Eurosat Land Use and Land Cover Classification Using Deep Learning

May 25, 2022

1 Eurosat Land Use and Land Cover Classification Using Deep Learning

1.1 1: Introduction

Human activities on earth have caused rapid changes on land cover over the last decades. Monitoring land cover changes is important to understand the relationship between man and nature to provide decision makers with relevant information. The proliferation of satellite image data and the advanced machine and deep learning technologies have made it easier to monitor land cover and quantify changes automatically.

In this project, I applied different machine and deep learning technologies on Eurosat, a European Land Cover dataset, to classify different categories of land cover. I also compared the performance of different models on this dataset. This study tries to figure out which model performs best and offers best insights on future land cover classification using satellite images.

1.2 2: Data

Indented block

I used the Eurosat dataset which is a Sentinel-2 satellite image covering 13 spectral bands. The data contained 10 classes in 27000 images. The classes were SeaLake, AnnualCrop, Forest, Herbaceous Vegetation, Highway, Industrial, Pasture, Permanent crop, Residential and River.

Each class has 2400 images of 64 x 64 pixels with three color channels, red, green, and blue.

1.3 3: Methods

In this research, I applied deep learning machine methods such as CNN and transfer learning. I did Principal Component Analysis to reduce the dimensionality of the data before feeding it into the keras deep learning models. I also fine tuned the parameters of each model where later on I compared the performance of each model.

1.4 3.1: Deep Learning Algorithms

1.5 3.1.1 CNN

CNN is a deep learning algorithm that learns convolutional filters from input data automatically and identifies and filters all the important features from the data. I used a multiclass and also transfer learning methods to classify the Eurosat dataset. The multiclass was fed into the 10 labels.

For the transfer learning methods, I used VGG16 to fit and test our own datasets. My input shape was (64, 64, 3). I froze the weights of the pretrained layers at the beginning, and later updated the dense layers to get the output labels.

1.6 4: Results

1.7 4.1 CNN (Multiclass)

The multi-class classification (10 labels) has a test accuracy of 60%. This model did not perform very well on predicting most classes as it often mismatched them with other classes.

1.8 4.2: CNN (Transfer Learning Methods)

1.9 4.2.1: VGG16

VGG16 shows a testing accuracy (val_categorical_accuracy) of 87% despite the convolutional layers being frozen. This also shows that this pre-trained model is good for land classification applications in satellite images. The confusion matrix shows high accuracy for forest, residential and industrial classes while being weaker in permanent crop and herbaceous vegetation.

1.10 5: Summary

In summary, the CNN model performed as expected achieving accuracies of 87%. This agrees with the findings of comparable literature such as Helber et al. (2019) which noted that the recent use of the state-of-the art convolutional neural networks (CNN) is able to achieve superior results in image classification than other classifiers. I was able to meet my goal of comparing a series of machine learning algorithms to identify the best classifier for classifying land cover.

1.11 Reference:

[1] Helber, P., Bischke, B., Dengel, A., & Borth, D. (2019). Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 12(7), 2217-2226.

[1]:
```
import pandas as pd
import numpy as np
```

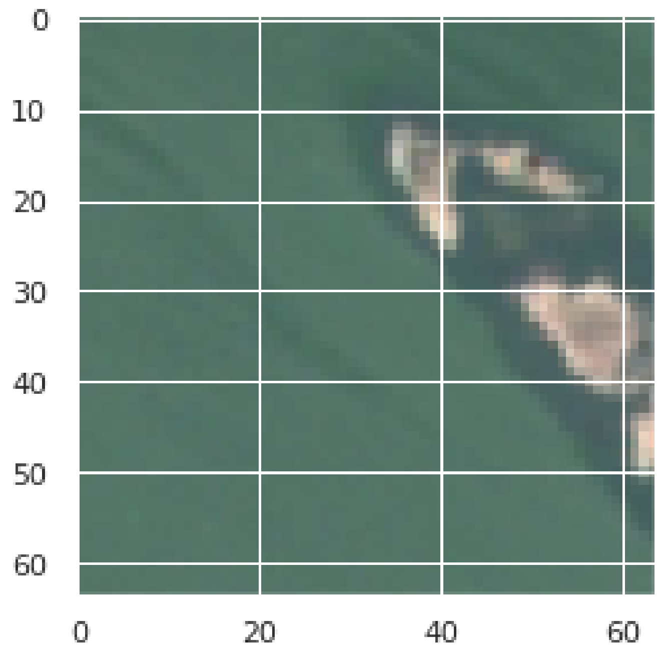
```python
import sklearn
import matplotlib.pyplot as plt
import keras
import seaborn as sns

from keras.utils import np_utils
from skimage.color import rgb2gray
from skimage.io import imread
from pathlib import Path
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

from keras.models import Sequential
from keras.layers import Conv2D, Flatten, Dropout, Dense, MaxPooling2D
from keras.utils.vis_utils import plot_model
from tensorflow.keras.optimizers import Adam
```

[2]:
```python
from google.colab import drive
drive.mount('/content/gdrive')
# root_path = 'gdrive/MyDrive/data/Eurosat/'
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call
drive.mount("/content/gdrive", force_remount=True).
```

[3]:
```python
import glob

imgdir = Path('/content/gdrive/MyDrive/data/2750')

imgfiles = []
for file in imgdir.glob("**/*.jp*g"):
  imgfiles.append(file)


# for file in glob.glob(imgdir + os.sep + "*" + os.sep + "*.jpg"):
#   imgfiles.append(file)
```

[37]:
```python
tmpimg = plt.imread(imgfiles[2400])
plt.imshow(tmpimg)
plt.show()
```




```
[5]: pd.set_option('display.max_columns', None)
     pd.set_option('display.width', 1000)
```

```
[6]: # df = pd.DataFrame({'path': list(imgdir.glob("**/*.jp*g"))})

     df = pd.DataFrame({'path': imgfiles})
```

```
[7]: df
```

```
[7]:                                                     path
     0      /content/gdrive/MyDrive/data/2750/SeaLake/SeaL…
     1      /content/gdrive/MyDrive/data/2750/SeaLake/SeaL…
     2      /content/gdrive/MyDrive/data/2750/SeaLake/SeaL…
     3      /content/gdrive/MyDrive/data/2750/SeaLake/SeaL…
     4      /content/gdrive/MyDrive/data/2750/SeaLake/SeaL…
     …                                                   …
     26995  /content/gdrive/MyDrive/data/2750/Pasture/Past…
     26996  /content/gdrive/MyDrive/data/2750/Pasture/Past…
     26997  /content/gdrive/MyDrive/data/2750/Pasture/Past…
     26998  /content/gdrive/MyDrive/data/2750/Pasture/Past…
     26999  /content/gdrive/MyDrive/data/2750/Pasture/Past…

     [27000 rows x 1 columns]
```

```
[8]: labels = df['path'].astype(str).str.split(r"/", expand=True)[7]
```

[9]:
```
new_labels = labels.str.split('_').str[0]
new_labels
```

[9]:
```
0        SeaLake
1        SeaLake
2        SeaLake
3        SeaLake
4        SeaLake
          …
26995    Pasture
26996    Pasture
26997    Pasture
26998    Pasture
26999    Pasture
Name: 7, Length: 27000, dtype: object
```

[10]:
```
new_labels.unique()
```

[10]:
```
array(['SeaLake', 'PermanentCrop', 'Forest', 'River', 'AnnualCrop',
       'HerbaceousVegetation', 'Industrial', 'Residential', 'Highway',
       'Pasture'], dtype=object)
```

[11]:
```
# labels = np.vstack()

eurosat_labels = np.vstack(new_labels).reshape(27000)

eurosat_labels
```

[11]:
```
array(['SeaLake', 'SeaLake', 'SeaLake', …, 'Pasture', 'Pasture',
       'Pasture'], dtype='<U20')
```

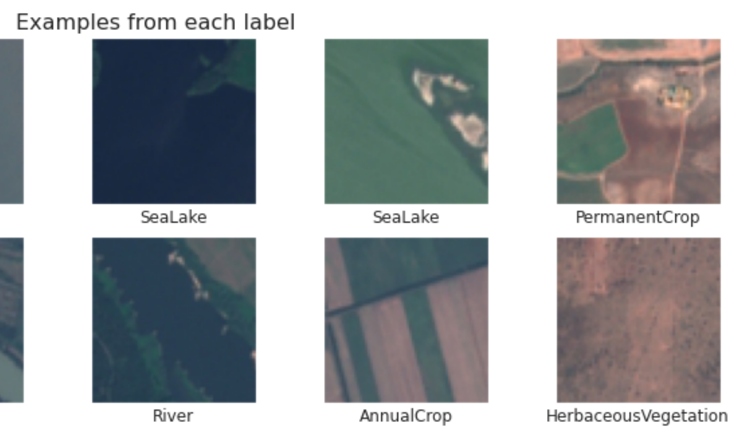
[12]:
```
# change label to integers
dictionary = { "SeaLake":0, "PermanentCrop":1, "Forest":2, "River":3,␣
↪"AnnualCrop":4,"HerbaceousVegetation":5,"Industrial":6,"Residential":
↪7,"Highway":8,"Pasture":9 }

final_labels = [dictionary[letter] for letter in eurosat_labels]
```

[13]:
```
from numpy import load
# load array
mat = load('/content/gdrive/MyDrive/data/final.npy')
# print the array
# print(data)
```

[14]:
```
mat.shape
```

[14]:
```
(27000, 64, 64, 3)
```

[78]:
```
fig = plt.figure(figsize=(15,5))
j=1
for i in [0,800,1600,2400,4800,7200,9600,10800,12000,14000]:
    plt.subplot(2,5,j)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(mat[i]/255) ## This step is because plt has a different␣
 ↪convention for color image axises
    plt.xlabel(new_labels[i])
    j=j+1
plt.suptitle("Examples from each label",fontsize=16,y=0.93)
plt.show()
```



[15]:
```
X_train, X_test, y_train, y_test = train_test_split(mat, final_labels, stratify␣
 ↪= final_labels, train_size=0.5, random_state = 10)
```

[16]:
```
X_train.shape
```

[16]: (13500, 64, 64, 3)

[17]:
```
# Convert class vectors to binary class matrices

num_classes = 10

y_train = np_utils.to_categorical(y_train, num_classes)
y_test = np_utils.to_categorical(y_test, num_classes)
```

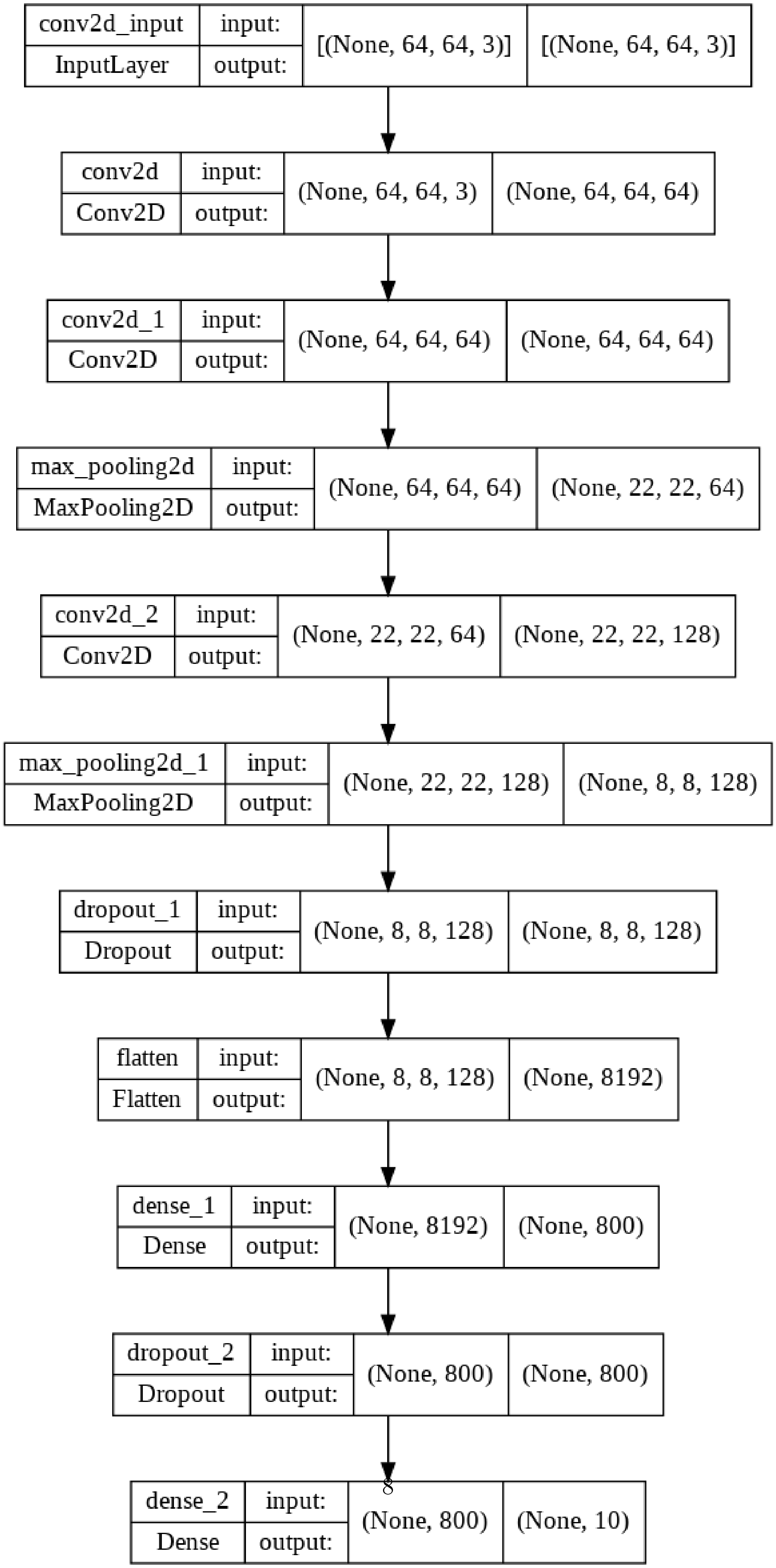
[82]:
```
y_train.shape
```

[82]: (13500, 10)

[83]:
```python
# Deep Learning Model

model = Sequential()
model.add(Conv2D(64, kernel_size=(3,3),
                 activation = 'relu',
                 input_shape = (64, 64, 3),
                 data_format='channels_last',
                 padding ="same"))
model.add(Conv2D(64, kernel_size=(3,3),
                 activation = 'relu',
                 data_format = 'channels_last',
                 padding = 'same'))
model.add(MaxPooling2D(pool_size = (3, 3), padding = "same"))
model.add(Conv2D(128, kernel_size=(3, 3),
                 activation = 'relu',
                 data_format = 'channels_last',
                 padding = 'same'))
model.add(MaxPooling2D(pool_size = (3, 3), padding = "same"))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(800, activation = "relu") )
model.add(Dropout(0.25))
model.add(Dense(10, activation = 'softmax'))
```

[84]:
```python
plot_model(model, show_shapes=True, show_layer_names=True)
```

[84]:



conv2d_input
InputLayer
input:
output:
[(None, 64, 64, 3)]
[(None, 64, 64, 3)]
conv2d
Conv2D
input:
output:
(None, 64, 64, 3)
(None, 64, 64, 64)
conv2d_1
Conv2D
input:
output:
(None, 64, 64, 64)
(None, 64, 64, 64)
max_pooling2d
MaxPooling2D
input:
output:
(None, 64, 64, 64)
(None, 22, 22, 64)
conv2d_2
Conv2D
input:
output:
(None, 22, 22, 64)
(None, 22, 22, 128)
max_pooling2d_1
MaxPooling2D
input:
output:
(None, 22, 22, 128)
(None, 8, 8, 128)
dropout_1
Dropout
input:
output:
(None, 8, 8, 128)
(None, 8, 8, 128)
flatten
Flatten
input:
output:
(None, 8, 8, 128)
(None, 8192)
dense_1
Dense
input:
output:
(None, 8192)
(None, 800)
dropout_2
Dropout
input:
output:
(None, 800)
(None, 800)
dense_2
Dense
input:
output:
(None, 800)
(None, 10)

[85]:
```
# compile the model

model.compile(loss ='categorical_crossentropy',
              optimizer=Adam(),
              metrics = ['accuracy'])
```

[86]:
```
# fit the model

history = model.fit(X_train, y_train,
                    batch_size=1200,
                    epochs=10,
                    verbose=1,
                    validation_data=(X_test, y_test))
```

```
Epoch 1/10
12/12 [==============================] - 16s 948ms/step - loss: 46.8017 -
accuracy: 0.1308 - val_loss: 2.4147 - val_accuracy: 0.1877
Epoch 2/10
12/12 [==============================] - 8s 655ms/step - loss: 2.0023 -
accuracy: 0.2820 - val_loss: 1.6555 - val_accuracy: 0.3722
Epoch 3/10
12/12 [==============================] - 8s 650ms/step - loss: 1.6278 -
accuracy: 0.3961 - val_loss: 1.5630 - val_accuracy: 0.4095
Epoch 4/10
12/12 [==============================] - 8s 642ms/step - loss: 1.4993 -
accuracy: 0.4379 - val_loss: 1.2546 - val_accuracy: 0.5358
Epoch 5/10
12/12 [==============================] - 7s 636ms/step - loss: 1.2928 -
accuracy: 0.5281 - val_loss: 1.1974 - val_accuracy: 0.5765
Epoch 6/10
12/12 [==============================] - 8s 648ms/step - loss: 1.2103 -
accuracy: 0.5701 - val_loss: 1.0335 - val_accuracy: 0.6409
Epoch 7/10
12/12 [==============================] - 8s 639ms/step - loss: 1.0912 -
accuracy: 0.6073 - val_loss: 0.9904 - val_accuracy: 0.6516
Epoch 8/10
12/12 [==============================] - 7s 636ms/step - loss: 1.0665 -
accuracy: 0.6187 - val_loss: 0.9497 - val_accuracy: 0.6656
Epoch 9/10
12/12 [==============================] - 8s 643ms/step - loss: 1.0410 -
accuracy: 0.6220 - val_loss: 0.9683 - val_accuracy: 0.6675
Epoch 10/10
12/12 [==============================] - 7s 630ms/step - loss: 0.9601 -
accuracy: 0.6605 - val_loss: 1.0718 - val_accuracy: 0.6064
```

[87]:
```python
# evaluate the model

score = model.evaluate(X_test, y_test, verbose = 0)
print("Test Score", score[0])
print("Test Accuracy", score[1])
```

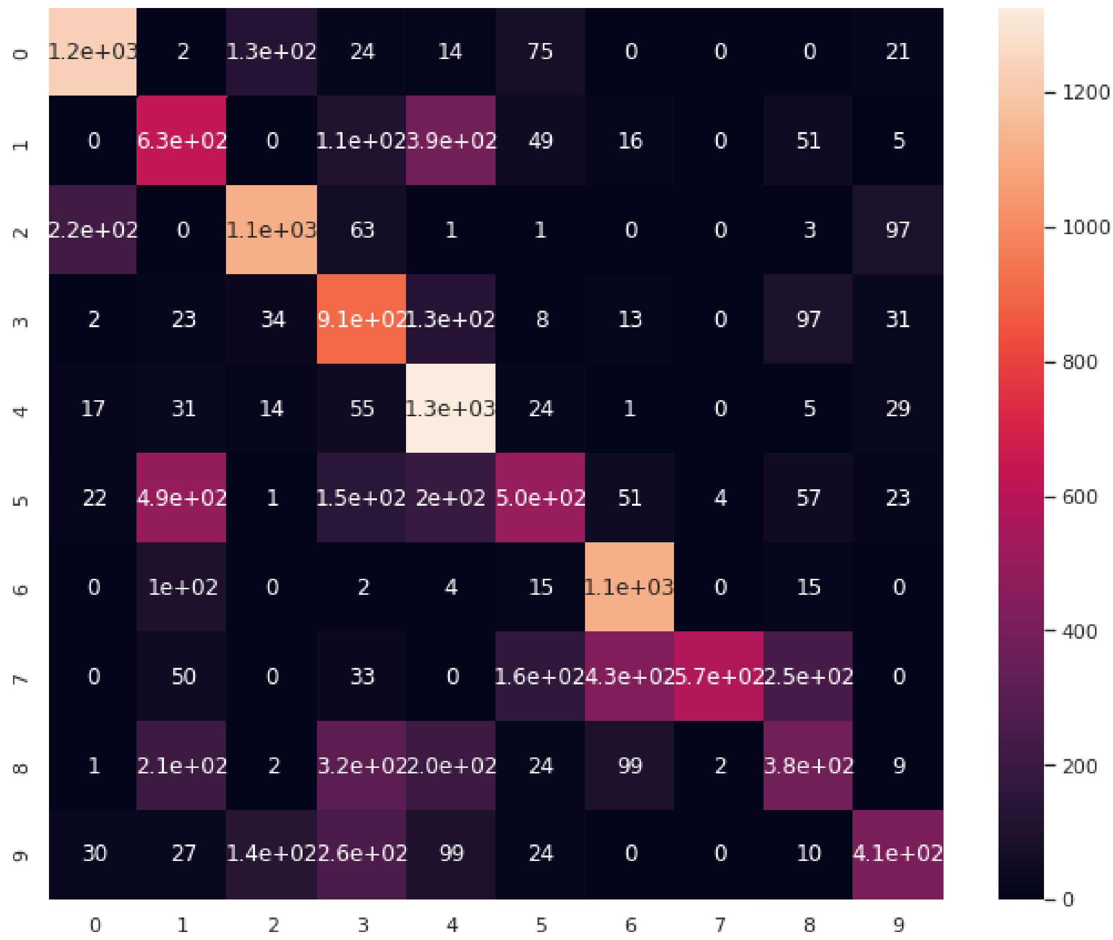
```
Test Score 1.0718187093734741
Test Accuracy 0.6063703894615173
```

[88]:
```python
# check labelling error of each class
results = model.predict(X_test)
results
```

[88]:
```
array([[6.55055919e-04, 1.55542046e-01, 1.96456071e-03, …,
        1.23027270e-03, 2.78194040e-01, 1.39225023e-02],
       [9.21449602e-01, 2.28366043e-04, 4.93983692e-03, …,
        1.26776595e-05, 1.60169060e-04, 5.88915199e-02],
       [2.89185409e-04, 8.51258263e-02, 7.17478979e-05, …,
        2.70051514e-05, 2.45492198e-02, 5.12546906e-03],
       …,
       [9.99575794e-01, 2.77302098e-10, 4.03843704e-04, …,
        4.47994069e-08, 5.29660440e-07, 2.08472642e-07],
       [6.37317717e-04, 7.61800259e-02, 4.70703846e-04, …,
        3.29842063e-04, 1.71053335e-01, 5.88374818e-03],
       [7.53801199e-09, 2.58354675e-02, 3.52820728e-09, …,
        3.72325485e-05, 7.94167165e-03, 2.44253260e-06]], dtype=float32)
```

[89]:
```python
y_pred = np.argmax(results, axis = 1)
y_test1 = np.argmax(y_test, axis = 1)

array = confusion_matrix(y_test1, y_pred)

df_cm = pd.DataFrame(array, range(10), range(10))
plt.figure(figsize = (11, 9))
sns.set(font_scale=1)
sns.heatmap(df_cm, annot = True, annot_kws = {"size": 12})
```

[89]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff14cc571d0>
```



[18]:

```python
# VGG16 Model

from keras.models import Model

from keras.applications.vgg16 import VGG16
from keras.applications.inception_v3 import InceptionV3, preprocess_input
```

[19]:

```python
# load the model

base_model = VGG16(weights='imagenet', include_top=False)
```

[20]:

```python
base_model.summary()
```

```
Model: "vgg16"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, None, None, 3)]   0
```

```
 block1_conv1 (Conv2D)       (None, None, None, 64)    1792

 block1_conv2 (Conv2D)       (None, None, None, 64)    36928

 block1_pool (MaxPooling2D)  (None, None, None, 64)    0

 block2_conv1 (Conv2D)       (None, None, None, 128)   73856

 block2_conv2 (Conv2D)       (None, None, None, 128)   147584

 block2_pool (MaxPooling2D)  (None, None, None, 128)   0

 block3_conv1 (Conv2D)       (None, None, None, 256)   295168

 block3_conv2 (Conv2D)       (None, None, None, 256)   590080

 block3_conv3 (Conv2D)       (None, None, None, 256)   590080

 block3_pool (MaxPooling2D)  (None, None, None, 256)   0

 block4_conv1 (Conv2D)       (None, None, None, 512)   1180160

 block4_conv2 (Conv2D)       (None, None, None, 512)   2359808

 block4_conv3 (Conv2D)       (None, None, None, 512)   2359808

 block4_pool (MaxPooling2D)  (None, None, None, 512)   0

 block5_conv1 (Conv2D)       (None, None, None, 512)   2359808

 block5_conv2 (Conv2D)       (None, None, None, 512)   2359808

 block5_conv3 (Conv2D)       (None, None, None, 512)   2359808

 block5_pool (MaxPooling2D)  (None, None, None, 512)   0

=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

[21]:
```
# freeze the model weights

for layer in base_model.layers:
  layer.trainable = False
```

[22]:
```
from keras.layers.pooling import GlobalAveragePooling2D
# construct the model

x = base_model.output
x = GlobalAveragePooling2D(name='avg_pool')(x)
x = Dropout(0.4)(x)
predictions = Dense(num_classes, activation='softmax')(x)
model = Model(inputs = base_model.input, outputs=predictions)
```

[23]:
```
model.compile(
    optimizer=Adam(),
    loss = 'categorical_crossentropy',
    metrics = ['accuracy']
)
```

[24]:
```
history = model.fit(X_train, y_train,
                    batch_size = 500,
                    epochs =12,
                    verbose = 1,
                    validation_data=(X_test, y_test))
```

```
Epoch 1/12
27/27 [==============================] - 31s 662ms/step - loss: 15.3157 -
accuracy: 0.1825 - val_loss: 4.6844 - val_accuracy: 0.4284
Epoch 2/12
27/27 [==============================] - 11s 418ms/step - loss: 6.8345 -
accuracy: 0.4150 - val_loss: 2.2050 - val_accuracy: 0.6956
Epoch 3/12
27/27 [==============================] - 11s 428ms/step - loss: 3.9486 -
accuracy: 0.5840 - val_loss: 1.5720 - val_accuracy: 0.7695
Epoch 4/12
27/27 [==============================] - 16s 609ms/step - loss: 2.8467 -
accuracy: 0.6607 - val_loss: 1.2443 - val_accuracy: 0.8085
Epoch 5/12
27/27 [==============================] - 16s 608ms/step - loss: 2.2179 -
accuracy: 0.7080 - val_loss: 1.0524 - val_accuracy: 0.8284
Epoch 6/12
27/27 [==============================] - 11s 424ms/step - loss: 1.8573 -
accuracy: 0.7362 - val_loss: 0.9251 - val_accuracy: 0.8393
Epoch 7/12
27/27 [==============================] - 11s 425ms/step - loss: 1.5210 -
accuracy: 0.7625 - val_loss: 0.8439 - val_accuracy: 0.8464
Epoch 8/12
27/27 [==============================] - 11s 425ms/step - loss: 1.3257 -
accuracy: 0.7744 - val_loss: 0.7757 - val_accuracy: 0.8528
Epoch 9/12
27/27 [==============================] - 16s 603ms/step - loss: 1.2030 -
```

```
accuracy: 0.7867 - val_loss: 0.6975 - val_accuracy: 0.8602
Epoch 10/12
27/27 [==============================] - 11s 421ms/step - loss: 1.0427 -
accuracy: 0.7996 - val_loss: 0.6424 - val_accuracy: 0.8667
Epoch 11/12
27/27 [==============================] - 16s 603ms/step - loss: 0.9681 -
accuracy: 0.8084 - val_loss: 0.6102 - val_accuracy: 0.8683
Epoch 12/12
27/27 [==============================] - 16s 601ms/step - loss: 0.8616 -
accuracy: 0.8147 - val_loss: 0.5759 - val_accuracy: 0.8720
```

[25]:
```
# evaluate the model

score = model.evaluate(X_test, y_test, verbose = 0)
print("Test Score", score[0])
print("Test Accuracy", score[1])
```

```
Test Score 0.5758934020996094
Test Accuracy 0.871999979019165
```

[26]:
```
# check labelling error of each class
results = model.predict(X_test)
```

[27]:
```
results
```

[27]:
```
array([[2.6370645e-19, 5.2178822e-10, 2.6698347e-30, …, 1.4582748e-14,
        3.6179107e-01, 1.5132398e-14],
       [9.9975985e-01, 5.6427165e-08, 2.1226161e-04, …, 1.2545919e-06,
        5.3337832e-08, 1.3842428e-06],
       [4.2031645e-09, 8.8120095e-10, 8.9835396e-12, …, 6.8299614e-13,
        3.7798327e-05, 4.3011874e-05],
       …,
       [9.9230313e-01, 2.5486517e-05, 6.1942190e-03, …, 2.9960470e-04,
        4.4459281e-05, 1.3861556e-04],
       [9.6188679e-10, 8.4348336e-05, 3.5271685e-14, …, 2.9702662e-07,
        3.4508082e-01, 1.7298460e-06],
       [0.0000000e+00, 4.4533372e-06, 3.2079273e-34, …, 6.0303617e-15,
        3.8098444e-03, 2.6290694e-22]], dtype=float32)
```

[28]:
```
y_pred = np.argmax(results, axis = 1)
y_test1 = np.argmax(y_test, axis = 1)
```

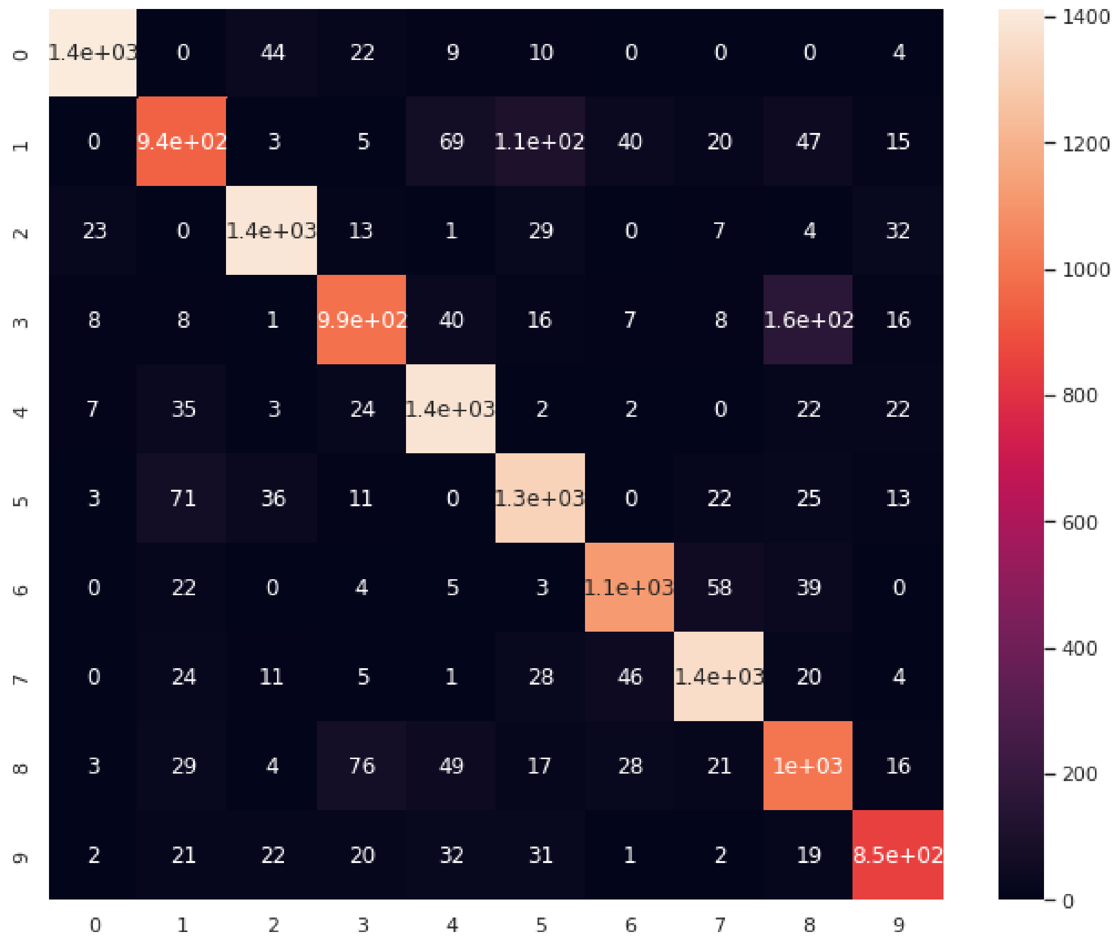
[29]:
```
confusion_matrix(y_test1, y_pred)
```

[29]:
```
array([[1411,    0,   44,   22,    9,   10,    0,    0,    0,    4],
       [   0,  944,    3,    5,   69,  107,   40,   20,   47,   15],
       [  23,    0, 1391,   13,    1,   29,    0,    7,    4,   32],
```

```
       [   8,    8,    1,  987,   40,   16,    7,    8,  159,   16],
       [   7,   35,    3,   24, 1383,    2,    2,    0,   22,   22],
       [   3,   71,   36,   11,    0, 1319,    0,   22,   25,   13],
       [   0,   22,    0,    4,    5,    3, 1119,   58,   39,    0],
       [   0,   24,   11,    5,    1,   28,   46, 1361,   20,    4],
       [   3,   29,    4,   76,   49,   17,   28,   21, 1007,   16],
       [   2,   21,   22,   20,   32,   31,    1,    2,   19,  850]])
```

[30]:
```python
array = confusion_matrix(y_test1, y_pred)

df_cm = pd.DataFrame(array, range(10), range(10))
plt.figure(figsize = (11, 9))
sns.set(font_scale=1)
sns.heatmap(df_cm, annot = True, annot_kws = {"size": 12})
```

[30]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff2e1e6a710>

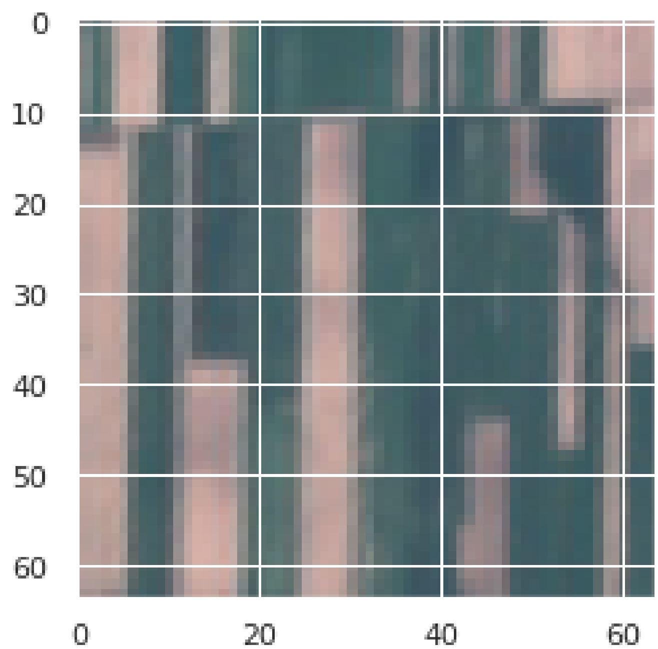


[31]:
```python
dictionary
```

[31]:
```
{'AnnualCrop': 4,
 'Forest': 2,
 'HerbaceousVegetation': 5,
 'Highway': 8,
 'Industrial': 6,
 'Pasture': 9,
 'PermanentCrop': 1,
 'Residential': 7,
 'River': 3,
 'SeaLake': 0}
```

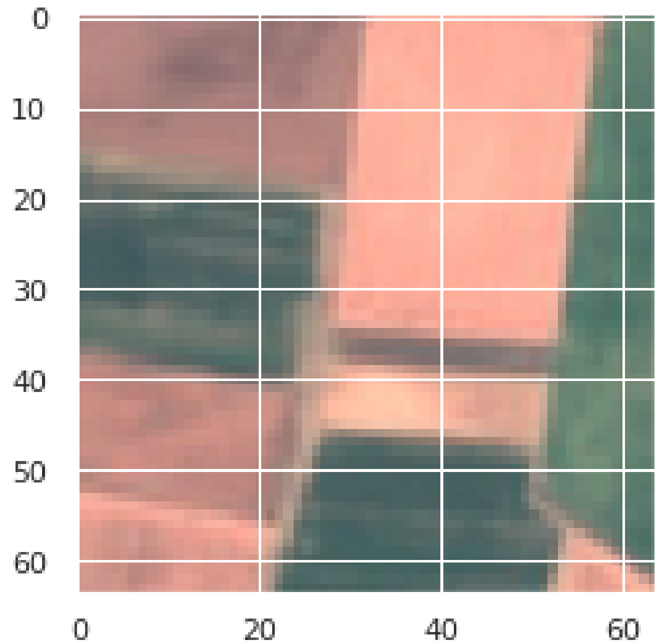
From the confusion matrix we find that:- - Herbaceous vegetation is prone to be misclassified as Permanent crop - River is prone to be misclassified as Highway :

[45]:
```
# Permanent crop successfully identified

trial1 = np.where((y_test1 == 1) & (y_pred == 1))[0][943]
tmpimg = imread(imgfiles[trial1])
plt.imshow(tmpimg)
plt.show()
```

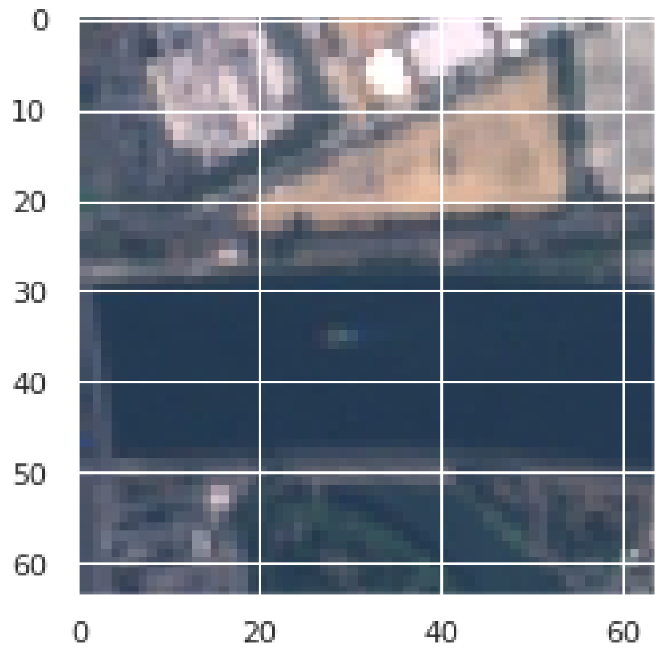


[43]:
```
# Herbaceous vegetation successfully identified

trial1 = np.where((y_test1 == 2) & (y_pred == 2))[0][1390]
tmpimg = imread(imgfiles[trial1])
plt.imshow(tmpimg)
```

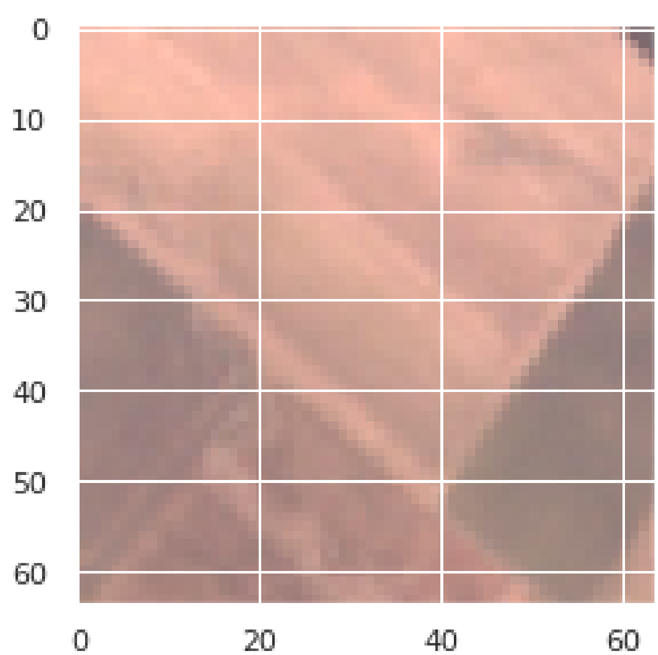
```
plt.show()
```



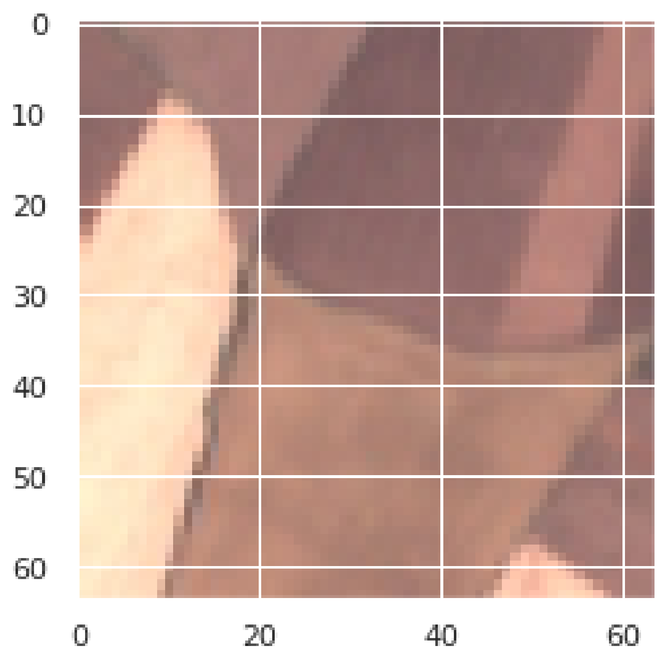
[51]:
```
# Industrial successfully identified

trial1 = np.where((y_test1 == 6) & (y_pred == 6))[0][800]
tmpimg = imread(imgfiles[trial1])
plt.imshow(tmpimg)
plt.show()
```



[55]:
```
# Herbaceous Vegetation being misclassified as Permanent crop

trial1 = np.where((y_test1 == 5) & (y_pred == 1))[0][66]
tmpimg = imread(imgfiles[trial1])
plt.imshow(tmpimg)
plt.show()
```



[62]:
```python
# River being misclassified as Highway

trial1 = np.where((y_test1 == 3) & (y_pred == 8))[0][158]
tmpimg = imread(imgfiles[trial1])
plt.imshow(tmpimg)
plt.show()
```



1.12 MobileNetV2

[ ]:
```

```

[ ]:
```

```