```
import tensorflow as tf
import cv2
import numpy as np
from matplotlib import pyplot as plt
from pathlib import Path
import imghdr
```

Import Dataset

```
from google.colab import drive
drive.mount('/content/drive')
data_dir = '/content/drive/My Drive/Colab Notebooks/Dataset/Dataset/'
image_extensions = [".png", ".jpg", ".jpeg"]
img_type_accepted_by_tf = ["jpg", "jpeg", "png", "gif", "bmp"]
```

```
Mounted at /content/drive
```

```
data = tf.keras.utils.image_dataset_from_directory(data_dir, image_size=(227,227),batch_size=16)
print(data.class_names)
class_names=data.class_names
```

```
Found 679 files belonging to 4 classes.
['Bolt', 'Carl', 'Olaf', 'Woody']
```

```
data_iterator = data.as_numpy_iterator()
print("data_iterator", data_iterator)
```

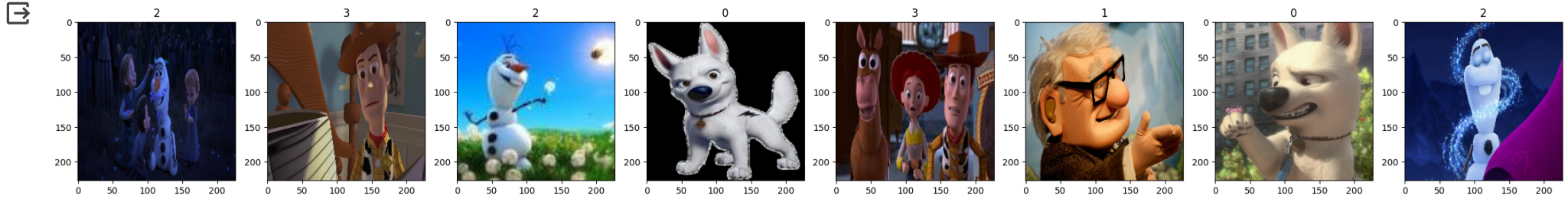
```
data_iterator <tensorflow.python.data.ops.dataset_ops.NumpyIterator object at 0x7aad5b562b80>
```

```
batch = data_iterator.next()
print("batch", batch)
```

```
       ...,
       [2.60000000e+01, 3.00000000e+01, 3.10000000e+01],
       [2.40000000e+01, 2.80000000e+01, 2.90000000e+01],
       [2.40000000e+01, 2.80000000e+01, 2.90000000e+01]],

      [[2.70000000e+01, 3.10000000e+01, 3.20000000e+01],
       [2.70000000e+01, 3.10000000e+01, 3.20000000e+01],
       [2.70000000e+01, 3.10000000e+01, 3.20000000e+01],
       ...,
       [2.60000000e+01, 3.00000000e+01, 3.10000000e+01],
       [2.40000000e+01, 2.80000000e+01, 2.90000000e+01],
       [2.40000000e+01, 2.80000000e+01, 2.90000000e+01]],

      [[2.70000000e+01, 3.10000000e+01, 3.20000000e+01],
       [2.70000000e+01, 3.10000000e+01, 3.20000000e+01],
       [2.70000000e+01, 3.10000000e+01, 3.20000000e+01],
       ...,
       [2.60000000e+01, 3.00000000e+01, 3.10000000e+01],
       [2.40000000e+01, 2.80000000e+01, 2.90000000e+01],
       [2.40000000e+01, 2.80000000e+01, 2.90000000e+01]],

      ...,

      [[2.40000000e+01, 2.80000000e+01, 2.90000000e+01],
       [2.40000000e+01, 2.80000000e+01, 2.90000000e+01],
       [2.43259888e+01, 2.83259888e+01, 2.93259888e+01],
       ...,
       [1.10000000e+01, 1.10000000e+01, 1.10000000e+01],
       [1.03480225e+01, 1.03480225e+01, 1.23480225e+01],
       [1.03480225e+01, 1.03480225e+01, 1.23480225e+01]],

      [[2.40000000e+01, 2.80000000e+01, 2.90000000e+01],
       [2.40000000e+01, 2.80000000e+01, 2.90000000e+01],
       [2.40000000e+01, 2.80000000e+01, 2.90000000e+01],
       ...,
       [1.10000000e+01, 1.10000000e+01, 1.10000000e+01],
       [9.00000000e+00, 9.00000000e+00, 1.10000000e+01],
       [9.00000000e+00, 9.00000000e+00, 1.10000000e+01]],

      [[2.40000000e+01, 2.80000000e+01, 2.90000000e+01],
       [2.40000000e+01, 2.80000000e+01, 2.90000000e+01],
       [2.30000000e+01, 2.70000000e+01, 2.80000000e+01],
       ...,
       [1.10000000e+01, 1.10000000e+01, 1.10000000e+01],
       [9.00000000e+00, 9.00000000e+00, 1.10000000e+01],
       [9.00000000e+00, 9.00000000e+00, 1.10000000e+01]]]],
     dtype=float32), array([2, 3, 2, 0, 3, 1, 0, 2, 2, 2, 1, 3, 1, 1, 1, 3], dtype=int32))
```

```
fig, ax = plt.subplots(ncols=8, figsize=(30,30))
for idx, img in enumerate(batch[0][:8]):
    ax[idx].imshow(img.astype(int))
```

```
    ax[idx].title.set_text(batch[1][idx])
```



NORMALISASI

```
data = data.map(lambda x, y: (x/255.0, y))
print("Data type after normalization: {}".format(data.element_spec))
print("Data shape after normalization: {}".format(data.element_spec))
print("Jumlah data", len(data))
```

```
Data type after normalization: (TensorSpec(shape=(None, 227, 227, 3), dtype=tf.float32, name=None), TensorSpec(shape=(None,), dtype=tf.int32, name=None))
Data shape after normalization: (TensorSpec(shape=(None, 227, 227, 3), dtype=tf.float32, name=None), TensorSpec(shape=(None,), dtype=tf.int32, name=None))
Jumlah data 43
```

BAGI DATASET JADI 3 BAGIAN : TRAIN, VALIDASI, DAN TESTING

```
train_size = int(0.8 * len(data))
val_size = int(0.1 * len(data))
test_size = int(0.1 * len(data))

print(train_size)
print(val_size)
print(test_size)
```

```
34
4
4
```

```
train = data.take(train_size)
val = data.skip(train_size).take(val_size)
test = data.skip(train_size+val_size).take(test_size)
```

MODEL ALEXNET (STE)

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Dropout


#inisisasi model kosong
model = Sequential()


# Layer 1
model.add(Conv2D(96, kernel_size=(11, 11), strides=(4, 4), activation='relu', input_shape=(227, 227, 3), padding='valid'))
model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))


# Layer 2
model.add(Conv2D(64, kernel_size=(5, 5), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))


# Layer 3
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu', padding='same'))


# Layer 4
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu', padding='same'))


# Layer 5
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))


# Flatten the output for the fully connected layers
model.add(Flatten())


# Fully Connected Layer 1
model.add(Dense(512, activation='relu'))

# Fully Connected Layer 2
model.add(Dense(256, activation='relu'))

model.add(Dropout(0.2))

# Output Layer
model.add(Dense(4, activation='softmax'))
```

```
#compile model dengan adamax optimizer
model.compile(optimizer='adamax', loss='sparse_categorical_crossentropy', metrics=['accuracy'])


model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 55, 55, 96)        34944

 max_pooling2d (MaxPooling2  (None, 27, 27, 96)        0
 D)

 conv2d_1 (Conv2D)           (None, 27, 27, 64)        153664

 max_pooling2d_1 (MaxPoolin  (None, 13, 13, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 13, 13, 128)       73856

 conv2d_3 (Conv2D)           (None, 13, 13, 128)       147584

 conv2d_4 (Conv2D)           (None, 13, 13, 64)        73792

 max_pooling2d_2 (MaxPoolin  (None, 6, 6, 64)          0
 g2D)

 flatten (Flatten)           (None, 2304)              0

 dense (Dense)               (None, 512)               1180160

 dense_1 (Dense)             (None, 256)               131328

 dropout (Dropout)           (None, 256)               0

 dense_2 (Dense)             (None, 4)                 1028

=================================================================
Total params: 1796356 (6.85 MB)
Trainable params: 1796356 (6.85 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
history = model.fit(train, validation_data=val, epochs=40)
```

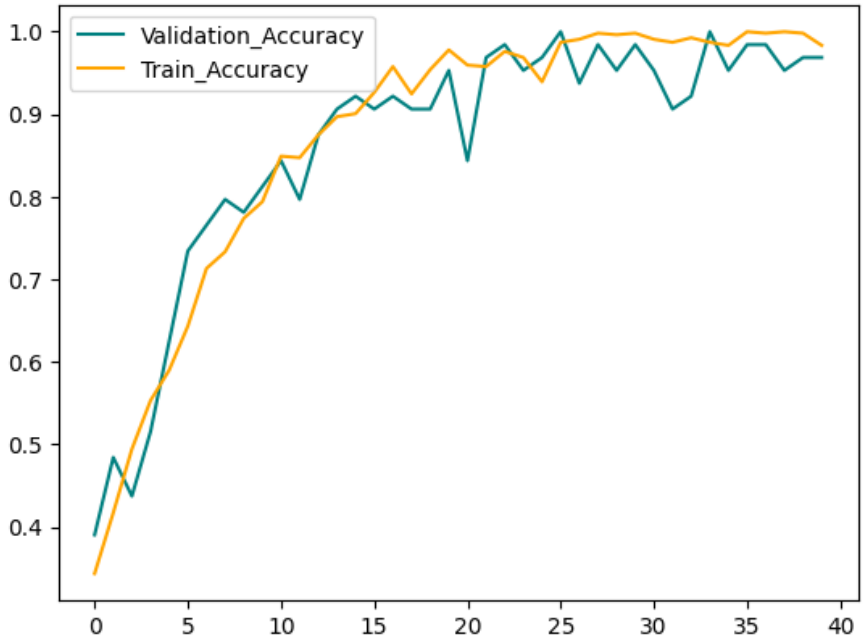
```
Epoch 1/40
34/34 [==============================] - 84s 2s/step - loss: 1.3485 - accuracy: 0.3438 - val_loss: 1.3206 - val_accuracy: 0.3906
Epoch 2/40
```

```
34/34 [==============================] - 42s 1s/step - loss: 1.2593 - accuracy: 0.4173 - val_loss: 1.0813 - val_accuracy: 0.4844
Epoch 3/40
34/34 [==============================] - 40s 1s/step - loss: 1.1184 - accuracy: 0.4945 - val_loss: 1.2042 - val_accuracy: 0.4375
Epoch 4/40
34/34 [==============================] - 41s 1s/step - loss: 1.0433 - accuracy: 0.5533 - val_loss: 1.0423 - val_accuracy: 0.5156
Epoch 5/40
34/34 [==============================] - 52s 2s/step - loss: 0.9817 - accuracy: 0.5901 - val_loss: 0.8995 - val_accuracy: 0.6250
Epoch 6/40
34/34 [==============================] - 42s 1s/step - loss: 0.8458 - accuracy: 0.6434 - val_loss: 0.7999 - val_accuracy: 0.7344
Epoch 7/40
34/34 [==============================] - 41s 1s/step - loss: 0.7277 - accuracy: 0.7132 - val_loss: 0.6029 - val_accuracy: 0.7656
Epoch 8/40
34/34 [==============================] - 41s 1s/step - loss: 0.6963 - accuracy: 0.7335 - val_loss: 0.6064 - val_accuracy: 0.7969
Epoch 9/40
34/34 [==============================] - 43s 1s/step - loss: 0.5881 - accuracy: 0.7739 - val_loss: 0.5913 - val_accuracy: 0.7812
Epoch 10/40
34/34 [==============================] - 41s 1s/step - loss: 0.5598 - accuracy: 0.7941 - val_loss: 0.4661 - val_accuracy: 0.8125
Epoch 11/40
34/34 [==============================] - 40s 1s/step - loss: 0.4515 - accuracy: 0.8493 - val_loss: 0.3855 - val_accuracy: 0.8438
Epoch 12/40
34/34 [==============================] - 41s 1s/step - loss: 0.4005 - accuracy: 0.8474 - val_loss: 0.4694 - val_accuracy: 0.7969
Epoch 13/40
34/34 [==============================] - 40s 1s/step - loss: 0.3707 - accuracy: 0.8750 - val_loss: 0.3283 - val_accuracy: 0.8750
Epoch 14/40
34/34 [==============================] - 41s 1s/step - loss: 0.3029 - accuracy: 0.8971 - val_loss: 0.4225 - val_accuracy: 0.9062
Epoch 15/40
34/34 [==============================] - 43s 1s/step - loss: 0.2765 - accuracy: 0.9007 - val_loss: 0.2260 - val_accuracy: 0.9219
Epoch 16/40
34/34 [==============================] - 45s 1s/step - loss: 0.2300 - accuracy: 0.9265 - val_loss: 0.2877 - val_accuracy: 0.9062
Epoch 17/40
34/34 [==============================] - 44s 1s/step - loss: 0.1504 - accuracy: 0.9577 - val_loss: 0.2628 - val_accuracy: 0.9219
Epoch 18/40
34/34 [==============================] - 41s 1s/step - loss: 0.2330 - accuracy: 0.9246 - val_loss: 0.2587 - val_accuracy: 0.9062
Epoch 19/40
34/34 [==============================] - 41s 1s/step - loss: 0.1424 - accuracy: 0.9540 - val_loss: 0.2700 - val_accuracy: 0.9062
Epoch 20/40
34/34 [==============================] - 41s 1s/step - loss: 0.1151 - accuracy: 0.9779 - val_loss: 0.1229 - val_accuracy: 0.9531
Epoch 21/40
34/34 [==============================] - 41s 1s/step - loss: 0.1247 - accuracy: 0.9596 - val_loss: 0.7359 - val_accuracy: 0.8438
Epoch 22/40
34/34 [==============================] - 40s 1s/step - loss: 0.1423 - accuracy: 0.9577 - val_loss: 0.1172 - val_accuracy: 0.9688
Epoch 23/40
34/34 [==============================] - 41s 1s/step - loss: 0.0764 - accuracy: 0.9761 - val_loss: 0.1029 - val_accuracy: 0.9844
Epoch 24/40
34/34 [==============================] - 44s 1s/step - loss: 0.1100 - accuracy: 0.9688 - val_loss: 0.1910 - val_accuracy: 0.9531
Epoch 25/40
34/34 [==============================] - 44s 1s/step - loss: 0.1601 - accuracy: 0.9393 - val_loss: 0.1468 - val_accuracy: 0.9688
Epoch 26/40
34/34 [==============================] - 46s 1s/step - loss: 0.0567 - accuracy: 0.9871 - val_loss: 0.0476 - val_accuracy: 1.0000
Epoch 27/40
34/34 [==============================] - 42s 1s/step - loss: 0.0306 - accuracy: 0.9908 - val_loss: 0.2898 - val_accuracy: 0.9375
```

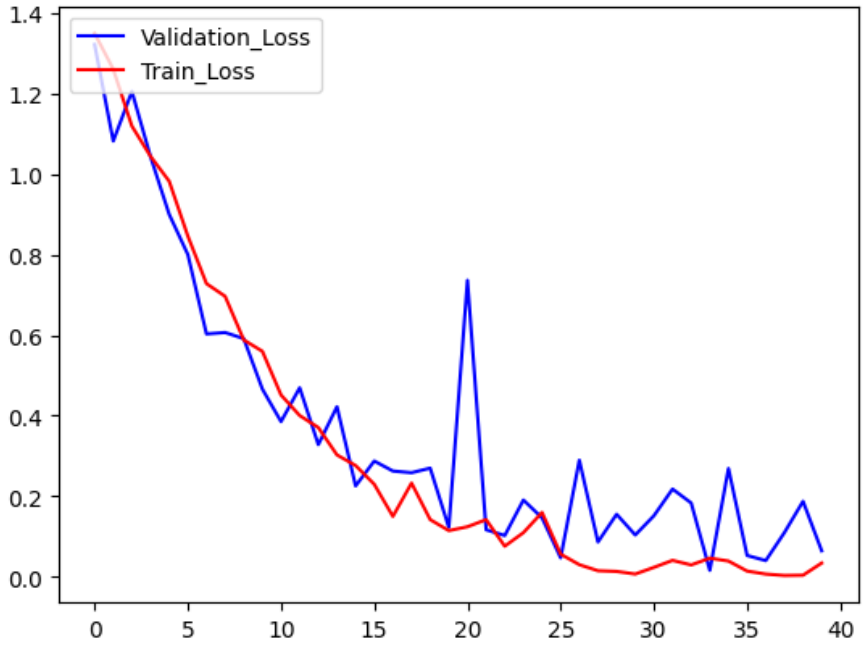
```
Epoch 28/40
34/34 [==============================] - 39s 1s/step - loss: 0.0156 - accuracy: 0.9982 - val_loss: 0.0869 - val_accuracy: 0.9844
Epoch 29/40
```

```
import pandas as pd
#pip install openpyxl

# Convert the training history to a DataFrame
history_df = pd.DataFrame(history.history)

# Save the DataFrame to an Excel file
history_df.to_excel('/content/drive/My Drive/Colab Notebooks/History_ALEXNET_B_SB 4.xlsx', index=False)
```

```
fig = plt.figure()
plt.plot(history.history['val_accuracy'], color='teal', label='Validation_Accuracy')
plt.plot(history.history['accuracy'], color='orange', label='Train_Accuracy')
plt.legend(loc="upper left")
plt.show()
```



```
fig = plt.figure()
plt.plot(history.history['val_loss'], color='blue', label='Validation_Loss')
plt.plot(history.history['loss'], color='red', label='Train_Loss')
plt.legend(loc="upper left")
plt.show()
```




```
model.evaluate(test)
```

```
4/4 [==============================] - 10s 161ms/step - loss: 0.0415 - accuracy: 0.9844
[0.04151067137718201, 0.984375]
```

```
model.save('/content/drive/My Drive/Colab Notebooks/model_alexnet.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file f
  saving_api.save_model(
```

```
import tensorflow as tf
import cv2
import numpy as np
from matplotlib import pyplot as plt
```

```
WARNING:tensorflow:From c:\Program Files\Python311\Lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is depreca
```

Mencari Unsupported Image

```
from pathlib import Path
import filetype

# RFC image file extensions supported by TensorFlow
img_exts = {"png", "jpg", "gif", "bmp"}

def searchUnsupportedImages(path):
    for file in path.iterdir():
        if file.is_dir():
            continue

        ext = filetype.guess_extension(file)

        if ext is None:
            print(f"'{file}': extension cannot be guessed from content")
        elif ext not in img_exts:
            print(f"'{file}': not a supported image file")


path1 = Path("Dataset/Olaf")
path2 = Path("Dataset/Woody")
path3 = Path("Dataset/Bolt")
path4 = Path("Dataset/Carl")

searchUnsupportedImages(path1)
searchUnsupportedImages(path2)
searchUnsupportedImages(path3)
searchUnsupportedImages(path4)
```

PERSIAPAN DATASET

```
data_dir = './Dataset/'
```

```
#BATCH UKURAN 16, IMAGE SESUAI MODEL
data = tf.keras.utils.image_dataset_from_directory(data_dir, image_size=(224,224),batch_size=16)
print(data.class_names)
class_names=data.class_names
```

```
Found 679 files belonging to 4 classes.
['Bolt', 'Carl', 'Olaf', 'Woody']
```

```
data_iterator = data.as_numpy_iterator()
print("data_iterator", data_iterator)
```

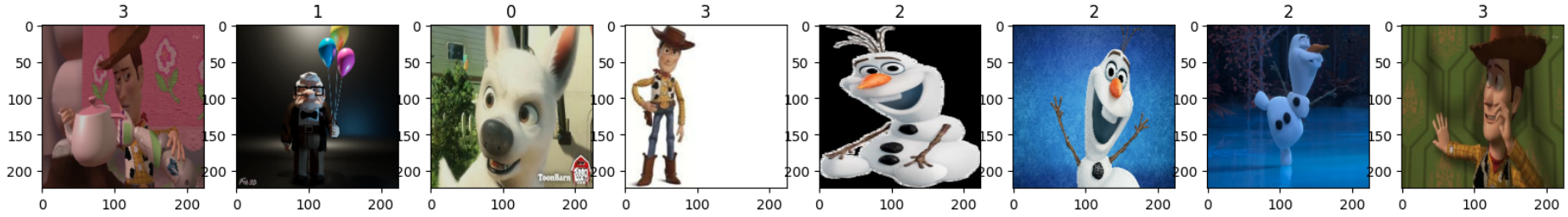
```
data_iterator <tensorflow.python.data.ops.dataset_ops.NumpyIterator object at 0x0000025B8AF0DAD0>
```

```
batch = data_iterator.next()
print("batch", batch)
```

```
batch (array([[[[122.44771  ,  95.42857  ,  87.71301  ],
         [122.       ,  95.       ,  87.64286  ],
         [116.07143  ,  89.07143  ,  82.07143  ],
         ...,
         [150.       ,  74.93497  ,  87.93497  ],
         [142.77171  ,  67.77171  ,  80.77171  ],
         [134.32149  ,  59.32148  ,  72.32148  ]],

        [[121.85715  ,  94.79975  ,  87.79975  ],
         [121.26785  ,  94.26785  ,  87.       ],
         [116.07143  ,  89.07143  ,  82.07143  ],
         ...,
         [153.14415  ,  78.125015 ,  91.125015 ],
         [148.26785  ,  73.26785  ,  86.26785  ],
         [143.07143  ,  68.07143  ,  81.07143  ]],

        [[122.625    ,  95.19005  ,  88.19005  ],
         [121.       ,  93.       ,  86.       ],
         [116.07143  ,  89.07143  ,  82.07143  ],
         ...,
         [134.58545  ,  60.624985 ,  74.624985 ],
         [139.50127  ,  65.21429  ,  79.05483  ],
         [142.44643  ,  68.21423  ,  82.21423  ]],

        ...,

        [[ 65.337425 ,  43.76599  ,  35.45602  ],
         [ 61.285713 ,  32.357143 ,  26.       ],
         [ 60.92857  ,  33.       ,  26.       ],
         ...,
         [107.03952  ,  74.58536  ,  71.       ],
         [105.55347  ,  75.55347  ,  71.55347  ],
         [103.55347  ,  73.4578   ,  71.988365 ]],

        [[ 57.94258  ,  36.371155 ,  27.789473 ],
```

```
       [ 62.024265 ,  33.357143 ,  27.       ],
       [ 61.94771  ,  33.267944 ,  27.       ],
       ...,
       [106.       ,  71.       ,  68.732056 ],
       [103.35718  ,  71.9043   ,  68.261475 ],
       [100.9426   ,  70.575226 ,  68.9426   ]],

      [[ 57.785713 ,  36.214287 ,  27.214287 ],
       [ 62.285713 ,  33.357143 ,  27.       ],
       [ 62.       ,  34.       ,  27.       ],
       ...,
       [108.089355 ,  73.089355 ,  70.089355 ],
       [103.35718  ,  72.       ,  69.       ],
       [ 99.       ,  68.78577  ,  67.78577  ]]],


     [[[ 17.131536 ,  16.86368  ,  13.1315365],
       [ 17.694515 ,  17.003508 ,  13.494579 ],
       [ 18.672514 ,  17.50287  ,  13.827329 ],
       ...,
       [ 14.       ,  12.       ,  10.49713  ],
       [ 12.789239 ,  12.       ,  10.39462  ],
       [ 12.136333 ,  11.863667 ,  10.       ]]
```

```
fig, ax = plt.subplots(ncols=8, figsize=(20,20))
for idx, img in enumerate(batch[0][:8]):
    ax[idx].imshow(img.astype(int))
    ax[idx].title.set_text(batch[1][idx])
```




```
data = data.map(lambda x, y: (x/255.0, y))
print("Data type after normalization: {}".format(data.element_spec))
print("Data shape after normalization: {}".format(data.element_spec))
print("Jumlah data", len(data))
```

```
Data type after normalization: (TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32, name=None), TensorSpec(shape=(None,), dtype=tf.int32, name=None))
Data shape after normalization: (TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32, name=None), TensorSpec(shape=(None,), dtype=tf.int32, name=None))
Jumlah data 43
```

BAGI DATASET JADI 3 BAGIAN: TRAIN, VALIDASI, TESTING

```
train_size = int(0.8 * len(data))
val_size = int(0.1 * len(data))
test_size = int(0.1 * len(data))

print(train_size)
print(val_size)
print(test_size)
```

```
34
4
4
```

```
train = data.take(train_size)
val = data.skip(train_size).take(val_size)
test = data.skip(train_size+val_size).take(test_size)
```

MODEL VGGNet

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Dropout
from tensorflow.keras.layers import BatchNormalization


#inisisasi model kosong
model = Sequential()
```

```
WARNING:tensorflow:From c:\Program Files\Python311\Lib\site-packages\keras\src\backend.py:873: The name tf.get_default_graph is deprecated. Please use tf
```

```
# 13 convolutional layers dan 5 max-pooling layers

# Block 1
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3), padding='same'))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))

# Block 2
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))
```

```
# Block 3
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))

# Block 4
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))

# Block 5
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))
```

```
WARNING:tensorflow:From c:\Program Files\Python311\Lib\site-packages\keras\src\layers\normalization\batch_normalization.py:979: The name tf.nn.fused_batc
```

+ Code + Text

```
# Flatten the output for the fully connected layers
model.add(Flatten())


# 3 Fully Connected Layer
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(4, activation='softmax'))


from tensorflow.keras.optimizers import Adamax
from tensorflow.keras.callbacks import LearningRateScheduler

def lr_schedule(epoch):
    lr = 0.001 * (0.9 ** epoch)
    return lr

model.compile(optimizer=Adamax(learning_rate=0.001), loss='sparse_categorical_crossentropy', metrics=['accuracy'])
lr_scheduler = LearningRateScheduler(lr_schedule)


model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 224, 224, 32)      896

 conv2d_1 (Conv2D)           (None, 224, 224, 32)      9248

 batch_normalization (Batch  (None, 224, 224, 32)      128
 Normalization)

 max_pooling2d (MaxPooling2  (None, 112, 112, 32)      0
 D)

 conv2d_2 (Conv2D)           (None, 112, 112, 64)      18496

 conv2d_3 (Conv2D)           (None, 112, 112, 64)      36928

 batch_normalization_1 (Bat  (None, 112, 112, 64)      256
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 56, 56, 64)        0
 g2D)

 conv2d_4 (Conv2D)           (None, 56, 56, 128)       73856

 conv2d_5 (Conv2D)           (None, 56, 56, 128)       147584

 conv2d_6 (Conv2D)           (None, 56, 56, 128)       147584

 batch_normalization_2 (Bat  (None, 56, 56, 128)       512
 chNormalization)

 max_pooling2d_2 (MaxPoolin  (None, 28, 28, 128)       0
 g2D)

 conv2d_7 (Conv2D)           (None, 28, 28, 256)       295168

 conv2d_8 (Conv2D)           (None, 28, 28, 256)       590080

 conv2d_9 (Conv2D)           (None, 28, 28, 256)       590080

 batch_normalization_3 (Bat  (None, 28, 28, 256)       1024
 chNormalization)

 max_pooling2d_3 (MaxPoolin  (None, 14, 14, 256)       0
 g2D)

 conv2d_10 (Conv2D)          (None, 14, 14, 256)       590080

 conv2d_11 (Conv2D)          (None, 14, 14, 256)       590080

 conv2d_12 (Conv2D)          (None, 14, 14, 256)       590080
```

```
 batch_normalization_4 (Bat  (None, 14, 14, 256)       1024
 chNormalization)

 max_pooling2d_4 (MaxPoolin  (None, 7, 7, 256)         0
```

```
history = model.fit(train, validation_data=val, epochs=35, callbacks=[lr_scheduler])
```

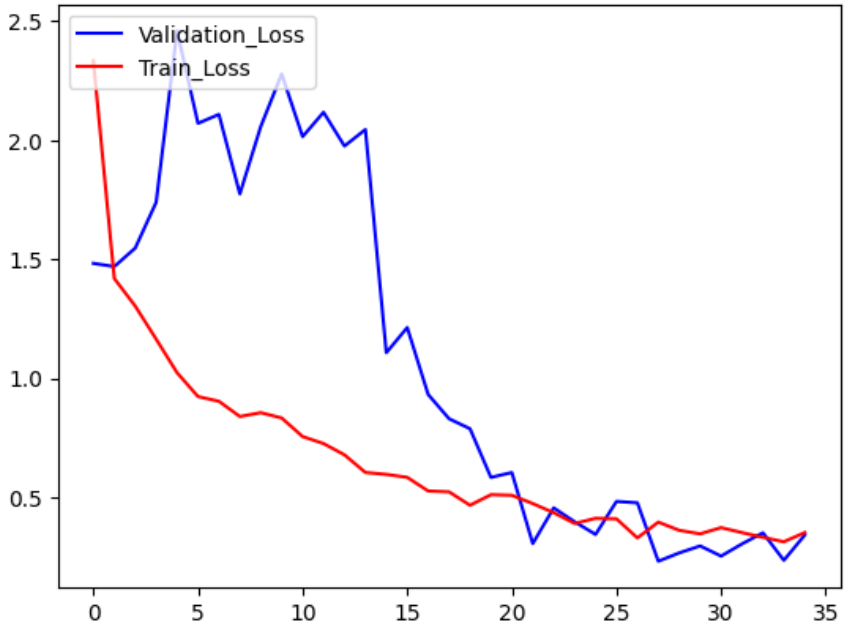
```
Epoch 1/35
WARNING:tensorflow:From c:\Program Files\Python311\Lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecate

WARNING:tensorflow:From c:\Program Files\Python311\Lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.executing_eagerly_outside_fu

34/34 [==============================] - 343s 10s/step - loss: 2.3336 - accuracy: 0.3107 - val_loss: 1.4826 - val_accuracy: 0.2031 - lr: 0.0010
Epoch 2/35
34/34 [==============================] - 375s 11s/step - loss: 1.4190 - accuracy: 0.4357 - val_loss: 1.4690 - val_accuracy: 0.2500 - lr: 9.0000e-04
Epoch 3/35
34/34 [==============================] - 319s 9s/step - loss: 1.3030 - accuracy: 0.5018 - val_loss: 1.5468 - val_accuracy: 0.2812 - lr: 8.1000e-04
Epoch 4/35
34/34 [==============================] - 339s 10s/step - loss: 1.1642 - accuracy: 0.5276 - val_loss: 1.7389 - val_accuracy: 0.1562 - lr: 7.2900e-04
Epoch 5/35
34/34 [==============================] - 284s 8s/step - loss: 1.0232 - accuracy: 0.6085 - val_loss: 2.4593 - val_accuracy: 0.1875 - lr: 6.5610e-04
Epoch 6/35
34/34 [==============================] - 296s 9s/step - loss: 0.9230 - accuracy: 0.6213 - val_loss: 2.0706 - val_accuracy: 0.1719 - lr: 5.9049e-04
Epoch 7/35
34/34 [==============================] - 293s 9s/step - loss: 0.9032 - accuracy: 0.6397 - val_loss: 2.1082 - val_accuracy: 0.3125 - lr: 5.3144e-04
Epoch 8/35
34/34 [==============================] - 326s 10s/step - loss: 0.8393 - accuracy: 0.6489 - val_loss: 1.7742 - val_accuracy: 0.2344 - lr: 4.7830e-04
Epoch 9/35
34/34 [==============================] - 281s 8s/step - loss: 0.8554 - accuracy: 0.6544 - val_loss: 2.0574 - val_accuracy: 0.2500 - lr: 4.3047e-04
Epoch 10/35
34/34 [==============================] - 266s 8s/step - loss: 0.8331 - accuracy: 0.6820 - val_loss: 2.2778 - val_accuracy: 0.2031 - lr: 3.8742e-04
Epoch 11/35
34/34 [==============================] - 245s 7s/step - loss: 0.7551 - accuracy: 0.6985 - val_loss: 2.0148 - val_accuracy: 0.1719 - lr: 3.4868e-04
Epoch 12/35
34/34 [==============================] - 267s 8s/step - loss: 0.7256 - accuracy: 0.7390 - val_loss: 2.1171 - val_accuracy: 0.2656 - lr: 3.1381e-04
Epoch 13/35
34/34 [==============================] - 288s 8s/step - loss: 0.6782 - accuracy: 0.7224 - val_loss: 1.9752 - val_accuracy: 0.2344 - lr: 2.8243e-04
Epoch 14/35
34/34 [==============================] - 307s 9s/step - loss: 0.6043 - accuracy: 0.7684 - val_loss: 2.0447 - val_accuracy: 0.2969 - lr: 2.5419e-04
Epoch 15/35
34/34 [==============================] - 361s 11s/step - loss: 0.5962 - accuracy: 0.7592 - val_loss: 1.1073 - val_accuracy: 0.5469 - lr: 2.2877e-04
Epoch 16/35
34/34 [==============================] - 314s 9s/step - loss: 0.5838 - accuracy: 0.7776 - val_loss: 1.2129 - val_accuracy: 0.4531 - lr: 2.0589e-04
Epoch 17/35
34/34 [==============================] - 382s 11s/step - loss: 0.5264 - accuracy: 0.8033 - val_loss: 0.9318 - val_accuracy: 0.6094 - lr: 1.8530e-04
Epoch 18/35
34/34 [==============================] - 341s 10s/step - loss: 0.5228 - accuracy: 0.8107 - val_loss: 0.8298 - val_accuracy: 0.6250 - lr: 1.6677e-04
Epoch 19/35
34/34 [==============================] - 341s 10s/step - loss: 0.4665 - accuracy: 0.8346 - val_loss: 0.7879 - val_accuracy: 0.7031 - lr: 1.5009e-04
Epoch 20/35
34/34 [==============================] - 294s 8s/step - loss: 0.5109 - accuracy: 0.8162 - val_loss: 0.5835 - val_accuracy: 0.7969 - lr: 1.3509e-04
Epoch 21/35
34/34 [==============================] - 299s 9s/step - loss: 0.5085 - accuracy: 0.8162 - val_loss: 0.6041 - val_accuracy: 0.7344 - lr: 1.2158e-04
```

```
Epoch 22/35
34/34 [==============================] - 301s 9s/step - loss: 0.4736 - accuracy: 0.8272 - val_loss: 0.3055 - val_accuracy: 0.9375 - lr: 1.0942e-04
Epoch 23/35
34/34 [==============================] - 299s 9s/step - loss: 0.4353 - accuracy: 0.8511 - val_loss: 0.4555 - val_accuracy: 0.7656 - lr: 9.8477e-05
Epoch 24/35
34/34 [==============================] - 288s 8s/step - loss: 0.3900 - accuracy: 0.8658 - val_loss: 0.3972 - val_accuracy: 0.8281 - lr: 8.8629e-05
Epoch 25/35
34/34 [==============================] - 288s 8s/step - loss: 0.4116 - accuracy: 0.8529 - val_loss: 0.3439 - val_accuracy: 0.8594 - lr: 7.9766e-05
Epoch 26/35
34/34 [==============================] - 289s 8s/step - loss: 0.4095 - accuracy: 0.8621 - val_loss: 0.4825 - val_accuracy: 0.8125 - lr: 7.1790e-05
Epoch 27/35
```

```
import pandas as pd
#pip install openpyxl

# Convert the training history to a DataFrame
history_df = pd.DataFrame(history.history)

# Save the DataFrame to an Excel file
history_df.to_excel('History_VGG_B_SB 4.xlsx', index=False)
```

```
fig = plt.figure()
plt.plot(history.history['val_accuracy'], color='teal', label='Validation_Accuracy')
plt.plot(history.history['accuracy'], color='orange', label='Train_Accuracy')
plt.legend(loc="upper left")
plt.show()
```

```
fig = plt.figure()
plt.plot(history.history['val_loss'], color='blue', label='Validation_Loss')
plt.plot(history.history['loss'], color='red', label='Train_Loss')
plt.legend(loc="upper left")
plt.show()
```




```
model.evaluate(test)
```

```
4/4 [==============================] - 13s 1s/step - loss: 0.1948 - accuracy: 0.9219
[0.1948060542345047, 0.921875]
```

```
model.save('model_vgg16.h5')
```

```
c:\Program Files\Python311\Lib\site-packages\keras\src\engine\training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`
  saving_api.save_model(
```

```
import tensorflow as tf
import cv2
import numpy as np
from matplotlib import pyplot as plt
```

```
WARNING:tensorflow:From c:\Program Files\Python311\Lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Ple
```

Mencari Unsupported Image

```
from pathlib import Path
import filetype

# RFC image file extensions supported by TensorFlow
img_exts = {"png", "jpg", "gif", "bmp"}

def searchUnsupportedImages(path):
    for file in path.iterdir():
        if file.is_dir():
            continue

        ext = filetype.guess_extension(file)

        if ext is None:
            print(f"'{file}': extension cannot be guessed from content")
        elif ext not in img_exts:
            print(f"'{file}': not a supported image file")


path1 = Path("Dataset/Olaf")
path2 = Path("Dataset/Woody")
path3 = Path("Dataset/Bolt")
path4 = Path("Dataset/Carl")

searchUnsupportedImages(path1)
searchUnsupportedImages(path2)
searchUnsupportedImages(path3)
searchUnsupportedImages(path4)
```

PERSIAPAN DATASET

```
data_dir = './Dataset/'
```

```
#BATCH UKURAN 16, IMAGE SESUAI MODEL
data = tf.keras.utils.image_dataset_from_directory(data_dir, image_size=(224,224),batch_size=16)
print(data.class_names)
class_names=data.class_names
```

```
Found 679 files belonging to 4 classes.
['Bolt', 'Carl', 'Olaf', 'Woody']
```

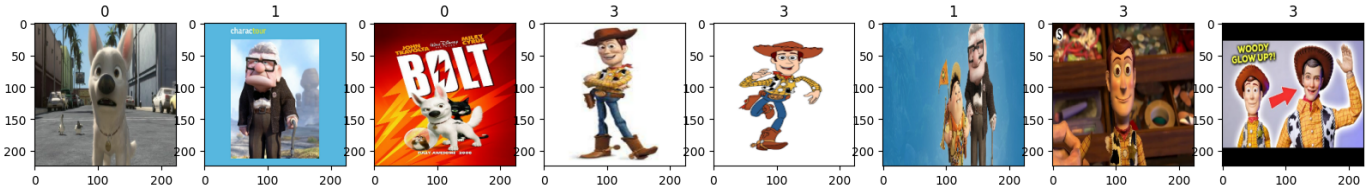
```
data_iterator = data.as_numpy_iterator()
print("data_iterator", data_iterator)
```

```
data_iterator <tensorflow.python.data.ops.dataset_ops.NumpyIterator object at 0x0000018A8D3B9C10>
```

```
batch = data_iterator.next()
print("batch", batch)
```

```
                [1.42878632e+02, 1.56488312e+02, 8.20737915e+01],
                [1.51772125e+02, 1.63058014e+02, 9.13436661e+01],
                [1.39101929e+02, 1.32530457e+02, 6.76095505e+01]],

               [[2.29123642e+02, 1.96123642e+02, 1.29123642e+02],
                [2.32599457e+02, 1.99599457e+02, 1.32599457e+02],
                [2.35482162e+02, 2.02482162e+02, 1.35482162e+02],
                ...,
                [1.40398972e+02, 1.48707596e+02, 7.93989792e+01],
                [1.43935654e+02, 1.55650009e+02, 8.19433136e+01],
                [1.36585556e+02, 1.30247482e+02, 6.85395966e+01]],

               [[2.35381409e+02, 2.02381409e+02, 1.35381409e+02],
                [2.34017883e+02, 2.01017883e+02, 1.34017883e+02],
                [2.31727066e+02, 1.98727066e+02, 1.31727066e+02],
                ...,
                [1.33245987e+02, 1.41103043e+02, 7.22459869e+01],
                [1.44870560e+02, 1.56584915e+02, 8.27277374e+01],
                [1.37982117e+02, 1.31267761e+02, 7.08392334e+01]]]],
              dtype=float32), array([0, 1, 0, 3, 3, 1, 3, 3, 1, 2, 3, 3, 0, 1, 3, 0]))
```

```
fig, ax = plt.subplots(ncols=8, figsize=(20,20))
for idx, img in enumerate(batch[0][:8]):
    ax[idx].imshow(img.astype(int))
    ax[idx].title.set_text(batch[1][idx])
```




```
data = data.map(lambda x, y: (x/255.0, y))
print("Data type after normalization: {}".format(data.element_spec))
print("Data shape after normalization: {}".format(data.element_spec))
print("Jumlah data", len(data))
```

```
Data type after normalization: (TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32, name=None), TensorSpec(shape=(None,), dtype=tf.int32, name=None))
Data shape after normalization: (TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32, name=None), TensorSpec(shape=(None,), dtype=tf.int32, name=None))
Jumlah data 43
```

BAGI DATASET JADI 3 BAGIAN: TRAIN, VALIDASI, TESTING

```
train_size = int(0.8 * len(data))
val_size = int(0.1 * len(data))
test_size = int(0.1 * len(data))

print(train_size)
print(val_size)
print(test_size)
```

```
34
4
4
```

```
train = data.take(train_size)
val = data.skip(train_size).take(val_size)
test = data.skip(train_size+val_size).take(test_size)
```

MODEL VGGNet

```
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, BatchNormalization, Activation, Add, GlobalAveragePooling2D, Dense
from tensorflow.keras.models import Model


def residual_block(x, filters, downsample=False):
    strides = 1
    if downsample:
        strides = 2
        identity = Conv2D(filters, 1, strides=strides, padding='same')(x)
        identity = BatchNormalization()(identity)
    else:
        identity = x

    x = Conv2D(filters, 3, strides=strides, padding='same')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(filters, 3, padding='same')(x)
    x = BatchNormalization()(x)

    x = Add()([x, identity])
    x = Activation('relu')(x)
    return x
```

```
def ResNet18(input_shape=(224, 224, 3), num_classes=1000):
    inputs = Input(shape=input_shape)

    x = Conv2D(64, 7, strides=2, padding='same')(inputs)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = MaxPooling2D(pool_size=3, strides=2, padding='same')(x)

    x = residual_block(x, 64)
    x = residual_block(x, 64)
    x = residual_block(x, 128, downsample=True)
    x = residual_block(x, 128)
    x = residual_block(x, 256, downsample=True)
    x = residual_block(x, 256)
    x = residual_block(x, 512, downsample=True)
    x = residual_block(x, 512)

    x = GlobalAveragePooling2D()(x)
    outputs = Dense(num_classes, activation='softmax')(x)

    model = Model(inputs=inputs, outputs=outputs)
    return model


model_resnet18 = ResNet18()
model_resnet18.compile(optimizer='adamax', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model_resnet18.summary()
```

```
 conv2d_18 (Conv2D)          (None, 7, 7, 512)           2359808   ['activation_14[0][0]']

 batch_normalization_18 (Ba  (None, 7, 7, 512)           2048      ['conv2d_18[0][0]']
 tchNormalization)

 activation_15 (Activation)  (None, 7, 7, 512)           0         ['batch_normalization_18[0][0]
                                                                   ']

 conv2d_19 (Conv2D)          (None, 7, 7, 512)           2359808   ['activation_15[0][0]']

 batch_normalization_19 (Ba  (None, 7, 7, 512)           2048      ['conv2d_19[0][0]']
 tchNormalization)

 add_7 (Add)                 (None, 7, 7, 512)           0         ['batch_normalization_19[0][0]
                                                                   ',
                                                                    'activation_14[0][0]']

 activation_16 (Activation)  (None, 7, 7, 512)           0         ['add_7[0][0]']

 global_average_pooling2d (  (None, 512)                 0         ['activation_16[0][0]']
 GlobalAveragePooling2D)

 dense (Dense)               (None, 1000)                513000    ['global_average_pooling2d[0][[
                                                                   0]']

==================================================================================================
Total params: 11703912 (44.65 MB)
Trainable params: 11694312 (44.61 MB)
Non-trainable params: 9600 (37.50 KB)
__________________________________________________________________________________________________
```

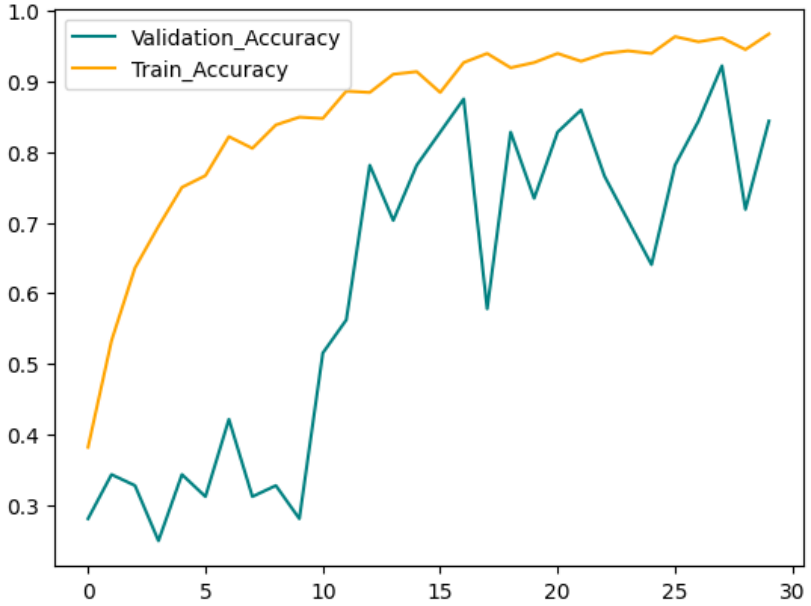
```
history_resnet18 = model_resnet18.fit(train, validation_data=val, epochs=30)
```

```
Epoch 1/30
WARNING:tensorflow:From c:\Program Files\Python311\Lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Ple

WARNING:tensorflow:From c:\Program Files\Python311\Lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.executing_eagerly_outside_function

34/34 [==============================] - 149s 4s/step - loss: 2.3795 - accuracy: 0.3824 - val_loss: 3.5276 - val_accuracy: 0.2812
Epoch 2/30
34/34 [==============================] - 127s 4s/step - loss: 1.1071 - accuracy: 0.5331 - val_loss: 1.9954 - val_accuracy: 0.3438
Epoch 3/30
34/34 [==============================] - 129s 4s/step - loss: 0.9174 - accuracy: 0.6360 - val_loss: 1.9970 - val_accuracy: 0.3281
Epoch 4/30
34/34 [==============================] - 130s 4s/step - loss: 0.7733 - accuracy: 0.6949 - val_loss: 1.8126 - val_accuracy: 0.2500
Epoch 5/30
34/34 [==============================] - 130s 4s/step - loss: 0.6985 - accuracy: 0.7500 - val_loss: 1.9172 - val_accuracy: 0.3438
Epoch 6/30
34/34 [==============================] - 129s 4s/step - loss: 0.6325 - accuracy: 0.7665 - val_loss: 1.8400 - val_accuracy: 0.3125
Epoch 7/30
34/34 [==============================] - 131s 4s/step - loss: 0.5236 - accuracy: 0.8217 - val_loss: 1.5181 - val_accuracy: 0.4219
Epoch 8/30
34/34 [==============================] - 125s 4s/step - loss: 0.5225 - accuracy: 0.8051 - val_loss: 4.2427 - val_accuracy: 0.3125
Epoch 9/30
```

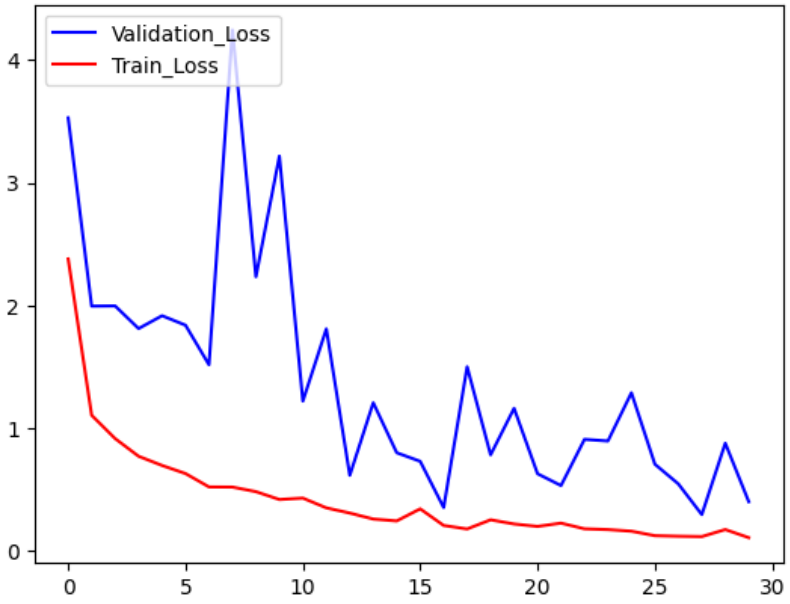
```
34/34 [==============================] - 134s 4s/step - loss: 0.4852 - accuracy: 0.8382 - val_loss: 2.2344 - val_accuracy: 0.3281
Epoch 10/30
34/34 [==============================] - 138s 4s/step - loss: 0.4214 - accuracy: 0.8493 - val_loss: 3.2161 - val_accuracy: 0.2812
Epoch 11/30
34/34 [==============================] - 143s 4s/step - loss: 0.4327 - accuracy: 0.8474 - val_loss: 1.2219 - val_accuracy: 0.5156
Epoch 12/30
34/34 [==============================] - 158s 5s/step - loss: 0.3533 - accuracy: 0.8860 - val_loss: 1.8103 - val_accuracy: 0.5625
Epoch 13/30
34/34 [==============================] - 159s 5s/step - loss: 0.3109 - accuracy: 0.8842 - val_loss: 0.6181 - val_accuracy: 0.7812
Epoch 14/30
34/34 [==============================] - 168s 5s/step - loss: 0.2623 - accuracy: 0.9099 - val_loss: 1.2089 - val_accuracy: 0.7031
Epoch 15/30
34/34 [==============================] - 160s 5s/step - loss: 0.2469 - accuracy: 0.9136 - val_loss: 0.8019 - val_accuracy: 0.7812
Epoch 16/30
34/34 [==============================] - 138s 4s/step - loss: 0.3453 - accuracy: 0.8842 - val_loss: 0.7320 - val_accuracy: 0.8281
Epoch 17/30
34/34 [==============================] - 122s 4s/step - loss: 0.2100 - accuracy: 0.9265 - val_loss: 0.3558 - val_accuracy: 0.8750
Epoch 18/30
34/34 [==============================] - 125s 4s/step - loss: 0.1816 - accuracy: 0.9393 - val_loss: 1.5022 - val_accuracy: 0.5781
Epoch 19/30
34/34 [==============================] - 119s 3s/step - loss: 0.2560 - accuracy: 0.9191 - val_loss: 0.7847 - val_accuracy: 0.8281
Epoch 20/30
34/34 [==============================] - 117s 3s/step - loss: 0.2222 - accuracy: 0.9265 - val_loss: 1.1636 - val_accuracy: 0.7344
Epoch 21/30
34/34 [==============================] - 127s 4s/step - loss: 0.2024 - accuracy: 0.9393 - val_loss: 0.6311 - val_accuracy: 0.8281
Epoch 22/30
34/34 [==============================] - 118s 3s/step - loss: 0.2290 - accuracy: 0.9283 - val_loss: 0.5347 - val_accuracy: 0.8594
Epoch 23/30
34/34 [==============================] - 118s 3s/step - loss: 0.1828 - accuracy: 0.9393 - val_loss: 0.9107 - val_accuracy: 0.7656
Epoch 24/30
34/34 [==============================] - 118s 3s/step - loss: 0.1765 - accuracy: 0.9430 - val_loss: 0.8976 - val_accuracy: 0.7031
Epoch 25/30
34/34 [==============================] - 119s 3s/step - loss: 0.1634 - accuracy: 0.9393 - val_loss: 1.2900 - val_accuracy: 0.6406
Epoch 26/30
34/34 [==============================] - 122s 4s/step - loss: 0.1269 - accuracy: 0.9632 - val_loss: 0.7088 - val_accuracy: 0.7812
```

```
import pandas as pd
#pip install openpyxl

# Convert the training history to a DataFrame
history_df = pd.DataFrame(history_resnet18.history)

# Save the DataFrame to an Excel file
history_df.to_excel('History_RESNET_B_SB 4.xlsx', index=False)
```

```
fig = plt.figure()
plt.plot(history_resnet18.history['val_accuracy'], color='teal', label='Validation_Accuracy')
plt.plot(history_resnet18.history['accuracy'], color='orange', label='Train_Accuracy')
plt.legend(loc="upper left")
plt.show()
```




```
fig = plt.figure()
plt.plot(history_resnet18.history['val_loss'], color='blue', label='Validation_Loss')
plt.plot(history_resnet18.history['loss'], color='red', label='Train_Loss')
plt.legend(loc="upper left")
plt.show()
```



```
model_resnet18.evaluate(test)
```

```
4/4 [==============================] - 10s 818ms/step - loss: 0.3301 - accuracy: 0.8750
[0.3300582766532898, 0.875]
```

```
model_resnet18.save('model_resnet.h5')
```

```
c:\Program Files\Python311\Lib\site-packages\keras\src\engine\training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This f
  saving_api.save_model(
```

Prediksi Model AlexNet

```
import tensorflow as tf
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
modelAlex = tf.keras.models.load_model('/content/drive/MyDrive/Colab Notebooks/Testing UAS/model_alexnet-1.h5')
```

```
modelAlex.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 55, 55, 96)        34944

 max_pooling2d (MaxPooling2  (None, 27, 27, 96)        0
 D)

 conv2d_1 (Conv2D)           (None, 27, 27, 64)        153664

 max_pooling2d_1 (MaxPoolin  (None, 13, 13, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 13, 13, 128)       73856

 conv2d_3 (Conv2D)           (None, 13, 13, 128)       147584

 conv2d_4 (Conv2D)           (None, 13, 13, 64)        73792

 max_pooling2d_2 (MaxPoolin  (None, 6, 6, 64)          0
 g2D)

 flatten (Flatten)           (None, 2304)              0
```

```
 dense (Dense)               (None, 512)               1180160

 dense_1 (Dense)             (None, 256)               131328

 dropout (Dropout)           (None, 256)               0

 dense_2 (Dense)             (None, 4)                 1028

=================================================================
Total params: 1796356 (6.85 MB)
Trainable params: 1796356 (6.85 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

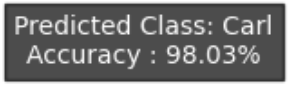
```
data = tf.keras.utils.image_dataset_from_directory('/content/drive/MyDrive/Colab Notebooks/Testing UAS/Dataset')
print(data.class_names)
class_names=data.class_names
```

```
Found 679 files belonging to 4 classes.
['Bolt', 'Carl', 'Olaf', 'Woody']
```

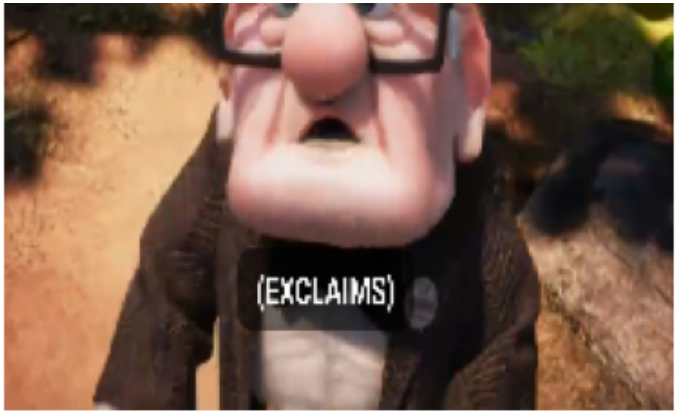
```
    def predict_test(img_path, model):
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        plt.imshow(img)

        img = cv2.resize(img, (227, 227))
        img = img / 255.0

        test_img = np.expand_dims(img, axis=0)
        print(test_img.shape)

        predictions = []

        pred = model.predict(test_img)
        print(pred)
        print(np.argmax(pred))
        accuracy = pred[0][np.argmax(pred)] * 100

        predicted_class = class_names[np.argmax(pred)]

        plt.imshow(img)
        plt.axis('off')  # Turn off axis labels

        text = "Predicted Class: {}\nAccuracy : {:.2f}%".format(predicted_class, accuracy)
        plt.text(0.5, -0.15, text, ha='center', va='center', transform=plt.gca().transAxes, color='white', fontsize=10, bbox=dict(facecolor='black', a

        predictions.append((predicted_class, accuracy))
        # Show the plot
        plt.show()

        return predictions
```

```
import os as os

data_dir = '/content/drive/MyDrive/Colab Notebooks/Test/Carl/'

for image in os.listdir(data_dir):
    print(image)
    predict_test(data_dir + image, modelAlex)
```

```
Carl9.jpg
(1, 227, 227, 3)
1/1 [==============================] - 0s 39ms/step
[[2.1135653e-04 9.8030061e-01 3.5020593e-03 1.5985822e-02]]
1
```

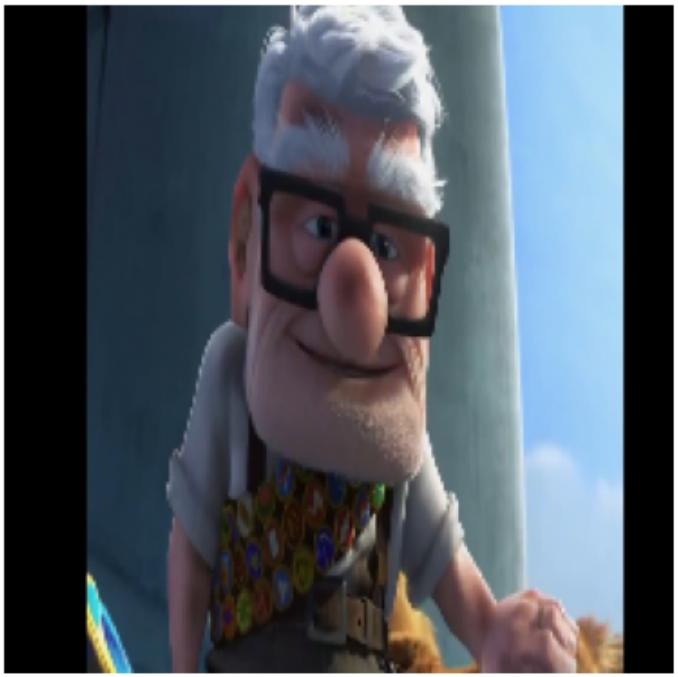


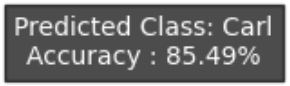

```
Carl6.jpg
(1, 227, 227, 3)
1/1 [==============================] - 0s 35ms/step
[[2.4218903e-11 1.0000000e+00 3.4797128e-12 5.4650244e-14]]
1
```



Predicted Class: Carl
Accuracy : 100.00%

```
Carl8.jpg
(1, 227, 227, 3)
1/1 [==============================] - 0s 33ms/step
[[0.00351563 0.8549298  0.1278831  0.01367147]]
1
```





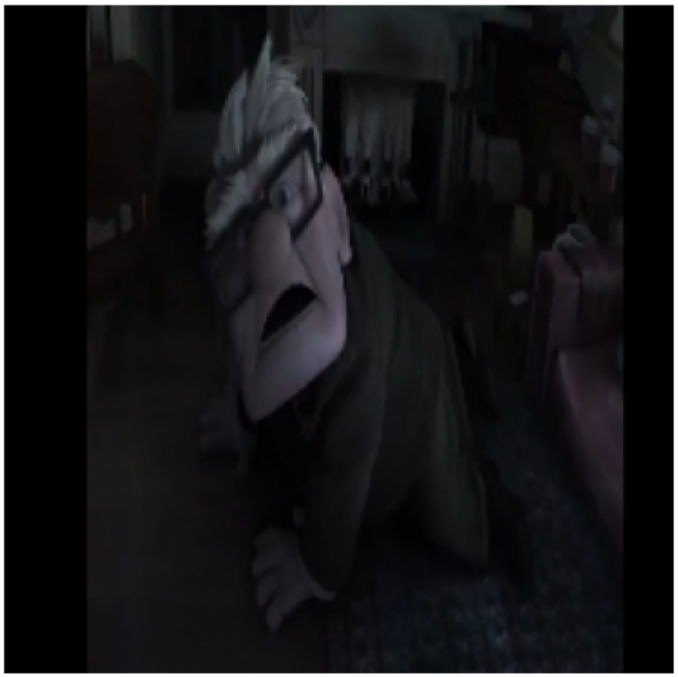

```
Carl7.jpg
(1, 227, 227, 3)
1/1 [==============================] - 0s 34ms/step
[[8.0240426e-13 1.0000000e+00 3.4895978e-11 5.0873531e-13]]
1
```

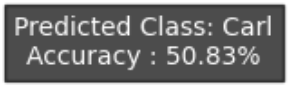

```
Carl2.jpg
(1, 227, 227, 3)
1/1 [==============================] - 0s 34ms/step
[[1.2435667e-05 9.9302930e-01 6.9155768e-03 4.2732143e-05]]
1
```

Predicted Class: Carl
Accuracy : 99.30%

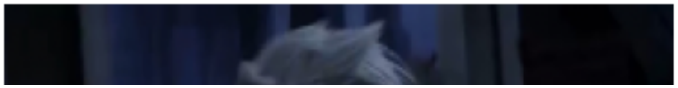
```
Carl3.jpg
(1, 227, 227, 3)
1/1 [==============================] - 0s 35ms/step
[[9.477745e-09 1.000000e+00 4.921103e-11 9.298457e-12]]
1
```

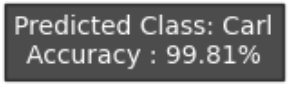

```
Carl4.jpg
(1, 227, 227, 3)
1/1 [==============================] - 0s 36ms/step
[[0.14522214 0.5083394  0.22999911 0.11643942]]
1
```






```
Carl5.jpg
(1, 227, 227, 3)
1/1 [==============================] - 0s 33ms/step
[[2.4797948e-04 9.9809307e-01 1.6243928e-03 3.4613910e-05]]
1
```

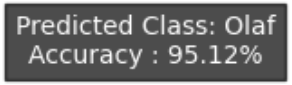




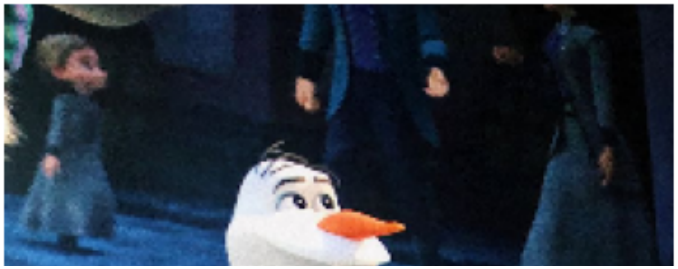

```
Carl10.png
(1, 227, 227, 3)
1/1 [==============================] - 0s 34ms/step
[[6.5550096e-12 1.0000000e+00 7.9214102e-10 2.4091408e-12]]
1
```

```
import os as os

data_dir = '/content/drive/MyDrive/Colab Notebooks/Test/Olaf/'

for image in os.listdir(data_dir):
    print(image)
    predict_test(data_dir + image, modelAlex)
```

```
Olaf1.jpeg
(1, 227, 227, 3)
1/1 [==============================] - 0s 61ms/step
[[1.0680542e-03 4.7725797e-02 9.5120323e-01 2.9112505e-06]]
2
```

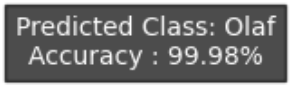



```
Olaf8.jpeg
(1, 227, 227, 3)
1/1 [==============================] - 0s 57ms/step
[[4.2892689e-01 1.2686551e-06 5.7070822e-01 3.6368682e-04]]
2
```

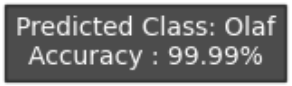


Predicted Class: Olaf
Accuracy : 57.07%

```
Olaf9.jpeg
(1, 227, 227, 3)
1/1 [==============================] - 0s 57ms/step
[[1.6294005e-05 2.3125311e-04 9.9975222e-01 2.8452118e-07]]
2
```



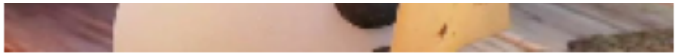

```
Olaf5.jpeg
(1, 227, 227, 3)
1/1 [==============================] - 0s 56ms/step
[[2.7254296e-07 1.5566113e-05 9.9989641e-01 8.7705273e-05]]
2
```



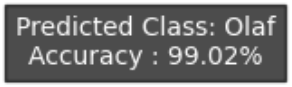

```
Olaf2.jpeg
(1, 227, 227, 3)
1/1 [==============================] - 0s 54ms/step
[[1.1779328e-05 8.5758939e-03 9.9138230e-01 3.0009598e-05]]
2
```

Predicted Class: Olaf
Accuracy : 99.14%

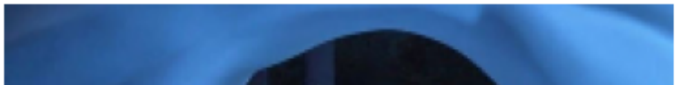
```
Olaf3.jpeg
(1, 227, 227, 3)
1/1 [==============================] - 0s 62ms/step
[[4.0845300e-09 1.0000000e+00 1.3303805e-08 2.5754730e-08]]
1
```




```
Olaf7.jpeg
(1, 227, 227, 3)
1/1 [==============================] - 0s 33ms/step
[[8.6036380e-03 6.6701847e-04 9.9015093e-01 5.7843316e-04]]
2
```

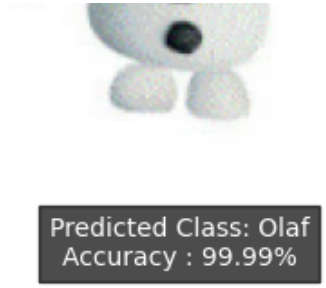



```
Olaf4.jpeg
(1, 227, 227, 3)
1/1 [==============================] - 0s 33ms/step
[[3.8909020e-05 1.9656903e-07 9.9996090e-01 4.4829123e-09]]
2
```



Predicted Class: Olaf
Accuracy : 100.00%

```
Olaf10.gif
(1, 227, 227, 3)
1/1 [==============================] - 0s 33ms/step
[[2.0789570e-05 5.8590067e-05 9.9992001e-01 5.9320570e-07]]
2
```




```
import os as os

data_dir = '/content/drive/MyDrive/Colab Notebooks/Test/Woody/'

for image in os.listdir(data_dir):
    print(image)
    predict_test(data_dir + image, modelAlex)
```

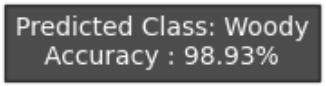
```
Woody1.jpeg
(1, 227, 227, 3)
1/1 [==============================] - 0s 37ms/step
[[9.9615418e-09 9.1137481e-10 5.2367458e-08 9.9999988e-01]]
3
```

Predicted Class: Woody
Accuracy : 100.00%

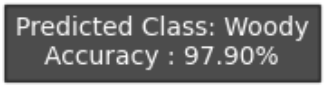
```
Woody3.jpg
(1, 227, 227, 3)
1/1 [==============================] - 0s 35ms/step
[[9.4711379e-15 2.5802433e-11 2.2767627e-10 1.0000000e+00]]
3
```

Predicted Class: Woody
Accuracy : 100.00%

```
Woody2.jpg
(1, 227, 227, 3)
1/1 [==============================] - 0s 33ms/step
[[5.27786653e-07 1.06034465e-02 6.00681997e-05 9.89335954e-01]]
3
```



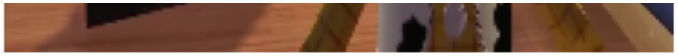

```
Woody4.png
(1, 227, 227, 3)
1/1 [==============================] - 0s 34ms/step
[[7.6578301e-04 1.9678794e-02 5.8907899e-04 9.7896636e-01]]
3
```

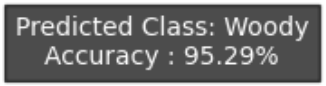



```
Woody5.png
(1, 227, 227, 3)
1/1 [==============================] - 0s 34ms/step
[[5.8215554e-03 2.2216456e-03 4.9255174e-05 9.9190760e-01]]
3
```

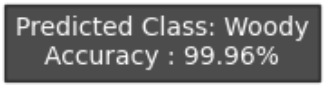
Predicted Class: Woody
Accuracy : 99.19%

```
Woody6.png
(1, 227, 227, 3)
1/1 [==============================] - 0s 36ms/step
[[0.00106689 0.04325056 0.00280513 0.95287746]]
3
```





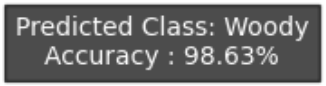

```
Woody7.png
(1, 227, 227, 3)
1/1 [==============================] - 0s 45ms/step
[[2.7862113e-04 2.1863581e-05 7.6254430e-05 9.9962318e-01]]
3
```




```
Woody8.png
(1, 227, 227, 3)
1/1 [==============================] - 0s 53ms/step
[[7.2526839e-04 1.2253028e-05 1.0972007e-05 9.9925143e-01]]
3
```

Predicted Class: Woody
Accuracy : 99.93%

```
Woody9.png
(1, 227, 227, 3)
1/1 [==============================] - 0s 57ms/step
[[9.3127172e-03 3.6978573e-03 7.0503535e-04 9.8628438e-01]]
3
```



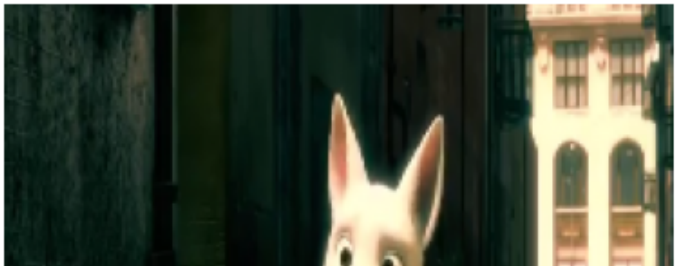

```
Woody10.png
(1, 227, 227, 3)
1/1 [==============================] - 0s 51ms/step
[[0.00950536 0.00113181 0.00297467 0.98638815]]
3
```

```
import os as os

data_dir = '/content/drive/MyDrive/Colab Notebooks/Test/Bolt/'

for image in os.listdir(data_dir):
    print(image)
    predict_test(data_dir + image, modelAlex)
```

```
Bolt2.png
(1, 227, 227, 3)
1/1 [==============================] - 0s 37ms/step
[[0.9788278  0.01417834 0.00436928 0.00262458]]
0
```

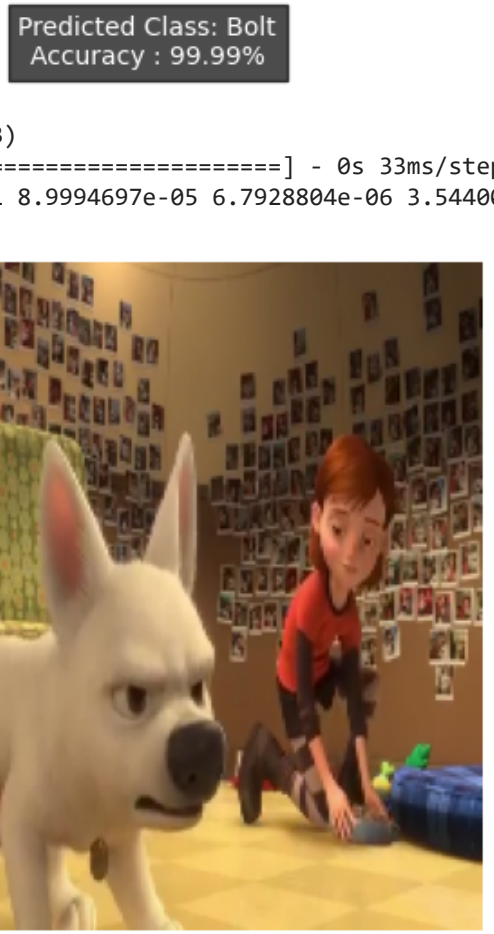
Predicted Class: Bolt
Accuracy : 97.88%

```
Bolt3.png
(1, 227, 227, 3)
1/1 [==============================] - 0s 33ms/step
[[7.3388082e-01 2.6507410e-01 1.0384995e-03 6.5045115e-06]]
0
```

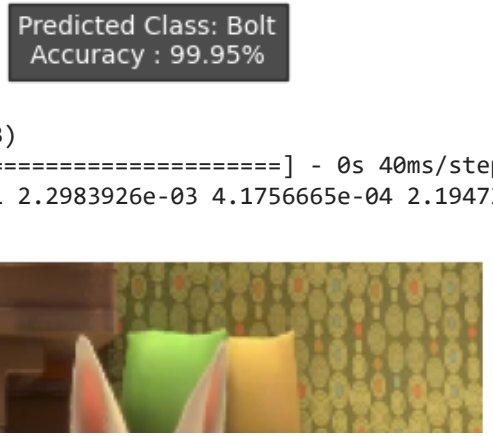


Predicted Class: Bolt
Accuracy : 73.39%

```
Bolt10.png
(1, 227, 227, 3)
1/1 [==============================] - 0s 45ms/step
[[9.9987519e-01 2.7159767e-05 1.2653221e-06 9.6271004e-05]]
0
```



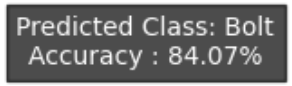

```
Bolt4.png
(1, 227, 227, 3)
1/1 [==============================] - 0s 33ms/step
[[9.9954885e-01 8.9994697e-05 6.7928804e-06 3.5440037e-04]]
0
```



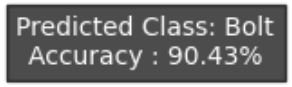

```
Bolt9.png
(1, 227, 227, 3)
1/1 [==============================] - 0s 40ms/step
[[9.9706465e-01 2.2983926e-03 4.1756665e-04 2.1947315e-04]]
0
```

Predicted Class: Bolt
Accuracy : 99.71%

```
Bolt7.png
(1, 227, 227, 3)
1/1 [==============================] - 0s 35ms/step
[[0.8407498  0.05740992 0.01890336 0.08293697]]
0
```




```
Bolt8.png
(1, 227, 227, 3)
1/1 [==============================] - 0s 44ms/step
[[9.0433443e-01 9.5540479e-02 7.9789112e-05 4.5288376e-05]]
0
```




```
Bolt1.jpg
(1, 227, 227, 3)
1/1 [==============================] - 0s 34ms/step
[[9.1851097e-01 7.9061992e-02 2.4254145e-03 1.5987795e-06]]
0
```


```
Bolt6.png
(1, 227, 227, 3)
1/1 [==============================] - 0s 31ms/step
[[0.90616006 0.00386495 0.0059636  0.08401146]]
0
```

Prediksi Model VGGNet

```
modelVGG = tf.keras.models.load_model('/content/drive/MyDrive/Colab Notebooks/Testing UAS/model_vgg16.h5')
```

```
modelVGG.summary()
```

```
max_pooling2d (MaxPooling2  (None, 112, 112, 32)      0
D)

conv2d_2 (Conv2D)           (None, 112, 112, 64)      18496

conv2d_3 (Conv2D)           (None, 112, 112, 64)      36928

batch_normalization_1 (Bat  (None, 112, 112, 64)      256
chNormalization)

max_pooling2d_1 (MaxPoolin  (None, 56, 56, 64)        0
g2D)

conv2d_4 (Conv2D)           (None, 56, 56, 128)       73856

conv2d_5 (Conv2D)           (None, 56, 56, 128)       147584

conv2d_6 (Conv2D)           (None, 56, 56, 128)       147584

batch_normalization_2 (Bat  (None, 56, 56, 128)       512
chNormalization)

max_pooling2d_2 (MaxPoolin  (None, 28, 28, 128)       0
g2D)

conv2d_7 (Conv2D)           (None, 28, 28, 256)       295168

conv2d_8 (Conv2D)           (None, 28, 28, 256)       590080

conv2d_9 (Conv2D)           (None, 28, 28, 256)       590080

batch_normalization_3 (Bat  (None, 28, 28, 256)       1024
chNormalization)

max_pooling2d_3 (MaxPoolin  (None, 14, 14, 256)       0
g2D)
```

```
max_pooling2d_4 (MaxPoolin  (None, 7, 7, 256)        0
g2D)

flatten (Flatten)           (None, 12544)            0

dense (Dense)               (None, 128)              1605760

dropout (Dropout)           (None, 128)              0
```

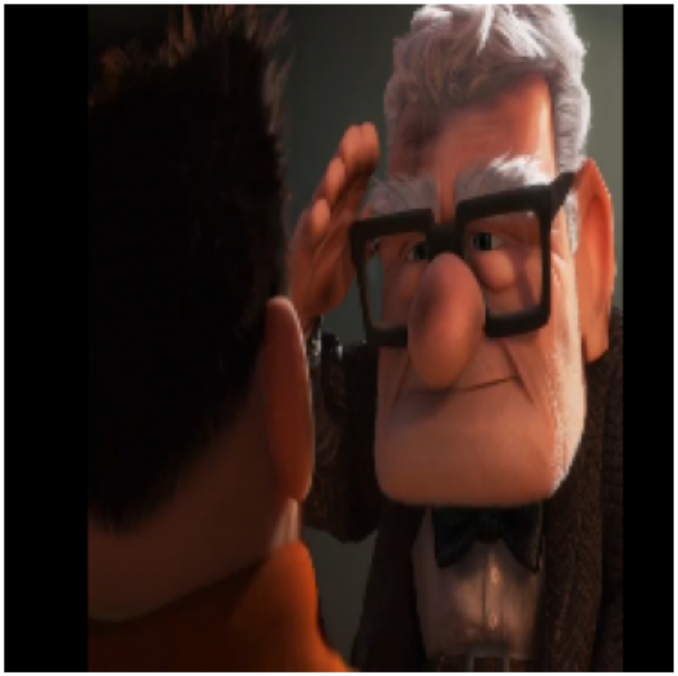
```
data = tf.keras.utils.image_dataset_from_directory('/content/drive/MyDrive/Colab Notebooks/Testing UAS/Dataset')
print(data.class_names)
class_names=data.class_names
```

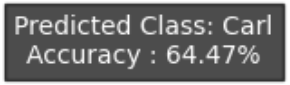
```
Found 679 files belonging to 4 classes.
['Bolt', 'Carl', 'Olaf', 'Woody']
```

```
    def predict_test(img_path, model):
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        plt.imshow(img)

        img = cv2.resize(img, (224, 224))
        img = img / 255.0

        test_img = np.expand_dims(img, axis=0)
        print(test_img.shape)

        predictions = []

        pred = model.predict(test_img)
        print(pred)
        print(np.argmax(pred))
        accuracy = pred[0][np.argmax(pred)] * 100

        predicted_class = class_names[np.argmax(pred)]

        plt.imshow(img)
        plt.axis('off')  # Turn off axis labels

        text = "Predicted Class: {}\nAccuracy : {:.2f}%".format(predicted_class, accuracy)
        plt.text(0.5, -0.15, text, ha='center', va='center', transform=plt.gca().transAxes, color='white', fontsize=10, bbox=dict(facecolor='black', a

        predictions.append((predicted_class, accuracy))
        # Show the plot
        plt.show()

        return predictions
```

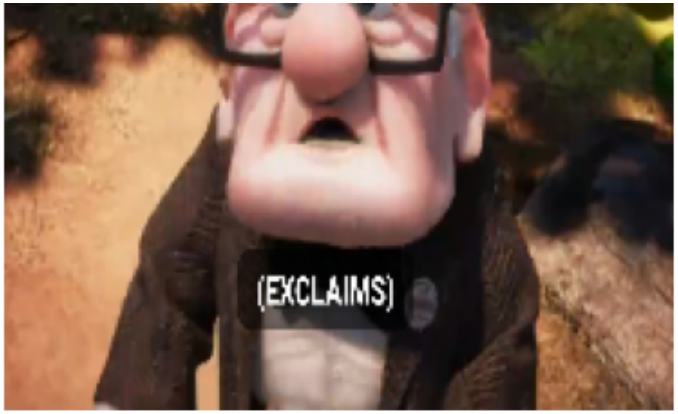
```
import os as os

data_dir = '/content/drive/MyDrive/Colab Notebooks/Test/Carl/'

for image in os.listdir(data_dir):
    print(image)
    predict_test(data_dir + image, modelVGG)
```

```
Carl9.jpg
(1, 224, 224, 3)
1/1 [==============================] - 0s 474ms/step
[[0.10282443 0.64469033 0.02054717 0.23193811]]
1
```



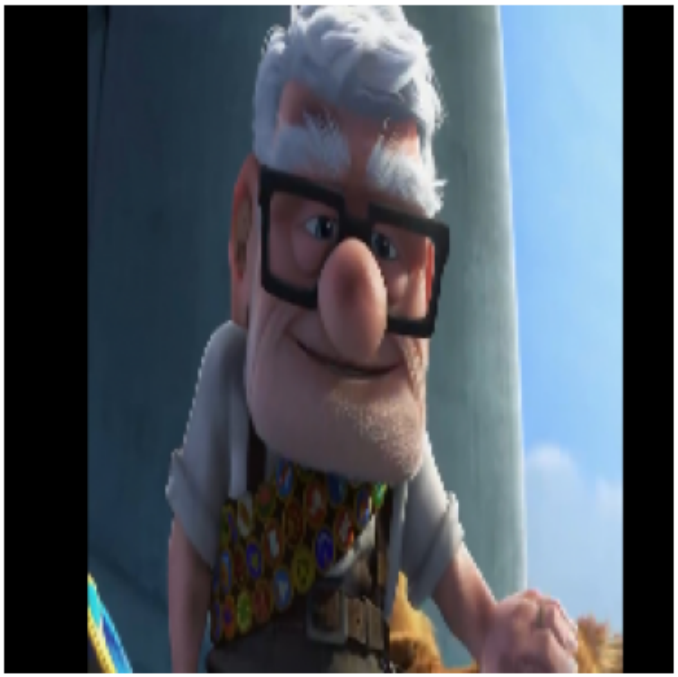



```
Carl6.jpg
(1, 224, 224, 3)
1/1 [==============================] - 0s 483ms/step
[[0.10007715 0.88328266 0.01011206 0.00652811]]
1
```



Predicted Class: Carl
Accuracy : 88.33%

```
Carl8.jpg
(1, 224, 224, 3)
1/1 [==============================] - 0s 321ms/step
[[0.09018173 0.7345402  0.07944535 0.09583267]]
1
```



Predicted Class: Carl
Accuracy : 73.45%

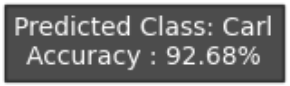
```
Carl7.jpg
(1, 224, 224, 3)
1/1 [==============================] - 0s 176ms/step
[[3.5853533e-03 9.9616903e-01 1.4623700e-04 9.9410943e-05]]
1
```

Predicted Class: Carl
Accuracy : 99.62%

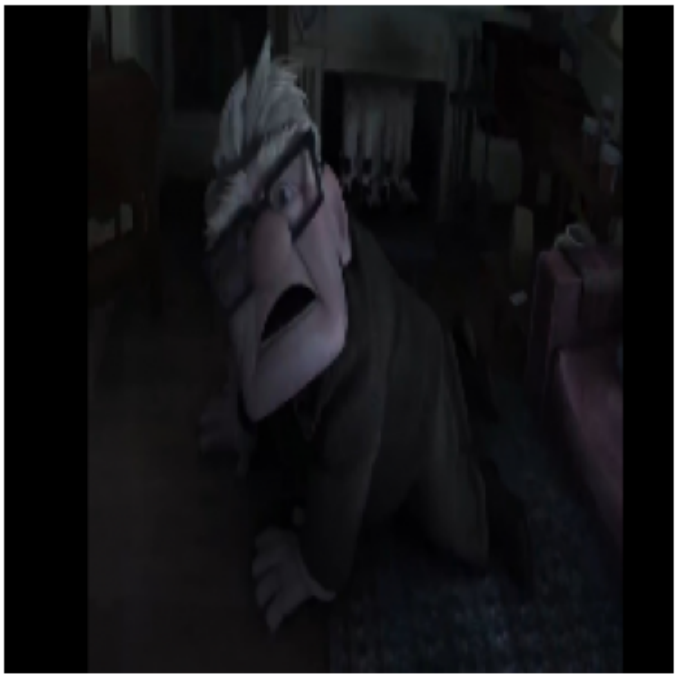
```
Carl2.jpg
(1, 224, 224, 3)
1/1 [==============================] - 0s 179ms/step
[[0.0310961  0.94129074 0.02510729 0.00250584]]
1
```

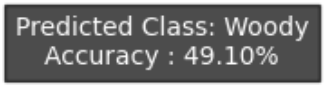
Predicted Class: Carl
Accuracy : 94.13%

```
Carl3.jpg
(1, 224, 224, 3)
1/1 [==============================] - 0s 195ms/step
[[0.06477115 0.9268057  0.00306606 0.00535715]]
1
```

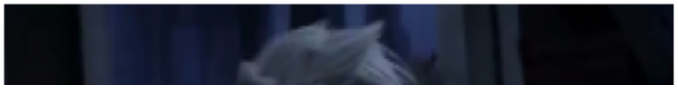



```
Carl4.jpg
(1, 224, 224, 3)
1/1 [==============================] - 0s 186ms/step
[[0.25555712 0.09552394 0.15790695 0.49101207]]
3
```






```
Carl5.jpg
(1, 224, 224, 3)
1/1 [==============================] - 0s 195ms/step
[[0.27885607 0.1523939  0.12383376 0.44491628]]
3
```

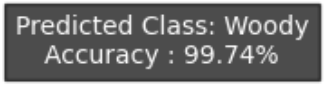


Predicted Class: Woody
Accuracy : 44.49%

```
Carl10.png
(1, 224, 224, 3)
1/1 [==============================] - 0s 195ms/step
[[2.4002607e-03 9.9749547e-01 5.4966331e-05 4.9328297e-05]]
1
```

```
import os as os

data_dir = '/content/drive/MyDrive/Colab Notebooks/Test/Woody/'

for image in os.listdir(data_dir):
    print(image)
    predict_test(data_dir + image, modelVGG)
```

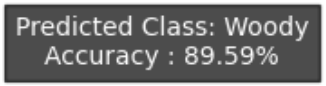
```
Woody1.jpeg
(1, 224, 224, 3)
1/1 [==============================] - 0s 168ms/step
[[4.3496687e-04 9.5808727e-04 1.2224481e-03 9.9738449e-01]]
3
```



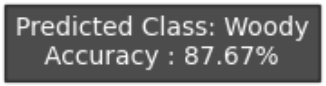

```
Woody3.jpg
(1, 224, 224, 3)
1/1 [==============================] - 0s 188ms/step
[[0.0050824  0.00618972 0.00957461 0.9791533 ]]
3
```

Predicted Class: Woody
Accuracy : 97.92%

```
Woody2.jpg
(1, 224, 224, 3)
1/1 [==============================] - 0s 177ms/step
[[0.05803839 0.02962398 0.01640307 0.8959345 ]]
3
```



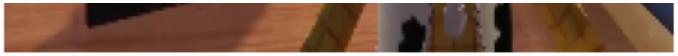

```
Woody4.png
(1, 224, 224, 3)
1/1 [==============================] - 0s 273ms/step
[[0.01582211 0.10027951 0.00724133 0.87665707]]
3
```

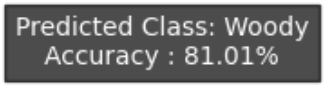



```
Woody5.png
(1, 224, 224, 3)
1/1 [==============================] - 0s 186ms/step
[[0.00789998 0.01507407 0.0147282  0.96229774]]
3
```

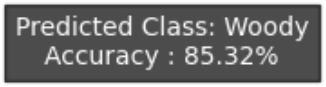
Predicted Class: Woody
Accuracy : 96.23%

```
Woody6.png
(1, 224, 224, 3)
1/1 [==============================] - 0s 180ms/step
[[0.0404716  0.08158326 0.06787181 0.8100734 ]]
3
```






```
Woody7.png
(1, 224, 224, 3)
1/1 [==============================] - 0s 219ms/step
[[0.04294319 0.01685954 0.08696512 0.85323215]]
3
```




```
Woody8.png
(1, 224, 224, 3)
1/1 [==============================] - 0s 259ms/step
[[0.05976226 0.02301965 0.05532119 0.8618969 ]]
3
```

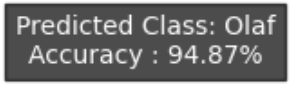
Predicted Class: Woody
Accuracy : 86.19%

```
Woody9.png
(1, 224, 224, 3)
1/1 [==============================] - 0s 291ms/step
[[0.04535459 0.03548438 0.01471423 0.9044468 ]]
3
```

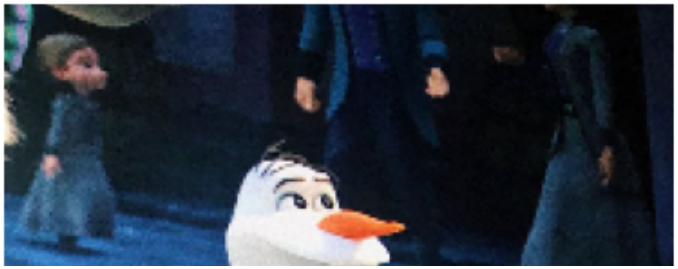
Predicted Class: Woody
Accuracy : 90.44%

```
import os as os

data_dir = '/content/drive/MyDrive/Colab Notebooks/Test/Olaf/'

for image in os.listdir(data_dir):
    print(image)
    predict_test(data_dir + image, modelVGG)
```

```
Olaf1.jpeg
(1, 224, 224, 3)
1/1 [==============================] - 0s 324ms/step
[[0.00342016 0.04088474 0.9487261  0.00696888]]
2
```

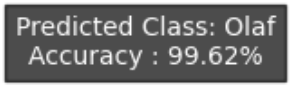



```
Olaf8.jpeg
(1, 224, 224, 3)
1/1 [==============================] - 0s 307ms/step
[[0.03519905 0.01061784 0.93499213 0.01919099]]
2
```

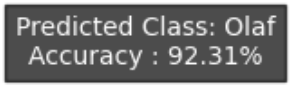


Predicted Class: Olaf
Accuracy : 93.50%

```
Olaf9.jpeg
(1, 224, 224, 3)
1/1 [==============================] - 0s 316ms/step
[[9.2227099e-04 2.5820481e-03 9.9619055e-01 3.0512101e-04]]
2
```




```
Olaf5.jpeg
(1, 224, 224, 3)
1/1 [==============================] - 0s 336ms/step
[[0.03117753 0.03569004 0.9230715  0.010061  ]]
2
```

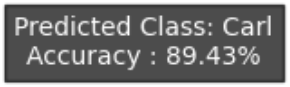



```
Olaf2.jpeg
(1, 224, 224, 3)
1/1 [==============================] - 0s 328ms/step
[[0.00509602 0.08849736 0.89333063 0.01307597]]
2
```

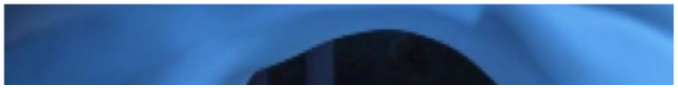
Predicted Class: Olaf
Accuracy : 89.33%

```
Olaf3.jpeg
(1, 224, 224, 3)
1/1 [==============================] - 0s 322ms/step
[[0.03879487 0.95540035 0.00278262 0.0030221 ]]
1
```

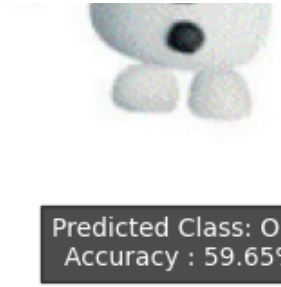

```
Olaf7.jpeg
(1, 224, 224, 3)
1/1 [==============================] - 0s 326ms/step
[[0.08512501 0.8943055  0.0123642  0.00820516]]
1
```




```
Olaf4.jpeg
(1, 224, 224, 3)
1/1 [==============================] - 0s 185ms/step
[[0.02066253 0.11631531 0.7636757  0.09934651]]
2
```




```
Olaf10.gif
(1, 224, 224, 3)
1/1 [==============================] - 0s 201ms/step
[[0.08915217 0.0559843  0.5964755  0.25838807]]
2
```



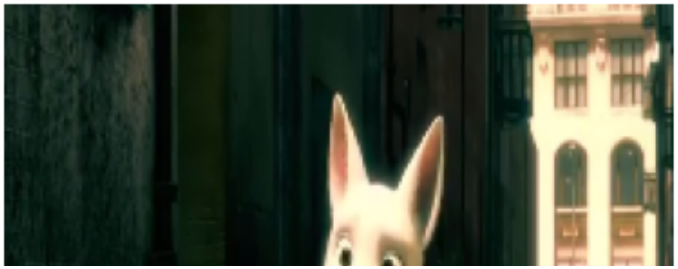

```
Olaf6.jpg
(1, 224, 224, 3)
1/1 [==============================] - 0s 209ms/step
[[0.05837734 0.14564012 0.78752166 0.00846097]]
2
```

```
import os as os

data_dir = '/content/drive/MyDrive/Colab Notebooks/Test/Bolt/'

for image in os.listdir(data_dir):
    print(image)
    predict_test(data_dir + image, modelVGG)
```

```
Bolt2.png
(1, 224, 224, 3)
1/1 [==============================] - 1s 521ms/step
[[0.51115555 0.34006462 0.01672738 0.1320525 ]]
0
```

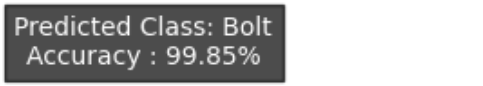
Predicted Class: Bolt
Accuracy : 51.12%

```
Bolt3.png
(1, 224, 224, 3)
1/1 [==============================] - 0s 321ms/step
[[0.8955111  0.02119838 0.01784425 0.06544622]]
0
```

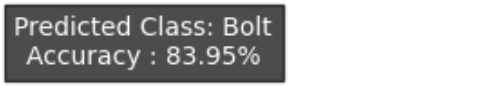


Predicted Class: Bolt
Accuracy : 89.55%

```
Bolt10.png
(1, 224, 224, 3)
1/1 [==============================] - 0s 303ms/step
[[9.9854255e-01 5.5684400e-04 1.9631130e-04 7.0435158e-04]]
0
```




```
Bolt4.png
(1, 224, 224, 3)
1/1 [==============================] - 0s 445ms/step
[[0.8394728  0.03675658 0.00842995 0.11534066]]
0
```




```
Bolt9.png
(1, 224, 224, 3)
1/1 [==============================] - 0s 473ms/step
[[0.76074725 0.20501451 0.00993759 0.02430057]]
0
```

Predicted Class: Bolt
Accuracy : 76.07%

```
Bolt7.png
(1, 224, 224, 3)
1/1 [==============================] - 0s 228ms/step
[[0.40319416 0.0985778  0.05713125 0.4410968 ]]
3
```

Predicted Class: Woody
Accuracy : 44.11%

```
Bolt8.png
(1, 224, 224, 3)
1/1 [==============================] - 0s 181ms/step
[[0.38876298 0.1914589  0.05551683 0.3642613 ]]
0
```

Predicted Class: Bolt
Accuracy : 38.88%

```
Bolt1.jpg
(1, 224, 224, 3)
1/1 [==============================] - 0s 182ms/step
[[0.46278185 0.1568579  0.28703463 0.09332555]]
0
```

Predicted Class: Bolt
Accuracy : 46.28%

```
Bolt6.png
(1, 224, 224, 3)
1/1 [==============================] - 0s 183ms/step
[[0.39275962 0.45049995 0.01203997 0.14470053]]
1
```

Prediksi Model ResNet

```
modelRes = tf.keras.models.load_model('/content/drive/MyDrive/Colab Notebooks/Testing UAS/model_resnet.h5')
```

```
modelAlex.summary()
```

```
Model: "sequential"
_________________________________________________________________
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 55, 55, 96)        34944

 max_pooling2d (MaxPooling2  (None, 27, 27, 96)        0
 D)

 conv2d_1 (Conv2D)           (None, 27, 27, 64)        153664

 max_pooling2d_1 (MaxPoolin  (None, 13, 13, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 13, 13, 128)       73856

 conv2d_3 (Conv2D)           (None, 13, 13, 128)       147584

 conv2d_4 (Conv2D)           (None, 13, 13, 64)        73792

 max_pooling2d_2 (MaxPoolin  (None, 6, 6, 64)          0
 g2D)

 flatten (Flatten)           (None, 2304)              0

 dense (Dense)               (None, 512)               1180160

 dense_1 (Dense)             (None, 256)               131328

 dropout (Dropout)           (None, 256)               0

 dense_2 (Dense)             (None, 4)                 1028

=================================================================
Total params: 1796356 (6.85 MB)
Trainable params: 1796356 (6.85 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
data = tf.keras.utils.image_dataset_from_directory('/content/drive/MyDrive/Colab Notebooks/Testing UAS/Dataset')
print(data.class_names)
class_names=data.class_names
```

```
Found 679 files belonging to 4 classes.
['Bolt', 'Carl', 'Olaf', 'Woody']
```

```
    def predict_test(img_path, model):
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        plt.imshow(img)

        img = cv2.resize(img, (224, 224))
        img = img / 255.0

        test_img = np.expand_dims(img, axis=0)
        print(test_img.shape)

        predictions = []

        pred = model.predict(test_img)
        print(pred.flatten()[:10])
        print(np.argmax(pred))
        accuracy = pred[0][np.argmax(pred)] * 100

        predicted_class = class_names[np.argmax(pred)]

        plt.imshow(img)
        plt.axis('off')  # Turn off axis labels

        text = "Predicted Class: {}\nAccuracy : {:.2f}%".format(predicted_class, accuracy)
        plt.text(0.5, -0.15, text, ha='center', va='center', transform=plt.gca().transAxes, color='white', fontsize=10, bbox=dict(facecolor='black', a

        predictions.append((predicted_class, accuracy))
        # Show the plot
        plt.show()

        return predictions
```

```
import os as os

data_dir = '/content/drive/MyDrive/Colab Notebooks/Test/Carl/'

for image in os.listdir(data_dir):
    print(image)
    predict_test(data_dir + image, modelRes)
```

Predicted Class: Carl
Accuracy : 99.93%

```
Carl1.jpg
(1, 224, 224, 3)
1/1 [==============================] - 0s 113ms/step
[1.4032157e-05 9.9998581e-01 1.6656497e-07 2.0080018e-09 1.8560329e-12
 2.9487180e-11 6.0297067e-12 4.7009050e-12 2.6333397e-12 1.7971839e-11]
1
```

Predicted Class: Carl
Accuracy : 100.00%