The applications of deep learning and Artifical Inteligence in Spatial Analysis

Spatial Analysis Course Project

Title: "*Land-Use Classification using AlexNet-based architecture*"

Group Memebers:

- Golsa Talebi

Professor:

- Dr. Bahramian

Load the Dataset from Google Drive

1. The dataset has been preprocessed
2. The dataset has been saved into a pickle dataset
3. We use pickle.load to load the dataset
4. Guide variable consists of name of each class

The dataset is available via Kaggle

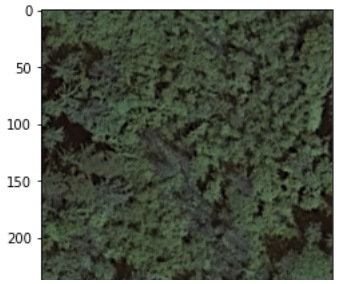
```
# Load the dataset
import pickle

pickle_in = open("/content/drive/MyDrive/Dataset/X.pickle","rb")
X = pickle.load(pickle_in)

pickle_in = open("/content/drive/MyDrive/Dataset/y.pickle","rb")
y = pickle.load(pickle_in)

pickle_in = open("/content/drive/MyDrive/Dataset/guide.pickle","rb")
guide = pickle.load(pickle_in)

Num_classes = 21
```

Testing the dataset randomly

Showing each picture and the class number related to it

1. Consists of 21 classes
2. Each class has 100 images
3. The size of each image is 256x256x3

```
# Testing the dataset
import matplotlib.pyplot as plt
import random
r1 = random.randint(0, 50)
plt.imshow(X[r1])
plt.show()
print('\nClass number related to the picture: '+str(y[r1])+'\n')
print('Class numbers guide:')
for i in range(4):
  print(guide[5*i:5*(i+1)])
print(guide[20])
```



Splitting train and test datasets

1. 25% of all the datasets are related to test and 75% is related to training of the network

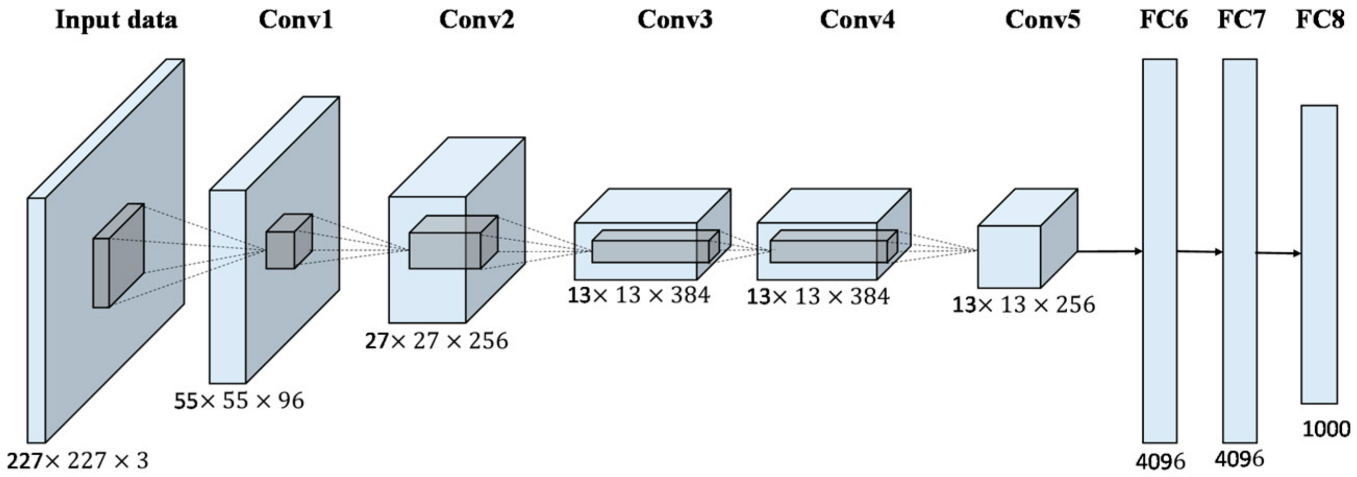
```
[['agricultural', 0], ['airplane', 1], ['baseballdiamond', 2], ['beach', 3], ['buildings
```

```
# Splits train and test
import numpy as np
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
y_train = np.array(y_train)
y_test = np.array(y_test)
X_train = np.array(X_train)
X_test = np.array(X_test)

y_train = y_train.reshape(-1, 1)
print('Number of images used in training process: '+str(len(y_train)))

y_test = y_test.reshape(-1, 1)
print('Number of images used in testing the network: '+str(len(y_test)))
```

```
Number of images used in training process: 1575
Number of images used in testing the network: 525
```

Creating the architecture of the network.

In this project we used AlexNet network with slight differences




```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
model=keras.Sequential([
    layers.Conv2D(filters=512, kernel_size=(5,5), strides=(4,4), activation='relu', input_shape=(256,256,3)),
    layers.BatchNormalization(),
    layers.MaxPool2D(pool_size=(2,2)),
    layers.Conv2D(filters=256, kernel_size=(5,5), strides=(1,1), activation='relu', padding="same"),
    layers.BatchNormalization(),
    layers.MaxPool2D(pool_size=(3,3)),
    # layers.Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), activation='relu', padding="same"),
    # layers.BatchNormalization(),
    layers.Conv2D(filters=256, kernel_size=(1,1), strides=(1,1), activation='relu', padding="same"),
```

```
    layers.BatchNormalization(),
    layers.Conv2D(filters=128, kernel_size=(3,3), strides=(1,1), activation='relu', padding="same"),
    layers.BatchNormalization(),
    # layers.Conv2D(filters=128, kernel_size=(1,1), strides=(1,1), activation='relu', padding="same"),
    # layers.BatchNormalization(),
    # layers.Conv2D(filters=56, kernel_size=(1,1), strides=(1,1), activation='relu', padding="same"),
    # layers.BatchNormalization(),
    layers.MaxPool2D(pool_size=(2,2)),
    layers.Flatten(),
    layers.Dense(1024,activation='relu'),
    layers.Dropout(0.7),
    layers.Dense(1024,activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(Num_classes,activation='softmax')

])


model.compile(
    loss='sparse_categorical_crossentropy',
    optimizer=tf.optimizers.SGD(learning_rate=0.001),
    metrics=['accuracy']
)
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 63, 63, 512)       38912

batch_normalization (BatchN  (None, 63, 63, 512)      2048
ormalization)

max_pooling2d (MaxPooling2D  (None, 31, 31, 512)      0
)

conv2d_1 (Conv2D)            (None, 31, 31, 256)       3277056

batch_normalization_1 (Batc  (None, 31, 31, 256)      1024
hNormalization)

max_pooling2d_1 (MaxPooling  (None, 10, 10, 256)      0
2D)

conv2d_2 (Conv2D)            (None, 10, 10, 256)       65792

batch_normalization_2 (Batc  (None, 10, 10, 256)      1024
hNormalization)

conv2d_3 (Conv2D)            (None, 10, 10, 128)       295040

batch_normalization_3 (Batc  (None, 10, 10, 128)      512
hNormalization)

max_pooling2d_2 (MaxPooling  (None, 5, 5, 128)        0
2D)

flatten (Flatten)            (None, 3200)              0

dense (Dense)                (None, 1024)              3277824

dropout (Dropout)            (None, 1024)              0

dense_1 (Dense)              (None, 1024)              1049600

dropout_1 (Dropout)          (None, 1024)              0

dense_2 (Dense)              (None, 21)                21525

=================================================================
Total params: 8,030,357
Trainable params: 8,028,053
Non-trainable params: 2,304
_________________________________________________________________
```

Training the model

```
from timeit import default_timer as timer
start = timer()
```

```
start = timer()
history = model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test))
print('\n\nTotal time of execution: '+str(timer()-start)+ 'seconds')
```

```
Epoch 1/10
50/50 [==============================] - 2s 44ms/step - loss: 0.6708 - accuracy: 0.7930 - val_loss: 1.1004 - val_accuracy: 0.6190
Epoch 2/10
50/50 [==============================] - 2s 42ms/step - loss: 0.6220 - accuracy: 0.7975 - val_loss: 1.4150 - val_accuracy: 0.5619
Epoch 3/10
50/50 [==============================] - 2s 43ms/step - loss: 0.5975 - accuracy: 0.8083 - val_loss: 1.1611 - val_accuracy: 0.6038
Epoch 4/10
50/50 [==============================] - 2s 43ms/step - loss: 0.6378 - accuracy: 0.7968 - val_loss: 1.1474 - val_accuracy: 0.6286
Epoch 5/10
50/50 [==============================] - 2s 43ms/step - loss: 0.6231 - accuracy: 0.7981 - val_loss: 1.2868 - val_accuracy: 0.6190
Epoch 6/10
50/50 [==============================] - 2s 43ms/step - loss: 0.5523 - accuracy: 0.8146 - val_loss: 1.0942 - val_accuracy: 0.6514
Epoch 7/10
50/50 [==============================] - 2s 43ms/step - loss: 0.5569 - accuracy: 0.8235 - val_loss: 1.0660 - val_accuracy: 0.6648
Epoch 8/10
50/50 [==============================] - 2s 43ms/step - loss: 0.5230 - accuracy: 0.8292 - val_loss: 1.0571 - val_accuracy: 0.6667
Epoch 9/10
50/50 [==============================] - 2s 43ms/step - loss: 0.5377 - accuracy: 0.8317 - val_loss: 1.7396 - val_accuracy: 0.5048
Epoch 10/10
50/50 [==============================] - 2s 43ms/step - loss: 0.5484 - accuracy: 0.8260 - val_loss: 1.0135 - val_accuracy: 0.6781


Total time of execution: 41.26560486800008seconds
```

Prediction using the model

By having the trained model in hand, we now can use it to predict the classes of test images.

By running each time, we will input the model a random test image and in return the model predicts its class name.

```
def show_result(guide, prediction):
  prediction_result = []
  for i in range(21):
    prediction_result.append([guide[i][0], prediction[0,i]])
  print(prediction_result)
  return prediction_result

def find_max_prob(pred_vector):
  index = np.where(pred_vector == max(pred_vector[0,:]))
  index_max = index[1][0]
  print('\nPredicted class: '+ res[index_max][0] +'\n Probability of class: '+ str(res[index_max][1])+'%\n\n')

def predict_model(X_test, img_number):
  x = X_test[img_number:img_number+1,]
  prediction = model.predict(x)
  return np.round(prediction*100, 2)

# x = X_test[0:1,]
# prediction = model.predict(x)
rnd_number = random.randint(0, 524)
prediction = predict_model(X_test, rnd_number)
# prediction = np.round(prediction*100, 2)
res = show_result(guide, prediction)
find_max_prob(prediction)

plt.imshow(X_test[rnd_number,])
plt.show()
```

```
1/1 [==============================] - 0s 18ms/step
[['agricultural', 0.01], ['airplane', 0.1], ['baseballdiamond', 1.67], ['beach', 0

Predicted class: freeway
 Probability of class: 93.84%
```

Save and Load the trained model with pickle

```
# Save the model to Google Drive
# filename = '/content/drive/MyDrive/Dataset/AlexNet_model.sav'
filename = '/content/drive/MyDrive/Dataset/refined_model.sav'
pickle.dump(model, open(filename, 'wb'))


# Load the model from Google Drive
filename = '/content/drive/MyDrive/Dataset/AlexNet_model.sav'
model = pickle.load(open(filename, 'rb'))
```