Aalto University

CS-C3240 - Machine Learning D

From Deserts to Golf Courts: Machine Learning Techniques for Aerial Image Categorization

Submitted: October 11, 2023

Figure 1: 7 diferent categories from the dataset with two images per category.

1 Introduction

More and more nations in the world have access to satellites, and thus the amount of data captured from above has exploded in recent decades. Many sectors eg. military and defence - for reconnaissance and surveillance -, urban planning, development and environmental management, mining and numerous other sectors are interested in insights that can be derived from aerial images. Large patches of land can be made more easily understandable for humans by labelling land cover types into certain categories that different sectors are interested in. Domain experts can focus their attention on certain areas of interest and analyze these images in detail saving time, or some large-scale analytics of land can be derived based on the land cover type labelling.

Using machine learning models for satellite imagery has already been implemented in research a lot, and although this report does not prove anything new, it at least establishes the ease of training these kinds of complex models with computing power that is accessible already for a significant proportion in the world. In this report, two different models are used to predict the land cover type based on pixel values from aerial images captured from satellites. The problem is formulated in section 2. The chosen models and the rationale behind the decision to use them are discussed in section 3. Section 4 presents a comparative analysis of the two models, and the best model is chosen. Section 5 sums up the findings and discusses some future trajectories this project could proceed with. The references section contains links to repositories where the majority of the code for the other model was taken from, and all the code used in this project is contained in the appendix.

2 Problem formulation

This project aims to classify aerial images that represent different land cover into 45 different scene classes that are used as labels for the problem. Altogether the dataset has 31 500 images with 700 images for each of the 45 classes. The image data is multidimensional 256x256x3 8-bit valued RGB images in JPG format, with 256 by 256 pixels in size and the latter 3 dimensions relating to the red, green and blue values in each pixel which can have values ranging from 0 to 255. There is some variation in the angle, lighting and other ambient factors between the images.

The dataset is from Northwestern Polytechnical University NWPU and was created in March 2022 (Junwei Han). The dataset categories range from desert to basketball and golf courts, power plants, train stations and many more. The human-labelled categories are used as the labels in the machine-learning model. The images, which are used as features for the model, have been taken from Google Earth and labelled by researchers. All of the land type cover categories can be found in Table 1 in the appendix.

3 Methods

The problem of classifying images into 45 different labels, that describe the type of landcover based on aerial images, is approached initially by training a convolutional neural network using supervised learning. To further make the labels clear they represent the type of land cover based on domain expert classification given to them. The images upon visual inspection by a layman look different in relation to colour, texture, and topology which motivates using them as features

for the model, and classifying them into different categories and assigning labels to each category. All of the different types of land cover can be found in the appendix in the code, and 7 different categories with two different images per category are displayed in Figure 1.

Convolutional neural networks have demonstrated good performance in image detection (Gu 2015). There exist many implementations of easily available CNNs such as ResNet50 developed by Microsoft Research. In this project, the afore-described scene classification is modelled with ResNet50 which stands for a residual network architecture that consists of 50 layers. The reason for using a convolutional neural network is based on the difficulty of the task. It is difficult to define what kind of filter should be used to transform the images, thus a more general approach with a model that does not require any domain-specific knowledge is a lot easier to implement. The neural network model is trained to recognize the images and extract relevant information from the images to predict labels. The residual network family employs skip connections in its' structure meaning that it can bypass certain hidden layers in the model. This kind of architecture enables the learning of more complex features from the training data (Gu, 2015) which fits the classifying problem under investigation.

Feature engineering on the images was performed by resizing the images. The images were downscaled from 256 by 256 pixels down to 64 by 64 pixels using anti-aliasing from the Python imaging library, Pillow. This step effectively shortened the training time for the model without losing too much detail and variety in the images.

In this project, cross-entropy loss is used as the model's loss function. It is particularly often used with convolutional neural networks when the aim is to classify more than two categories. The formula is displayed in Equation 1. In particular, it penalises overly confident models because it measures the difference between the true distribution and the distribution predicted by the model. Furthermore, it is based on logarithms which suit the problem particularly well when there are small deviations from the optimal predictions. Meaning that when the model predicts an almost correct class the penalty will not be very large compared to overconfident wrong predictions.

The training, validation and test data are split from the main data. To effectively train the convolutional neural network in a short timeframe, the main data is split in a 4:3 ratio. This means that 400 images from each class are used for training and validation. Furthermore, the training and validation dataset is split using a 20/80 split originating from the Pareto principle. 80% of the data is used for training and the rest 20% is used for validation and testing. Thus there will be 14 400 images for the training set and the rest, 3600 for validation which will determine the best model for the problem. The rest, 3 parts of the data or 13 500 images are used as the final test set to evaluate the best chosen model. This kind of split ensures that there is enough data to train the model and to further prevent overfitting for too small datasets.

The second machine learning model is logistic regression. It was chosen for its relative simplicity in relation to more complex models like the CNNs (convolutional neural networks, ie. ResNet50). Although images can be very complex, there can exist some linear relationships in them, for example, the amount of blue in an image can effectively contribute to how much water it contains, and probably correlates quite well with it being a lake, sea or beach land cover type. For logistic regression, a one-vs-all approach is used that expands binary classification to multiclass classification. Logistic regression also uses cross-entropy loss for a multiclass classification problem. Unlike CNNs, logistic regression models don't handle image data directly, and thus each image was converted into a 1D feature vector which represents all the pixel values in the images, and also contains all of the red green and blue values, as one column. Meaning that all of the data in one image is contained in a single column vector. The feature vectors for each image end up containing 12 288 values, with 64 x 64 pixels and 3 values for red, green and blue values. The logistic regression serves as a baseline, providing contrast to the more detailed and complex ResNet50 CNN model.

For the CNN training adaptive moment estimation was used, which is also known as 'adam' which dynamically adjusts the network weights and learning rate to achieve faster convergence for the model.

4 Results

For logistic regression, the training accuracy is 15% and validation accuracy is 2.2%, while for the convolutional neural network, the training accuracy is 86.9% and validation accuracy is 49.5%. On the other hand, the training error from cross-entropy loss is 0.45 for CNN, while the validation error is 2.3. For the logistic regression, the training error is 0.20 and the validation error is 5.2. Both of these errors are calculated using the cross-entropy loss also known as log loss.

Based on the metrics CNN was chosen as the best method, since the validation accuracy is that much higher in comparison to logistic regression. It is also clear that the logistic regression is overfitting the training data since the model accuracy is smaller and the loss is greater in the validation set. The test error for the final chosen CNN employing ResNet50 architecture is 2.4 and the accuracy is 49.7% using the 300 images left in each class that add together to 13 500 images. The performance is shown in figure 4.

Visual inspection of the confusion matrix for logistic regression shows that it is able to classify certain land cover types with a higher precision than other types of land cover. For desert land type labels the precision is 64%, and around 50% precision is also achieved for island, lake and meadow land type labels. From a heuristic standpoint, it seems likely the logistic regression can extract well certain colours from the images and make predictions based on colour since these types of land cover have a linear relationship to the distinctive colours compared to the other types. By a blind guess, the expected likelihood of getting a label correct for an image is $1/45$ and thus the validation accuracy of 2.2% is similar to blind guessing the labels. Logistic regression is overfitting the training data since the accuracy is around 7 times higher for the training data. All in all, this is expected from a naive implementation using a logistic regression model, that does not understand more complex patterns in the image. The convolutional neural network performs surprisingly well with just 16 epochs over the training data, ie. the model is trained 16 times over the same data with iterations having a tuning effect on the network weights. Based on how the training, and validation data accuracy change in relation to the epochs in Figure 6.2 it is clear that the model has not achieved a plateau in training. More computing time, by iterating more times over the training data, could make the model perform even better. It seems plausible that the model is slightly overfitting the training data since the validation accuracy is quite a lot worse with 80% in training versus 50% in validation and testing. Furthermore, the model seems to hit a plateau in the later epochs in terms of the worse performance in the validation set, but it is hard to make any greater conclusions based on such a small amount of epochs. For complex CNN models, and this kind of task of classifying images into 45 different categories more training effort is expected. To continue this project, an interesting trajectory could be to try to train the model with at least 100 epochs, ie. going over the training data 100 times. One downside that should be mentioned in relation to the CNN is the effort to train the model. The CNN was trained for 6 hours, with each epoch taking around 20 minutes, while the logistic regression was trained for 30 minutes with 1000 iterations over the training data. Based on the metrics, namely accuracy, the CNN is clearly a better model for the classification problem which was expected based on the literature.

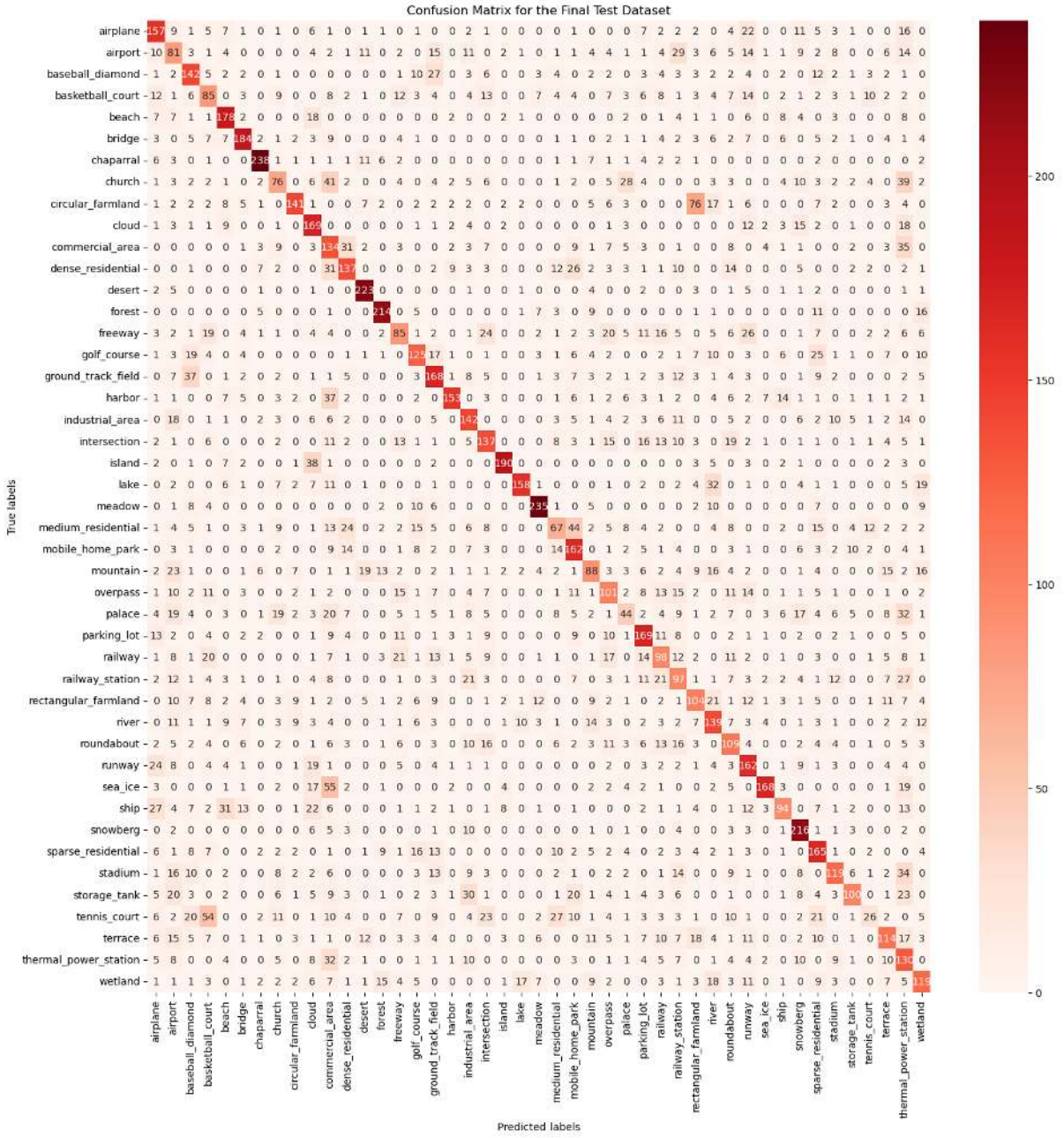


Figure 2: Confusion matrix for CNN ResNet50 for the test data.

5 Conclusion

This report describes the application of two different machine learning models to classify aerial images into 45 different categories. Convolutional neural networks are seen as a better option for the classification problem as was concluded based on the metrics in section 4.

The aerial images were downscaled from 256 by 256 to 64 pixels. The downscaling has most likely led to the disappearance of some topological features from the aerial images. By using higher-quality images, and possibly a wider band stretching towards the infrared region in the spectrum, the model could perform with better results. Furthermore, it seems there is a lot of room for improvement since the error of the model seems to go down with newer epochs meaning that we could continue training the model with new iterations over the training data, and build a better, more accurate model. This would require more computing power and time to train the model. Considering a baseline implementation the problem has been solved satisfactorily. If a domain expert would want to pick up certain land cover types that are of interest this convolutional neural network model would give correct labels to images around 50 percent of the time, and considering there are 45 categories even for a human it would not be an easy task.

References

[1] Gu 2015, https://arxiv.org/abs/1512.07108

[2] Han 2023, RESISC45 aerial image data set

`[Accessed 22 Sep. 2023], http://www.escience.cn/people/JunweiHan/NWPU-RESISC45.html`
`https://meta-album.github.io/datasets/RESISC.html`

[3] `https://github.com/iamtekson/DL-for-LULC-prediction/blob/master/lulc_classification_euroS`
`EuroSAT code github`

[4] `https://github.com/Sudhandar/ResNet-50-model/blob/master/main_script.py`
`ResNet50 github Sundhandar`

This report uses a different dataset, but identical functions as in the repository in references [3] and [4], and some of the same plotting and visualisation with minor changes. It is specified by blocks in the appendix whether or not code has been directly copied from the above repository (the logistic regression code is not taken from any specific repository).

6 Appendices

6.1 Table 1

airplane	airport	baseball_diamond	basketball_court	
beach	bridge	chaparral	church	
circular_farmland	cloud	commercial_area	dense_residential	
desert	forest	freeway	golf_course	
ground_track_field	harbor	industrial_area	intersection	
island	lake	meadow	medium_residential	
mobile_home_park	mountain	overpass	palace	
parking_lot	railway	railway_station	rectangular_farmland	
river	roundabout	runway	sea_ice	
ship	snowberg	sparse_residential	stadium	
storage_tank	tennis_court	terrace	thermal_power_station	
wetland				

Table 1: List of land cover types

6.2 Equation 1

Cross-entropy loss or log loss:

$$L(y,p) = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(p_i) + (1-y_i)\log(1-p_i)\right] \tag{1}$$

N number of samples, y is binary indicator (0 or 1) if belongs to sample, and p is the model prediction probability of belonging to certain class

Multi-class classification, the cross-entropy loss is written as:

$$L(y,p) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(p_{ij}) \tag{2}$$

M number of classes, yij the binary indicator, pij the model prediction of sample belonging to class j

Figure 3: Containing the full 45 categories, two images per category with small labels on top of each picture. All of the images are downscaled to 64 by 64 pixels. How many could you categorize correctly?

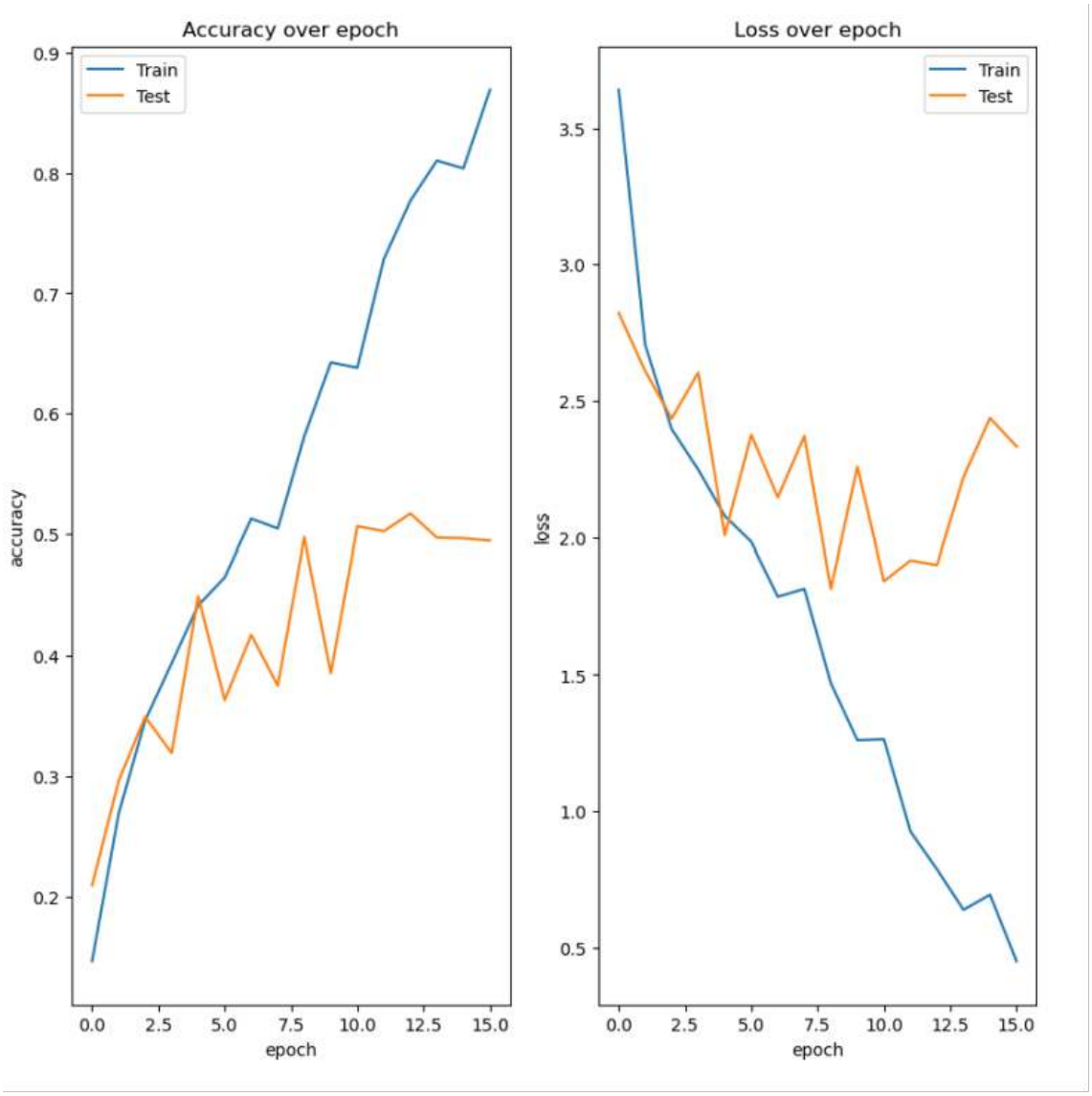


Figure 4: CNN model accuracy and loss over epochs.

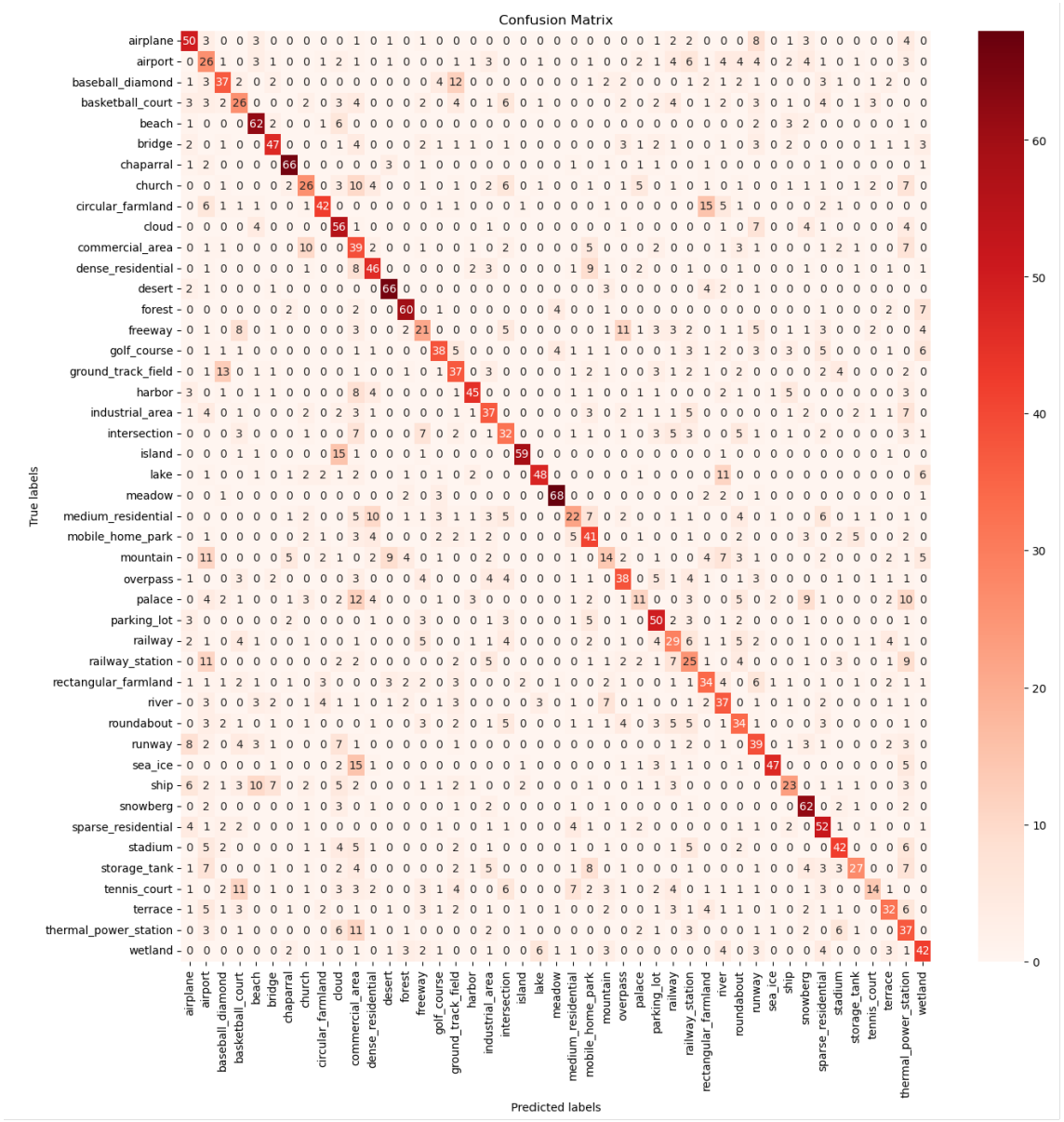


Figure 5: Confusion matrix for CNN ResNet50 for the validation data.

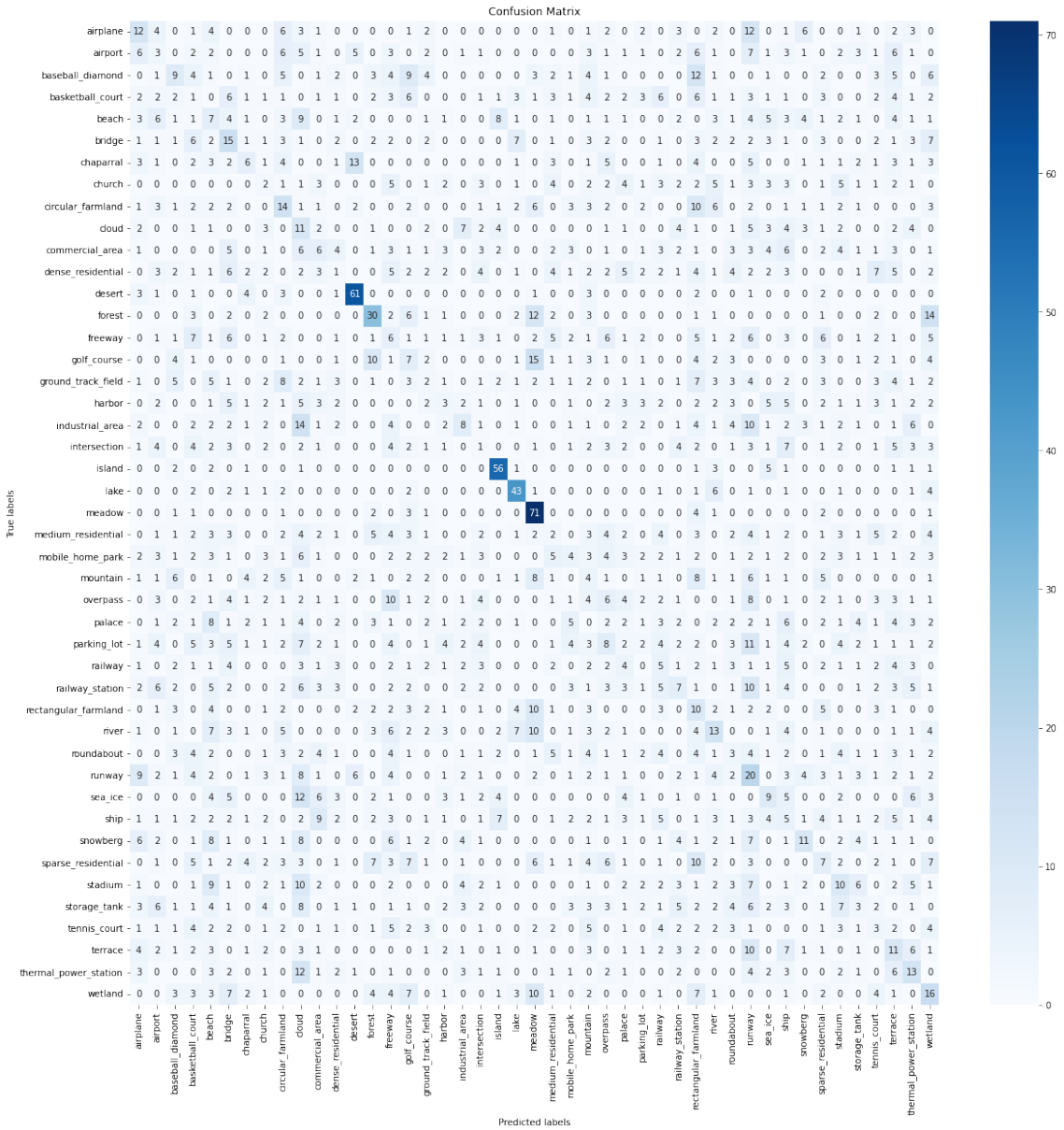


Figure 6: Confusion matrix for the logistic regression model for the validation data.

many parts of this code are copied from the github repository defined in the code blocks, the code was clear and from a pedacogic standpoint shows the underlying stucture of the ResNet50 model, that is why it was used

test_resnet-Copy1

October 10, 2023

[1]:
```
import os
import glob

import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

#tensorflow for ResNet50
import tensorflow.python.keras as k
import tensorflow as tf
from tensorflow.keras.layers import Input, Add, Dense, Activation,␣
 ↪ZeroPadding2D, BatchNormalization, Flatten, Conv2D, AveragePooling2D,␣
 ↪MaxPooling2D, GlobalMaxPooling2D
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, confusion_matrix
%matplotlib inline
```

[2]:
```
# import zipfile
# with zipfile.ZipFile('NWPU_lowy400.zip', 'r') as zip_ref:
#     zip_ref.extractall('.')
```

[3]:
```
#!rm -r NWPU_medy400.zip

#Feature Engineering; resizing the images. Based on experimental verification
#with 256 x 256 pixels and full dataset, training the model would take␣
 ↪according to Jupyter 100+ hours which is not feasible for the time
from PIL import Image
import os

def resize_images(input_directory, output_directory, target_size=(64, 64)):
    os.makedirs(output_directory)
    for foldername in os.listdir(input_directory):
        folder_path = os.path.join(input_directory, foldername)
        output_folder_path = os.path.join(output_directory, foldername)
```

```
        for filename in os.listdir(folder_path):
            image_path = os.path.join(folder_path, filename)
            image = Image.open(image_path)
            image_resized = image.resize(target_size, Image.ANTIALIAS)
            output_image_path = os.path.join(output_folder_path, filename)
            image_resized.save(output_image_path)

input_dir = "NWPU-RESISC45"
output_dir = "NWPU_medy"
resize_images(input_dir, output_dir)
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[3], line 22
     20 input_dir = "NWPU-RESISC45"
     21 output_dir = "NWPU_medy"
---> 22 resize_images(input_dir, output_dir)

Cell In[3], line 10, in resize_images(input_directory, output_directory,␣
 ↪target_size)
      8 def resize_images(input_directory, output_directory, target_size=(64,␣
 ↪64)):
      9     os.makedirs(output_directory)
---> 10     for foldername in os.listdir(input_directory):
     11         folder_path = os.path.join(input_directory, foldername)
     12         output_folder_path = os.path.join(output_directory, foldername)

FileNotFoundError: [Errno 2] No such file or directory: 'NWPU-RESISC45'
```

[3]:
```
img_height = 64
img_width = 64
dataset_u = r'NWPU_lowy400'
batch_size = 32
validation_split=0.2
rescale=1.0/255
```

[4]:
```
#source directly from github, initially from Coursera: https://github.com/
 ↪amanchadha/coursera-deep-learning-specialization/blob/master/
 ↪C4%20-%20Convolutional%20Neural%20Networks/Week%202/ResNets/
 ↪Residual_Networks_v2a.ipynb
datagen = tf.keras.preprocessing.image.
 ↪ImageDataGenerator(validation_split=validation_split, rescale=rescale)
dataset = tf.keras.preprocessing.image_dataset_from_directory(dataset_u,␣
 ↪image_size=(img_height, img_width), batch_size=batch_size)
```

```
Found 18000 files belonging to 45 classes.
```

```
2023-10-10 09:18:00.487692: I tensorflow/core/platform/cpu_feature_guard.cc:151]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  SSE4.1 SSE4.2 AVX
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
```

[5]:
```
#source directly from github, initially from Coursera: https://github.com/
 ↪iamtekson/DL-for-LULC-prediction/blob/master/lulc_classification_euroSAT.
 ↪ipynb
train_dataset = datagen.flow_from_directory(batch_size=batch_size,
                                            directory=dataset_u,
                                            shuffle=True,
                                            target_size=(img_height, img_width),
                                            subset="training",
                                            class_mode='categorical')
```

```
Found 14400 images belonging to 45 classes.
```

[6]:
```
#source directly from github, initially from Coursera: https://github.com/
 ↪iamtekson/DL-for-LULC-prediction/blob/master/lulc_classification_euroSAT.
 ↪ipynb
test_dataset = datagen.flow_from_directory(batch_size=batch_size,
                                            directory=dataset_u,
                                            shuffle=True,
                                            target_size=(img_height, img_width),
                                            subset="validation",
                                            class_mode='categorical')
```

```
Found 3600 images belonging to 45 classes.
```

[7]:
```
#source directly github (with modification in range and size), initially from␣
 ↪Coursera: https://github.com/iamtekson/DL-for-LULC-prediction/blob/master/
 ↪lulc_classification_euroSAT.ipynb
class_names = dataset.class_names
plt.figure(figsize=(8, 8))
for images, labels in dataset.take(1):
    for i in range(32):
        ax = plt.subplot(7, 7, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[labels[i]])
        plt.axis("off")
```

[8]:
```
#function taken directly from github, initially from Coursera: https://github.
↪com/amanchadha/coursera-deep-learning-specialization/blob/master/
↪C4%20-%20Convolutional%20Neural%20Networks/Week%202/ResNets/
↪Residual_Networks_v2a.ipynb
def identity_block(X, f, filters, training=True, initializer=random_uniform):
    """
    Implementation of the identity block as defined in Figure 4

    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 path
    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 = initializer(seed=0))(X)
    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_initializer = initializer(seed=0))(X)
    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_initializer = initializer(seed=0))(X)
    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
```

[9]:
```
#function taken directly from github, initially from Coursera: https://github.
 ↪com/amanchadha/coursera-deep-learning-specialization/blob/master/
 ↪C4%20-%20Convolutional%20Neural%20Networks/Week%202/ResNets/
 ↪Residual_Networks_v2a.ipynb
def convolutional_block(X, f, filters, s = 2, training=True,␣
 ↪initializer=glorot_uniform):
    """
    Implementation of the convolutional block as defined in Figure 4

    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 path
    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 = initializer(seed=0))(X)
    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)
    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=0))(X)
    X = BatchNormalization(axis = 3)(X, training = training)

    ##### SHORTCUT PATH #### ( 2 lines)
    X_shortcut = Conv2D(F3, (1, 1), strides = (s, s), padding = 'valid',␣
↪kernel_initializer = initializer(seed=0))(X_shortcut)
    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 through a RELU activation
    X = Add()([X, X_shortcut])
    X = Activation('relu')(X)
```

```
    return X
```

[10]:
```
#function taken directly from github, initially from Coursera: https://github.
↪com/amanchadha/coursera-deep-learning-specialization/blob/master/
↪C4%20-%20Convolutional%20Neural%20Networks/Week%202/ResNets/
↪Residual_Networks_v2a.ipynb
def ResNet50(input_shape = (64, 64, 3), classes = 45):
    """
    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)
    #X = tf.keras.layers.GlobalAveragePooling2D()(X) #this was tested in our␣
 ↪model for Aalto ML



    # 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
```

[11]:
```
#source github, initially from Coursera: https://github.com/iamtekson/
 ↪DL-for-LULC-prediction/blob/master/lulc_classification_euroSAT.ipynb
model = ResNet50(input_shape=(64,64,3), classes=45)
```

[12]:
```
#source github, initially from Coursera: https://github.com/iamtekson/
 ↪DL-for-LULC-prediction/blob/master/lulc_classification_euroSAT.ipynb
model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
```

[13]:
```
#source github, initially from Coursera: https://github.com/iamtekson/
 ↪DL-for-LULC-prediction/blob/master/lulc_classification_euroSAT.ipynb
#can be increased to more than 9, but now in the timescale we have opt for 9
history = model.fit(train_dataset, validation_data=test_dataset, epochs=16,␣
 ↪batch_size=32)
```

```
Epoch 1/16
```

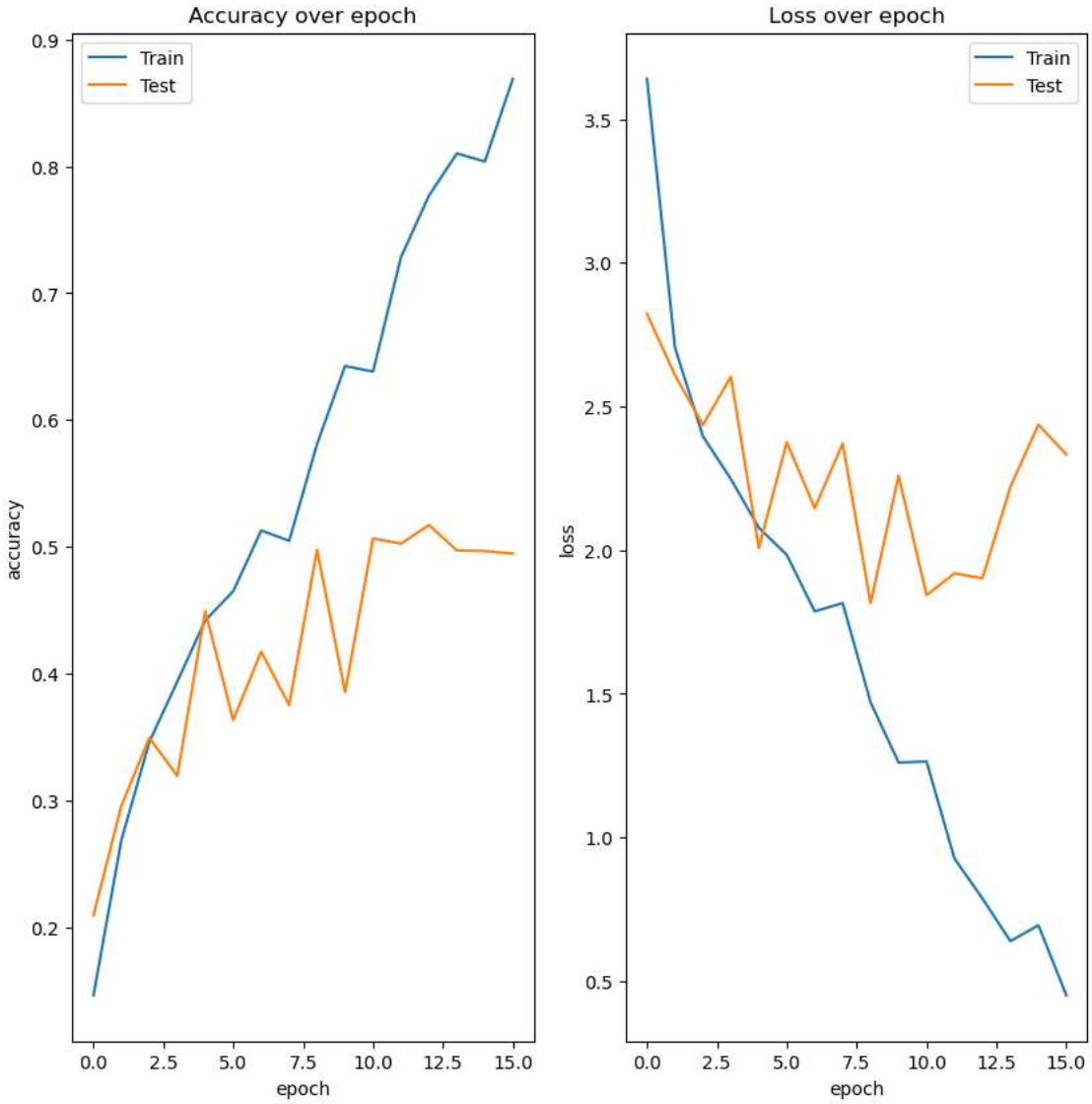
```
450/450 [==============================] - 1330s 3s/step - loss: 3.6408 -
accuracy: 0.1465 - val_loss: 2.8226 - val_accuracy: 0.2094
Epoch 2/16
450/450 [==============================] - 1252s 3s/step - loss: 2.7057 -
accuracy: 0.2689 - val_loss: 2.6093 - val_accuracy: 0.2961
Epoch 3/16
450/450 [==============================] - 1253s 3s/step - loss: 2.3963 -
accuracy: 0.3464 - val_loss: 2.4353 - val_accuracy: 0.3494
Epoch 4/16
450/450 [==============================] - 1253s 3s/step - loss: 2.2477 -
accuracy: 0.3939 - val_loss: 2.6038 - val_accuracy: 0.3194
Epoch 5/16
450/450 [==============================] - 1247s 3s/step - loss: 2.0780 -
accuracy: 0.4420 - val_loss: 2.0075 - val_accuracy: 0.4494
Epoch 6/16
450/450 [==============================] - 1281s 3s/step - loss: 1.9836 -
accuracy: 0.4651 - val_loss: 2.3758 - val_accuracy: 0.3633
Epoch 7/16
450/450 [==============================] - 1334s 3s/step - loss: 1.7868 -
accuracy: 0.5129 - val_loss: 2.1463 - val_accuracy: 0.4175
Epoch 8/16
450/450 [==============================] - 1384s 3s/step - loss: 1.8154 -
accuracy: 0.5048 - val_loss: 2.3715 - val_accuracy: 0.3753
Epoch 9/16
450/450 [==============================] - 1353s 3s/step - loss: 1.4697 -
accuracy: 0.5813 - val_loss: 1.8154 - val_accuracy: 0.4978
Epoch 10/16
450/450 [==============================] - 1291s 3s/step - loss: 1.2607 -
accuracy: 0.6426 - val_loss: 2.2589 - val_accuracy: 0.3856
Epoch 11/16
450/450 [==============================] - 1267s 3s/step - loss: 1.2641 -
accuracy: 0.6382 - val_loss: 1.8431 - val_accuracy: 0.5067
Epoch 12/16
450/450 [==============================] - 1301s 3s/step - loss: 0.9260 -
accuracy: 0.7284 - val_loss: 1.9192 - val_accuracy: 0.5025
Epoch 13/16
450/450 [==============================] - 1322s 3s/step - loss: 0.7871 -
accuracy: 0.7769 - val_loss: 1.9021 - val_accuracy: 0.5172
Epoch 14/16
450/450 [==============================] - 1293s 3s/step - loss: 0.6387 -
accuracy: 0.8101 - val_loss: 2.2200 - val_accuracy: 0.4972
Epoch 15/16
450/450 [==============================] - 1296s 3s/step - loss: 0.6939 -
accuracy: 0.8037 - val_loss: 2.4373 - val_accuracy: 0.4967
Epoch 16/16
450/450 [==============================] - 1285s 3s/step - loss: 0.4501 -
accuracy: 0.8690 - val_loss: 2.3330 - val_accuracy: 0.4947
```

[14]:
```
#source github, initially from Coursera: https://github.com/iamtekson/
↪DL-for-LULC-prediction/blob/master/lulc_classification_euroSAT.ipynb
model.save('CNN_16_epoch')
```

```
2023-10-10 15:05:13.207786: W tensorflow/python/util/util.cc:368] Sets are not
currently considered sequences, but this may change in the future, so consider
avoiding using them.

INFO:tensorflow:Assets written to: CNN_16_epoch/assets
```

[15]:
```
#source directly from github, initially from Coursera: https://github.com/
↪iamtekson/DL-for-LULC-prediction/blob/master/lulc_classification_euroSAT.
↪ipynb
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 10))
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")
```

[15]:
```
<matplotlib.legend.Legend at 0x7f25f46d1540>
```



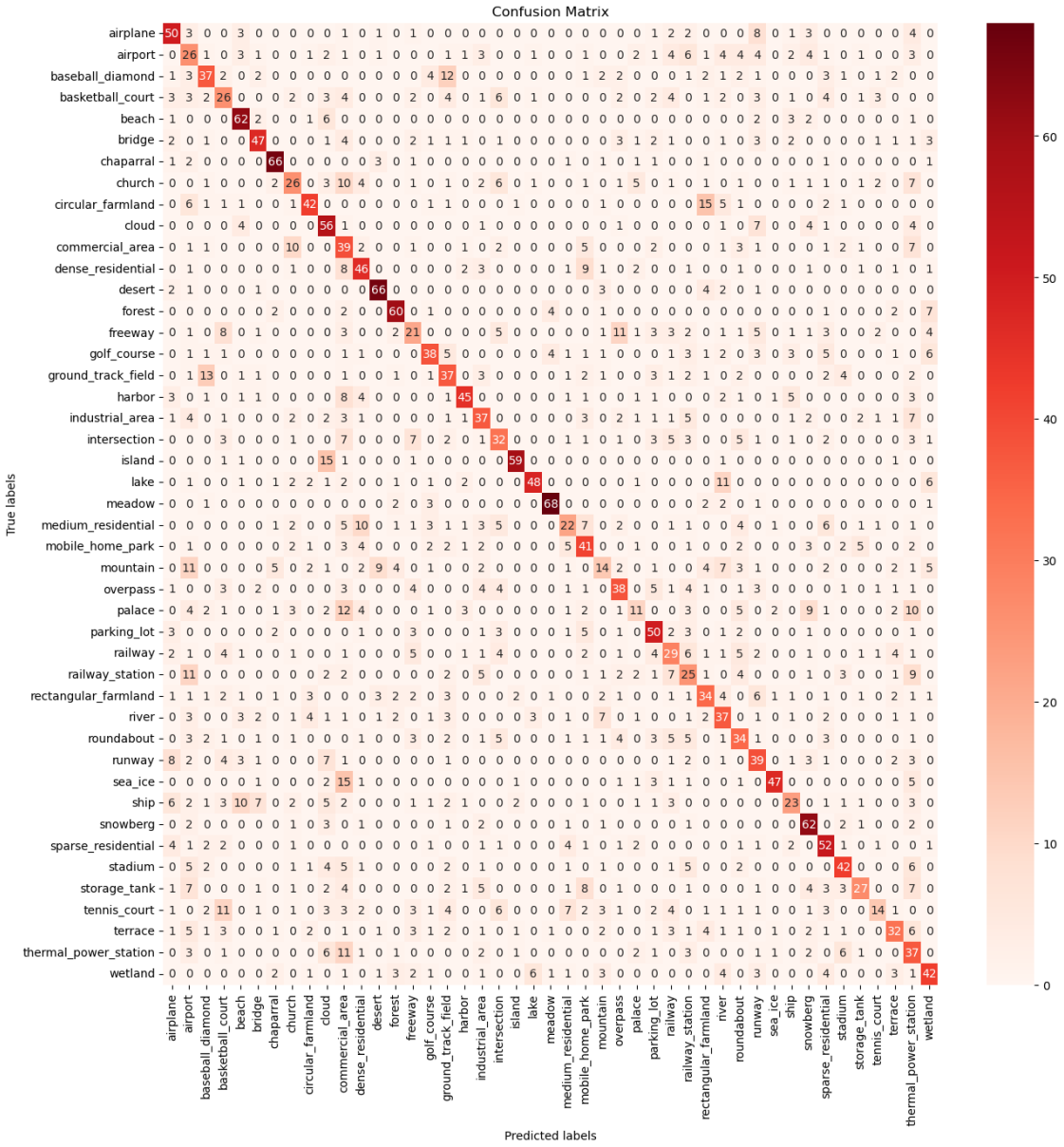
[16]:
```
y_pred = []#predicted values
y_true = []#true values, labeled in the dataset

#SOURCE,for loop structure from github same as above, for Aalto ML changed if␣
 ↪structure in the loop
for i, (image_batch, label_batch) in enumerate(test_dataset):
    y_true.append(label_batch)
    preds = model.predict(image_batch)
    y_pred.append(np.argmax(preds, axis =  1))
    if i==112:
        break
#SOURCE from github same as above
```

```
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)
```

[20]:
```
# print(len(y_pred))
# print(y_pred)
#from sklearn.metrics import classification_report
#print("\nClassification Report:\n", classification_report(y_true, y_pred))
cm = confusion_matrix(correct_labels, predicted_labels)
cm
```

[20]:
```
array([[50,  3,  0, …,  0,  4,  0],
       [ 0, 26,  1, …,  0,  3,  0],
       [ 1,  3, 37, …,  2,  0,  0],
       …,
       [ 1,  5,  1, …, 32,  6,  0],
       [ 0,  3,  0, …,  0, 37,  0],
       [ 0,  0,  0, …,  3,  1, 42]])
```

[21]:
```
folders = os.listdir('NWPU_lowy400')
plt.figure(figsize=(15, 15))
sns.heatmap(cm, annot=True, fmt='g', cmap='Reds', xticklabels=folders,␣
 ↪yticklabels=folders)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```



[22]:
```
print(folders)
```

```
['airplane', 'airport', 'baseball_diamond', 'basketball_court', 'beach',
'bridge', 'chaparral', 'church', 'circular_farmland', 'cloud',
'commercial_area', 'dense_residential', 'desert', 'forest', 'freeway',
'golf_course', 'ground_track_field', 'harbor', 'industrial_area',
'intersection', 'island', 'lake', 'meadow', 'medium_residential',
'mobile_home_park', 'mountain', 'overpass', 'palace', 'parking_lot', 'railway',
'railway_station', 'rectangular_farmland', 'river', 'roundabout', 'runway',
'sea_ice', 'ship', 'snowberg', 'sparse_residential', 'stadium', 'storage_tank',
'tennis_court', 'terrace', 'thermal_power_station', 'wetland']
```

[23]:
```
# Calculate validation accuracy
accuracy = cm.diagonal().sum() / cm.sum()
print(f"Validation Accuracy: {accuracy:.4f}")
```

```
Validation Accuracy: 0.4961
```

[ ]:
```

```

final_562CNN

October 11, 2023

[1]:
```
test_dataset_new = r'NWPU_lowy400test'
```

[2]:
```
import os
import glob

import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

#tensorflow for ResNet50
import tensorflow.python.keras as k
import tensorflow as tf
from tensorflow.keras.layers import Input, Add, Dense, Activation,␣
 ↪ZeroPadding2D, BatchNormalization, Flatten, Conv2D, AveragePooling2D,␣
 ↪MaxPooling2D, GlobalMaxPooling2D
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, confusion_matrix
%matplotlib inline
```

[3]:
```
img_height = 64
img_width = 64
dataset_u = r'NWPU_lowy400test'
batch_size = 32
validation_split=0.2
rescale=1.0/255
```

[4]:
```
test_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=rescale)
```

[5]:
```
model  = tf.keras.models.load_model('CNN_16_epoch')
```

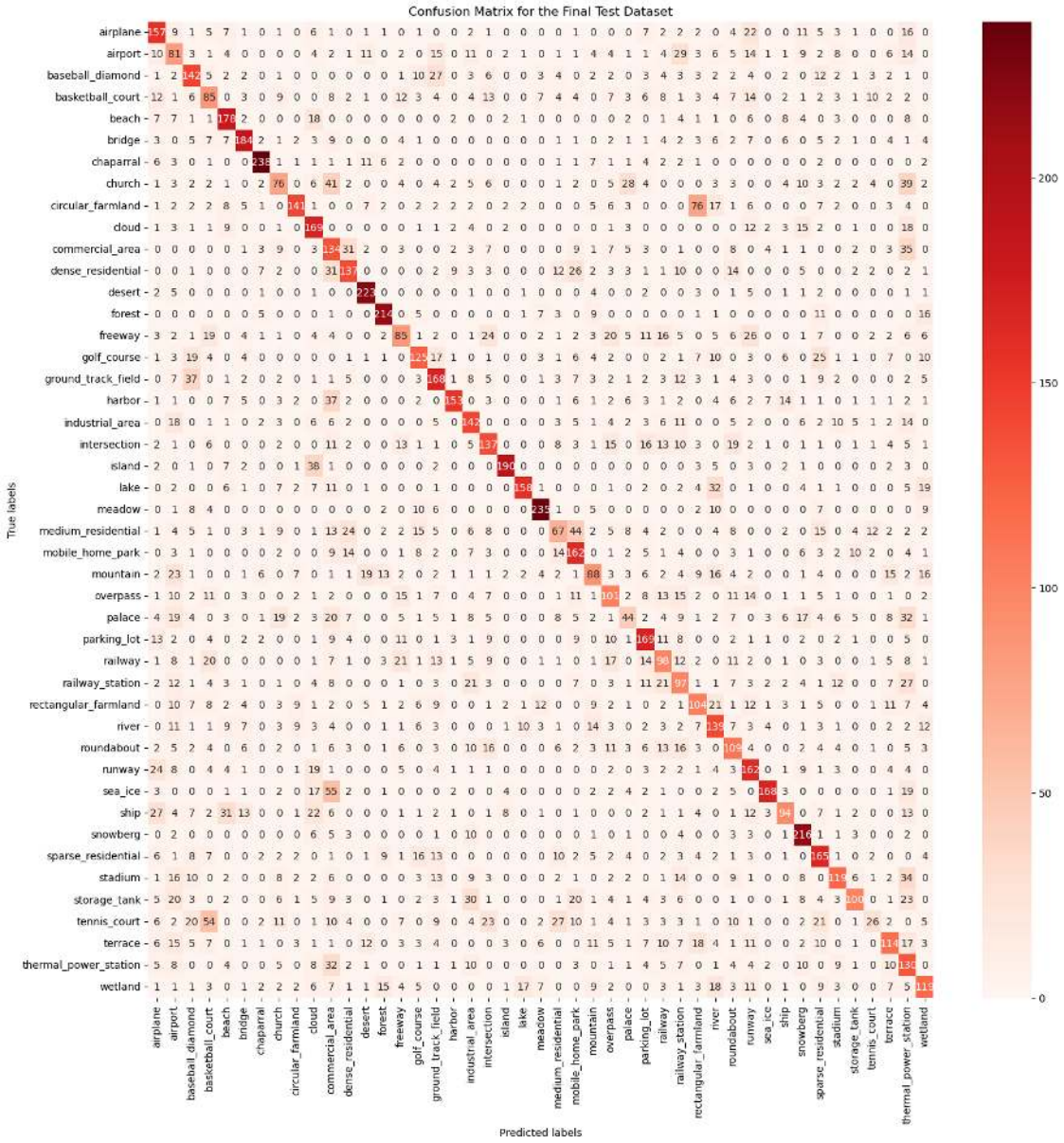
```
2023-10-11 11:12:34.883568: I tensorflow/core/platform/cpu_feature_guard.cc:151]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  SSE4.1 SSE4.2 AVX
```

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

[6]:
```
final_test_dataset = test_datagen.flow_from_directory(batch_size=batch_size,
                                                      ␣
 ↪directory=test_dataset_new,
                                                       shuffle=True,
                                                       target_size=(img_height,␣
 ↪img_width),
                                                       class_mode='categorical')
```

Found 13500 images belonging to 45 classes.

[11]:
```
y_pred = []  #predicted values
y_true = []  #true values

for i, (image_batch, label_batch) in enumerate(final_test_dataset):
    y_true.append(label_batch)
    preds = model.predict(image_batch)
    y_pred.append(np.argmax(preds, axis=1))
    if i == 385:#562:
        break

#concatenate the results
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)


cm = confusion_matrix(correct_labels, predicted_labels)

folders = os.listdir('NWPU_lowy400test')

plt.figure(figsize=(17, 17))
sns.heatmap(cm, annot=True, fmt='g', cmap='Reds', xticklabels=folders,␣
 ↪yticklabels=folders)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix for the Final Test Dataset')
plt.show()

# accuracy calculation
accuracy = cm.diagonal().sum() / cm.sum()
print(f"Final Test Accuracy: {accuracy:.4f}")
```



Final Test Accuracy: 0.4975

[12]:
```
results = model.evaluate(final_test_dataset)
```

422/422 [==============================] - 321s 761ms/step - loss: 2.3848 - accuracy: 0.4972

[13]:
```
print(results)
```

[2.3847761154174805, 0.4971851706504822]

[ ]:

In [ ]:
```
import os
import numpy as np
from PIL import Image
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

base_path = 'NWPU_lowy400'
folders = os.listdir(base_path)

X = []  # images that will be dim(1800,3(rgb)*64(height)*64(height)); features
y = []  # labels, 45 different categories


for label, folder in enumerate(folders):
    folder_path = os.path.join(base_path, folder)
    for image_name in os.listdir(folder_path):
        image_path = os.path.join(folder_path, image_name)
        img = Image.open(image_path)  # load with color
        X.append(np.array(img))
        y.append(label)
X = np.array(X)
y = np.array(y)
```

In [ ]:
```

```

In [ ]:
```
X = X.reshape(X.shape[0], -1)  #flatten each image to a 1D vector
X = X / 255.0  # normalize pixel values [0,1]

print(X.shape)
```

```
(18000, 12288)
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=40) #20/80 split

# using logistic regression
clf = LogisticRegression(max_iter=1000, verbose=1)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)
print(f"Accuracy: {accuracy_score(y_test, y_pred)*100:.2f}%")
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
c:\Users\Juliu\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed: 26.3min finished
Accuracy: 15.14%
```

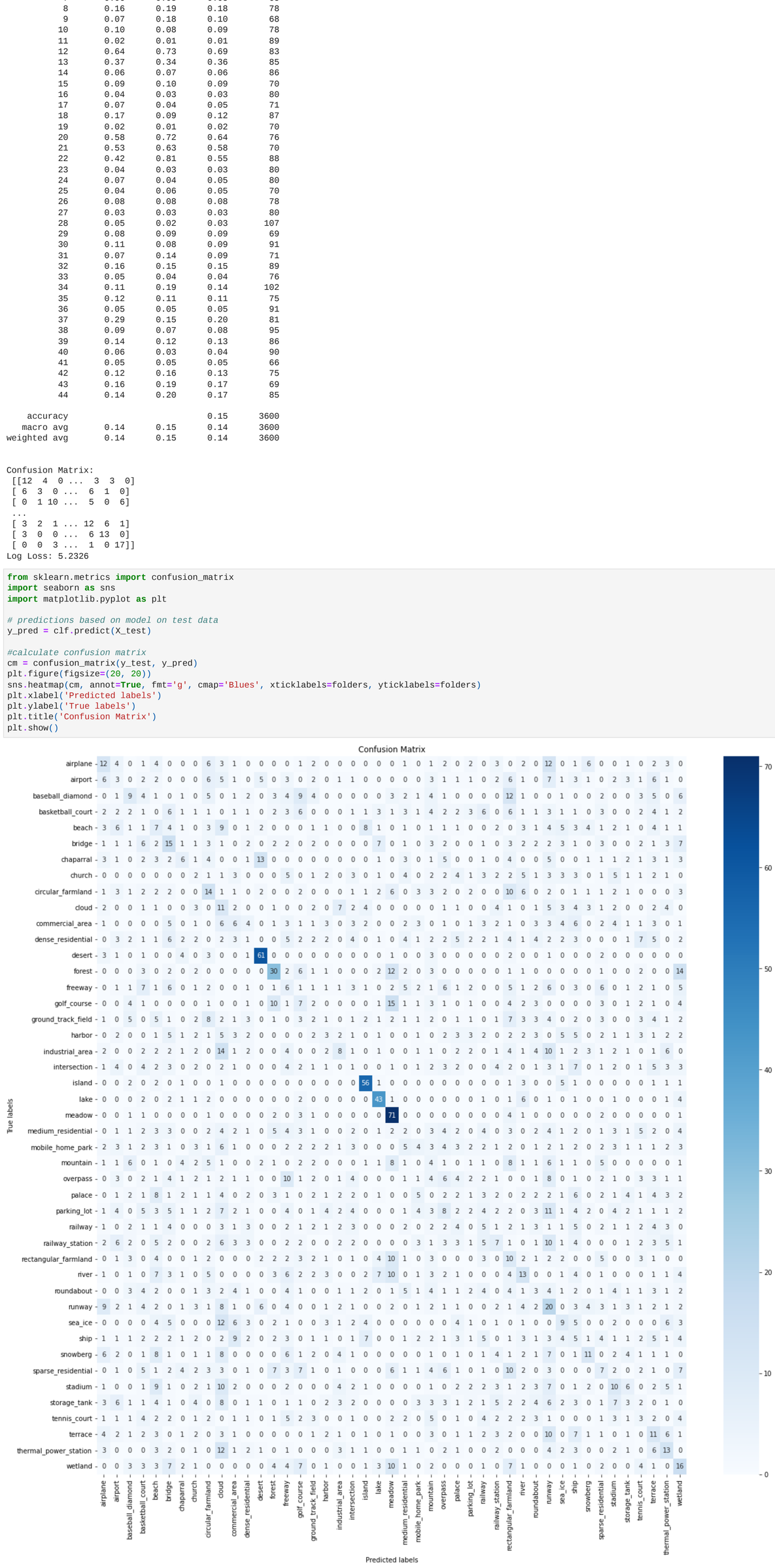
In [ ]:
```
from sklearn.metrics import log_loss

# Predict the probabilities on the training set
y_train_pred_proba = clf.predict_proba(X_train)

# Calculate log loss (cross-entropy loss) for the training set
train_loss = log_loss(y_train, y_train_pred_proba)

print(f"Training Log Loss: {train_loss:.4f}")
```

```
Training Log Loss: 0.1984
```

In [ ]:
```
from sklearn.metrics import classification_report, confusion_matrix

print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))

from sklearn.metrics import log_loss

y_pred_proba = clf.predict_proba(X_test)

# Calculate cross-entropy loss AKA log loss
loss = log_loss(y_test, y_pred_proba)
print(f"Log Loss: {loss:.4f}")
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.18      0.17      0.18        70
           1       0.04      0.04      0.04        77
           2       0.17      0.12      0.14        85
           3       0.01      0.01      0.01        81
           4       0.07      0.09      0.08        85
           5       0.13      0.17      0.15        83
           6       0.16      0.09      0.11        69
           7       0.06      0.05      0.05        65
           8       0.16      0.19      0.18        78
           9       0.07      0.18      0.10        68
          10       0.10      0.08      0.09        78
          11       0.02      0.01      0.01        89
          12       0.64      0.73      0.69        83
          13       0.37      0.34      0.36        85
          14       0.06      0.07      0.06        86
          15       0.09      0.10      0.09        70
          16       0.04      0.03      0.03        80
          17       0.07      0.04      0.05        71
          18       0.17      0.09      0.12        87
          19       0.02      0.01      0.02        70
          20       0.58      0.72      0.64        76
          21       0.53      0.63      0.58        70
          22       0.42      0.81      0.55        88
          23       0.04      0.03      0.03        80
          24       0.07      0.04      0.05        80
          25       0.04      0.06      0.05        70
          26       0.08      0.08      0.08        78
          27       0.03      0.03      0.03        80
          28       0.05      0.02      0.03       107
          29       0.08      0.09      0.09        69
          30       0.11      0.08      0.09        91
          31       0.07      0.14      0.09        71
          32       0.16      0.15      0.15        89
          33       0.05      0.04      0.04        76
          34       0.11      0.19      0.14       102
          35       0.12      0.11      0.11        75
          36       0.05      0.05      0.05        91
          37       0.29      0.15      0.20        81
          38       0.09      0.07      0.08        95
          39       0.14      0.12      0.13        86
          40       0.06      0.03      0.04        90
          41       0.05      0.05      0.05        66
          42       0.12      0.16      0.13        75
          43       0.16      0.19      0.17        69
          44       0.14      0.20      0.17        85

    accuracy                           0.15      3600
   macro avg       0.14      0.15      0.14      3600
weighted avg       0.14      0.15      0.14      3600


Confusion Matrix:
 [[12  4  0 ...  3  3  0]
 [ 6  3  0 ...  6  1  0]
 [ 0  1 10 ...  5  0  6]
 ...
 [ 3  2  1 ... 12  6  1]
 [ 3  0  0 ...  6 13  0]
 [ 0  0  3 ...  1  0 17]]
Log Loss: 5.2326
```

In [ ]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# predictions based on model on test data
y_pred = clf.predict(X_test)

#calculate confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(20, 20))
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues', xticklabels=folders, yticklabels=folders)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```



In [ ]:
```
# Calculate validation accuracy
accuracy = cm.diagonal().sum() / cm.sum()
print(f"Validation Accuracy: {accuracy:.4f}")
```

```
Validation Accuracy: 0.1503
```

In [ ]:
```
from sklearn.metrics import log_loss

y_test_pred_proba = clf.predict_proba(X_test)
loss_train = log_loss(y_test, y_test_pred_proba)

print(f"Validation Log Loss: {loss_train:.4f}")
```

```
Validation Log Loss: 5.2326
```