In [1]:
```
from google.colab import drive
drive.mount("/content/drive")
```

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

In [3]:
```
import os
from PIL import Image

# Set the path to your dataset directory
dataset_dir = '/content/drive/MyDrive/waterbottle'

# Initialize an empty list to store the image data
image_list = []

# Iterate over the images in the dataset directory
for filename in os.listdir(dataset_dir):
    if filename.endswith('.jpg') or filename.endswith('.png'):  # Adjust the file
        image_path = os.path.join(dataset_dir, filename)
        try:
            # Open the image using PIL
            image = Image.open(image_path)
            # Convert the image data to a list
            image_data = list(image.getdata())
            # Append the image data to the list
            image_list.append(image_data)
        except Exception as e:
            print(f"Error processing image: {filename}")
            print(e)

# Print the total number of images converted to a list
print(f"Total images converted: {len(image_list)}")
```

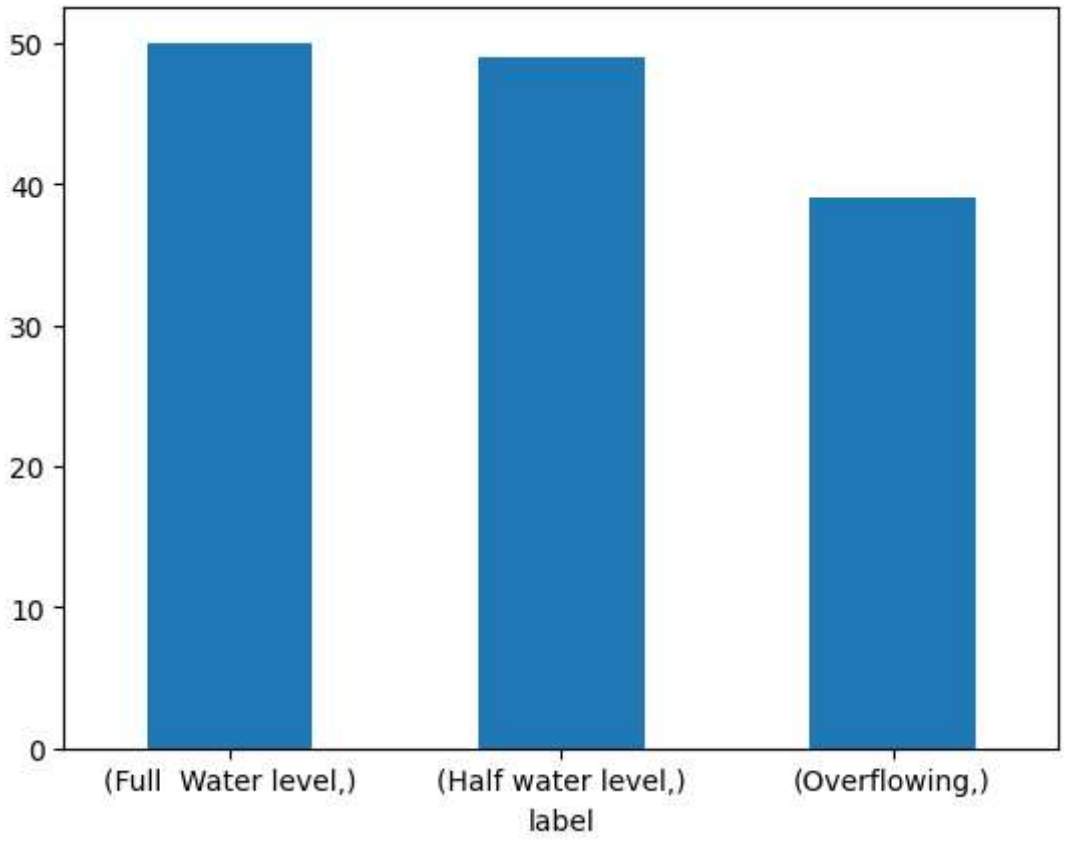
```
Total images converted: 0
```

In [13]:
```
import os
import cv2
import numpy as np

import warnings
warnings.filterwarnings('ignore') # Hide all warnings

data = []
labels = []
input_size = 64
image_size = (input_size, input_size)

# Access the directory and sub-directories and so on
# directory = "water-bottle-dataset"
directory = "/content/drive/MyDrive/waterbottle"

# Extract all images file inside the folders and stored them into list
for sub_folder in os.listdir(directory):
    sub_folder_path = os.path.join(directory, sub_folder)
    # for sub_sub_folder in os.listdir(sub_folder_path):
    #     sub_sub_folder_path = os.path.join(sub_folder_path, sub_sub_folder)
    for image_file in os.listdir(sub_folder_path):
        if image_file.endswith(".jpeg") or image_file.endswith(".png"): # Check t
            image_path = os.path.join(sub_folder_path, image_file)
            # Read the image using OpenCV
            image = cv2.imread(image_path) # the decoded images stored in **B G R
            # Resize the image to a standard size
            image = cv2.resize(image, image_size)
            # Append the image to the data list
            data.append(image)
            # Append the label to the labels list
            labels.append(sub_folder)

# Convert the data and labels lists into numpy arrays
data = np.array(data)
labels = np.array(labels)

# Print the dimension of the dataset
print(f'data shape: {data.shape}')
print(f'labels shape: {labels.shape}')
```

```
data shape: (138, 64, 64, 3)
labels shape: (138,)
```

In [14]:

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.DataFrame({"label":labels})
df.value_counts().plot(kind='bar')
plt.xticks(rotation = 0) # Rotates X-Axis Ticks by 45-degrees
plt.show()
```



In [15]:
```
# Generate augmented data

from keras.preprocessing.image import ImageDataGenerator

# Load the data
X = data # array of preprocessed data
y = labels # array of labels
n_gen = 40

# Create data generator
datagen = ImageDataGenerator(
        rotation_range=0, #0
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')

# Fit the data generator on the data
datagen.fit(X)

# Generate augmented data
X_augmented, y_augmented = [], []

'''
1st Option multiple dataset with same ratio
'''
# # Non resampling
# for X_batch, y_batch in datagen.flow(X, y, batch_size=32):
#     X_augmented.append(X_batch)
#     y_augmented.append(y_batch)
#     if len(X_augmented) >= 100: # Setting generated augmented data
#         break

'''
2nd Option resampling with equaly labels ratio
'''
# With resampling
for X_batch, y_batch in datagen.flow(X[:308], y[:308], batch_size=32):
    X_augmented.append(X_batch)
    y_augmented.append(y_batch)
    if len(X_augmented) >= n_gen: # Setting generated augmented data
        break

for X_batch, y_batch in datagen.flow(X[308:447], y[308:447], batch_size=32):
    X_augmented.append(X_batch)
    y_augmented.append(y_batch)
    if len(X_augmented) >= n_gen*2.3: # Setting generated augmented data
        break

for X_batch, y_batch in datagen.flow(X[447:], y[447:], batch_size=32):
    X_augmented.append(X_batch)
    y_augmented.append(y_batch)
    if len(X_augmented) >= n_gen*4.2: # Setting generated augmented data
        break
```

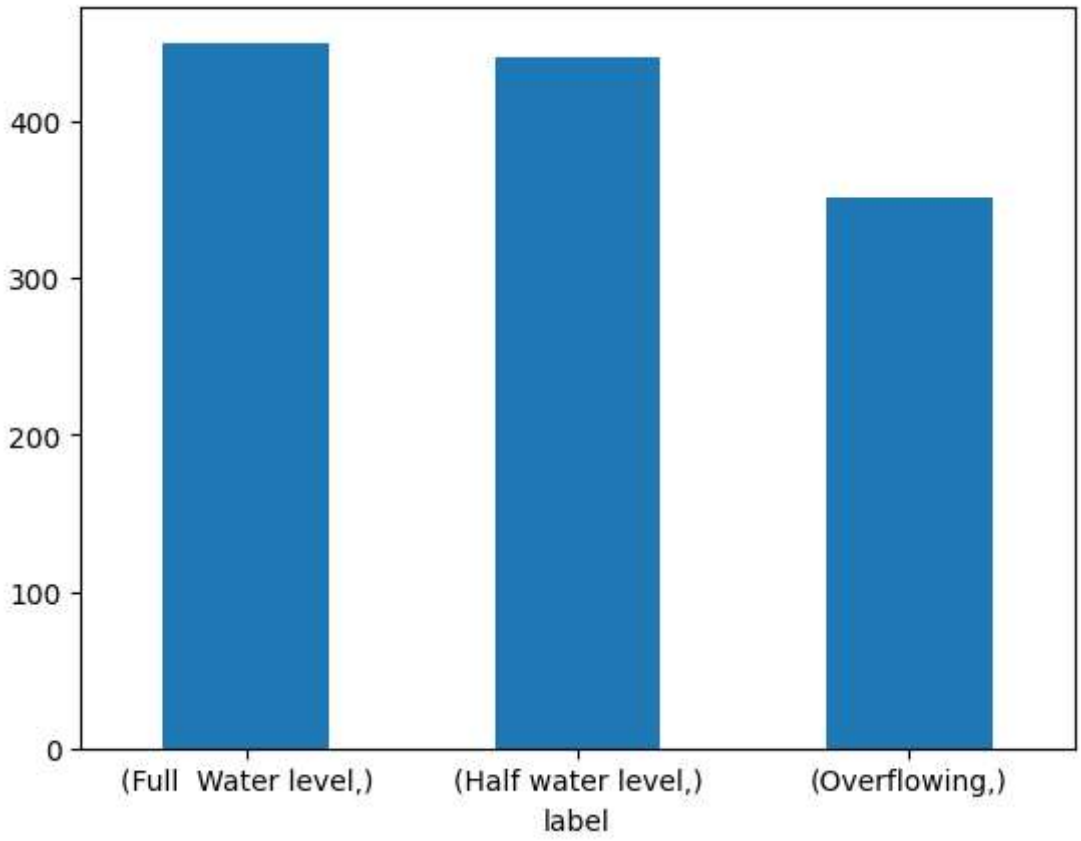
```
# Concatenate augmented data with original data
data = np.concatenate((X, np.concatenate(X_augmented)))
labels = np.concatenate((y, np.concatenate(y_augmented)))

print(f"data augmented shape : {data.shape}")
print(f"labels augmented shape : {labels.shape}")

import pandas as pd
df = pd.DataFrame({"label":labels})
df.value_counts()
```

```
data augmented shape : (1242, 64, 64, 3)
labels augmented shape : (1242,)
```

Out[15]:

```
label
Full  Water level    450
Half water level     441
Overflowing          351
dtype: int64
```

In [16]:
```
'''
See how many numbers of each labels.
After I regenerated data.
'''

import pandas as pd
import matplotlib.pyplot as plt
df = pd.DataFrame({"label":labels})
df.value_counts().plot(kind='bar')
plt.xticks(rotation = 0) # Rotates X-Axis Ticks by 45-degrees
plt.show()
```



In [17]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2,

data = X_train # Split training data
labels = y_train # Split training labels

X_test = X_test # Test data
y_test = y_test # Test labels
```

In [18]:
```
import pandas as pd

print(f'data shape:{data.shape}')
print(f'labels shape:{labels.shape}')
df = pd.DataFrame({"label":labels})
print(df.value_counts())
print("")
print(f'test_date shape:{X_test.shape}')
print(f'test_labels shape:{y_test.shape}')
df = pd.DataFrame({"test_labels":y_test})
print(df.value_counts())
```

```
data shape:(993, 64, 64, 3)
labels shape:(993,)
label
Half water level     363
Full  Water level    362
Overflowing          268
dtype: int64

test_date shape:(249, 64, 64, 3)
test_labels shape:(249,)
test_labels
Full  Water level    88
Overflowing          83
Half water level     78
dtype: int64
```

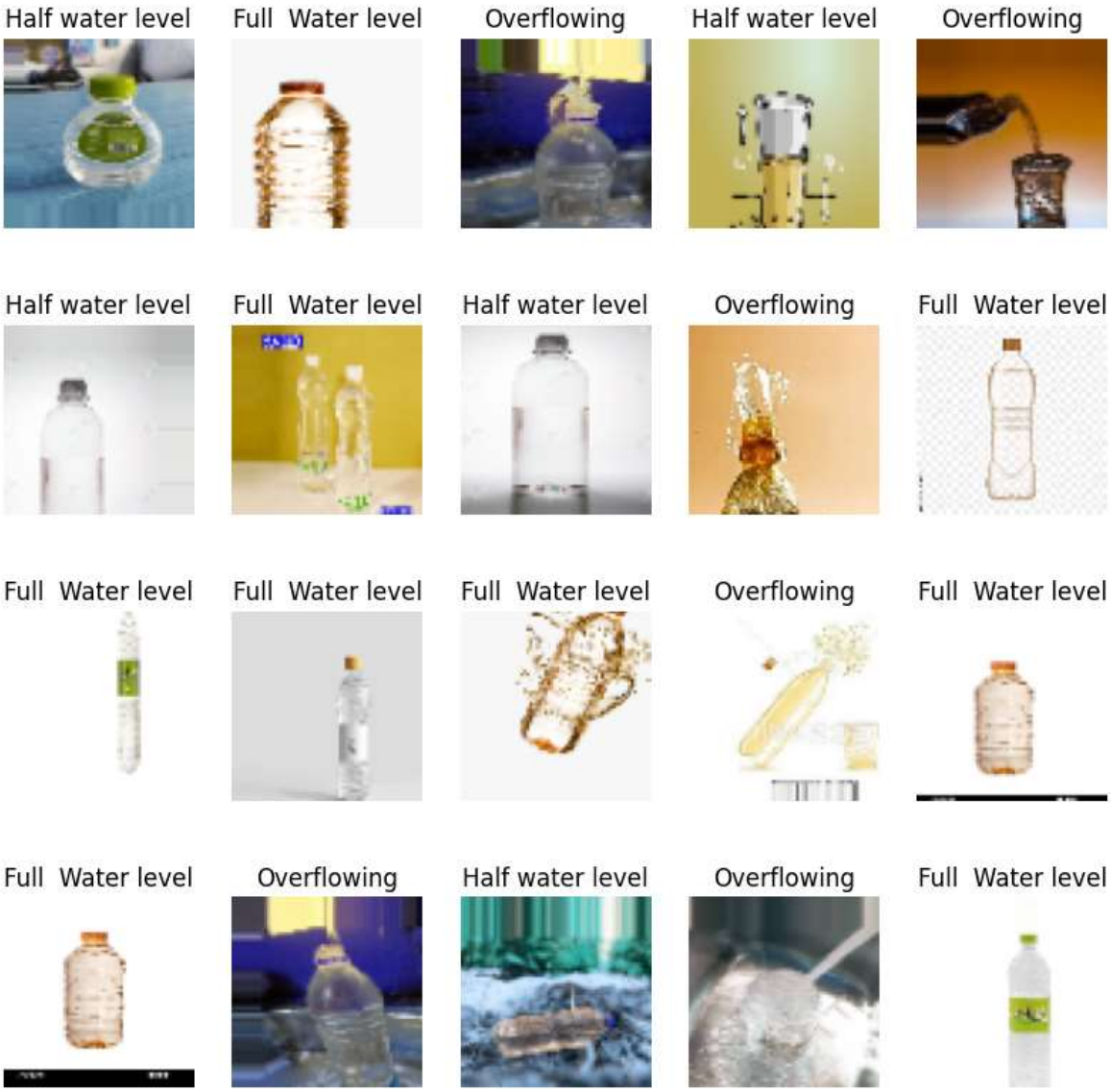
In [19]:
```
# Normalize the pixel values to a range between 0 and 1
data = data / 255.0
X_test = X_test / 225.0
```

In [20]:
```
labels = labels
# Convert the labels into one-hot encoded arrays
labels_one_hot = np.zeros((labels.shape[0], 3))

for i, label in enumerate(labels):
    if label == "Full  Water level":
        labels_one_hot[i, 0] = 1
    elif label == "Half water level":
        labels_one_hot[i, 1] = 1
    else:
        labels_one_hot[i, 2] = 1
```

In [21]:
```
# Show converted output
print(labels_one_hot[0])
```

```
[0. 0. 1.]
```

In [22]:
```
'''
Show a sample of images from the dataset
'''

import matplotlib.pyplot as plt

# Load the data
data = data

# choose 20 random indices
indices = np.random.randint(0, len(data), 20)

# Get 20 sample images
sample_images = data[indices]

# Plot the images
fig = plt.figure(figsize=(10,10))
for i, img in enumerate(sample_images):
    plt.subplot(4, 5, i+1)
    plt.imshow(img)
    plt.axis('off')
    plt.title(labels[indices[i]])

plt.show()
```


Half water level
Full Water level
Overflowing
Half water level
Overflowing
Half water level
Full Water level
Half water level
Overflowing
Full Water level
Full Water level
Full Water level
Full Water level
Overflowing
Full Water level
Full Water level
Overflowing
Half water level
Overflowing
Full Water level

In [23]:
```
'''
# Save augmented images to specific directory --- Uncomment to use
# create new directory to save augmented images
import os

# Check existing directory, if not: crate new directory
if not os.path.exists("augmented_images"):
    os.makedirs("augmented_images")

augmented_data = data
labels = labels
# loop through each image in the augmented data
for i, image in enumerate(augmented_data):
    # convert the image back to its original form
    image = (image).astype("uint8")

    # save the image to the new directory
    cv2.imwrite(f"augmented_images/augmented_{labels[i]}_{i}.jpeg", image)
'''
```

Out[23]:
```
'\n# Save augmented images to specific directory --- Uncomment to use\n# create
new directory to save augmented images\nimport os\n\n# Check existing director
y, if not: crate new directory\nif not os.path.exists("augmented_images"):\n
os.makedirs("augmented_images")\n\naugmented_data = data\nlabels = labels\n# lo
op through each image in the augmented data\nfor i, image in enumerate(augmente
d_data):\n    # convert the image back to its original form\n    image = (imag
e).astype("uint8")\n    \n    # save the image to the new directory\n    cv2.im
write(f"augmented_images/augmented_{labels[i]}_{i}.jpeg", image)\n'
```

CNN MODEL

In [24]:
```
def run_custom_model(batch_size, epochs):

    import tensorflow as tf
    from tensorflow import keras
    from tensorflow.keras import layers
    from tensorflow.keras.optimizers import Adam, SGD

    # set seed value for randomization
    # np.random.seed(42)
    tf.random.set_seed(42)

    # Build the model using a Convolutional Neural Network
    model = keras.Sequential([
        keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(input_size
        keras.layers.Conv2D(32, (3,3), activation='relu'),
        keras.layers.MaxPooling2D(2,2),
        keras.layers.Dropout(0.2),

        keras.layers.Conv2D(64, (3,3), activation='relu'),
        keras.layers.Conv2D(64, (3,3), activation='relu'),
        keras.layers.MaxPooling2D(2,2),
        keras.layers.Dropout(0.2),

        keras.layers.Conv2D(256, (3,3), activation='relu'),
        keras.layers.Conv2D(256, (3,3), activation='relu'),
        keras.layers.MaxPooling2D(2,2),
        keras.layers.Dropout(0.2),

        keras.layers.Flatten(),
        keras.layers.Dense(1024, activation='relu'),
        keras.layers.Dropout(0.5),
        keras.layers.Dense(3, activation='softmax')
    ])


    # Compile the model
    model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['ac

    # See an overview of the model architecture and to debug issues related to th
    model.summary()

###############################################################################

    import time
    start_time = time.time() #To show the training time

    # Train the model

    # set an early stopping mechanism
    # set patience to be tolerant against random validation loss increases
    early_stopping = tf.keras.callbacks.EarlyStopping(patience=5)

    # history = model.fit(data, labels_one_hot, batch_size=32, epochs=10, validat
    history = model.fit(x=data,
                        y=labels_one_hot,
                        batch_size=batch_size,
```

```
                    epochs=epochs,
                    validation_split=0.2,)
#                     callbacks=[early_stopping])

# Evaluate the model
print("Test accuracy: ", max(history.history['val_accuracy']))

# Assign the trained model
self_train_model = history

end_time = time.time() # To show the training time
training_time = end_time - start_time
print("Training time:", training_time, "seconds")

self_train_model_time = training_time

return self_train_model, self_train_model_time
```

Res modief

In [25]:
```
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Model
import tensorflow as tf

import time
start_time = time.time() #To show the training time

X=data
y=labels_one_hot

# set seed value for randomization
tf.random.set_seed(42)

# Load pre-trained ResNet50 model
resnet = ResNet50(include_top=False, input_shape=(input_size, input_size, 3))

# Freeze layers in ResNet50 model
for layer in resnet.layers:
    layer.trainable = False

# Add new classification layers
x = Flatten()(resnet.output)
x = Dense(128, activation='relu')(x)
x = Dense(3, activation='softmax')(x)

# Create new model
model = Model(inputs=resnet.input, outputs=x)

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura

# Train the model
history = model.fit(X, y, epochs=100, batch_size=256, validation_split=0.2)

# Evaluate the model
print("Test accuracy: ", max(history.history['val_accuracy']))

# Assign the trained model
pre_train_model = history

end_time = time.time() # To show the training time
training_time = end_time - start_time
print("Training time:", training_time, "seconds")

pre_train_model_time = training_time
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applic
ations/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5 (https://
storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights
_tf_dim_ordering_tf_kernels_notop.h5)
94765736/94765736 [==============================] - 1s 0us/step
Epoch 1/100
4/4 [==============================] - 25s 5s/step - loss: 1.2579 - accurac
y: 0.3791 - val_loss: 1.3082 - val_accuracy: 0.3719
Epoch 2/100
```

```
4/4 [==============================] - 18s 5s/step - loss: 1.3703 - accurac
y: 0.3791 - val_loss: 1.0740 - val_accuracy: 0.4774
Epoch 3/100
4/4 [==============================] - 18s 4s/step - loss: 1.1070 - accurac
y: 0.4219 - val_loss: 1.2494 - val_accuracy: 0.3266
Epoch 4/100
4/4 [==============================] - 18s 5s/step - loss: 1.0870 - accurac
y: 0.4270 - val_loss: 1.0639 - val_accuracy: 0.3869
Epoch 5/100
4/4 [==============================] - 19s 5s/step - loss: 1.0619 - accurac
```

In [26]:

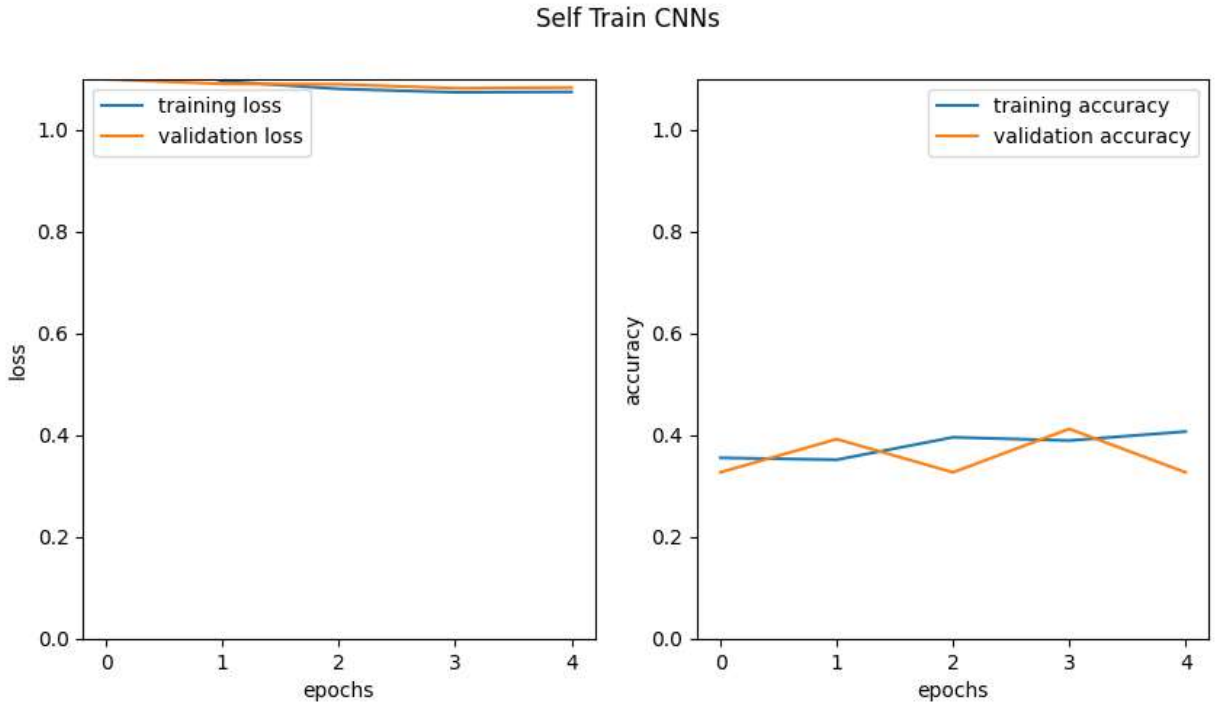
```python
def plot_model_loss_and_acc(model, name):
    import matplotlib.pyplot as plt

    # Assign model to variable 'history'
    history = model

    # Set Figure size
    plt.figure(figsize=(10,5))

    # Plot the training and validation loss
    plt.subplot(1,2,1)
    plt.plot(history.history['loss'], label='training loss')
    plt.plot(history.history['val_loss'], label='validation loss')
    plt.xlabel('epochs')
    plt.ylabel('loss')
    plt.legend()
    plt.ylim(0,1.1)



    # Plot the training and validation accuracy
    plt.subplot(1,2,2)
    plt.plot(history.history['accuracy'], label='training accuracy')
    plt.plot(history.history['val_accuracy'], label='validation accuracy')
    plt.xlabel('epochs')
    plt.ylabel('accuracy')
    plt.legend()
    plt.ylim(0,1.1)

    plt.suptitle(name)
    plt.show()
```

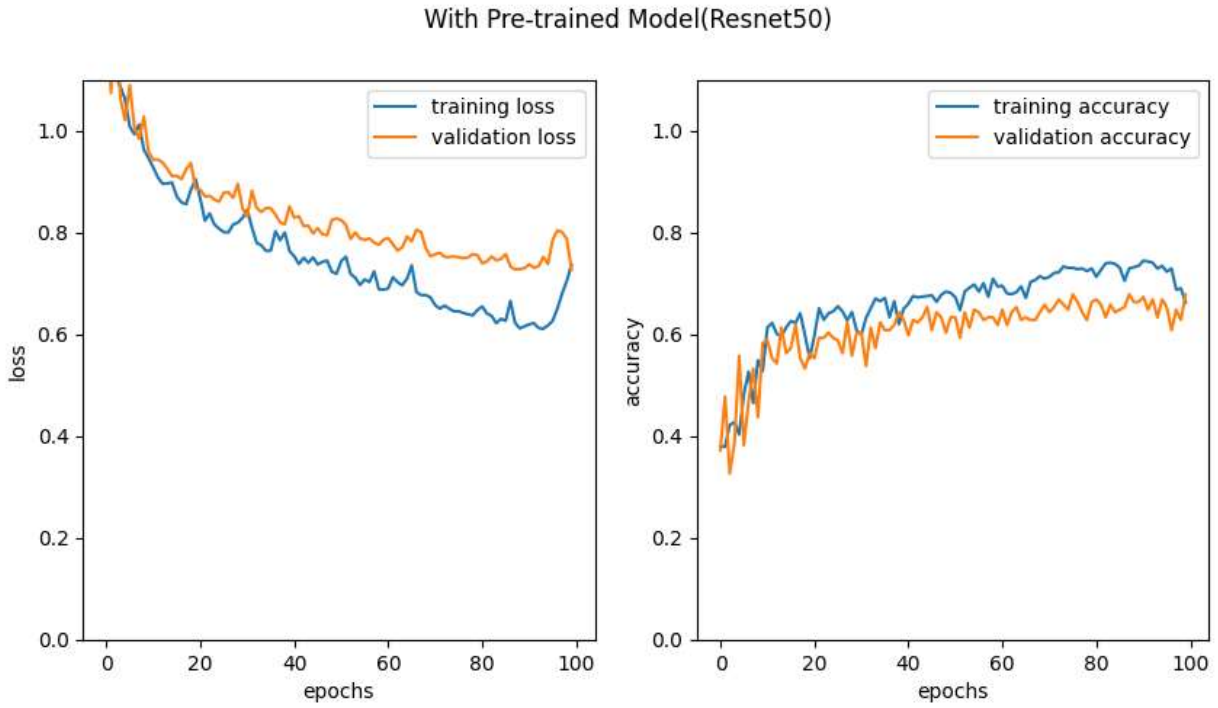
In [27]:
```
# Run CNN model
self_train_model, self_train_model_time = run_custom_model(batch_size = 256,epoch
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 32)        896

 conv2d_1 (Conv2D)           (None, 60, 60, 32)        9248

 max_pooling2d (MaxPooling2D  (None, 30, 30, 32)       0
 )

 dropout (Dropout)           (None, 30, 30, 32)        0

 conv2d_2 (Conv2D)           (None, 28, 28, 64)        18496

 conv2d_3 (Conv2D)           (None, 26, 26, 64)        36928

 max_pooling2d_1 (MaxPooling  (None, 13, 13, 64)       0
 2D)

 dropout_1 (Dropout)         (None, 13, 13, 64)        0

 conv2d_4 (Conv2D)           (None, 11, 11, 256)       147712

 conv2d_5 (Conv2D)           (None, 9, 9, 256)         590080

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 256)        0
 2D)

 dropout_2 (Dropout)         (None, 4, 4, 256)         0

 flatten_1 (Flatten)         (None, 4096)              0

 dense_2 (Dense)             (None, 1024)              4195328

 dropout_3 (Dropout)         (None, 1024)              0

 dense_3 (Dense)             (None, 3)                 3075

=================================================================
Total params: 5,001,763
Trainable params: 5,001,763
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
4/4 [==============================] - 33s 7s/step - loss: 1.1598 - accuracy:
0.3552 - val_loss: 1.0991 - val_accuracy: 0.3266
Epoch 2/5
4/4 [==============================] - 34s 7s/step - loss: 1.0954 - accuracy:
0.3514 - val_loss: 1.0900 - val_accuracy: 0.3920
Epoch 3/5
4/4 [==============================] - 31s 7s/step - loss: 1.0802 - accuracy:
0.3955 - val_loss: 1.0893 - val_accuracy: 0.3266
```

```
Epoch 4/5
4/4 [==============================] - 30s 7s/step - loss: 1.0734 - accuracy:
0.3892 - val_loss: 1.0815 - val_accuracy: 0.4121
Epoch 5/5
4/4 [==============================] - 32s 7s/step - loss: 1.0742 - accuracy:
0.4068 - val_loss: 1.0829 - val_accuracy: 0.3266
Test accuracy:  0.41206029057502747
Training time: 203.80512976646423 seconds
```

In [28]:
```
plot_model_loss_and_acc(self_train_model, 'Self Train CNNs')
plot_model_loss_and_acc(pre_train_model, 'With Pre-trained Model(Resnet50)')
```





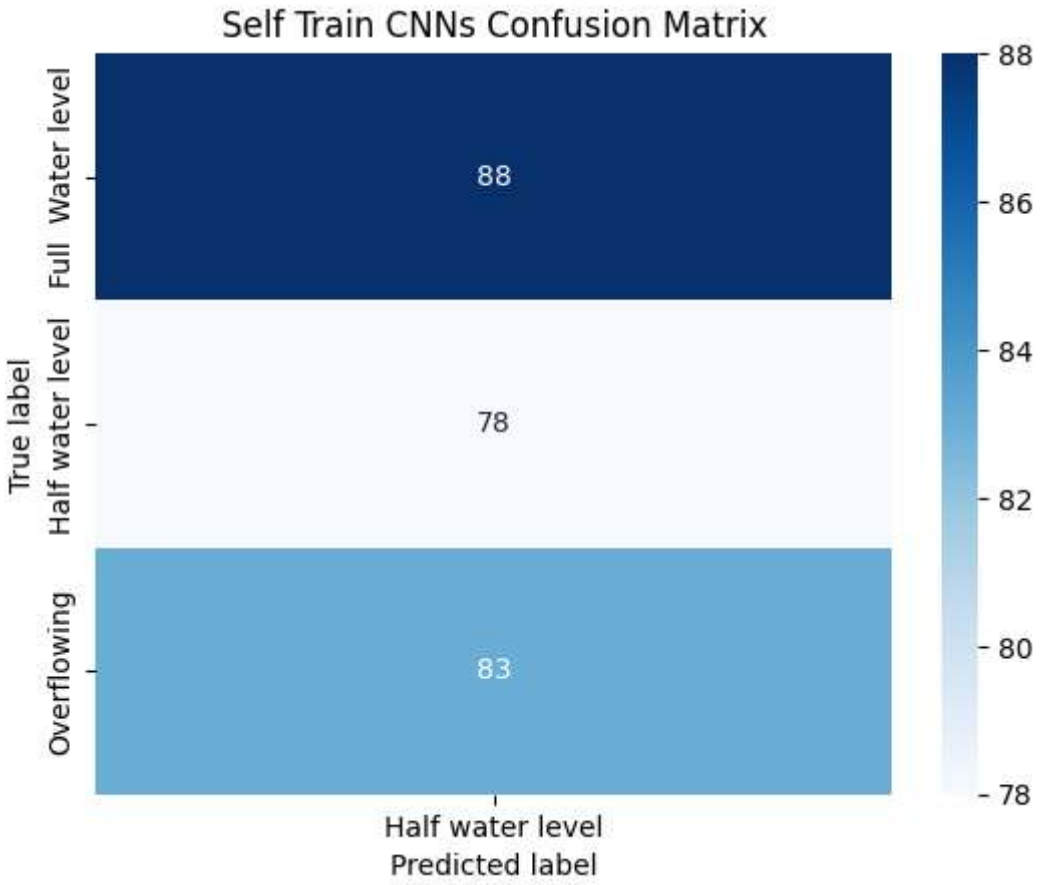
In [29]:
```
'''
Convert np.ndarray(n,3) into List of predicted labels
'''
def output_converter(model_output):

    import numpy as np

    output = model_output

    # assume that 'output' is a numpy array of shape (n, 3)
    output_labels = ['Full  Water level', 'Half water level', 'Overflowing']
    predictions = np.argmax(output, axis=1)
    predicted_labels = [output_labels[p] for p in predictions]

    return predicted_labels
```

In [30]:
```
'''
Plot a Heatmap-Crosstab table out of predicted labels and True labels
'''
def plot_hm_ct(y_true, y_pred):
    import pandas as pd
    import seaborn as sns
    import matplotlib.pyplot as plt

    # create a DataFrame from y_true and y_pred
    df = pd.DataFrame({'y_true': y_true, 'y_pred': y_pred})

    # create cross-tabulation matrix
    ctab = pd.crosstab(df['y_true'], df['y_pred'])

    # create heatmap using seaborn
    sns.heatmap(ctab, annot=True, cmap='Blues', fmt='d')

    # add labels and title
    plt.xlabel('Predicted label')
    plt.ylabel('True label')
    plt.title('Confusion Matrix')

    # show the plot
    plt.show()
```

In [32]:
```
'''
Generate confusion matrix from trained model
'''
def generate_cf(model, name):

    import pandas as pd
    import seaborn as sns
    import matplotlib.pyplot as plt

    # Assign model to variable 'history'
    history = model

    # Load output data
    y_pred = output_converter(history.model.predict(X_test))
    y_true = y_test

    # Plot the confusion matrix
    # create a DataFrame from y_true and y_pred
    df = pd.DataFrame({'y_true': y_true, 'y_pred': y_pred})

    # create cross-tabulation matrix
    ctab = pd.crosstab(df['y_true'], df['y_pred'])

    # create heatmap using seaborn
    sns.heatmap(ctab, annot=True, cmap='Blues', fmt='d')

    # add labels and title
    plt.xlabel('Predicted label')
    plt.ylabel('True label')
    plt.title('{} Confusion Matrix'.format(name))

    # show the plot
    plt.show()

    # Calculate accuracy score
    from sklearn.metrics import accuracy_score
    accuracy = accuracy_score(y_true, y_pred)
    print("{} accuracy score: {}".format(name, accuracy))
```

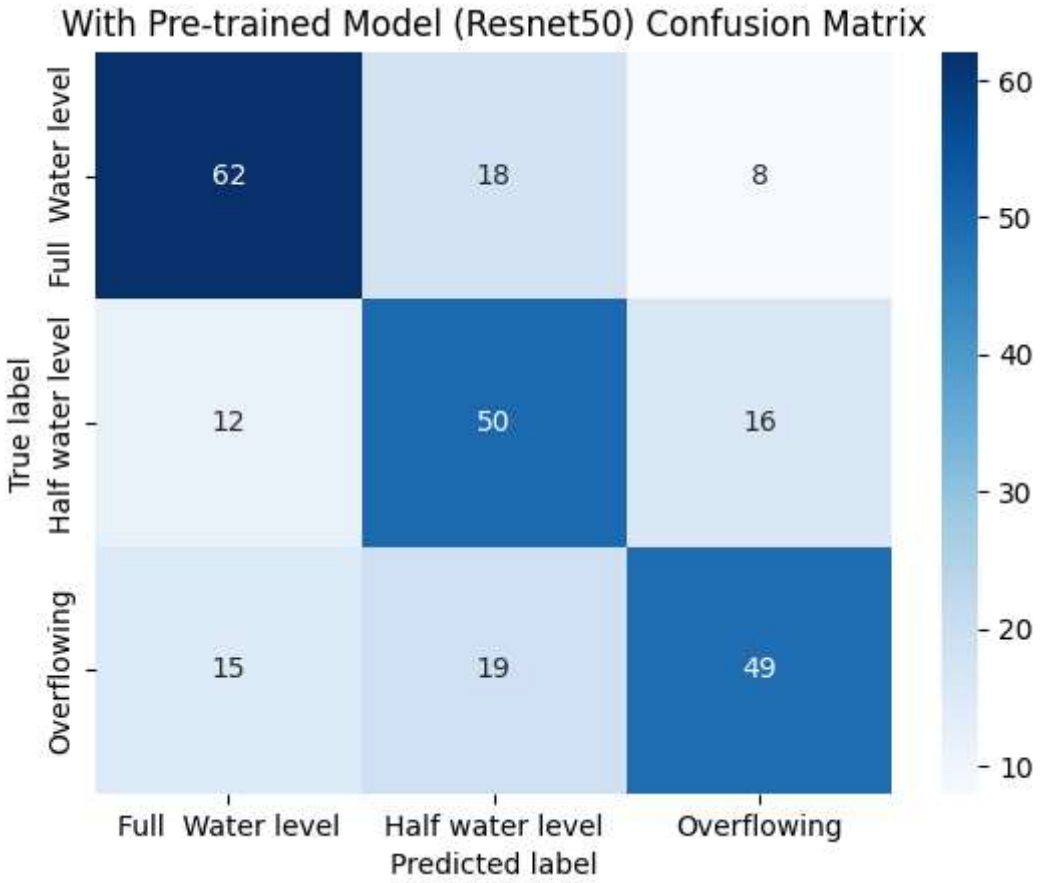
In [33]:
```
generate_cf(self_train_model, 'Self Train CNNs')
print("")
print("")
print("")
generate_cf(pre_train_model, 'With Pre-trained Model (Resnet50)')
```

```
8/8 [==============================] - 2s 196ms/step
```




```
Self Train CNNs accuracy score: 0.3132530120481928
```

```
8/8 [==============================] - 7s 717ms/step
```




```
With Pre-trained Model (Resnet50) accuracy score: 0.6465863453815262
```

In [ ]: