Import the essential libraries

In [4]:
```
#pip install tensorflow
```

In [18]:
```
# Essential and common packages
import os
import glob

# Plots and bars
import matplotlib.pyplot as plt

# Computation library
import numpy as np

# Tensorflow for building the resnet50 model
import tensorflow.python.keras as k
import tensorflow as tf
from tensorflow.keras.layers import Input, Add, Dense, Activation, ZeroPadding2D, BatchNormalization,
from tensorflow.keras.initializers import random_uniform, glorot_uniform
from tensorflow.keras.models import Model

# Sklearn for confusion matrix
import itertools
from sklearn.metrics import confusion_matrix
#from sklearn.metrics import plot_confusion_matrix

# For visualization of plots without plt.show()
%matplotlib inline
```

In [44]:
```
tf.__version__
```

Out[44]:
```
'2.12.0'
```

Define the required variable

In [19]:
```

dataset_url = r'C:\Users\mdsoh\OneDrive\Documents\Deep Learning Projects\Project - EuroSAT Land Cover
batch_size = 32
img_height = 64
img_width = 64
validation_split=0.2
rescale=1.0/255
```

Data preparation for the model

In [20]:
```
datagen = tf.keras.preprocessing.image.ImageDataGenerator(validation_split=validation_split, rescale=r
dataset = tf.keras.preprocessing.image_dataset_from_directory(dataset_url, image_size=(img_height, img
```

```
Found 27000 files belonging to 10 classes.
```

In [21]:
```
train_dataset = datagen.flow_from_directory(batch_size=batch_size,
                                            directory=dataset_url,
                                            shuffle=True,
                                            target_size=(img_height, img_width),
                                            subset="training",
                                            class_mode='categorical')
```

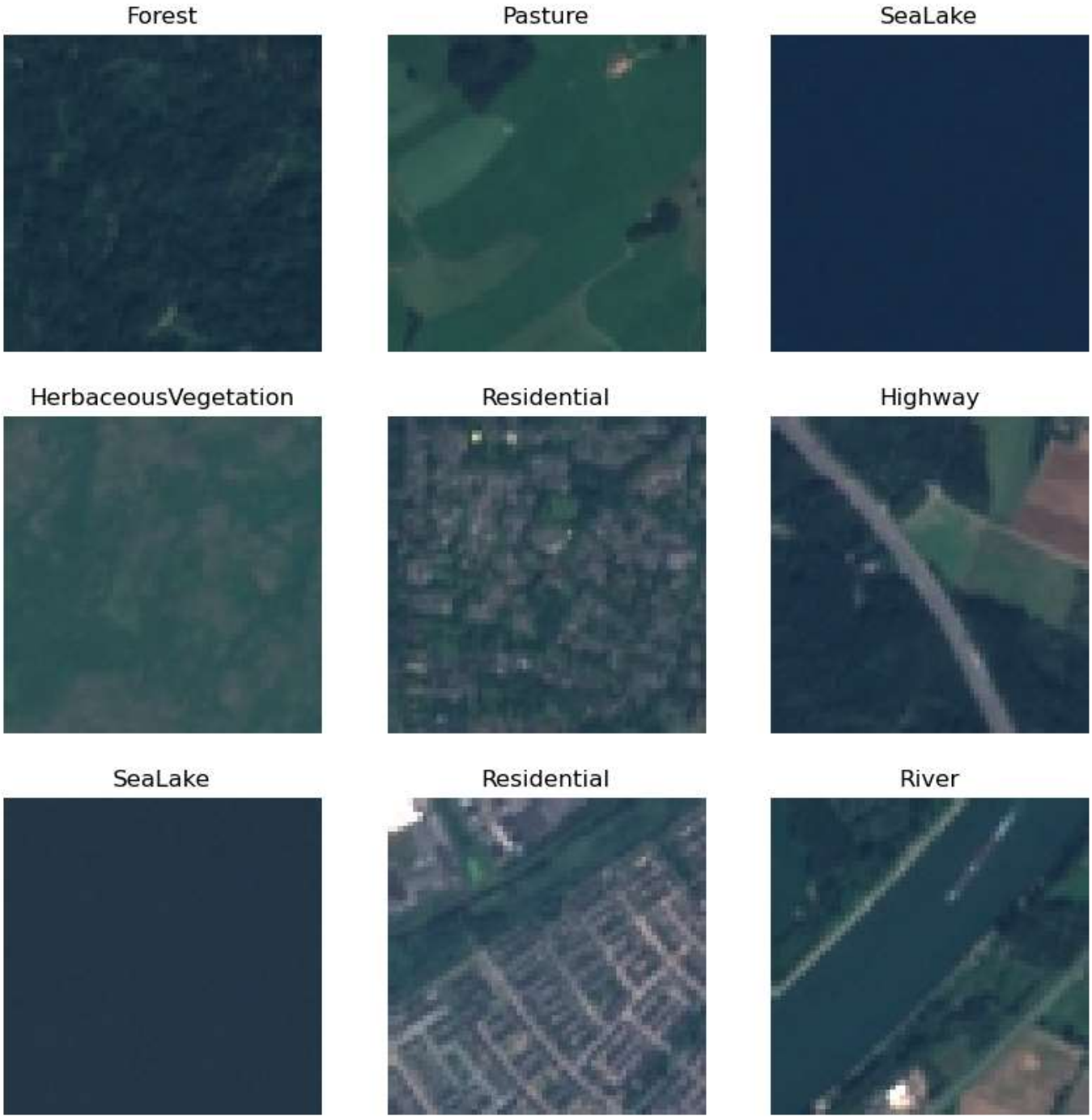
```
Found 21600 images belonging to 10 classes.
```

In [22]:
```
test_dataset = datagen.flow_from_directory(batch_size=batch_size,
                                           directory=dataset_url,
                                           shuffle=True,
                                           target_size=(img_height, img_width),
                                           subset="validation",
                                           class_mode='categorical')
```

```
Found 5400 images belonging to 10 classes.
```

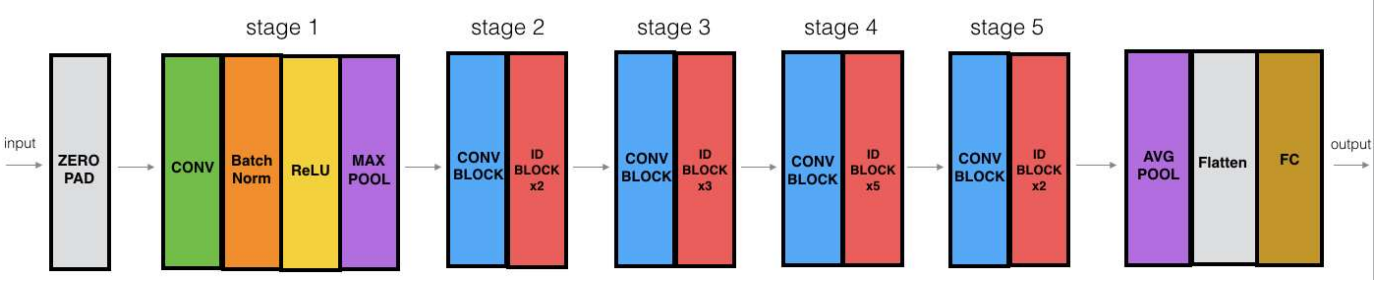
Visualization of input datasets

In [25]:
```
class_names = dataset.class_names
plt.figure(figsize=(10, 10))
for images, labels in dataset.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[labels[i]])
        plt.axis("off")
```

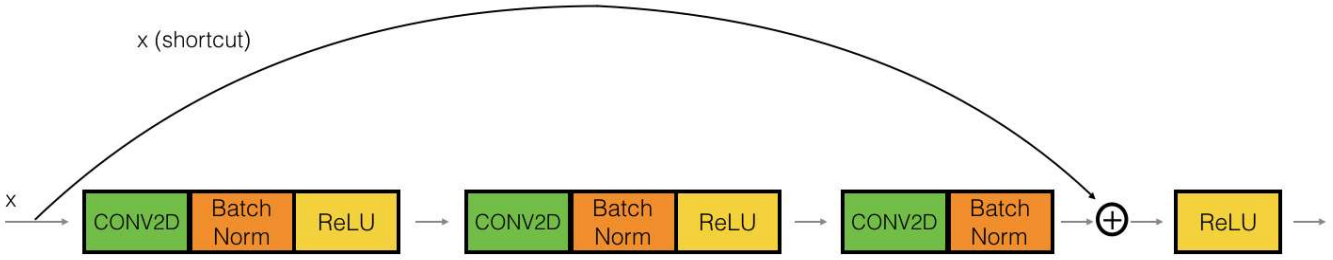


ResNet50 Model building

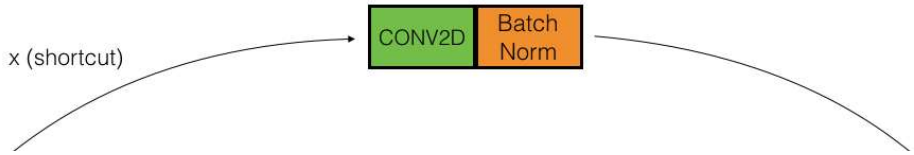
ResNet50 Architecture



ResNet Identity block



ResNet Convolution Block



In [26]:
```
def identity_block(X, f, filters, training=True, initializer=random_uniform):
    """
    Implementation of the identity block

    Arguments:
    X -- input tensor of shape (m, n_H_prev, n_W_prev, n_C_prev)
    f -- integer, specifying the shape of the middle CONV's window for the main path
    filters -- python list of integers, defining the number of filters in the CONV layers of the main
    training -- True: Behave in training mode
                False: Behave in inference mode
    initializer -- to set up the initial weights of a layer. Equals to random uniform initializer

    Returns:
    X -- output of the identity block, tensor of shape (n_H, n_W, n_C)
    """

    # Retrieve Filters
    F1, F2, F3 = filters

    # Save the input value.
    X_shortcut = X
    cache = []
    # First component of main path
    X = Conv2D(filters = F1, kernel_size = 1, strides = (1, 1), padding = 'valid', kernel_initializer
    X = BatchNormalization(axis = 3)(X, training = training) # Default axis
    X = Activation('relu')(X)

    # Second component of main path (≈3 lines)
    X = Conv2D(filters = F2, kernel_size = (f, f), strides = (1, 1), padding = 'same', kernel_initiali
    X = BatchNormalization(axis = 3)(X, training = training)
    X = Activation('relu')(X)

    # Third component of main path (≈2 lines)
    X = Conv2D(filters = F3, kernel_size = (1, 1), strides = (1, 1), padding = 'valid', kernel_initial
    X = BatchNormalization(axis = 3)(X, training = training)

    # Final step: Add shortcut value to main path, and pass it through a RELU activation (≈2 lines)
    X = Add()([X_shortcut, X])
    X = X = Activation('relu')(X, training = training)

    return X
```

In [27]:
```
def convolutional_block(X, f, filters, s = 2, training=True, initializer=glorot_uniform):
    """
    Implementation of the convolutional block

    Arguments:
    X -- input tensor of shape (m, n_H_prev, n_W_prev, n_C_prev)
    f -- integer, specifying the shape of the middle CONV's window for the main path
    filters -- python list of integers, defining the number of filters in the CONV layers of the main
    s -- Integer, specifying the stride to be used
    training -- True: Behave in training mode
                False: Behave in inference mode
    initializer -- to set up the initial weights of a layer. Equals to Glorot uniform initializer,
                   also called Xavier uniform initializer.

    Returns:
    X -- output of the convolutional block, tensor of shape (n_H, n_W, n_C)
    """

    # Retrieve Filters
    F1, F2, F3 = filters

    # Save the input value
    X_shortcut = X


    ##### MAIN PATH #####

    # First component of main path glorot_uniform(seed=0)
    X = Conv2D(filters = F1, kernel_size = 1, strides = (s, s), padding='valid', kernel_initializer =
    X = BatchNormalization(axis = 3)(X, training=training)
    X = Activation('relu')(X)


    # Second component of main path (≈3 lines)
    X = Conv2D(F2, (f, f), strides = (1, 1), padding = 'same', kernel_initializer = initializer(seed=0
    X = BatchNormalization(axis = 3)(X, training = training)
    X = Activation('relu')(X)

    # Third component of main path (≈2 lines)
    X = Conv2D(F3, (1, 1), strides = (1, 1), padding = 'valid', kernel_initializer = initializer(seed=
    X = BatchNormalization(axis = 3)(X, training = training)

    ##### SHORTCUT PATH #### (≈2 lines)
    X_shortcut = Conv2D(F3, (1, 1), strides = (s, s), padding = 'valid', kernel_initializer = initiali
    X_shortcut = BatchNormalization(axis = 3)(X_shortcut, training = training)

    # Final step: Add shortcut value to main path (Use this order [X, X_shortcut]), and pass it throug
    X = Add()([X, X_shortcut])
    X = Activation('relu')(X)

    return X
```

In [28]:
```
def ResNet50(input_shape = (64, 64, 3), classes = 6):
    """
    Stage-wise implementation of the architecture of the popular ResNet50:
    CONV2D -> BATCHNORM -> RELU -> MAXPOOL -> CONVBLOCK -> IDBLOCK*2 -> CONVBLOCK -> IDBLOCK*3
    -> CONVBLOCK -> IDBLOCK*5 -> CONVBLOCK -> IDBLOCK*2 -> AVGPOOL -> FLATTEN -> DENSE

    Arguments:
    input_shape -- shape of the images of the dataset
    classes -- integer, number of classes

    Returns:
    model -- a Model() instance in Keras
    """

    # Define the input as a tensor with shape input_shape
    X_input = Input(input_shape)


    # Zero-Padding
    X = ZeroPadding2D((3, 3))(X_input)

    # Stage 1
    X = Conv2D(64, (7, 7), strides = (2, 2), kernel_initializer = glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis = 3)(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((3, 3), strides=(2, 2))(X)

    # Stage 2
    X = convolutional_block(X, f = 3, filters = [64, 64, 256], s = 1)
    X = identity_block(X, 3, [64, 64, 256])
    X = identity_block(X, 3, [64, 64, 256])

   # Stage 3 (≈4 lines)
    X = convolutional_block(X, f = 3, filters = [128, 128, 512], s = 2)
    X = identity_block(X, 3, [128, 128, 512])
    X = identity_block(X, 3, [128, 128, 512])
    X = identity_block(X, 3, [128, 128, 512])

    # Stage 4 (≈6 lines)
    X = convolutional_block(X, f = 3, filters = [256, 256, 1024], s = 2)
    X = identity_block(X, 3, [256, 256, 1024])
    X = identity_block(X, 3, [256, 256, 1024])
    X = identity_block(X, 3, [256, 256, 1024])
    X = identity_block(X, 3, [256, 256, 1024])
    X = identity_block(X, 3, [256, 256, 1024])

    # Stage 5 (≈3 lines)
    X = convolutional_block(X, f = 3, filters = [512, 512, 2048], s = 2)
    X = identity_block(X, 3, [512, 512, 2048])
    X = identity_block(X, 3, [512, 512, 2048])

    # AVGPOOL (≈1 line). Use "X = AveragePooling2D(...)(X)"
    X = AveragePooling2D(pool_size = (2, 2), name = 'avg_pool')(X)


    # output layer
    X = Flatten()(X)
    X = Dense(classes, activation='softmax', kernel_initializer = glorot_uniform(seed=0))(X)


    # Create model
    model = Model(inputs = X_input, outputs = X)

    return model
```

Model train

In [29]:
```
model = ResNet50(input_shape=(64,64,3), classes=10)
model.summary()
```

```
 add_14 (Add)                   (None, 2, 2, 2048)   0           ['activation_42[0][0]',
                                                                  'batch_normalization_49[0][0]']

 activation_45 (Activation)     (None, 2, 2, 2048)   0           ['add_14[0][0]']

 conv2d_50 (Conv2D)             (None, 2, 2, 512)    1049088     ['activation_45[0][0]']

 batch_normalization_50 (BatchN  (None, 2, 2, 512)   2048        ['conv2d_50[0][0]']
 ormalization)

 activation_46 (Activation)     (None, 2, 2, 512)    0           ['batch_normalization_50[0][0]']

 conv2d_51 (Conv2D)             (None, 2, 2, 512)    2359808     ['activation_46[0][0]']

 batch_normalization_51 (BatchN  (None, 2, 2, 512)   2048        ['conv2d_51[0][0]']
 ormalization)

 activation_47 (Activation)     (None, 2, 2, 512)    0           ['batch_normalization_51[0][0]']
```

In [31]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
# please increase the epoch for higher accuracy (epochs=100)
history = model.fit(train_dataset, validation_data=test_dataset, epochs=20, batch_size=32)
```

In [33]:
```
model.save(r'model/sohaib_model_20_epoch.h5')
```

In [13]:
```
model.save(r'model/model_100_epoch.h5')
```

```
C:\Users\tek\anaconda3\envs\tf\lib\site-packages\tensorflow\python\keras\utils\generic_utils.py:494:
CustomMaskWarning: Custom mask layers require a config and must override get_config. When loading, th
e custom mask layer must be passed to the custom_objects argument.
  warnings.warn('Custom mask layers require a config and must override '

INFO:tensorflow:Assets written to: lulc_20_epoch\assets
```

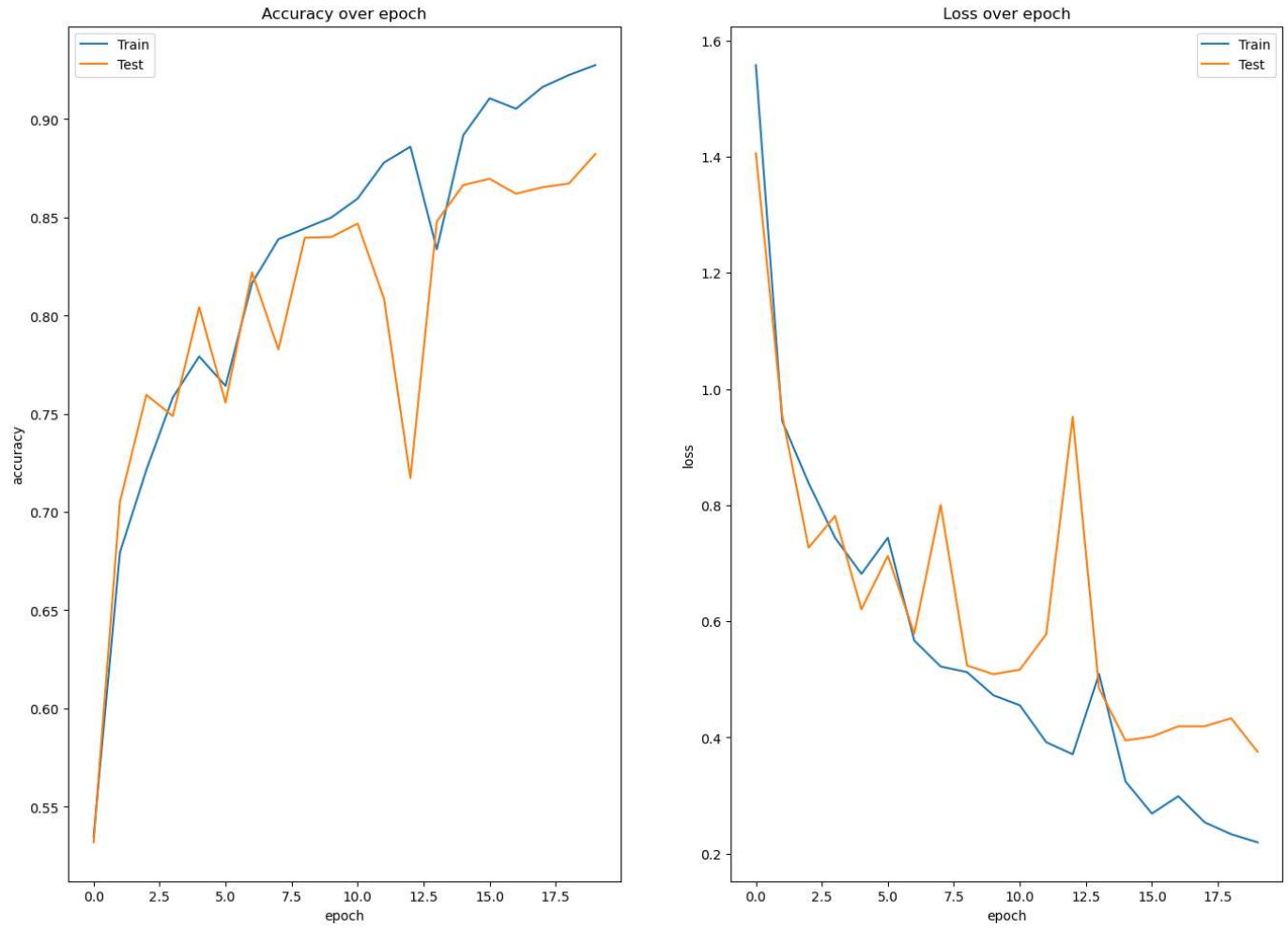
Load model

In [34]:
```
from tensorflow.keras.models import load_model
model = load_model(r"model/sohaib_model_20_epoch.h5")
#model = load_model(r"model/model_100_epoch.h5")
```

analyzing results and visualization

In [35]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16.53, 11.69))
ax1.plot(history.history['accuracy'])
ax1.plot(history.history['val_accuracy'])
ax1.set_xlabel('epoch')
ax1.set_ylabel('accuracy')
ax1.set_title('Accuracy over epoch')
ax1.legend(['Train', 'Test'], loc='upper left')

ax2.plot(history.history['loss'])
ax2.plot(history.history['val_loss'])
ax2.set_xlabel('epoch')
ax2.set_ylabel('loss')
ax2.set_title('Loss over epoch')
ax2.legend(['Train', 'Test'], loc="upper right")
```

Out[35]:
```
<matplotlib.legend.Legend at 0x1ea0b4df040>
```



Confusion matrix

In [36]:

```
y_pred = []  # store predicted labels
y_true = []  # store true labels

# iterate over the dataset
for i, (image_batch, label_batch) in enumerate(test_dataset):   # use dataset.unbatch() with repeat
    # append true labels
    y_true.append(label_batch)
    # compute predictions
    preds = model.predict(image_batch)
    # append predicted labels
    y_pred.append(np.argmax(preds, axis =  1))
    if i==300:
        break

# convert the true and predicted labels into tensors
correct_labels = tf.concat([item for item in y_true], axis = 0)
correct_labels = np.argmax(correct_labels, axis=1)
predicted_labels = tf.concat([item for item in y_pred], axis = 0)
```

```
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 0s 203ms/step
1/1 [==============================] - 0s 179ms/step
1/1 [==============================] - 0s 188ms/step
1/1 [==============================] - 0s 165ms/step
1/1 [==============================] - 0s 178ms/step
1/1 [==============================] - 0s 176ms/step
1/1 [==============================] - 0s 186ms/step
1/1 [==============================] - 0s 181ms/step
1/1 [==============================] - 0s 186ms/step
1/1 [==============================] - 0s 187ms/step
1/1 [==============================] - 0s 182ms/step
1/1 [==============================] - 0s 194ms/step
1/1 [==============================] - 0s 181ms/step
1/1 [==============================] - 0s 183ms/step
1/1 [==============================] - 0s 181ms/step
1/1 [==============================] - 0s 170ms/step
1/1 [==============================] - 0s 171ms/step
1/1 [==============================] - 0s 184ms/step
1/1 [==============================] - 0s 180ms/step
1/1 [==============================] - 0s 175ms/step
1/1 [==============================] - 0s 163ms/step
1/1 [==============================] - 0s 182ms/step
1/1 [==============================] - 0s 176ms/step
1/1 [==============================] - 0s 176ms/step
1/1 [==============================] - 0s 165ms/step
1/1 [==============================] - 0s 172ms/step
1/1 [==============================] - 0s 179ms/step
1/1 [==============================] - 0s 173ms/step
1/1 [==============================] - 0s 175ms/step
1/1 [==============================] - 0s 167ms/step
1/1 [==============================] - 0s 174ms/step
1/1 [==============================] - 0s 183ms/step
1/1 [==============================] - 0s 176ms/step
1/1 [==============================] - 0s 163ms/step
1/1 [==============================] - 0s 183ms/step
1/1 [==============================] - 0s 183ms/step
1/1 [==============================] - 0s 180ms/step
1/1 [==============================] - 0s 174ms/step
1/1 [==============================] - 0s 177ms/step
1/1 [==============================] - 0s 182ms/step
1/1 [==============================] - 0s 180ms/step
1/1 [==============================] - 0s 174ms/step
1/1 [==============================] - 0s 186ms/step
1/1 [==============================] - 0s 183ms/step
1/1 [==============================] - 0s 186ms/step
1/1 [==============================] - 0s 180ms/step
1/1 [==============================] - 0s 187ms/step
1/1 [==============================] - 0s 182ms/step
1/1 [==============================] - 0s 183ms/step
1/1 [==============================] - 0s 169ms/step
1/1 [==============================] - 0s 180ms/step
1/1 [==============================] - 0s 176ms/step
1/1 [==============================] - 0s 180ms/step
1/1 [==============================] - 0s 177ms/step
1/1 [==============================] - 0s 176ms/step
1/1 [==============================] - 0s 184ms/step
1/1 [==============================] - 0s 175ms/step
1/1 [==============================] - 0s 206ms/step
1/1 [==============================] - 0s 196ms/step
1/1 [==============================] - 0s 216ms/step
1/1 [==============================] - 0s 233ms/step
1/1 [==============================] - 0s 281ms/step
1/1 [==============================] - 0s 287ms/step
1/1 [==============================] - 0s 241ms/step
1/1 [==============================] - 0s 242ms/step
1/1 [==============================] - 0s 199ms/step
1/1 [==============================] - 0s 190ms/step
1/1 [==============================] - 0s 191ms/step
1/1 [==============================] - 0s 172ms/step
```

```
1/1 [==============================] - 0s 184ms/step
1/1 [==============================] - 0s 186ms/step
1/1 [==============================] - 0s 174ms/step
1/1 [==============================] - 0s 170ms/step
1/1 [==============================] - 0s 172ms/step
1/1 [==============================] - 0s 180ms/step
1/1 [==============================] - 0s 179ms/step
1/1 [==============================] - 0s 185ms/step
1/1 [==============================] - 0s 173ms/step
1/1 [==============================] - 0s 184ms/step
1/1 [==============================] - 0s 189ms/step
1/1 [==============================] - 0s 172ms/step
1/1 [==============================] - 0s 168ms/step
1/1 [==============================] - 0s 174ms/step
1/1 [==============================] - 0s 179ms/step
1/1 [==============================] - 0s 170ms/step
1/1 [==============================] - 0s 181ms/step
1/1 [==============================] - 0s 171ms/step
1/1 [==============================] - 0s 174ms/step
1/1 [==============================] - 0s 172ms/step
1/1 [==============================] - 0s 168ms/step
1/1 [==============================] - 0s 171ms/step
1/1 [==============================] - 0s 192ms/step
1/1 [==============================] - 0s 176ms/step
1/1 [==============================] - 0s 171ms/step
1/1 [==============================] - 0s 185ms/step
1/1 [==============================] - 0s 187ms/step
1/1 [==============================] - 0s 187ms/step
1/1 [==============================] - 0s 185ms/step
1/1 [==============================] - 0s 173ms/step
1/1 [==============================] - 0s 166ms/step
1/1 [==============================] - 0s 192ms/step
1/1 [==============================] - 0s 193ms/step
1/1 [==============================] - 0s 183ms/step
1/1 [==============================] - 0s 187ms/step
1/1 [==============================] - 0s 187ms/step
1/1 [==============================] - 0s 194ms/step
1/1 [==============================] - 0s 183ms/step
1/1 [==============================] - 0s 195ms/step
1/1 [==============================] - 0s 186ms/step
1/1 [==============================] - 0s 192ms/step
1/1 [==============================] - 0s 188ms/step
1/1 [==============================] - 0s 191ms/step
1/1 [==============================] - 0s 191ms/step
1/1 [==============================] - 0s 184ms/step
1/1 [==============================] - 0s 181ms/step
1/1 [==============================] - 0s 191ms/step
1/1 [==============================] - 0s 189ms/step
1/1 [==============================] - 0s 178ms/step
1/1 [==============================] - 0s 184ms/step
1/1 [==============================] - 0s 186ms/step
1/1 [==============================] - 0s 193ms/step
1/1 [==============================] - 0s 183ms/step
1/1 [==============================] - 0s 173ms/step
1/1 [==============================] - 0s 209ms/step
1/1 [==============================] - 0s 228ms/step
1/1 [==============================] - 0s 282ms/step
1/1 [==============================] - 0s 188ms/step
1/1 [==============================] - 0s 186ms/step
1/1 [==============================] - 0s 177ms/step
1/1 [==============================] - 0s 177ms/step
1/1 [==============================] - 0s 171ms/step
1/1 [==============================] - 0s 174ms/step
1/1 [==============================] - 0s 176ms/step
1/1 [==============================] - 0s 177ms/step
1/1 [==============================] - 0s 172ms/step
1/1 [==============================] - 0s 172ms/step
1/1 [==============================] - 0s 175ms/step
1/1 [==============================] - 0s 178ms/step
1/1 [==============================] - 0s 169ms/step
```

```
1/1 [==============================] - 0s 169ms/step
1/1 [==============================] - 0s 179ms/step
1/1 [==============================] - 0s 176ms/step
1/1 [==============================] - 0s 173ms/step
1/1 [==============================] - 0s 162ms/step
1/1 [==============================] - 0s 174ms/step
1/1 [==============================] - 0s 181ms/step
1/1 [==============================] - 0s 187ms/step
1/1 [==============================] - 0s 172ms/step
1/1 [==============================] - 0s 165ms/step
1/1 [==============================] - 0s 215ms/step
1/1 [==============================] - 0s 224ms/step
1/1 [==============================] - 0s 209ms/step
1/1 [==============================] - 0s 214ms/step
```

```
1/1 [==============================] - 0s 188ms/step
1/1 [==============================] - 0s 215ms/step
1/1 [==============================] - 0s 276ms/step
1/1 [==============================] - 0s 268ms/step
1/1 [==============================] - 0s 262ms/step
1/1 [==============================] - 0s 283ms/step
1/1 [==============================] - 0s 276ms/step
1/1 [==============================] - 0s 264ms/step
1/1 [==============================] - 0s 220ms/step
1/1 [==============================] - 0s 199ms/step
1/1 [==============================] - 0s 196ms/step
1/1 [==============================] - 0s 187ms/step
1/1 [==============================] - 0s 191ms/step
1/1 [==============================] - 0s 197ms/step
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 0s 178ms/step
1/1 [==============================] - 0s 183ms/step
1/1 [==============================] - 0s 191ms/step
1/1 [==============================] - 0s 190ms/step
1/1 [==============================] - 0s 176ms/step
1/1 [==============================] - 0s 190ms/step
1/1 [==============================] - 0s 224ms/step
1/1 [==============================] - 0s 202ms/step
1/1 [==============================] - 0s 281ms/step
1/1 [==============================] - 0s 269ms/step
1/1 [==============================] - 0s 297ms/step
1/1 [==============================] - 0s 203ms/step
1/1 [==============================] - 0s 204ms/step
1/1 [==============================] - 0s 247ms/step
1/1 [==============================] - 0s 316ms/step
1/1 [==============================] - 0s 194ms/step
1/1 [==============================] - 0s 249ms/step
1/1 [==============================] - 0s 279ms/step
1/1 [==============================] - 0s 268ms/step
1/1 [==============================] - 0s 273ms/step
1/1 [==============================] - 0s 271ms/step
1/1 [==============================] - 0s 279ms/step
1/1 [==============================] - 0s 196ms/step
1/1 [==============================] - 0s 189ms/step
1/1 [==============================] - 0s 174ms/step
1/1 [==============================] - 0s 256ms/step
1/1 [==============================] - 0s 187ms/step
1/1 [==============================] - 0s 212ms/step
1/1 [==============================] - 0s 258ms/step
1/1 [==============================] - 0s 207ms/step
1/1 [==============================] - 0s 199ms/step
1/1 [==============================] - 0s 186ms/step
1/1 [==============================] - 0s 175ms/step
1/1 [==============================] - 0s 185ms/step
1/1 [==============================] - 0s 191ms/step
1/1 [==============================] - 0s 194ms/step
1/1 [==============================] - 0s 185ms/step
1/1 [==============================] - 0s 182ms/step
1/1 [==============================] - 0s 184ms/step
1/1 [==============================] - 0s 183ms/step
1/1 [==============================] - 0s 169ms/step
1/1 [==============================] - 0s 181ms/step
1/1 [==============================] - 0s 198ms/step
1/1 [==============================] - 0s 185ms/step
1/1 [==============================] - 0s 173ms/step
1/1 [==============================] - 0s 168ms/step
1/1 [==============================] - 0s 171ms/step
1/1 [==============================] - 0s 179ms/step
1/1 [==============================] - 0s 180ms/step
1/1 [==============================] - 0s 168ms/step
1/1 [==============================] - 0s 173ms/step
1/1 [==============================] - 0s 196ms/step
1/1 [==============================] - 0s 196ms/step
1/1 [==============================] - 0s 188ms/step
1/1 [==============================] - 0s 179ms/step
```

```
1/1 [==============================] - 0s 183ms/step
1/1 [==============================] - 0s 193ms/step
1/1 [==============================] - 0s 184ms/step
1/1 [==============================] - 0s 179ms/step
1/1 [==============================] - 0s 189ms/step
1/1 [==============================] - 0s 200ms/step
1/1 [==============================] - 0s 191ms/step
1/1 [==============================] - 0s 186ms/step
1/1 [==============================] - 0s 186ms/step
1/1 [==============================] - 0s 196ms/step
1/1 [==============================] - 0s 187ms/step
1/1 [==============================] - 0s 188ms/step
1/1 [==============================] - 0s 183ms/step
1/1 [==============================] - 0s 187ms/step
1/1 [==============================] - 0s 186ms/step
1/1 [==============================] - 0s 182ms/step
1/1 [==============================] - 0s 187ms/step
1/1 [==============================] - 0s 197ms/step
1/1 [==============================] - 0s 191ms/step
1/1 [==============================] - 0s 176ms/step
1/1 [==============================] - 0s 191ms/step
1/1 [==============================] - 0s 216ms/step
1/1 [==============================] - 0s 208ms/step
1/1 [==============================] - 0s 202ms/step
1/1 [==============================] - 0s 199ms/step
1/1 [==============================] - 0s 201ms/step
1/1 [==============================] - 0s 193ms/step
1/1 [==============================] - 0s 174ms/step
1/1 [==============================] - 0s 172ms/step
1/1 [==============================] - 0s 178ms/step
1/1 [==============================] - 0s 180ms/step
1/1 [==============================] - 0s 181ms/step
1/1 [==============================] - 0s 169ms/step
1/1 [==============================] - 0s 172ms/step
1/1 [==============================] - 0s 180ms/step
1/1 [==============================] - 0s 181ms/step
1/1 [==============================] - 0s 174ms/step
1/1 [==============================] - 0s 171ms/step
1/1 [==============================] - 0s 179ms/step
1/1 [==============================] - 0s 181ms/step
1/1 [==============================] - 0s 182ms/step
1/1 [==============================] - 0s 172ms/step
1/1 [==============================] - 0s 172ms/step
1/1 [==============================] - 0s 181ms/step
1/1 [==============================] - 0s 182ms/step
1/1 [==============================] - 0s 175ms/step
1/1 [==============================] - 0s 173ms/step
1/1 [==============================] - 0s 176ms/step
1/1 [==============================] - 0s 183ms/step
1/1 [==============================] - 0s 182ms/step
1/1 [==============================] - 0s 171ms/step
1/1 [==============================] - 0s 167ms/step
1/1 [==============================] - 0s 179ms/step
1/1 [==============================] - 0s 178ms/step
1/1 [==============================] - 0s 172ms/step
1/1 [==============================] - 0s 170ms/step
1/1 [==============================] - 0s 173ms/step
1/1 [==============================] - 0s 175ms/step
1/1 [==============================] - 0s 178ms/step
1/1 [==============================] - 0s 167ms/step
1/1 [==============================] - 0s 168ms/step
1/1 [==============================] - 0s 180ms/step
1/1 [==============================] - 0s 177ms/step
1/1 [==============================] - 0s 175ms/step
1/1 [==============================] - 0s 176ms/step
1/1 [==============================] - 0s 185ms/step
1/1 [==============================] - 0s 196ms/step
1/1 [==============================] - 0s 187ms/step
1/1 [==============================] - 0s 181ms/step
1/1 [==============================] - 0s 180ms/step
```

```
1/1 [==============================] - 0s 187ms/step
1/1 [==============================] - 0s 188ms/step
1/1 [==============================] - 0s 180ms/step
1/1 [==============================] - 0s 186ms/step
1/1 [==============================] - 0s 190ms/step
1/1 [==============================] - 0s 198ms/step
1/1 [==============================] - 0s 181ms/step
```

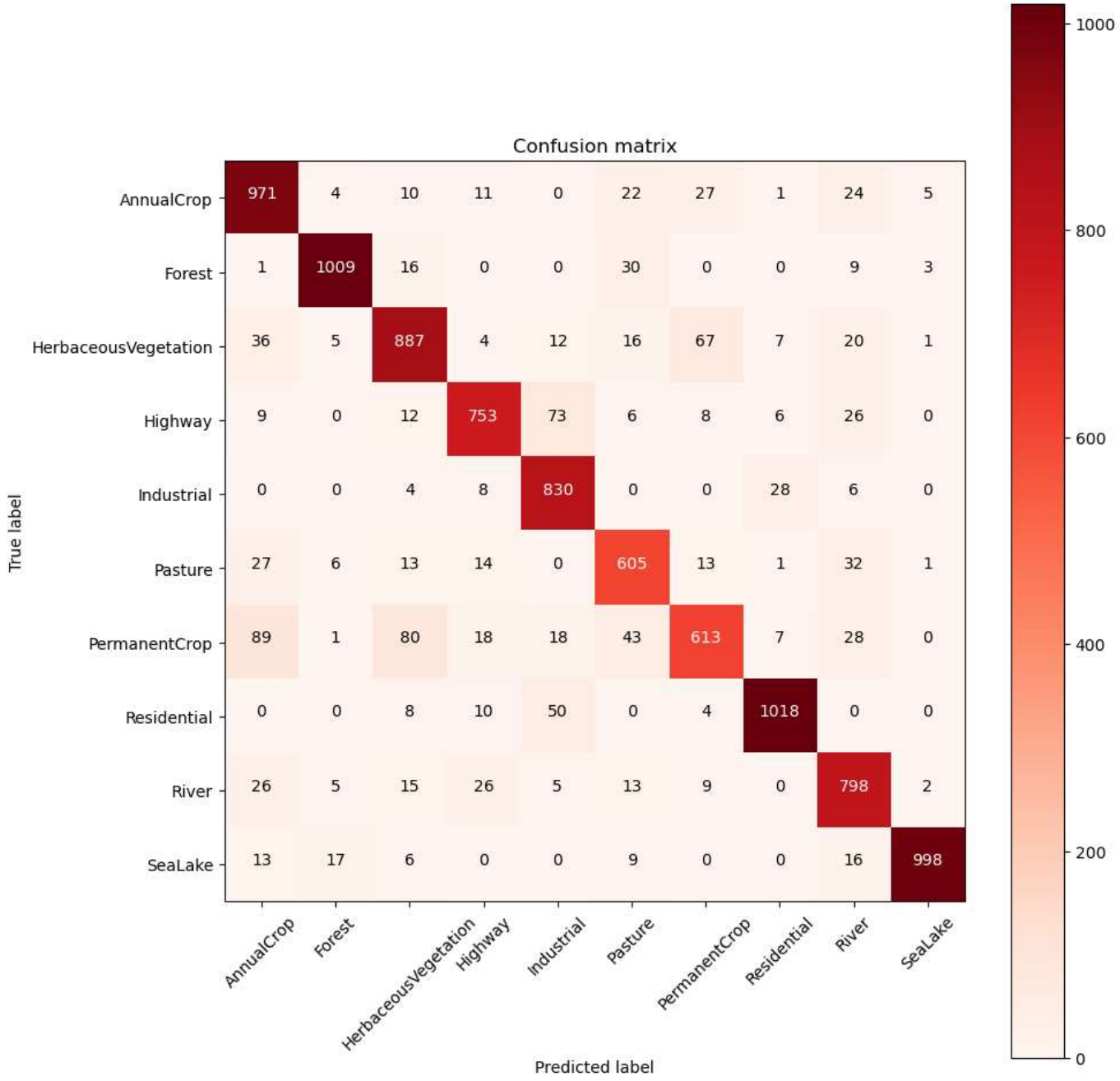
In [43]:
```
cm = confusion_matrix(correct_labels, predicted_labels)
cm
```

Out[43]:
```
array([[ 971,    4,   10,   11,    0,   22,   27,    1,   24,    5],
       [   1, 1009,   16,    0,    0,   30,    0,    0,    9,    3],
       [  36,    5,  887,    4,   12,   16,   67,    7,   20,    1],
       [   9,    0,   12,  753,   73,    6,    8,    6,   26,    0],
       [   0,    0,    4,    8,  830,    0,    0,   28,    6,    0],
       [  27,    6,   13,   14,    0,  605,   13,    1,   32,    1],
       [  89,    1,   80,   18,   18,   43,  613,    7,   28,    0],
       [   0,    0,    8,   10,   50,    0,    4, 1018,    0,    0],
       [  26,    5,   15,   26,    5,   13,    9,    0,  798,    2],
       [  13,   17,    6,    0,    0,    9,    0,    0,   16,  998]],
      dtype=int64)
```

In [41]:
```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          figsize=(10, 10),
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.figure(figsize=figsize)
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
            horizontalalignment="center",
            color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

In [42]:
```
plot_confusion_matrix(cm, train_dataset.class_indices, cmap='Reds')
```

```
Confusion matrix, without normalization
[[ 971    4   10   11    0   22   27    1   24    5]
 [   1 1009   16    0    0   30    0    0    9    3]
 [  36    5  887    4   12   16   67    7   20    1]
 [   9    0   12  753   73    6    8    6   26    0]
 [   0    0    4    8  830    0    0   28    6    0]
 [  27    6   13   14    0  605   13    1   32    1]
 [  89    1   80   18   18   43  613    7   28    0]
 [   0    0    8   10   50    0    4 1018    0    0]
 [  26    5   15   26    5   13    9    0  798    2]
 [  13   17    6    0    0    9    0    0   16  998]]
```



Thank you