jupyter hw5 最后检查: 3 分钟前 (自动保存)

Logout

File Edit View Insert Cell Kernel Widgets Help 不可信 Python 3

HW5 - EuroSAT Land Use and Land Cover Classification using Deep Learning

In this homework your task is to implement deep learning models to solve a typical problem in satellite imaging using a benchmark dataset. The homework was designed to make you work on increasingly more complex models. We hope that the homework will be very helpful to improve your skills and knowledge in deep learning!

S1:

- Visit the EuroSAT data description page and download the data: https://github.com/phelber/eurosat
- Split the data into training (50%) and testing sets (50%), stratified on class labels (equal percentage of each class type in train and test sets).

In [9]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image, ImageFilter, ImageOps
import sklearn
import glob
import imageio
from skimage.color import rgb2gray
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import RMSprop
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.layers import Dense, Flatten, Activation
from keras.optimizers import SGD

import tensorflow as tf
from tensorflow.keras import Model, layers
from sklearn.model_selection import StratifiedKFold
```

In [10]:
```
# read the pictures in the EuroSAT
annualcrop = glob.glob('D:/MUSA_Fall/RS/EuroSAT/2750/AnnualCrop/*.jpg')
forest = glob.glob('D:/MUSA_Fall/RS/EuroSAT/2750/Forest/*.jpg')
herbaceous = glob.glob('D:/MUSA_Fall/RS/EuroSAT/2750/HerbaceousVegetation/*.jpg')
highway = glob.glob('D:/MUSA_Fall/RS/EuroSAT/2750/Highway/*.jpg')
industrial = glob.glob('D:/MUSA_Fall/RS/EuroSAT/2750/Industrial/*.jpg')
pasture = glob.glob('D:/MUSA_Fall/RS/EuroSAT/2750/Pasture/*.jpg')
permanentcrop = glob.glob('D:/MUSA_Fall/RS/EuroSAT/2750/PermanentCrop/*.jpg')
residential = glob.glob('D:/MUSA_Fall/RS/EuroSAT/2750/Residential/*.jpg')
river = glob.glob('D:/MUSA_Fall/RS/EuroSAT/2750/River/*.jpg')
sealake = glob.glob('D:/MUSA_Fall/RS/EuroSAT/2750/Sealake/*.jpg')
```

In [11]:
```
#combine them together
all_image = np.concatenate((annualcrop,forest,herbaceous,highway,industrial,
                            pasture,permanentcrop,residential,river,sealake))
```

In [62]:
```
# read the first image
tmpRGB = np.asarray(Image.open(annualcrop[0]))
# get the image size
imgSize = np.array(tmpRGB.shape[0:2])
imgSize
```

Out[62]:
```
array([64, 64])
```

In [19]:
```
#create a zero array
dMat = np.zeros([len(all_image), np.prod(imgSize)]).astype(np.uint8)
```

In [20]:
```
for i in range(len(all_image)):
    tmpRGB = Image.open(all_image[i])
    tmpGray = np.asarray(ImageOps.grayscale(tmpRGB)).astype(np.uint8).flatten()
    dMat[i,:] = tmpGray
```

In [21]:
```
dMat.shape
```

Out[21]:
```
(27000, 4096)
```

In [28]:
```
label = []

for i in range(len(annualcrop)):
    classtype = 0
    label.append(classtype)

for i in range(len(forest)):
    classtype = 1
    label.append(classtype)

for i in range(len(herbaceous)):
    classtype = 2
    label.append(classtype)

for i in range(len(highway)):
    classtype = 3
    label.append(classtype)

for i in range(len(industrial)):
    classtype = 4
    label.append(classtype)

for i in range(len(pasture)):
    classtype = 5
    label.append(classtype)

for i in range(len(permanentcrop)):
    classtype = 6
    label.append(classtype)

for i in range(len(residential)):
    classtype = 7
    label.append(classtype)

for i in range(len(river)):
    classtype = 8
    label.append(classtype)

for i in range(len(sealake)):
    classtype = 9
    label.append(classtype)
```

In [51]:
```
X_train, X_test, y_train, y_test = train_test_split(dMat, label, stratify = label,test_size=.5, random_state=42)
```

In [52]:
```
X_train.shape
```

Out[52]:
```
(13500, 4096)
```

S2:

- Convert each RBG image to grayscale and flatten the images into a data matrix (n x p: n = #samples, p = #pixels in each image)
- Implement a first deep learning model (M.1) using a fully connected network with a single fully connected layer (i.e: input layer + fully connected layer as the output layer).
- Q1: Calculate classification accuracy on the test data.
  classification accuracy on the test data is 0.1113

In [53]:
```
# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)

y_train.shape
```

Out[53]:
```
(13500, 10)
```

In [54]:
```
model_1 = Sequential()
```

```
model_1.add(Dense(1024, activation='relu', input_shape=(4096,)))
model_1.add(Dense(10, activation='softmax'))

model_1.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_14 (Dense)             (None, 1024)              4195328
_________________________________________________________________
dense_15 (Dense)             (None, 10)                10250
=================================================================
Total params: 4,205,578
Trainable params: 4,205,578
Non-trainable params: 0
_________________________________________________________________
```

In [55]:

```
# compile the model
model_1.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])
```

In [56]:

```
batch_size = 128
epochs = 10

fit = model_1.fit(X_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(X_test, y_test))
```

```
Train on 13500 samples, validate on 13500 samples
Epoch 1/10
13500/13500 [==============================] - 7s 520us/step - loss: 977.7904 - accuracy: 0.1031 - val_loss: 604.1355 - val_accuracy:
0.0926
Epoch 2/10
13500/13500 [==============================] - 7s 517us/step - loss: 186.0090 - accuracy: 0.1072 - val_loss: 128.8050 - val_accuracy:
0.0926
Epoch 3/10
13500/13500 [==============================] - 7s 498us/step - loss: 81.0833 - accuracy: 0.1030 - val_loss: 35.9198 - val_accuracy: 0.1
279
Epoch 4/10
13500/13500 [==============================] - 7s 500us/step - loss: 7.1827 - accuracy: 0.1121 - val_loss: 2.3009 - val_accuracy: 0.111
6
Epoch 5/10
13500/13500 [==============================] - 7s 504us/step - loss: 2.2992 - accuracy: 0.1111 - val_loss: 2.2978 - val_accuracy: 0.111
6
Epoch 6/10
13500/13500 [==============================] - 7s 518us/step - loss: 2.2971 - accuracy: 0.1111 - val_loss: 2.2963 - val_accuracy: 0.111
6
Epoch 7/10
13500/13500 [==============================] - 7s 523us/step - loss: 2.2960 - accuracy: 0.1111 - val_loss: 2.2955 - val_accuracy: 0.111
6
Epoch 8/10
13500/13500 [==============================] - 7s 485us/step - loss: 2.2954 - accuracy: 0.1096 - val_loss: 2.2951 - val_accuracy: 0.111
6
Epoch 9/10
13500/13500 [==============================] - 7s 500us/step - loss: 2.2952 - accuracy: 0.1084 - val_loss: 2.2949 - val_accuracy: 0.111
6
Epoch 10/10
13500/13500 [==============================] - 7s 489us/step - loss: 2.2950 - accuracy: 0.1090 - val_loss: 2.2948 - val_accuracy: 0.111
6
```

In [ ]:

In [ ]:

S3:

- Implement a second deep learning model (M.2) adding an additional fully connected hidden layer (with an arbitrary number of nodes) to the previous model.
- Q1: Calculate classification accuracy on the test data.

  The classification accuracy on the test data is 0.1116

In [40]:
```
model_2 = Sequential()
model_2.add(Dense(1024, activation='relu', input_shape=(4096,)))
model_2.add(Dense(1024, activation='relu'))
model_2.add(Dense(10, activation='softmax'))

model_2.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 1024)              4195328
_________________________________________________________________
dense_6 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dense_7 (Dense)              (None, 10)                10250
=================================================================
Total params: 5,255,178
Trainable params: 5,255,178
Non-trainable params: 0
_________________________________________________________________
```

In [41]:
```
#compile the model
model_2.compile(loss='categorical_crossentropy',
                optimizer = RMSprop(),
                metrics = ['accuracy'])
```

In [42]:
```
fit = model_2.fit(X_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(X_test, y_test))
```

```
Train on 13500 samples, validate on 13500 samples
Epoch 1/10
13500/13500 [==============================] - 9s 650us/step - loss: 346.3634 - accuracy: 0.1078 - val_loss: 2.3004 - val_accuracy: 0.1
111
Epoch 2/10
13500/13500 [==============================] - 8s 624us/step - loss: 4.9072 - accuracy: 0.1108 - val_loss: 2.2980 - val_accuracy: 0.111
0
Epoch 3/10
13500/13500 [==============================] - 9s 640us/step - loss: 2.2955 - accuracy: 0.1084 - val_loss: 2.2973 - val_accuracy: 0.110
9
Epoch 4/10
13500/13500 [==============================] - 9s 643us/step - loss: 2.2955 - accuracy: 0.1098 - val_loss: 2.2975 - val_accuracy: 0.111
1
Epoch 5/10
13500/13500 [==============================] - 8s 622us/step - loss: 2.2957 - accuracy: 0.1101 - val_loss: 2.2978 - val_accuracy: 0.111
0
Epoch 6/10
13500/13500 [==============================] - 9s 643us/step - loss: 2.2956 - accuracy: 0.1073 - val_loss: 2.2976 - val_accuracy: 0.111
3
Epoch 7/10
13500/13500 [==============================] - 8s 616us/step - loss: 2.2955 - accuracy: 0.1016 - val_loss: 2.2976 - val_accuracy: 0.111
3
Epoch 8/10
13500/13500 [==============================] - 9s 642us/step - loss: 2.2954 - accuracy: 0.1080 - val_loss: 2.2976 - val_accuracy: 0.111
3
Epoch 9/10
13500/13500 [==============================] - 8s 614us/step - loss: 2.2953 - accuracy: 0.1120 - val_loss: 2.2984 - val_accuracy: 0.111
1
Epoch 10/10
13500/13500 [==============================] - 9s 643us/step - loss: 2.2956 - accuracy: 0.1076 - val_loss: 2.2979 - val_accuracy: 0.111
3
```

In [ ]:

In [ ]:

In [0]:

S4:

- Implement a third deep learning model (M.3) adding two additional fully connected hidden layers (with arbitrary number of nodes) as well as drop-out layers to the previous model.

- Q1: Calculate classification accuracy on the test data.

  The classification accuracy on the test data is 0.1115.
- Q2: Compare against previous models. Which model was the "best"? Why?

  I think Model3 suppose to be the best because Model3 has dropout function which can avoid the issue about overfitting.

In [58]:
```
model_3 = Sequential()
model_3.add(Dense(1024, activation='relu', input_shape=(4096,)))
model_3.add(Dense(1024, activation='relu'))
model_3.add(Dense(1024, activation='relu'))
model_3.add(Dropout(0.2))
model_3.add(Dense(10, activation='softmax'))

model_3.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_19 (Dense)             (None, 1024)              4195328
_________________________________________________________________
dense_20 (Dense)             (None, 1024)              1049600
_________________________________________________________________
dense_21 (Dense)             (None, 1024)              1049600
_________________________________________________________________
dropout_2 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_22 (Dense)             (None, 10)                10250
=================================================================
Total params: 6,304,778
Trainable params: 6,304,778
Non-trainable params: 0
_________________________________________________________________
```

In [59]:
```
model_3.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])
```

In [60]:
```
fit = model_3.fit(X_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(X_test, y_test))
```

```
Train on 13500 samples, validate on 13500 samples
Epoch 1/10
13500/13500 [==============================] - 11s 808us/step - loss: 223.6087 - accuracy: 0.1070 - val_loss: 2.3133 - val_accuracy: 0.
1111
Epoch 2/10
13500/13500 [==============================] - 10s 755us/step - loss: 6.0374 - accuracy: 0.1087 - val_loss: 2.2968 - val_accuracy: 0.11
16
Epoch 3/10
13500/13500 [==============================] - 11s 782us/step - loss: 2.2956 - accuracy: 0.1085 - val_loss: 2.2957 - val_accuracy: 0.11
24
Epoch 4/10
13500/13500 [==============================] - 11s 788us/step - loss: 2.2954 - accuracy: 0.1121 - val_loss: 2.2955 - val_accuracy: 0.11
25
Epoch 5/10
13500/13500 [==============================] - 10s 745us/step - loss: 2.2956 - accuracy: 0.1066 - val_loss: 2.2964 - val_accuracy: 0.11
15
Epoch 6/10
13500/13500 [==============================] - 11s 782us/step - loss: 2.2956 - accuracy: 0.1097 - val_loss: 2.2959 - val_accuracy: 0.11
20
Epoch 7/10
13500/13500 [==============================] - 10s 770us/step - loss: 2.2955 - accuracy: 0.1047 - val_loss: 2.2960 - val_accuracy: 0.11
17
Epoch 8/10
13500/13500 [==============================] - 10s 750us/step - loss: 2.2952 - accuracy: 0.1088 - val_loss: 2.2960 - val_accuracy: 0.11
18
Epoch 9/10
13500/13500 [==============================] - 11s 794us/step - loss: 2.2954 - accuracy: 0.1061 - val_loss: 2.2964 - val_accuracy: 0.11
16
Epoch 10/10
13500/13500 [==============================] - 10s 755us/step - loss: 2.2953 - accuracy: 0.1085 - val_loss: 2.2964 - val_accuracy: 0.11
15
```

In [ ]:

In [0]:

S5:

- Using RGB images (without vectorizing them), implement a fourth CNN model (M.4) that includes the following layers: Conv2D, MaxPooling2D, Dropout, Flatten, Dense.
- Q1: Calculate classification accuracy on the test data.

The classification accuracy on the test data is 0.77.

- Q2: Compare against previous models. Which model was the "best"? Why?

The 4th model performs best. This model uses all three bands and uses CNN which makes the model more complicated.

In [76]:
```
# read the first image
tmpRGB = np.asarray(Image.open(annualcrop[0]))

imgSize = np.array(tmpRGB.shape)
imgSize
```
Out[76]:
```
array([64, 64,  3])
```

In [77]:
```
#create a zero array
dMat2 = np.zeros([len(all_image), imgSize[0],imgSize[1],imgSize[2]]).astype(np.uint8)
```

In [78]:
```
for i in range(len(all_image)):
    tmpRGB = np.asarray(Image.open(all_image[i]))
    dMat2[i,:,:,:] = tmpRGB
```

In [103]:
```
X_train, X_test, y_train, y_test = train_test_split(dMat2, label, stratify = label,test_size=.5, random_state=42)
```

In [104]:
```
X_train.shape
```
Out[104]:
```
(13500, 64, 64, 3)
```

In [105]:
```
X_train = X_train/255
X_test = X_test/255
```

In [106]:
```
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)

y_train.shape
```
Out[106]:
```
(13500, 10)
```

In [107]:
```
# network design
input_shape = (64,64,3)
model_4 = Sequential()
model_4.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model_4.add(Conv2D(64, (3, 3), activation='relu'))
model_4.add(MaxPooling2D(pool_size=(2, 2)))
model_4.add(Dropout(0.25))
model_4.add(Flatten())
model_4.add(Dense(128, activation='relu'))
model_4.add(Dropout(0.5))
model_4.add(Dense(10, activation='softmax'))

model_4.summary()
```

```
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 62, 62, 32)        896
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 60, 60, 64)        18496
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 30, 30, 64)        0
_________________________________________________________________
dropout_11 (Dropout)         (None, 30, 30, 64)        0
```

```
_________________________________________________________________
flatten_5 (Flatten)          (None, 57600)             0
_________________________________________________________________
dense_30 (Dense)             (None, 128)               7372928
_________________________________________________________________
dropout_12 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_31 (Dense)             (None, 10)                1290
=================================================================
Total params: 7,393,610
Trainable params: 7,393,610
Non-trainable params: 0
_________________________________________________________________
```

In [108]:

```
sgd = SGD(lr=0.01, momentum=0.9, nesterov=True)
model_4.compile(
    loss='categorical_crossentropy',
    optimizer=sgd,
    metrics=['accuracy'])
```

In [109]:

```
model_4.fit(X_train, y_train,
              batch_size=128,
              epochs=10,
              verbose=1,
              validation_data=(X_test, y_test))
```

```
Train on 13500 samples, validate on 13500 samples
Epoch 1/10
13500/13500 [==============================] - 107s 8ms/step - loss: 1.9513 - accuracy: 0.2650 - val_loss: 1.8664 - val_accuracy: 0.291
0
Epoch 2/10
13500/13500 [==============================] - 106s 8ms/step - loss: 1.5401 - accuracy: 0.4440 - val_loss: 1.2798 - val_accuracy: 0.554
1
Epoch 3/10
13500/13500 [==============================] - 107s 8ms/step - loss: 1.2052 - accuracy: 0.5719 - val_loss: 0.9684 - val_accuracy: 0.669
2
Epoch 4/10
13500/13500 [==============================] - 106s 8ms/step - loss: 1.0032 - accuracy: 0.6410 - val_loss: 0.8083 - val_accuracy: 0.719
0
Epoch 5/10
13500/13500 [==============================] - 106s 8ms/step - loss: 0.9144 - accuracy: 0.6780 - val_loss: 1.3581 - val_accuracy: 0.537
9
Epoch 6/10
13500/13500 [==============================] - 108s 8ms/step - loss: 0.8387 - accuracy: 0.6974 - val_loss: 0.7757 - val_accuracy: 0.711
4
Epoch 7/10
13500/13500 [==============================] - 107s 8ms/step - loss: 0.7822 - accuracy: 0.7219 - val_loss: 0.8611 - val_accuracy: 0.716
8
Epoch 8/10
13500/13500 [==============================] - 106s 8ms/step - loss: 0.6757 - accuracy: 0.7611 - val_loss: 0.6834 - val_accuracy: 0.760
8
Epoch 9/10
13500/13500 [==============================] - 106s 8ms/step - loss: 0.6295 - accuracy: 0.7805 - val_loss: 0.6368 - val_accuracy: 0.782
5
Epoch 10/10
13500/13500 [==============================] - 108s 8ms/step - loss: 0.5594 - accuracy: 0.8016 - val_loss: 0.6479 - val_accuracy: 0.770
7
```

Out[109]:

```
<keras.callbacks.callbacks.History at 0x2a51123aac8>
```

In [ ]:

In [ ]:

S6:

- Using RGB images (without vectorizing them), implement a fifth deep learning model (M.5) targeting accuracy that will outperform all previous models. You are free to use any tools and techniques, as well as pre-trained models for transfer learning.
- Q1: Describe the model you built, and why you chose it.

I add CNN layer and maxpooling layer to the 5th model and two dropout layers, two dense layers which can keep the essence information of the dataset.

- Q2: Calculate classification accuracy on the test data.

The classification accuracy on the test data is 0.6134.

- Q3: Compare against previous models. Which model was the "best"? Why?

The 4th is the best for it has the highest accuraccy which is 0.77.

- Q4: What are the two classes with the highest labeling error? Explain using data and showing mis-classified examples.

The two classes with the highest labeling error are Annual Crop with 0.89 error rate and Highway with 0.61 error rate.

In [112]:
```
# build the model
model_5 = Sequential()

model_5.add(Conv2D(64, (10, 10), padding='same', activation='relu'))
model_5.add(MaxPooling2D(pool_size=(2, 2)))
model_5.add(Dropout(0.25))

# Adding new Layers
model_5.add(Flatten())
model_5.add(Dense(1024, activation='relu'))
model_5.add(Dropout(0.5))
model_5.add(Dense(10, activation='softmax'))
```

In [118]:
```
# Compile the model
from keras import optimizers

model_5.compile(loss='categorical_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
```

In [120]:
```
model_5.fit(X_train, y_train,
              batch_size=128,
              epochs=10,
              verbose=1,
              validation_data=(X_test, y_test))
```

```
Train on 13500 samples, validate on 13500 samples
Epoch 1/10
13500/13500 [==============================] - 308s 23ms/step - loss: 1.9353 - acc: 0.2920 - val_loss: 1.5507 - val_acc: 0.4507
Epoch 2/10
13500/13500 [==============================] - 323s 24ms/step - loss: 1.5449 - acc: 0.4320 - val_loss: 1.4501 - val_acc: 0.4255
Epoch 3/10
13500/13500 [==============================] - 330s 24ms/step - loss: 1.4099 - acc: 0.4923 - val_loss: 1.4194 - val_acc: 0.5037
Epoch 4/10
13500/13500 [==============================] - 416s 31ms/step - loss: 1.3382 - acc: 0.5252 - val_loss: 1.2946 - val_acc: 0.5267
Epoch 5/10
13500/13500 [==============================] - 431s 32ms/step - loss: 1.2690 - acc: 0.5456 - val_loss: 1.3750 - val_acc: 0.5117
Epoch 6/10
13500/13500 [==============================] - 436s 32ms/step - loss: 1.2200 - acc: 0.5662 - val_loss: 1.2181 - val_acc: 0.5650
Epoch 7/10
13500/13500 [==============================] - 432s 32ms/step - loss: 1.1642 - acc: 0.5881 - val_loss: 1.1837 - val_acc: 0.5547
Epoch 8/10
13500/13500 [==============================] - 450s 33ms/step - loss: 1.1396 - acc: 0.5995 - val_loss: 1.5630 - val_acc: 0.4372
Epoch 9/10
13500/13500 [==============================] - 450s 33ms/step - loss: 1.0937 - acc: 0.6141 - val_loss: 1.3794 - val_acc: 0.4739
Epoch 10/10
13500/13500 [==============================] - 452s 33ms/step - loss: 1.0550 - acc: 0.6239 - val_loss: 1.0782 - val_acc: 0.6134
```

Out[120]: <keras.callbacks.callbacks.History at 0x2a511370860>

In [121]:
```
# predict label
result_label_test = model_5.predict_classes(X_test)
```

In [122]:
```
pre_label = keras.utils.to_categorical(result_label_test, 10)
```

In [145]:
```
error = []

for i in range(pre_label.shape[1]):
    error.append(1-np.sum((pre_label[:,i]==y_test[:,i])&(pre_label[:,i]==1))/np.sum(y_test[:,i]==1))
```

In [172]:
```
error_df = pd.DataFrame(error)
```

In [157]:
```
labels = ['Annual_crop','Forest','HerbaceousVegetation','Highway','Industrial','Pasture',
         'PermanentCrop','Residential','River','Sealake']
```

In [173]:
```
error_df['labels'] = labels
error_df = error_df.sort_values(error_df.columns[0],ascending=False)
error_df
```

Out[173]:

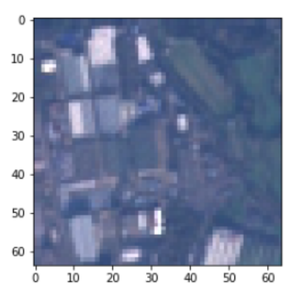
	0	labels
0	0.895333	Annual_crop
3	0.611200	Highway
2	0.510000	HerbaceousVegetation
8	0.416800	River
5	0.328000	Pasture
9	0.256000	Sealake
1	0.246667	Forest
6	0.216000	PermanentCrop
4	0.198400	Industrial
7	0.150667	Residential

In [176]:
```
#find the misclassified Industrial images
np.where((pre_label[:,4]!=y_test[:,4])&(y_test[:,4]==1))
```

Out[176]:
```
(array([   16,    61,    68,   197,   214,   387,   412,   413,   428,
          471,   612,   668,   720,   760,   816,   893,   910,   945,
         1031,  1047,  1077,  1226,  1247,  1263,  1316,  1334,  1362,
         1420,  1422,  1435,  1443,  1469,  1512,  1526,  1563,  1630,
         1647,  1719,  1765,  1864,  1868,  2049,  2074,  2087,  2093,
         2401,  2402,  2497,  2625,  2633,  2862,  3028,  3061,  3153,
         3559,  3564,  3581,  3624,  3727,  3744,  3833,  3835,  3895,
         4016,  4033,  4073,  4181,  4205,  4272,  4301,  4306,  4364,
         4434,  4449,  4451,  4584,  4627,  4641,  4673,  4685,  4716,
         4850,  5083,  5090,  5101,  5107,  5110,  5202,  5382,  5396,
         5442,  5574,  5605,  5684,  5696,  5750,  5788,  5835,  5843,
         5923,  6068,  6122,  6147,  6209,  6219,  6236,  6240,  6270,
         6311,  6312,  6431,  6442,  6461,  6462,  6499,  6512,  6515,
         6574,  6575,  6661,  6663,  6728,  6854,  6900,  6943,  7098,
         7120,  7125,  7158,  7413,  7464,  7477,  7497,  7541,  7551,
         7581,  7596,  7674,  7709,  7715,  7748,  8087,  8096,  8157,
         8253,  8270,  8286,  8332,  8380,  8479,  8643,  8739,  8811,
         8897,  9060,  9114,  9117,  9165,  9221,  9236,  9262,  9285,
         9286,  9336,  9395,  9403,  9459,  9567,  9605,  9653,  9697,
         9728,  9789,  9792,  9847,  9851,  9855,  9871,  9883,  9900,
         9957,  9967, 10054, 10057, 10072, 10108, 10162, 10195, 10222,
        10290, 10537, 10545, 10567, 10592, 10601, 10725, 10889, 10951,
        11171, 11264, 11309, 11357, 11510, 11560, 11645, 11653, 11690,
        11704, 11751, 11799, 11952, 12006, 12024, 12049, 12111, 12172,
        12204, 12277, 12320, 12334, 12357, 12386, 12465, 12494, 12545,
        12546, 12576, 12631, 12666, 12684, 12806, 12812, 12829, 12907,
        12919, 12939, 12955, 13059, 13073, 13078, 13113, 13161, 13279,
        13290, 13313, 13447, 13468, 13496], dtype=int64),)
```

In [177]:
```
# show one of these misclassified image
plt.imshow(X_test[16,:,:])
```

Out[177]:
```
<matplotlib.image.AxesImage at 0x2a511c59240>
```



In [178]:
```
# This Industrial image was misclassified as Residential
pre_label[16,:]
```

Out[178]:
```
array([0., 0., 0., 0., 0., 0., 0., 1., 0., 0.], dtype=float32)
```

```
In [179]: #find the misclassified Highway images
np.where((pre_label[:,3]!=y_test[:,3])&(y_test[:,3]==1))
```

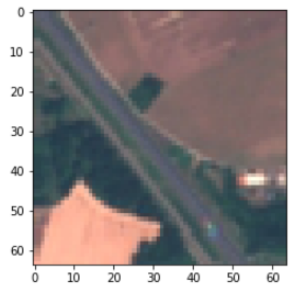
```
Out[179]: (array([   11,    27,    55,    74,   126,   172,   190,   217,   226,
                245,   270,   301,   306,   312,   326,   330,   377,   384,
                399,   410,   422,   431,   446,   462,   474,   506,   510,
                513,   537,   566,   568,   580,   615,   629,   674,   682,
                698,   700,   726,   739,   750,   756,   764,   765,   780,
                806,   812,   814,   825,   849,   919,   923,   924,   930,
                932,   954,   989,  1045,  1059,  1118,  1147,  1151,  1158,
               1165,  1168,  1180,  1231,  1251,  1308,  1339,  1341,  1344,
               1360,  1361,  1383,  1423,  1432,  1444,  1456,  1463,  1473,
               1490,  1494,  1511,  1516,  1525,  1586,  1591,  1592,  1616,
               1619,  1626,  1642,  1643,  1649,  1681,  1707,  1711,  1712,
               1745,  1751,  1754,  1763,  1764,  1780,  1803,  1835,  1838,
               1847,  1865,  1877,  1894,  1925,  1945,  1966,  1977,  1993,
               2024,  2043,  2051,  2067,  2070,  2144,  2165,  2203,  2207,
               2208,  2223,  2228,  2263,  2306,  2307,  2322,  2334,  2335,
               2348,  2371,  2372,  2379,  2385,  2403,  2407,  2408,  2415,
               2453,  2474,  2505,  2532,  2537,  2548,  2556,  2560,  2569,
               2593,  2605,  2617,  2636,  2707,  2717,  2731,  2734,  2753,
               2763,  2772,  2845,  2849,  2867,  2911,  2935,  2952,  2999,
               3001,  3023,  3059,  3064,  3104,  3105,  3156,  3172,  3184,
               3237,  3259,  3266,  3268,  3292,  3295,  3302,  3345,  3350,
               3364,  3374,  3430,  3445,  3451,  3452,  3454,  3460,  3462,
               3468,  3470,  3472,  3475,  3484,  3518,  3522,  3532,  3570,
               3595,  3604,  3618,  3636,  3638,  3650,  3659,  3666,  3683,
               3705,  3728,  3729,  3733,  3758,  3759,  3768,  3769,  3773,
               3790,  3803,  3829,  3836,  3882,  3884,  3901,  3907,  3916,
               3917,  3924,  3951,  3963,  3967,  3968,  3973,  3978,  3981,
               3990,  3998,  4019,  4021,  4032,  4035,  4044,  4056,  4063,
               4067,  4140,  4176,  4180,  4208,  4217,  4220,  4238,  4250,
               4275,  4285,  4309,  4333,  4343,  4347,  4358,  4360,  4393,
               4410,  4430,  4444,  4473,  4492,  4522,  4621,  4649,  4728,
               4737,  4745,  4793,  4812,  4818,  4829,  4842,  4852,  4867,
               4869,  4885,  4896,  4903,  4904,  4940,  4942,  4990,  4994,
               5001,  5003,  5032,  5054,  5092,  5116,  5117,  5120,  5132,
               5158,  5174,  5189,  5240,  5297,  5341,  5344,  5365,  5419,
               5429,  5451,  5472,  5494,  5550,  5564,  5576,  5581,  5613,
               5623,  5624,  5632,  5657,  5675,  5679,  5717,  5728,  5742,
               5764,  5787,  5791,  5857,  5860,  5866,  5867,  5876,  5882,
               5883,  5895,  5897,  5909,  5916,  5950,  5954,  5957,  5986,
               6003,  6005,  6123,  6139,  6149,  6153,  6162,  6178,  6186,
               6217,  6229,  6264,  6306,  6310,  6332,  6352,  6399,  6408,
               6464,  6486,  6526,  6541,  6615,  6627,  6631,  6644,  6666,
               6684,  6689,  6699,  6709,  6732,  6746,  6784,  6802,  6805,
               6905,  6926,  6930,  6931,  6968,  6975,  6996,  7016,  7019,
               7022,  7039,  7051,  7058,  7077,  7118,  7119,  7132,  7133,
               7143,  7144,  7147,  7155,  7156,  7162,  7178,  7197,  7217,
               7254,  7269,  7277,  7283,  7341,  7361,  7379,  7380,  7391,
               7400,  7414,  7444,  7456,  7495,  7508,  7513,  7519,  7544,
               7546,  7553,  7554,  7556,  7575,  7582,  7602,  7607,  7610,
               7611,  7623,  7624,  7648,  7662,  7665,  7675,  7680,  7683,
               7690,  7705,  7728,  7737,  7751,  7815,  7823,  7826,  7841,
               7848,  7915,  7962,  7993,  7995,  7997,  8009,  8026,  8030,
               8041,  8060,  8107,  8130,  8156,  8188,  8196,  8248,  8281,
               8284,  8298,  8302,  8323,  8333,  8346,  8364,  8391,  8396,
               8405,  8408,  8424,  8427,  8438,  8457,  8467,  8486,  8498,
               8503,  8514,  8529,  8537,  8551,  8572,  8581,  8583,  8587,
               8589,  8601,  8604,  8616,  8623,  8635,  8689,  8705,  8708,
               8720,  8729,  8730,  8770,  8803,  8813,  8819,  8862,  8865,
               8880,  8947,  8949,  8956,  8964,  8980,  8985,  9053,  9064,
               9065,  9073,  9077,  9082,  9087,  9215,  9243,  9250,  9268,
               9272,  9274,  9282,  9299,  9335,  9351,  9354,  9367,  9376,
               9385,  9390,  9414,  9417,  9428,  9435,  9501,  9503,  9525,
               9544,  9548,  9549,  9563,  9577,  9602,  9616,  9628,  9660,
               9714,  9719,  9725,  9745,  9760,  9769,  9796,  9838,  9857,
               9887,  9897,  9906,  9917,  9955,  9996, 10007, 10052, 10113,
              10117, 10119, 10151, 10166, 10191, 10199, 10212, 10258, 10259,
              10265, 10271, 10278, 10301, 10314, 10317, 10346, 10370, 10450,
              10480, 10488, 10496, 10502, 10503, 10516, 10519, 10541, 10562,
              10571, 10585, 10620, 10675, 10706, 10737, 10751, 10762, 10777,
              10848, 10849, 10872, 10907, 10919, 10926, 10931, 10932, 10995,
              11006, 11011, 11083, 11107, 11113, 11114, 11132, 11136, 11140,
              11149, 11152, 11169, 11195, 11202, 11225, 11242, 11291, 11340,
              11345, 11350, 11354, 11384, 11415, 11434, 11486, 11533, 11549,
              11555, 11580, 11593, 11603, 11626, 11656, 11669, 11687, 11753,
              11759, 11773, 11828, 11834, 11844, 11848, 11943, 11973, 11974,
              11981, 12030, 12046, 12048, 12080, 12084, 12085, 12088, 12091,
```

```
         12094, 12107, 12166, 12171, 12189, 12208, 12210, 12213, 12217,
         12218, 12233, 12243, 12283, 12288, 12376, 12377, 12432, 12439,
         12440, 12451, 12466, 12488, 12510, 12540, 12613, 12620, 12627,
         12660, 12679, 12693, 12698, 12706, 12737, 12754, 12776, 12859,
         12901, 12921, 12925, 12937, 12946, 12950, 12964, 12972, 12983,
         12989, 13015, 13018, 13027, 13039, 13060, 13096, 13103, 13115,
         13148, 13165, 13186, 13189, 13224, 13230, 13236, 13255, 13273,
         13280, 13289, 13292, 13295, 13297, 13355, 13367, 13375, 13415,
         13427, 13432, 13440, 13448, 13452, 13458, 13462, 13492],
        dtype=int64),)
```

In [180]:

```
# show one of these misclassified image
plt.imshow(X_test[11, :, :])
```

Out[180]: <matplotlib.image.AxesImage at 0x2a511cf8978>



In [182]:

```
# This Industrial image was misclassified as Permanent Crop
pre_label[11, :]
```

Out[182]: array([0., 0., 0., 0., 0., 0., 1., 0., 0., 0.], dtype=float32)

In [ ]:

S7:

- Apply your best model on multispectral images.
- Q1: Calculate classification accuracy on the test data.
- Q2: Compare against results using RGB images.

In [0]:

In [0]:

In [0]: