plt.title('Kucing')
        else:
            plt.title('Anjing')
        break
```

```
plt.tight_layout()
plt.show()
```



```
[ ]: def plotImages(images_arr):
         fig, axes = plt.subplots(1, 5, figsize=(20, 20))
```

```
    axes = axes.flatten()
    for img, ax in zip(images_arr, axes):
        ax.imshow(img)
        ax.axis('off')
    plt.tight_layout()
    plt.show()

augmented_images = [train_generator[0][0][0] for i in range(5)]
plotImages(augmented_images)
```



1.0.2 Model SVM

[ ]:
```
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.regularizers import l2
```

[ ]:
```
model_cnn = Sequential([
    tf.keras.layers.Conv2D(32,padding="same",kernel_size=3, activation='relu',␣
 ↪strides=2, input_shape=[image_size[0], image_size[1], 3]),
    tf.keras.layers.MaxPool2D(pool_size=2, strides=2),

    tf.keras.layers.Conv2D(64,padding="same",kernel_size=3, activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=2, strides=2),

    tf.keras.layers.Conv2D(128,padding="same",kernel_size=3, activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=2, strides=2),

    tf.keras.layers.Flatten(),

    tf.keras.layers.Dense(units=128, activation='relu'),
    tf.keras.layers.Dense(units=1, kernel_regularizer=l2(0.01),␣
 ↪activation='linear')
])

model_cnn.summary()
```

```
Model: "sequential_5"
```

```
 Layer (type)                    Output Shape                  Param #

 conv2d_11 (Conv2D)              (None, 68, 68, 32)                896

 max_pooling2d_11 (MaxPooling2D)  (None, 34, 34, 32)                  0

 conv2d_12 (Conv2D)              (None, 34, 34, 64)             18,496

 max_pooling2d_12 (MaxPooling2D)  (None, 17, 17, 64)                  0

 conv2d_13 (Conv2D)              (None, 17, 17, 128)            73,856

 max_pooling2d_13 (MaxPooling2D)  (None, 8, 8, 128)                   0

 flatten_5 (Flatten)             (None, 8192)                         0

 dense_10 (Dense)                (None, 128)                  1,048,704

 dense_11 (Dense)                (None, 1)                          129


 Total params: 1,142,081 (4.36 MB)

 Trainable params: 1,142,081 (4.36 MB)

 Non-trainable params: 0 (0.00 B)
```

```
[ ]: model_cnn.compile(optimizer='adam', loss='hinge', metrics=['accuracy'])
```

```
[ ]: history = model_cnn.fit(train_generator, validation_data=valid_generator,␣
      ↪epochs=100)
```

```
Epoch 1/100

d:\anaconda3\envs\yolov8\lib\site-
packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:120:
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()

63/63                 23s 310ms/step -
accuracy: 0.5215 - loss: 1.0225 - val_accuracy: 0.5920 - val_loss: 0.9719
Epoch 2/100
63/63                 21s 313ms/step -
```

```
accuracy: 0.5382 - loss: 0.9806 - val_accuracy: 0.6450 - val_loss: 0.8624
Epoch 3/100
63/63            22s 323ms/step -
accuracy: 0.5555 - loss: 0.9194 - val_accuracy: 0.5060 - val_loss: 0.9079
Epoch 4/100
63/63            21s 304ms/step -
accuracy: 0.5447 - loss: 0.8996 - val_accuracy: 0.5560 - val_loss: 0.9444
Epoch 5/100
63/63            21s 306ms/step -
accuracy: 0.6121 - loss: 0.8498 - val_accuracy: 0.6120 - val_loss: 0.7381
Epoch 6/100
63/63            21s 307ms/step -
accuracy: 0.6301 - loss: 0.8006 - val_accuracy: 0.6390 - val_loss: 0.7689
Epoch 7/100
63/63            22s 330ms/step -
accuracy: 0.6019 - loss: 0.8017 - val_accuracy: 0.6830 - val_loss: 0.7374
Epoch 8/100
63/63            26s 371ms/step -
accuracy: 0.6484 - loss: 0.7569 - val_accuracy: 0.6950 - val_loss: 0.6678
Epoch 9/100
63/63            23s 334ms/step -
accuracy: 0.6843 - loss: 0.6799 - val_accuracy: 0.6970 - val_loss: 0.6680
Epoch 10/100
63/63            28s 414ms/step -
accuracy: 0.6902 - loss: 0.6626 - val_accuracy: 0.6890 - val_loss: 0.7175
Epoch 11/100
63/63            33s 498ms/step -
accuracy: 0.6835 - loss: 0.6785 - val_accuracy: 0.6810 - val_loss: 0.6868
Epoch 12/100
63/63            29s 429ms/step -
accuracy: 0.6783 - loss: 0.6759 - val_accuracy: 0.7000 - val_loss: 0.6422
Epoch 13/100
63/63            25s 375ms/step -
accuracy: 0.6955 - loss: 0.6536 - val_accuracy: 0.6910 - val_loss: 0.6499
Epoch 14/100
63/63            25s 372ms/step -
accuracy: 0.6959 - loss: 0.6278 - val_accuracy: 0.7250 - val_loss: 0.6476
Epoch 15/100
63/63            26s 386ms/step -
accuracy: 0.6936 - loss: 0.6496 - val_accuracy: 0.7190 - val_loss: 0.5972
Epoch 16/100
63/63            24s 352ms/step -
accuracy: 0.7122 - loss: 0.6181 - val_accuracy: 0.7030 - val_loss: 0.6062
Epoch 17/100
63/63            24s 353ms/step -
accuracy: 0.7055 - loss: 0.5934 - val_accuracy: 0.6990 - val_loss: 0.6205
Epoch 18/100
63/63            27s 393ms/step -
```

```
accuracy: 0.7099 - loss: 0.6205 - val_accuracy: 0.7360 - val_loss: 0.6050
Epoch 19/100
63/63            29s 415ms/step -
accuracy: 0.7349 - loss: 0.5707 - val_accuracy: 0.6560 - val_loss: 0.7213
Epoch 20/100
63/63            28s 415ms/step -
accuracy: 0.6967 - loss: 0.6099 - val_accuracy: 0.7460 - val_loss: 0.5982
Epoch 21/100
63/63            28s 415ms/step -
accuracy: 0.7286 - loss: 0.5569 - val_accuracy: 0.7440 - val_loss: 0.6232
Epoch 22/100
63/63            29s 416ms/step -
accuracy: 0.7427 - loss: 0.5705 - val_accuracy: 0.7380 - val_loss: 0.6196
Epoch 23/100
63/63            29s 419ms/step -
accuracy: 0.7480 - loss: 0.5631 - val_accuracy: 0.7710 - val_loss: 0.5936
Epoch 24/100
63/63            28s 409ms/step -
accuracy: 0.7511 - loss: 0.5340 - val_accuracy: 0.7540 - val_loss: 0.6105
Epoch 25/100
63/63            26s 381ms/step -
accuracy: 0.7450 - loss: 0.5545 - val_accuracy: 0.7430 - val_loss: 0.5571
Epoch 26/100
63/63            22s 323ms/step -
accuracy: 0.7551 - loss: 0.5352 - val_accuracy: 0.7680 - val_loss: 0.5475
Epoch 27/100
63/63            22s 322ms/step -
accuracy: 0.7407 - loss: 0.5720 - val_accuracy: 0.7650 - val_loss: 0.6155
Epoch 28/100
63/63            22s 316ms/step -
accuracy: 0.7595 - loss: 0.5208 - val_accuracy: 0.7670 - val_loss: 0.5528
Epoch 29/100
63/63            22s 314ms/step -
accuracy: 0.7599 - loss: 0.5280 - val_accuracy: 0.7690 - val_loss: 0.5074
Epoch 30/100
63/63            22s 318ms/step -
accuracy: 0.7540 - loss: 0.5051 - val_accuracy: 0.7760 - val_loss: 0.5306
Epoch 31/100
63/63            25s 364ms/step -
accuracy: 0.7588 - loss: 0.5210 - val_accuracy: 0.7630 - val_loss: 0.5703
Epoch 32/100
63/63            22s 322ms/step -
accuracy: 0.7772 - loss: 0.4856 - val_accuracy: 0.7790 - val_loss: 0.5129
Epoch 33/100
63/63            21s 312ms/step -
accuracy: 0.7795 - loss: 0.4854 - val_accuracy: 0.7790 - val_loss: 0.5097
Epoch 34/100
63/63            22s 316ms/step -
```

```
accuracy: 0.7689 - loss: 0.5136 - val_accuracy: 0.7350 - val_loss: 0.5383
Epoch 35/100
63/63             22s 315ms/step -
accuracy: 0.7823 - loss: 0.4699 - val_accuracy: 0.7690 - val_loss: 0.5330
Epoch 36/100
63/63             22s 313ms/step -
accuracy: 0.7796 - loss: 0.4779 - val_accuracy: 0.7900 - val_loss: 0.5205
Epoch 37/100
63/63             21s 313ms/step -
accuracy: 0.7495 - loss: 0.5372 - val_accuracy: 0.7830 - val_loss: 0.5175
Epoch 38/100
63/63             23s 329ms/step -
accuracy: 0.7645 - loss: 0.4991 - val_accuracy: 0.7890 - val_loss: 0.4757
Epoch 39/100
63/63             21s 311ms/step -
accuracy: 0.7875 - loss: 0.4521 - val_accuracy: 0.7870 - val_loss: 0.4815
Epoch 40/100
63/63             22s 319ms/step -
accuracy: 0.7631 - loss: 0.5071 - val_accuracy: 0.7940 - val_loss: 0.5593
Epoch 41/100
63/63             21s 313ms/step -
accuracy: 0.7894 - loss: 0.4488 - val_accuracy: 0.7870 - val_loss: 0.5246
Epoch 42/100
63/63             21s 309ms/step -
accuracy: 0.7732 - loss: 0.5042 - val_accuracy: 0.7950 - val_loss: 0.4846
Epoch 43/100
63/63             21s 309ms/step -
accuracy: 0.7910 - loss: 0.4276 - val_accuracy: 0.7830 - val_loss: 0.4671
Epoch 44/100
63/63             21s 312ms/step -
accuracy: 0.7949 - loss: 0.4581 - val_accuracy: 0.7800 - val_loss: 0.5272
Epoch 45/100
63/63             21s 312ms/step -
accuracy: 0.7896 - loss: 0.4664 - val_accuracy: 0.7500 - val_loss: 0.5120
Epoch 46/100
63/63             21s 310ms/step -
accuracy: 0.7683 - loss: 0.4842 - val_accuracy: 0.7790 - val_loss: 0.4778
Epoch 47/100
63/63             21s 311ms/step -
accuracy: 0.8012 - loss: 0.4382 - val_accuracy: 0.7730 - val_loss: 0.5042
Epoch 48/100
63/63             23s 342ms/step -
accuracy: 0.7999 - loss: 0.4317 - val_accuracy: 0.7870 - val_loss: 0.5398
Epoch 49/100
63/63             25s 362ms/step -
accuracy: 0.8036 - loss: 0.4265 - val_accuracy: 0.7970 - val_loss: 0.4714
Epoch 50/100
63/63             24s 354ms/step -
```

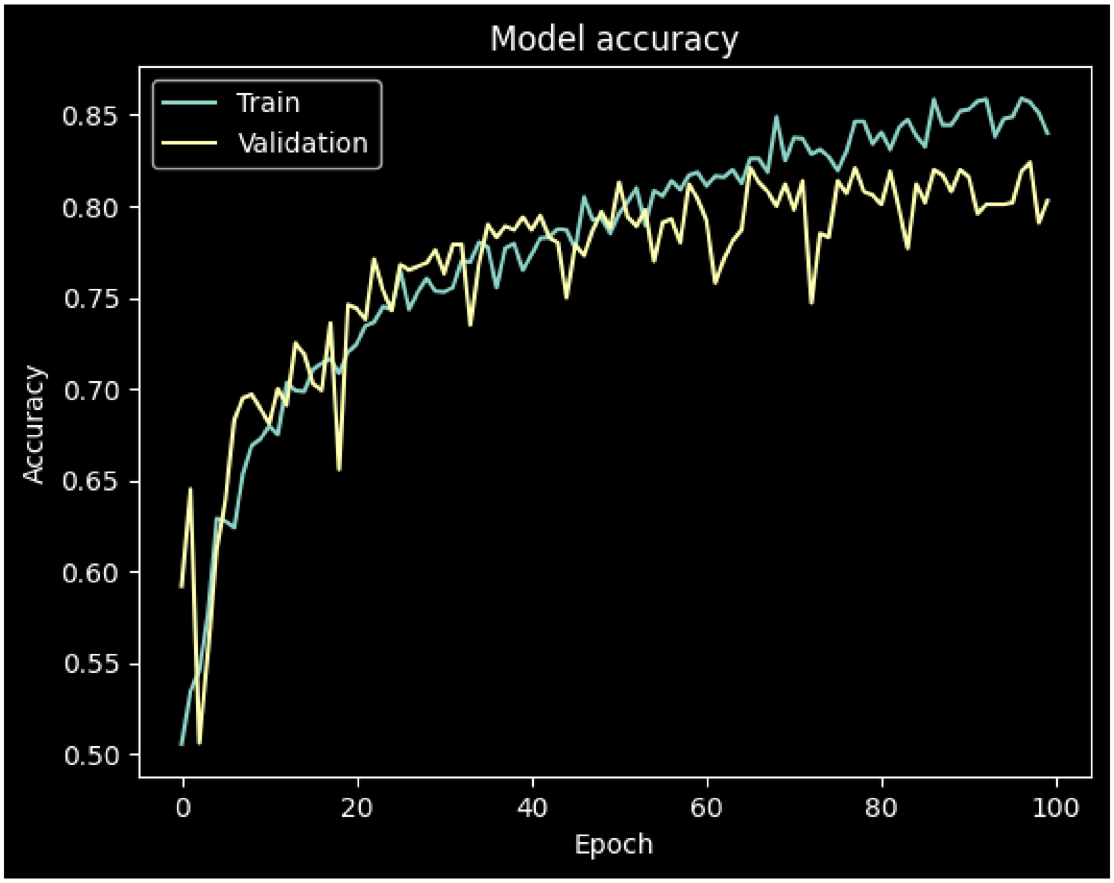
```
accuracy: 0.7745 - loss: 0.4744 - val_accuracy: 0.7880 - val_loss: 0.4820
Epoch 51/100
63/63            24s 357ms/step -
accuracy: 0.7965 - loss: 0.4261 - val_accuracy: 0.8130 - val_loss: 0.4928
Epoch 52/100
63/63            24s 343ms/step -
accuracy: 0.8084 - loss: 0.4240 - val_accuracy: 0.7940 - val_loss: 0.4909
Epoch 53/100
63/63            24s 344ms/step -
accuracy: 0.8155 - loss: 0.4082 - val_accuracy: 0.7890 - val_loss: 0.4681
Epoch 54/100
63/63            24s 348ms/step -
accuracy: 0.7913 - loss: 0.4384 - val_accuracy: 0.7980 - val_loss: 0.4380
Epoch 55/100
63/63            24s 345ms/step -
accuracy: 0.8285 - loss: 0.3962 - val_accuracy: 0.7700 - val_loss: 0.6393
Epoch 56/100
63/63            23s 333ms/step -
accuracy: 0.8036 - loss: 0.4603 - val_accuracy: 0.7910 - val_loss: 0.5268
Epoch 57/100
63/63            21s 308ms/step -
accuracy: 0.8020 - loss: 0.4261 - val_accuracy: 0.7930 - val_loss: 0.4938
Epoch 58/100
63/63            21s 304ms/step -
accuracy: 0.8142 - loss: 0.4033 - val_accuracy: 0.7800 - val_loss: 0.4514
Epoch 59/100
63/63            21s 310ms/step -
accuracy: 0.8046 - loss: 0.4072 - val_accuracy: 0.8120 - val_loss: 0.4661
Epoch 60/100
63/63            21s 308ms/step -
accuracy: 0.8308 - loss: 0.3834 - val_accuracy: 0.8040 - val_loss: 0.5296
Epoch 61/100
63/63            21s 305ms/step -
accuracy: 0.8177 - loss: 0.4049 - val_accuracy: 0.7920 - val_loss: 0.4345
Epoch 62/100
63/63            21s 308ms/step -
accuracy: 0.8128 - loss: 0.3996 - val_accuracy: 0.7580 - val_loss: 0.4969
Epoch 63/100
63/63            22s 312ms/step -
accuracy: 0.8098 - loss: 0.4149 - val_accuracy: 0.7710 - val_loss: 0.4764
Epoch 64/100
63/63            21s 306ms/step -
accuracy: 0.8172 - loss: 0.3771 - val_accuracy: 0.7810 - val_loss: 0.4643
Epoch 65/100
63/63            22s 319ms/step -
accuracy: 0.8185 - loss: 0.3992 - val_accuracy: 0.7870 - val_loss: 0.4509
Epoch 66/100
63/63            21s 305ms/step -
```

```
accuracy: 0.8339 - loss: 0.3722 - val_accuracy: 0.8210 - val_loss: 0.4822
Epoch 67/100
63/63            21s 311ms/step -
accuracy: 0.8280 - loss: 0.3708 - val_accuracy: 0.8130 - val_loss: 0.4572
Epoch 68/100
63/63            21s 312ms/step -
accuracy: 0.8281 - loss: 0.3861 - val_accuracy: 0.8080 - val_loss: 0.4368
Epoch 69/100
63/63            21s 305ms/step -
accuracy: 0.8509 - loss: 0.3529 - val_accuracy: 0.8000 - val_loss: 0.4669
Epoch 70/100
63/63            21s 307ms/step -
accuracy: 0.8403 - loss: 0.3635 - val_accuracy: 0.8120 - val_loss: 0.4709
Epoch 71/100
63/63            21s 312ms/step -
accuracy: 0.8398 - loss: 0.3574 - val_accuracy: 0.7980 - val_loss: 0.4677
Epoch 72/100
63/63            21s 306ms/step -
accuracy: 0.8477 - loss: 0.3378 - val_accuracy: 0.8140 - val_loss: 0.4869
Epoch 73/100
63/63            25s 375ms/step -
accuracy: 0.8354 - loss: 0.3691 - val_accuracy: 0.7470 - val_loss: 0.5071
Epoch 74/100
63/63            21s 305ms/step -
accuracy: 0.8451 - loss: 0.3454 - val_accuracy: 0.7850 - val_loss: 0.4678
Epoch 75/100
63/63            21s 309ms/step -
accuracy: 0.8275 - loss: 0.3685 - val_accuracy: 0.7830 - val_loss: 0.4578
Epoch 76/100
63/63            21s 308ms/step -
accuracy: 0.8148 - loss: 0.4006 - val_accuracy: 0.8140 - val_loss: 0.4748
Epoch 77/100
63/63            21s 307ms/step -
accuracy: 0.8421 - loss: 0.3538 - val_accuracy: 0.8070 - val_loss: 0.4408
Epoch 78/100
63/63            21s 308ms/step -
accuracy: 0.8501 - loss: 0.3313 - val_accuracy: 0.8210 - val_loss: 0.4672
Epoch 79/100
63/63            21s 305ms/step -
accuracy: 0.8537 - loss: 0.3328 - val_accuracy: 0.8080 - val_loss: 0.4763
Epoch 80/100
63/63            21s 306ms/step -
accuracy: 0.8311 - loss: 0.3570 - val_accuracy: 0.8060 - val_loss: 0.5457
Epoch 81/100
63/63            21s 303ms/step -
accuracy: 0.8402 - loss: 0.3526 - val_accuracy: 0.8010 - val_loss: 0.4249
Epoch 82/100
63/63            21s 307ms/step -
```

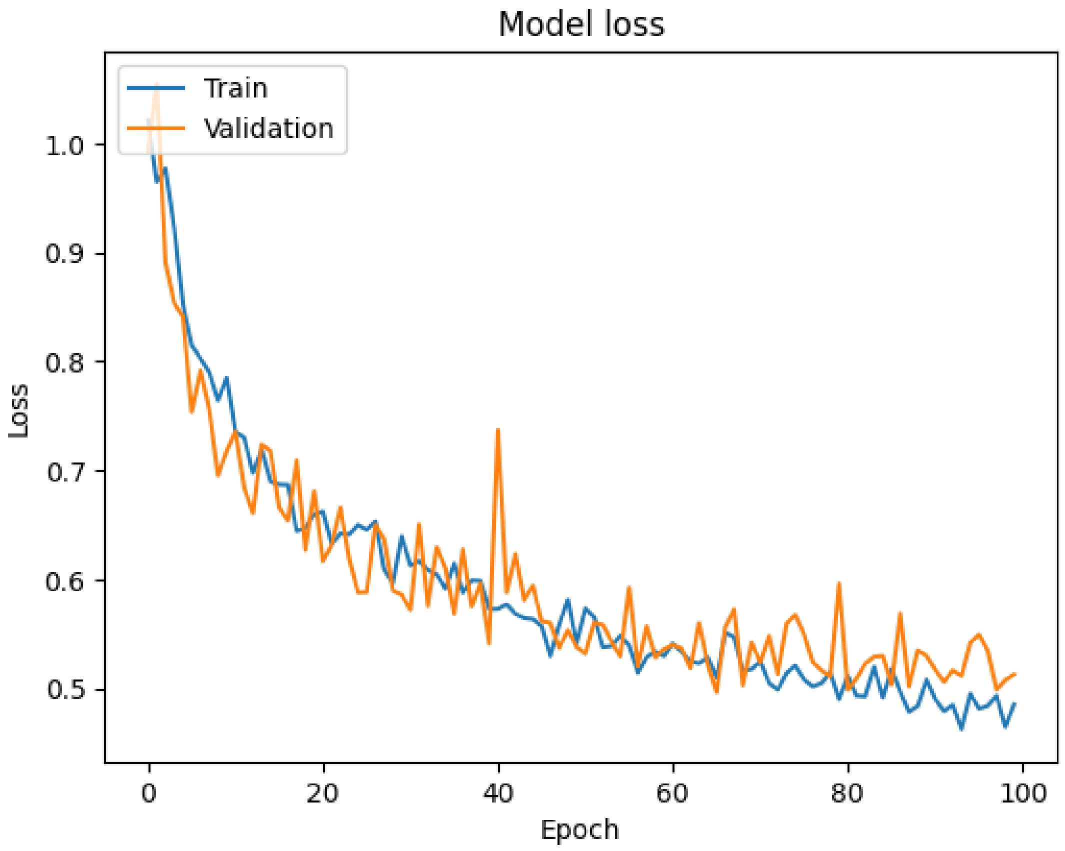
```
accuracy: 0.8363 - loss: 0.3628 - val_accuracy: 0.8190 - val_loss: 0.5471
Epoch 83/100
63/63            21s 306ms/step -
accuracy: 0.8282 - loss: 0.3719 - val_accuracy: 0.7990 - val_loss: 0.4496
Epoch 84/100
63/63            21s 307ms/step -
accuracy: 0.8579 - loss: 0.3137 - val_accuracy: 0.7770 - val_loss: 0.4678
Epoch 85/100
63/63            21s 305ms/step -
accuracy: 0.8356 - loss: 0.3703 - val_accuracy: 0.8120 - val_loss: 0.4589
Epoch 86/100
63/63            21s 310ms/step -
accuracy: 0.8257 - loss: 0.3617 - val_accuracy: 0.8020 - val_loss: 0.4746
Epoch 87/100
63/63            22s 315ms/step -
accuracy: 0.8507 - loss: 0.3283 - val_accuracy: 0.8200 - val_loss: 0.4606
Epoch 88/100
63/63            21s 305ms/step -
accuracy: 0.8495 - loss: 0.3218 - val_accuracy: 0.8170 - val_loss: 0.4382
Epoch 89/100
63/63            21s 303ms/step -
accuracy: 0.8569 - loss: 0.3181 - val_accuracy: 0.8080 - val_loss: 0.4395
Epoch 90/100
63/63            21s 303ms/step -
accuracy: 0.8520 - loss: 0.3184 - val_accuracy: 0.8200 - val_loss: 0.4378
Epoch 91/100
63/63            21s 303ms/step -
accuracy: 0.8569 - loss: 0.3217 - val_accuracy: 0.8160 - val_loss: 0.4329
Epoch 92/100
63/63            21s 301ms/step -
accuracy: 0.8621 - loss: 0.3214 - val_accuracy: 0.7960 - val_loss: 0.4787
Epoch 93/100
63/63            21s 303ms/step -
accuracy: 0.8682 - loss: 0.3110 - val_accuracy: 0.8010 - val_loss: 0.4475
Epoch 94/100
63/63            21s 302ms/step -
accuracy: 0.8467 - loss: 0.3247 - val_accuracy: 0.8010 - val_loss: 0.4355
Epoch 95/100
63/63            21s 307ms/step -
accuracy: 0.8564 - loss: 0.3006 - val_accuracy: 0.8010 - val_loss: 0.4469
Epoch 96/100
63/63            21s 312ms/step -
accuracy: 0.8358 - loss: 0.3458 - val_accuracy: 0.8020 - val_loss: 0.4296
Epoch 97/100
63/63            21s 302ms/step -
accuracy: 0.8652 - loss: 0.2942 - val_accuracy: 0.8190 - val_loss: 0.4392
Epoch 98/100
63/63            21s 302ms/step -
```

```
accuracy: 0.8675 - loss: 0.3097 - val_accuracy: 0.8240 - val_loss: 0.4283
Epoch 99/100
63/63                 21s 302ms/step -
accuracy: 0.8510 - loss: 0.3395 - val_accuracy: 0.7910 - val_loss: 0.4515
Epoch 100/100
63/63                 21s 309ms/step -
accuracy: 0.8537 - loss: 0.3283 - val_accuracy: 0.8030 - val_loss: 0.4845
```

[ ]:
```
# Plot Accruacy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```



[ ]:
```
# Plot Loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```



[ ]:
```
model_cnn.save('model_cnn_best.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.
```

2 Prediksi citra dengan load model yang telah di export/di simpan

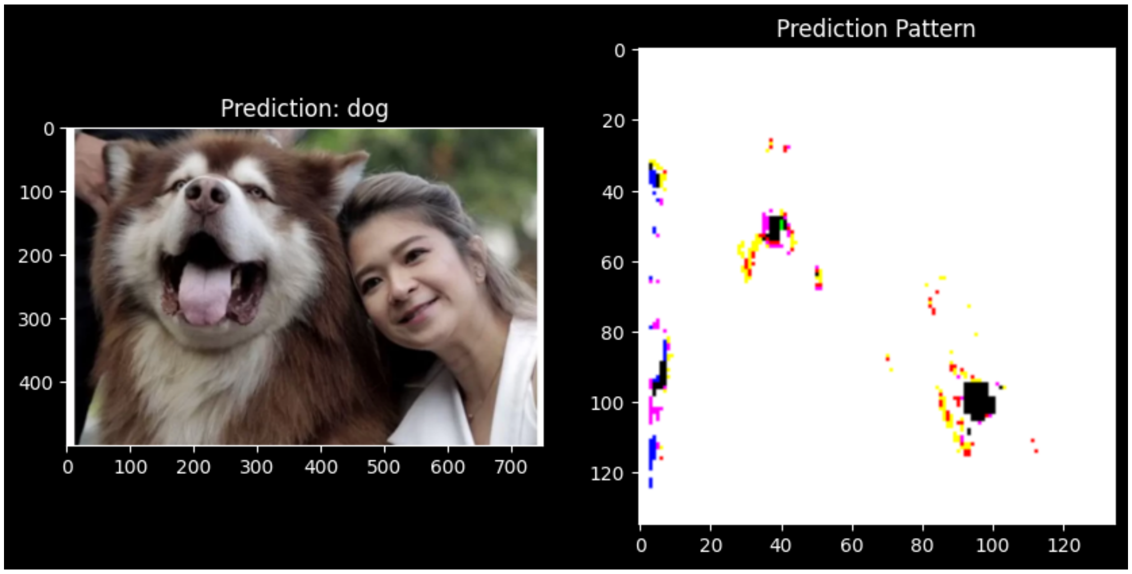
```
[ ]: from tensorflow.keras.models import load_model

     loaded_model = load_model('model_cnn_best.h5')

     data_image = './testing_data/9_anjing.webp'

     image_predict =  tf.keras.preprocessing.image.load_img(data_image,␣
      ↪target_size=(image_size[0], image_size[1]))
     image_predict =  tf.keras.preprocessing.image.img_to_array(image_predict)
     image_predict =  np.expand_dims(image_predict, axis=0)
     result = loaded_model.predict(image_predict)
     print(result)
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
built. `model.compile_metrics` will be empty until you train or evaluate the
model.
```

```
1/1             0s 91ms/step
[[591.368]]
```

```
[ ]: fig, axs = plt.subplots(1, 2, figsize=(10, 5))
     fig = plt.style.use('dark_background')

     # Plot original image
     axs[0].imshow(tf.keras.preprocessing.image.load_img(data_image))
     axs[0].set_title('Prediction: ' + ('cat' if result[0] < 0 else 'dog'))

     # Plot prediction pattern
     axs[1].imshow(image_predict[0])
     axs[1].set_title('Prediction Pattern')

     plt.show()

     if result[0] < 0:
         prediction = 'cat'
     else:
         prediction = 'dog'

     print(prediction)
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with
RGB data ([0..1] for floats or [0..255] for integers).
```



```
dog
```