Import the essential libraries

In [1]:
```
# Essential and common packages
import os
import glob

# Read and visualize the raster data
import rasterio
from rasterio.plot import show

# Plots and bars
import matplotlib.pyplot as plt
import seaborn as sns

# 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, ZeroPadding
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, plot_confusion_matrix

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

Define the required variable

In [2]:
```
dataset_url = r'C:\Users\BALLA\Desktop\try\DL-for-LULC-prediction\EuroSAT\2750
batch_size = 32
img_height = 64
img_width = 64
validation_split=0.2
rescale=1.0/255
```

Data preparation for the model

In [3]:
```
datagen = tf.keras.preprocessing.image.ImageDataGenerator(validation_split=val
dataset = tf.keras.preprocessing.image_dataset_from_directory(dataset_url, ima
```

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

In [4]:
```
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 [5]:
```
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 [6]:
```
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

In [7]:
```
def identity_block(X, f, filters, training=True, initializer=random_uniform):

    # 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 = 'val
    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 =
    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 =
    X = BatchNormalization(axis = 3)(X, training = training)

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

    return X
```

In [8]:
```
def convolutional_block(X, f, filters, s = 2, training=True, initializer=gloro


    # 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
    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_initiali
    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_initial
    X = BatchNormalization(axis = 3)(X, training = training)

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

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

    return X
```

In [9]:
```
def ResNet50(input_shape = (64, 64, 3), classes = 6):


    # 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_unifo
    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_unifo


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

    return model
```

Model train

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

In [11]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc
```

In [12]:
```
# please increase the epoch for higher accuracy (epochs=100)
history = model.fit(train_dataset, validation_data=test_dataset, epochs=20, ba
```

```
Epoch 1/20
675/675 [==============================] - 260s 379ms/step - loss: 1.5566 - a
ccuracy: 0.5439 - val_loss: 1.1688 - val_accuracy: 0.6430
Epoch 2/20
675/675 [==============================] - 254s 376ms/step - loss: 1.0657 - a
ccuracy: 0.6463 - val_loss: 1.2341 - val_accuracy: 0.6067
Epoch 3/20
675/675 [==============================] - 255s 378ms/step - loss: 0.9766 - a
ccuracy: 0.6866 - val_loss: 1.0293 - val_accuracy: 0.6533
Epoch 4/20
675/675 [==============================] - 255s 377ms/step - loss: 0.8460 - a
ccuracy: 0.7325 - val_loss: 0.7216 - val_accuracy: 0.7611
Epoch 5/20
675/675 [==============================] - 254s 376ms/step - loss: 0.7362 - a
ccuracy: 0.7642 - val_loss: 0.8269 - val_accuracy: 0.7193
Epoch 6/20
675/675 [==============================] - 253s 375ms/step - loss: 0.6699 - a
ccuracy: 0.7824 - val_loss: 0.6173 - val_accuracy: 0.7919
Epoch 7/20
675/675 [==============================] - 254s 376ms/step - loss: 0.5544 - a
ccuracy: 0.8200 - val_loss: 0.5773 - val_accuracy: 0.8122
Epoch 8/20
675/675 [==============================] - 254s 376ms/step - loss: 0.6272 - a
ccuracy: 0.8104 - val_loss: 1.1135 - val_accuracy: 0.7078
Epoch 9/20
675/675 [==============================] - 254s 376ms/step - loss: 0.6083 - a
ccuracy: 0.8040 - val_loss: 0.5825 - val_accuracy: 0.8089
Epoch 10/20
675/675 [==============================] - 256s 379ms/step - loss: 1.1006 - a
ccuracy: 0.6621 - val_loss: 0.8815 - val_accuracy: 0.6978
Epoch 11/20
675/675 [==============================] - 256s 380ms/step - loss: 0.8871 - a
ccuracy: 0.7219 - val_loss: 0.6925 - val_accuracy: 0.7787
Epoch 12/20
675/675 [==============================] - 255s 377ms/step - loss: 0.6545 - a
ccuracy: 0.7852 - val_loss: 0.7961 - val_accuracy: 0.7278
Epoch 13/20
675/675 [==============================] - 254s 377ms/step - loss: 0.5854 - a
ccuracy: 0.8012 - val_loss: 0.6216 - val_accuracy: 0.7730
Epoch 14/20
675/675 [==============================] - 254s 377ms/step - loss: 0.5504 - a
ccuracy: 0.8171 - val_loss: 0.8009 - val_accuracy: 0.7278
Epoch 15/20
675/675 [==============================] - 256s 380ms/step - loss: 0.5710 - a
ccuracy: 0.8167 - val_loss: 0.6722 - val_accuracy: 0.7806
Epoch 16/20
675/675 [==============================] - 254s 376ms/step - loss: 0.6764 - a
ccuracy: 0.7836 - val_loss: 1.0433 - val_accuracy: 0.6793
Epoch 17/20
675/675 [==============================] - 255s 379ms/step - loss: 0.5936 - a
ccuracy: 0.7984 - val_loss: 0.5053 - val_accuracy: 0.8361
Epoch 18/20
675/675 [==============================] - 254s 377ms/step - loss: 0.4573 - a
ccuracy: 0.8428 - val_loss: 0.4915 - val_accuracy: 0.8402
Epoch 19/20
675/675 [==============================] - 252s 373ms/step - loss: 0.3720 - a
ccuracy: 0.8717 - val_loss: 0.4607 - val_accuracy: 0.8456
```

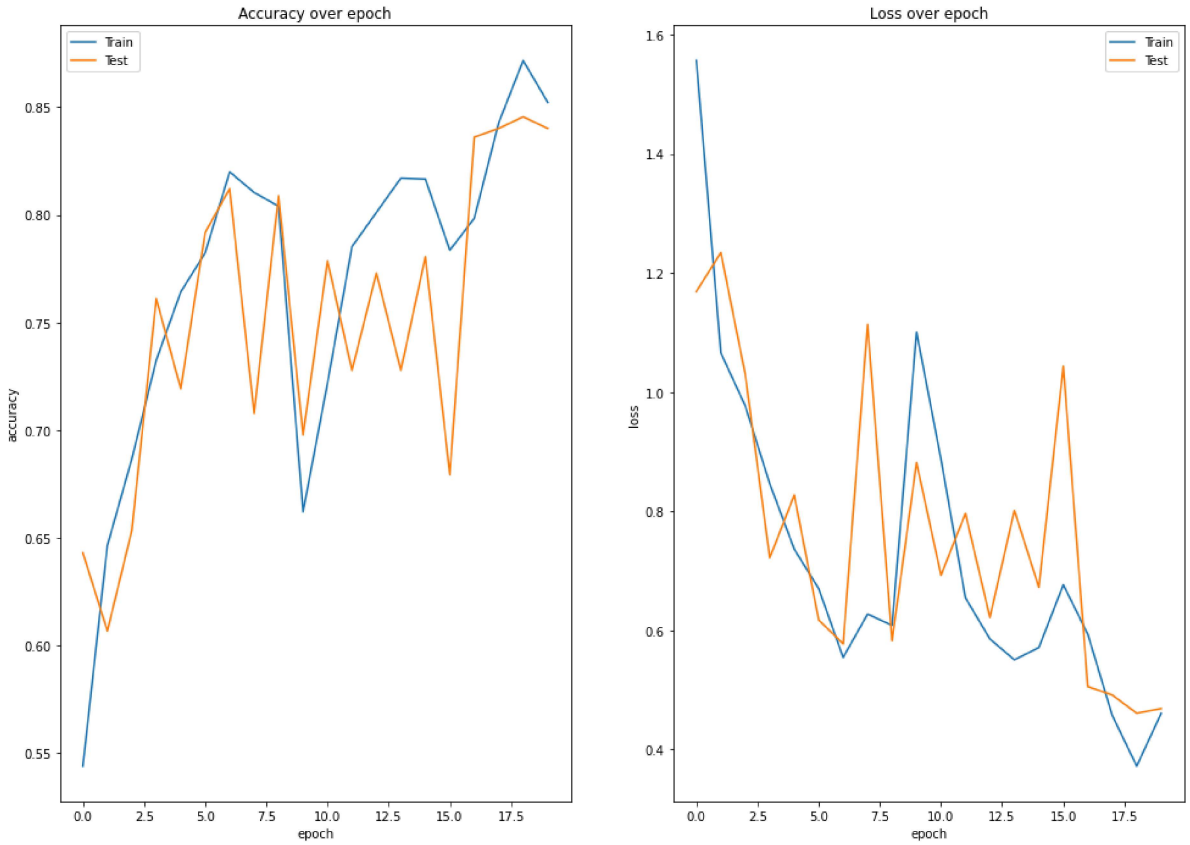
```
Epoch 20/20
675/675 [==============================] - 251s 372ms/step - loss: 0.4605 - a
ccuracy: 0.8523 - val_loss: 0.4685 - val_accuracy: 0.8402
```

In [ ]:
```
model.save('lulc_20_epoch')
```

analyzing results and visualization

In [14]:
```
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[14]:
```
<matplotlib.legend.Legend at 0x263ba4c8490>
```



Confusion matrix

In [15]:
```
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.
    # 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)
```

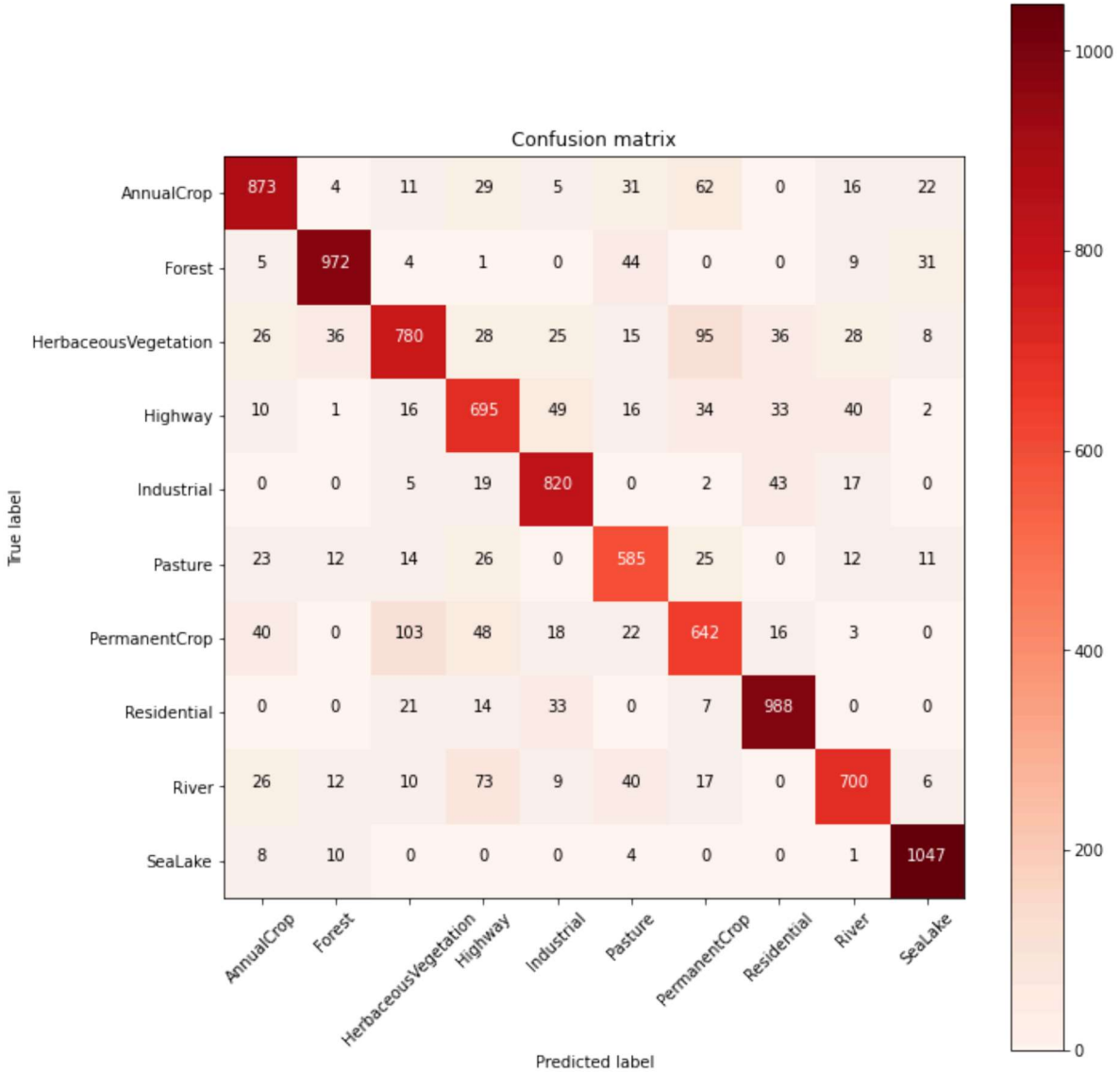
In [16]:
```
cm = confusion_matrix(correct_labels, predicted_labels)
cm
```

Out[16]:
```
array([[ 873,    4,   11,   29,    5,   31,   62,    0,   16,   22],
       [   5,  972,    4,    1,    0,   44,    0,    0,    9,   31],
       [  26,   36,  780,   28,   25,   15,   95,   36,   28,    8],
       [  10,    1,   16,  695,   49,   16,   34,   33,   40,    2],
       [   0,    0,    5,   19,  820,    0,    2,   43,   17,    0],
       [  23,   12,   14,   26,    0,  585,   25,    0,   12,   11],
       [  40,    0,  103,   48,   18,   22,  642,   16,    3,    0],
       [   0,    0,   21,   14,   33,    0,    7,  988,    0,    0],
       [  26,   12,   10,   73,    9,   40,   17,    0,  700,    6],
       [   8,   10,    0,    0,    0,    4,    0,    0,    1, 1047]],
      dtype=int64)
```

In [17]:
```
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 [18]:
```
plot_confusion_matrix(cm, train_dataset.class_indices, cmap='Reds')
```

```
Confusion matrix, without normalization
[[ 873    4   11   29    5   31   62    0   16   22]
 [   5  972    4    1    0   44    0    0    9   31]
 [  26   36  780   28   25   15   95   36   28    8]
 [  10    1   16  695   49   16   34   33   40    2]
 [   0    0    5   19  820    0    2   43   17    0]
 [  23   12   14   26    0  585   25    0   12   11]
 [  40    0  103   48   18   22  642   16    3    0]
 [   0    0   21   14   33    0    7  988    0    0]
 [  26   12   10   73    9   40   17    0  700    6]
 [   8   10    0    0    0    4    0    0    1 1047]]
```



In [19]:
```
# model = tf.keras.applications.ResNet101(
#     include_top=False,
#     input_tensor=None,
#     input_shape=(64,64, 3),
#     classes=10,
# )
# model.summary()
```

Load model

In [20]:
```
from tensorflow.keras.models import load_model
model = load_model(r"lulc")
```

Thank you