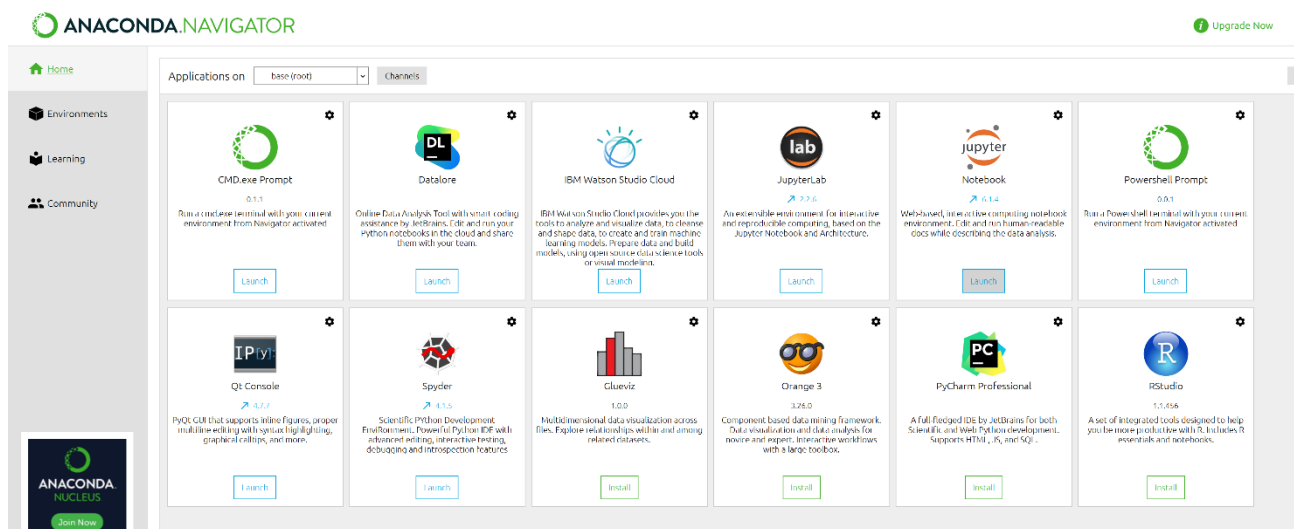


Day 3-4

We are not going to lie. Day 3-5 is going to be a lot more difficult as compared to Day 1-2. There will be a lot of Python versions, dependencies and libraries issues. On top of that, there are many lines of commands that the participants will need to input into the system. Nonetheless, Day 3-5 covers the application codes that will automate many analytics processes, which is why we will try to make everything as simplified as possible.

First of all, initiate Anaconda Navigator on your PC.



Select CMD.exe Prompt. A black command line interface should turn up. Refer below:

```
Microsoft Windows [Version 10.0.19042.1237]
(c) Microsoft Corporation. All rights reserved.

(base) C:\Users\ >conda create -n mysa -c conda-forge gdal
```

Type the following commands:

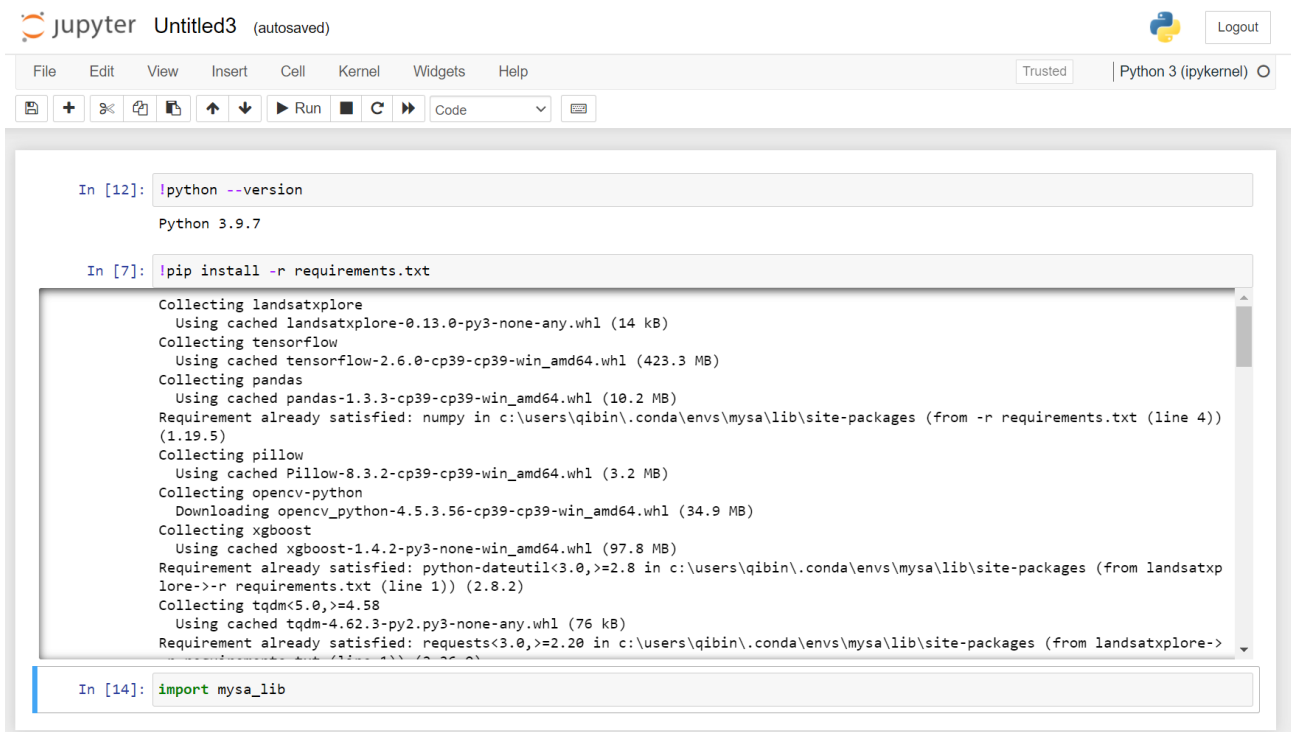
```
conda create -n mysa -c conda-forge gdal
conda activate mysa
conda install -c conda-forge rasterio
pip install jupyter
cd <DIRECTORY OF CHOICE WITH TRAINING FILES>
jupyter notebook
```

THERAPY

With this command, we are essentially creating an independent anaconda “environment” called “mysa”. In this environment, we will be installing the required libraries (gdal, rasterio and jupyter). By typing “jupyter notebook” and pressing ENTER, Jupyter notebook will be initiated in your default browser. You will see the following:



Click New on the upper right, and select Python3. This will create an empty “notebook” where you can test out the codes that we have prepared for you. Follow the commands in the following diagram.

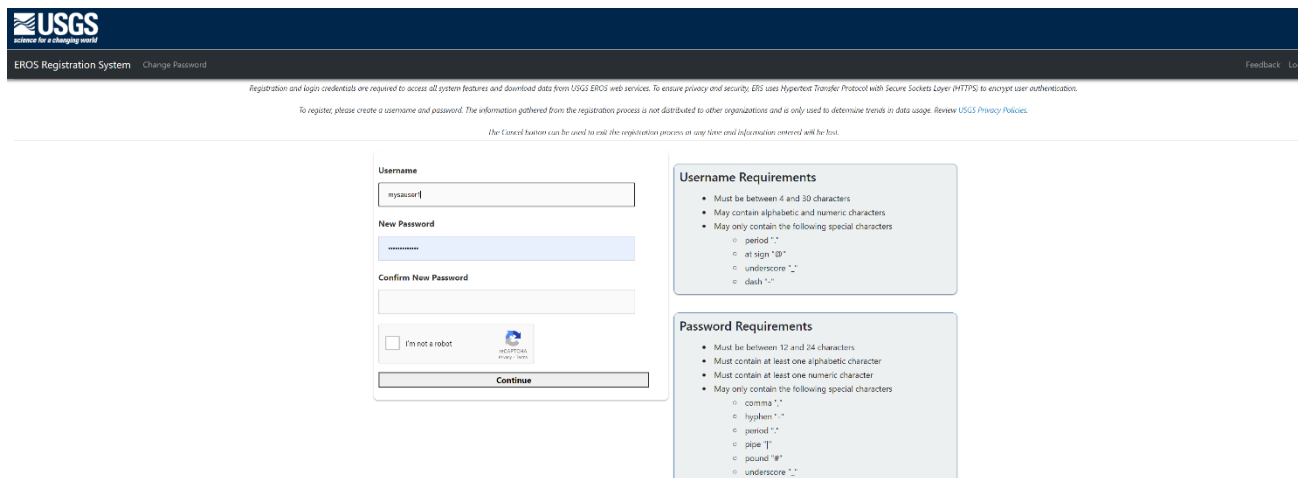


With this, the Python environment required should already be set. Make sure that Python version is higher than version 3.7. Also check for any errors as this will affect the next part of the class.

Automation in Satellite Images Downloading for LandSat8

The tutorial we have prepared here is a fast-track course, designed specifically for MYSA.

First of all, register for an ID here <https://ers.cr.usgs.gov/register/>.



USGS
science for a changing world

EROS Registration System [Change Password](#) [Feedback](#) [Log Out](#)

Registration and login credentials are required to access all system features and download data from USGS EROS web services. To ensure privacy and security, EROS uses Hypertext Transfer Protocol with Secure Sockets Layer (HTTPS) to encrypt user authentication.


To register, please create a username and password. The information gathered from the registration process is not distributed to other organizations and is only used to determine trends in data usage. [Review USGS Privacy Policies](#).

The Cancel button can be used to end the registration process at any time and information entered will be lost.

Username

New Password

Confirm New Password

☐ I'm not a robot 

Continue

Username Requirements

- Must be between 4 and 30 characters
- May contain alphabetic and numeric characters
- May only contain the following special characters
 - period "."
 - at sign "@"
 - underscore "_"
 - dash "-"

Password Requirements

- Must be between 12 and 24 characters
- Must contain at least one alphabetic character
- Must contain at least one numeric character
- May only contain the following special characters
 - comma ","
 - hyphen "-"
 - period "."
 - pipe "|"
 - pound "#"
 - underscore "_"

After verifying your account, login to <https://earthexplorer.usgs.gov/>

Search CriteriaData SetsAdditional CriteriaResults

1. Enter Search Criteria

To narrow your search area: type in an address or place name, enter coordinates or click the map to define your search area (for advanced map tools, view the help documentation), and/or choose a date range.

GeocoderKML/Shapfile Upload

Select a Geocoding Method

Feature (GNIS)

Search Limits: The search result limit is 100 records; select a Country, Feature Class, and/or Feature Type to reduce your chances of exceeding this limit.

US FeaturesWorld Features

Feature Name

(use % as wildcard)

State

All

Feature Type

All

ShowClear

PolygonCirclePredefined Area

Degree/Minute/SecondDecimal

1. Lat: 03° 08' 42" N, Lon: 101° 45' 07" E

Use MapAdd CoordinateClear Coordinates

Date RangeCloud CoverResult Options

Search from: mm/dd/yyyyto: mm/dd/yyyy

Search months: (all)

Data Sets »Additional Criteria »Results »

Search Criteria Summary (Show)Clear Search Criteria

(03° 07' 03" N, 101° 45' 16" E)Options

Map of Malaysia showing Kuala Lumpur and surrounding areas. A blue pin is located near Kuala Lumpur. The map includes labels for various locations such as Kuala Lumpur, Port Dickson, and various states like Selangor, Pahang, and Negeri Sembilan.

Loaded | Tiles © Esri — Source: Esri, DeLorme, USDA, USGS, AEX, GeoEye, GeoTran, IGN, IGP, UPR-EGP, and the GIS User Community. ESRI

The provided maps are not for purchase or for download; it is to be used as a guide for reference and search purposes only.

With this UI, we can get our region of interest in the form of coordinates. Of course, this can be done via QGIS or Google Earth as well.

Next, let's get back to the Jupyter notebook that we have created. Type these commands. Remember to substitute the username and password with the one you have registered with <https://ers.cr.usgs.gov>.

THERAPY

```
import landsatxplore.api
from landsatxplore.earthexplorer import EarthExplorer

sat_data='landsat_ot_c2_l2'
username='mysauser1'
password='malaysia#1my'
api = landsatxplore.api.API(username=username,password=password)
scenes = api.search(
    dataset=sat_data,
    latitude=3.0445,
    longitude=101.6244,
    start_date='1-8-2021',
    end_date='31-8-2021',max_results=200,
    max_cloud_cover=50)
l8_ls=[scene['display_id'] for scene in scenes]
print(l8_ls)
```

The above code basically allow us to search for scenes between the dates of “1-8-2021” and “31-8-2021” for the latitude 3.0445 and longitude 101.6244. In addition, we specify that max cloud cover should be 50% (usually we will recommend 5 or 10%) . The maximum results returned by USGS API should be less than 200. The output generated from the above code should only give you a single scene.

In this case, we have selected the satellite dataset known as “landsat_ot_c2_l2”, which is essentially Landsat 8 Collection 2 Level 2 (The importance of this dataset will be shared later), as shown by the table below.

Dataset Name	Dataset ID
Landsat 5 TM Collection 1 Level 1	landsat_tm_c1
Landsat 5 TM Collection 2 Level 1	landsat_tm_c2_11
Landsat 5 TM Collection 2 Level 2	landsat_tm_c2_12
Landsat 7 ETM+ Collection 1 Level 1	landsat_etm_c1
Landsat 7 ETM+ Collection 2 Level 1	landsat_etm_c2_11
Landsat 7 ETM+ Collection 2 Level 2	landsat_etm_c2_12
Landsat 8 Collection 1 Level 1	landsat_8_c1
Landsat 8 Collection 2 Level 1	landsat_ot_c2_11
Landsat 8 Collection 2 Level 2	landsat_ot_c2_12

Feel free to retry the above commands with different date ranges, different maximum cloud cover, different satellite dataset or different geolocations. Let us know the result.

THERAPY

```
import platform
print(platform.python_version())
import requests
import json

host = 'https://espa.cr.usgs.gov/api/v1/'

def espa_api(endpoint, verb='get', body=None, uauth=None):
    auth_tup = uauth if uauth else (username, password)
    response = getattr(requests, verb)(host + endpoint, auth=auth_tup, json=body)
    print('{} {}'.format(response.status_code, response.reason))
    data = response.json()
    if isinstance(data, dict):
        messages = data.pop("messages", None)
        if messages:
            print(json.dumps(messages, indent=4))
    try:
        response.raise_for_status()
    except Exception as e:
        print(e)
        return None
    else:
        return data

print('GET /api/v1/available-products')
order = espa_api('available-products', body=dict(inputs=18_ls))
print(json.dumps(order, indent=4))

# Add in the rest of the order information
order['format'] = 'gtiff'
order['resampling_method'] = 'cc'
order['note'] = 'MYSA automation test!!'

# Place the order
print('POST /api/v1/order')
resp = espa_api('order', verb='post', body=order)
print(json.dumps(resp, indent=4))
```

Now that we have our scene of interest, the above command allows us to order that particular scene via USGS API (<https://espa.cr.usgs.gov/api/v1/>). The result returned should look like below:

3.7.4

GET /api/v1/available-products

200 OK

```
{
  "olotirs8_collection_2_l2": {
    "products": [
      "11",
      "sr_ndvi",
      "sr_evi",
      "sr_savi",
      "sr_msavi",
      "sr_ndmi",
      "sr_nbr",
      "sr_nbr2",
      "stats"
    ],
    "inputs": [
      "LC08_L2SP_127058_20210802_20210811_02_T1"
    ]
  }
}
POST /api/v1/order
201 CREATED
{
  "orderid": "espa-mysauser1@gmail.com-08292021-000415-450",
  "status": "ordered"
}
```

The output shows that the scene/asset “LC08_L2SP_127058_20210802_20210811_02_T1” is available for download, which consists of products "sr_ndvi", "sr_evi", "sr_savi", "sr_msavi", "sr_ndmi", "sr_nbr", "sr_nbr2". The description of these products were as follow:

Product	Description
sr_ndvi	the normalized difference vegetation index from surface reflectance imagery
sr_evi	the enhanced vegetation index from surface reflectance imagery
sr_savi	the soil adjusted vegetation index from surface reflectance imagery
sr_msavi	the modified soil adjusted vegetation index from surface reflectance imagery
sr_ndmi	the normalized difference moisture index from surface reflectance imagery
sr_nbr	the normalized burn ratio from surface reflectance imagery
sr_nbr2	the normalized burn ratio 2 index from surface reflectance imagery

THERAPY

The status is “ordered” at the moment, we need to wait for it to be activated before we can download it. It will usually take a few days for activation to be done (for the purpose of this tutorial, we might have activated it on your behalf).

The code below will allow us to check on the activation status.

```
orderid = resp['orderid']
print('GET /api/v1/order-status/{}'.format(orderid))
resp = espa_api('order-status/{}'.format(orderid))
print(json.dumps(resp, indent=4))
if resp['status'] == "complete":
    print('GET /api/v1/item-status/{0}'.format(orderid))
    resp = espa_api('item-status/{0}'.format(orderid), body={'status': 'complete'})
    print(json.dumps(resp[orderid], indent=4))
```

```
GET /api/v1/order-status/espa-mysauser1@gmail.com-08292021-000415-450
200 OK
{
  "orderid": "espa-mysauser1@gmail.com-08292021-000415-450",
  "status": "complete"
}
GET /api/v1/item-status/espa-mysauser1@gmail.com-08292021-000415-450
200 OK
[
  {
    "cksum_download_url": "https://edclpdsftp.cr.usgs.gov/orders/espa-mysauser1@gmail.com-08292021-000415-450/LC081270582021080202T1-SC20210829050826.md5",
    "completion_date": "2021-08-29 00:10:16.629949",
    "name": "LC08_L2SP_127058_20210802_20210811_02_T1",
    "note": "",
    "product_dload_url": "https://edclpdsftp.cr.usgs.gov/orders/espa-mysauser1@gmail.com-08292021-000415-450/LC081270582021080202T1-SC20210829050826.tar.gz",
    "status": "complete"
  }
]
```

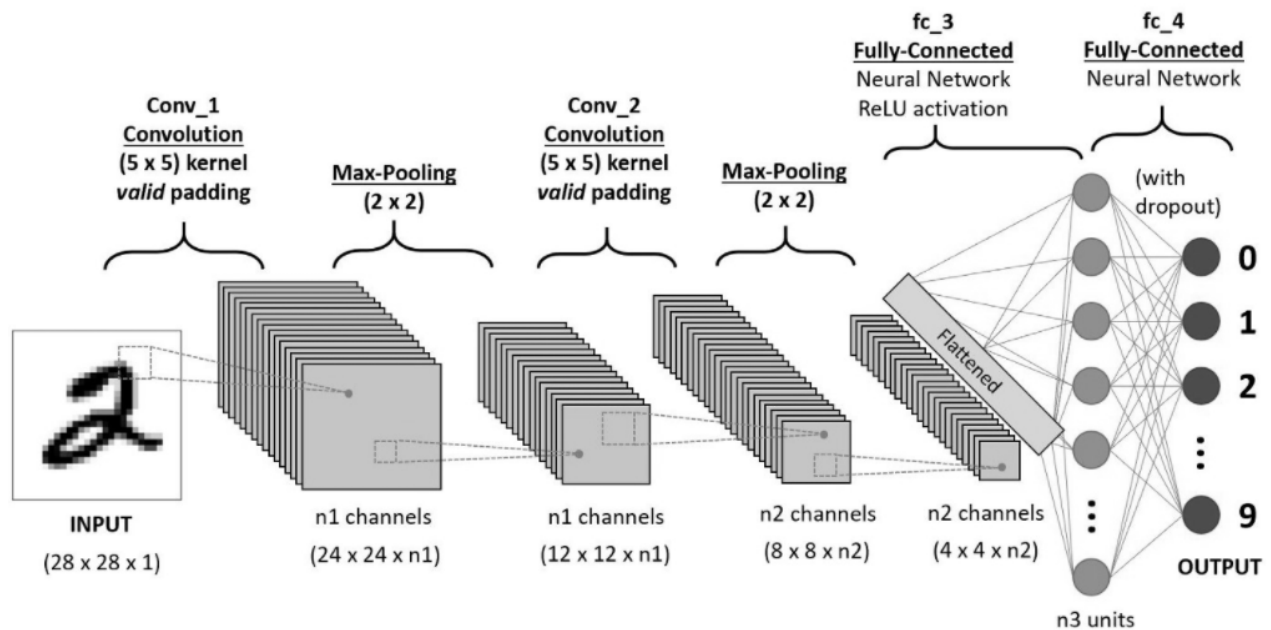
Once the status has been updated to complete, the download link will be made available. In this case, download link is now available at <https://edclpdsftp.cr.usgs.gov/orders/espa-mysauser1@gmail.com-08292021-000415-450/LC081270582021080202T1-SC20210829050826.tar.gz>.

Deep Learning with MNIST

The most famous handwritten digits, MNIST database contains 60,000 training images and 10,000 testing images. It is commonly used to train image processing systems and used to train/test different machine learning methods. Below are some examples of the images used.



For our first approach in building a deep learning model, we will be building a CNN model to recognize handwritten digits using MNIST dataset.



A typical neural network looks something like the above, with convolutional layers separated by max-pooling layers, before being flattened into fully-connected neural network layers.

The code to train the model can be found below:

THERAPY

```
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

from tensorflow.keras import layers
from tensorflow.keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', padding="valid", input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5, batch_size=64)
model.save('mnist_no_regularized.h5')

#model evaluation
test_loss, test_acc = model.evaluate(test_images, test_labels)
test_acc
```

The training process is going to take a while if you are running the process without using a GPU. To enable GPU-based training, different versions of CUDA and cuDNN libraries need to be installed in your computer, depending on your GPU driver and its manufacturer (not covered in this training).

```
Epoch 1/5
938/938 [=====] - 16s 16ms/step - loss: 0.1725 - accuracy: 0.9465
Epoch 2/5
938/938 [=====] - 15s 16ms/step - loss: 0.0480 - accuracy: 0.9853
Epoch 3/5
938/938 [=====] - 15s 16ms/step - loss: 0.0316 - accuracy: 0.9901
Epoch 4/5
938/938 [=====] - 15s 16ms/step - loss: 0.0241 - accuracy: 0.9927
Epoch 5/5
938/938 [=====] - 15s 16ms/step - loss: 0.0191 - accuracy: 0.9941
313/313 [=====] - 1s 3ms/step - loss: 0.0320 - accuracy: 0.9906
0.9905999898910522
```

THERAPY

Above shows the training log for our CNN model. It seems that the accuracy of our model is 99%, which is rather good. In this case, the formula of accuracy is quite straightforward,

$$Accuracy = \frac{\text{No of correct predictions}}{\text{Total no of predictions}}$$

Let's try out the model on some sample images. The code for testing the model can be found below:

```
from tensorflow import keras
import cv2
import numpy as np
model = keras.models.load_model('mnist_no_regularized.h5')

list1=list(range(0,10))
in1=cv2.imread('example/example_17.jpg')
resized = cv2.resize(in1, (28,28), interpolation = cv2.INTER_AREA)
resized=resized[:, :,0]
answer_index=model.predict(resized.reshape(1,28,28,1))
#Print the answer
print(list1[int(np.where(answer_index[0]==1)[0])])
```

So, let's compare the predicted digit versus the selected example. If trained properly, both will yield the result "7". Try to test the model on other examples.

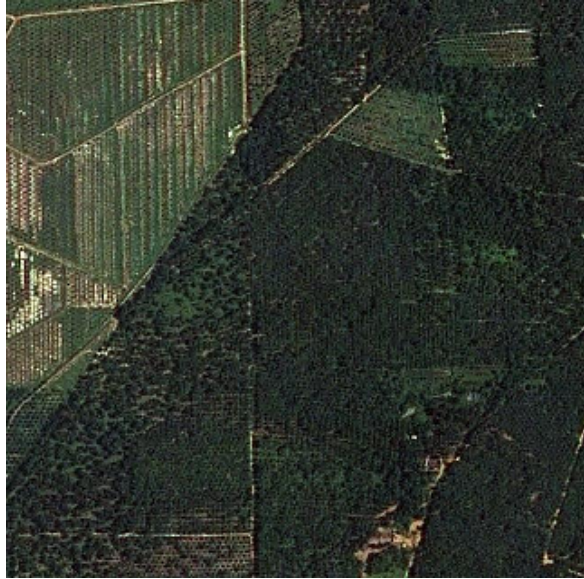
Deep Learning with MYSA Agricultural Lands Vs. Forest Dataset (BONUS)

While doing remote sensing studies, it is sometimes important for us to classify the images into different classes. Take for example, how do we know how much of the forest has been converted into agricultural lands/plantations? Manually looking through the images is going to take a long time. Imagine having an automated system that can screen through all Malaysia satellite images and automatically summarizes forest coverage in Malaysia, or measures the rate of deforestation in Malaysia every single year. To simplify the problem (for this training), as long as part of the image contains agricultural lands, we are labelling it as agricultural lands.

In this example, we are building a Convolutional Neural Network (CNN) model that can classify agricultural land from forest. Since we are dealing with a rather complex dataset as compared with the previous MNIST dataset, we will be using the architecture of ResNet101 as the backbone of the CNN model. To build the model, we have prepared 551 images for training and 226 for validation based on images provided by MYSA. Some example images:



Forest



Agriculture

Run the code below to build the model:

```

from tensorflow.keras.applications import ResNet101
from tensorflow.keras.callbacks import EarlyStopping
conv_base = ResNet101(weights=None, include_top=True, input_shape=(160, 160, 3))
model = models.Sequential()
model.add(conv_base)
model.add(layers.Flatten())
#model.add(layers.Dropout(0.1))
model.add(layers.Dense(256, activation='relu', kernel_regularizer='l2'))
model.add(layers.Dense(1, activation='sigmoid'))
model.summary()

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import optimizers
train_dir='farm_forest/train'
validation_dir='farm_forest/validation'
train_datagen = ImageDataGenerator(rescale=1./255, rotation_range=40, width_shift_range=0.2,
                                   height_shift_range=0.2, shear_range=0.2, zoom_range=0.2,
                                   horizontal_flip=True, fill_mode='nearest')
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(train_dir, target_size=(160, 160),
                                                    batch_size=10, class_mode='binary')
validation_generator = test_datagen.flow_from_directory(validation_dir, target_size=(160, 160),
                                                        batch_size=10, class_mode='binary')

model.compile(loss='binary_crossentropy', optimizer=optimizers.RMSprop(lr=2e-5), metrics=['acc'])
es = EarlyStopping(monitor='val_acc', mode='max', patience=30, restore_best_weights=True)
history = model.fit_generator(train_generator, steps_per_epoch=20, epochs=500,
                             validation_data=validation_generator, validation_steps=20, callbacks=[es])

```

We will not dive deep into the code above. Essentially, it is reading both training and validation datasets, and then resizing them into the size of (160, 160) before using the training dataset to build the model. To make sure that we are getting the best performing model, we will be using a method known as “earlystopping”. Essentially, we are stopping the model training the moment the accuracy of the model drops after some epochs. The regularizer here is mainly to avoid model overfitting, while the optimizer used is to reduce the training time. During each epoch, the model will be evaluated using the validation dataset. The diagram below shows the training process and the resulting loss values and accuracies. Both loss values and accuracies are metrics that we are using to determine the performance of the model.

THERAPY

```
20/20 [=====] - 3/5 2s/step - loss: 3.3950 - acc: 0.9250 - val_loss: 3.4500 - val_acc: 0.4550
Epoch 22/500
20/20 [=====] - 37s 2s/step - loss: 3.3395 - acc: 0.9200 - val_loss: 3.3937 - val_acc: 0.4750
Epoch 23/500
20/20 [=====] - 37s 2s/step - loss: 3.2783 - acc: 0.9500 - val_loss: 3.3469 - val_acc: 0.4400
Epoch 24/500
20/20 [=====] - 38s 2s/step - loss: 3.2346 - acc: 0.8912 - val_loss: 3.2876 - val_acc: 0.4800
Epoch 25/500
20/20 [=====] - 37s 2s/step - loss: 3.1730 - acc: 0.9300 - val_loss: 3.2370 - val_acc: 0.4750
Epoch 26/500
20/20 [=====] - 37s 2s/step - loss: 3.1202 - acc: 0.9300 - val_loss: 3.1832 - val_acc: 0.4900
Epoch 27/500
20/20 [=====] - 36s 2s/step - loss: 3.0710 - acc: 0.9119 - val_loss: 3.1199 - val_acc: 0.5550
Epoch 28/500
20/20 [=====] - 37s 2s/step - loss: 3.0160 - acc: 0.9200 - val_loss: 3.0689 - val_acc: 0.5600
Epoch 29/500
20/20 [=====] - 36s 2s/step - loss: 2.9639 - acc: 0.9275 - val_loss: 2.9987 - val_acc: 0.6600
Epoch 30/500
20/20 [=====] - 37s 2s/step - loss: 2.9073 - acc: 0.9600 - val_loss: 2.9198 - val_acc: 0.7900
Epoch 31/500
20/20 [=====] - 36s 2s/step - loss: 2.8608 - acc: 0.9326 - val_loss: 2.8545 - val_acc: 0.8550
Epoch 32/500
20/20 [=====] - 37s 2s/step - loss: 2.8122 - acc: 0.9300 - val_loss: 2.7964 - val_acc: 0.9000
Epoch 33/500
20/20 [=====] - 37s 2s/step - loss: 2.7591 - acc: 0.9500 - val_loss: 2.7386 - val_acc: 0.9350
Epoch 34/500
20/20 [=====] - 37s 2s/step - loss: 2.7219 - acc: 0.9100 - val_loss: 2.6911 - val_acc: 0.9400
Epoch 35/500
20/20 [=====] - 37s 2s/step - loss: 2.6728 - acc: 0.9150 - val_loss: 2.6385 - val_acc: 0.9550
Epoch 36/500
20/20 [=====] - 36s 2s/step - loss: 2.6228 - acc: 0.9275 - val_loss: 2.5933 - val_acc: 0.9550
Epoch 37/500
20/20 [=====] - 37s 2s/step - loss: 2.5790 - acc: 0.9200 - val_loss: 2.5472 - val_acc: 0.9550
Epoch 38/500
20/20 [=====] - 36s 2s/step - loss: 2.5386 - acc: 0.8964 - val_loss: 2.5009 - val_acc: 0.9600
Epoch 39/500
20/20 [=====] - 37s 2s/step - loss: 2.4888 - acc: 0.9300 - val_loss: 2.4558 - val_acc: 0.9600
Epoch 40/500
20/20 [=====] - 37s 2s/step - loss: 2.4477 - acc: 0.9050 - val_loss: 2.4100 - val_acc: 0.9650
```

As you can see, the final model achieves an accuracy of $> 95\%$, which is surprisingly good. A side note, usually the validation accuracy (val_acc) will never be higher than the training accuracy (acc), this is only observed because a small portion of training images were used for validation due to the limited dataset that we have.

Let's now try to test the model on some other images.

THERAPY

```
#0-agriculture,1-forest
test_dir='farm_forest/test'
test_generator = test_datagen.flow_from_directory(test_dir,target_size=(160, 160))
predictions = (model.predict(test_generator) > 0.5).astype("int32")
predictions
```

Found 6 images belonging to 2 classes.

```
array([[0],
       [0],
       [0],
       [1],
       [0],
       [1]], dtype=int32)
```