Configuration Manual

MSc Research Project
MSc. in Data Analytics

Venkatesh Mukhopadhyay
Student ID: x20107790

School of Computing
National College of Ireland

Supervisor: Dr. Barry Haycock

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Venkatesh Mukhopadhyay ……………………………………………………………………………………………………		
Student ID:	X20107790………………………………………………………………………………………		
Programme:	MSc. in Data Analytic	**Year:**	2021……….
Module:	MSc. Research Project ………		
Lecturer:	16/12/2021……………………………………………………………………………………		
Submission Due Date:	………		
Project Title:	Semantic Crop Segmentation Using Deep Learning Technique		
Word Count:	1200………………………	**Page Count:**	16………………………………………

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Venkatesh……………………………………………………………………………………………

Date: 16/12/2021……………………………………………………………………………………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

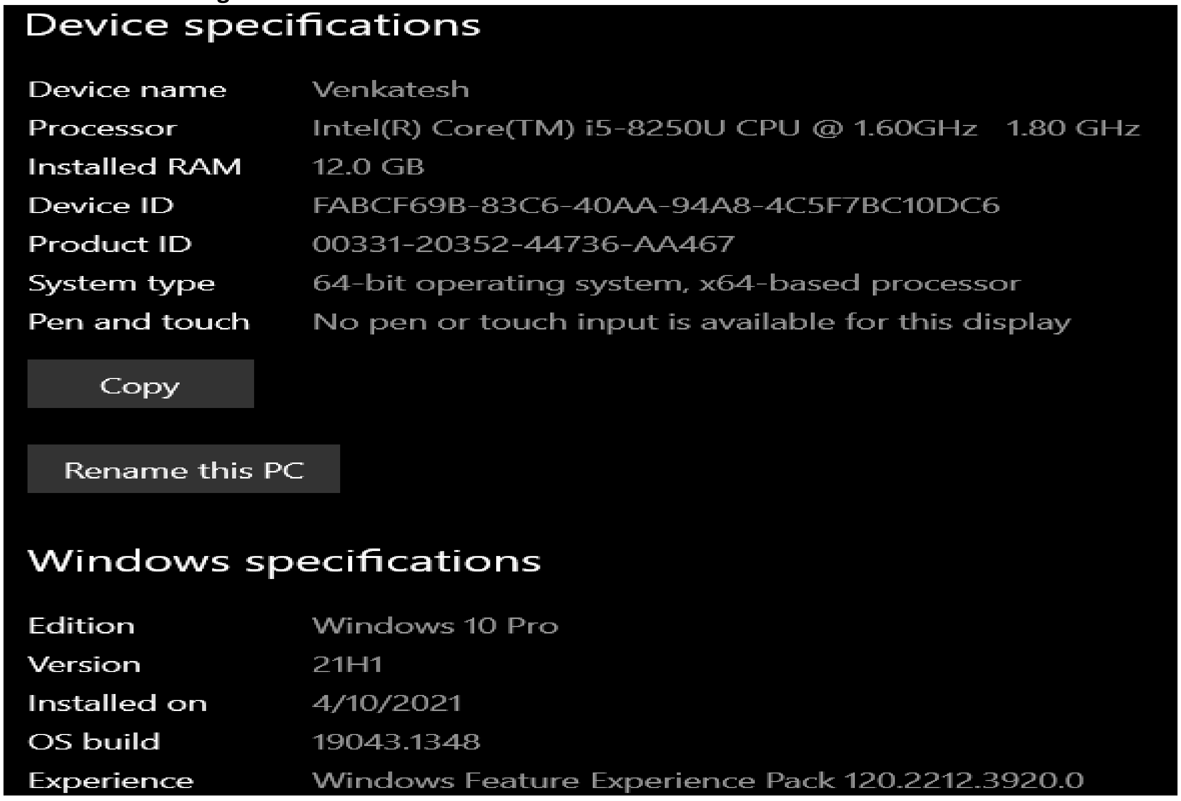
Venkatesh Mukhoadhyay
Student ID: x20107790

1 Introduction

The main purpose of this documentation is to compile all the process that must be completed throughout the research project implementation and execution stage. Core technologies , hardware, Library Dependencies , software and prerequisites are provided in order to replicate the research work in the future. This document covers all procedures that are required to run the code from end to end, this document also covers the deployment process.

2 System Configurations

2.1 **Hardware Configuration**



Device specifications

Device name	Venkatesh
Processor	Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz
Installed RAM	12.0 GB
Device ID	FABCF69B-83C6-40AA-94A8-4C5F7BC10DC6
Product ID	00331-20352-44736-AA467
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Windows specifications

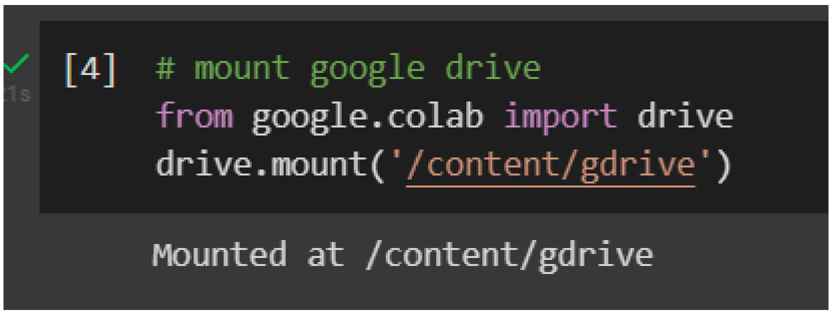
Edition	Windows 10 Pro
Version	21H1
Installed on	4/10/2021
OS build	19043.1348
Experience	Windows Feature Experience Pack 120.2212.3920.0

2.2 **Software Configuration**

This segment of the configuration offers details about the software which is used to carry out the study, as well as its specs.

2.2.1 **Google Colaboratory**

The entire dataset is uploaded to google drive and then the google drive is connected to the colab notebook



```
[4]  # mount google drive
     from google.colab import drive
     drive.mount('/content/gdrive')

     Mounted at /content/gdrive
```

Figure 2: Mounting Google Drive on Google Colab

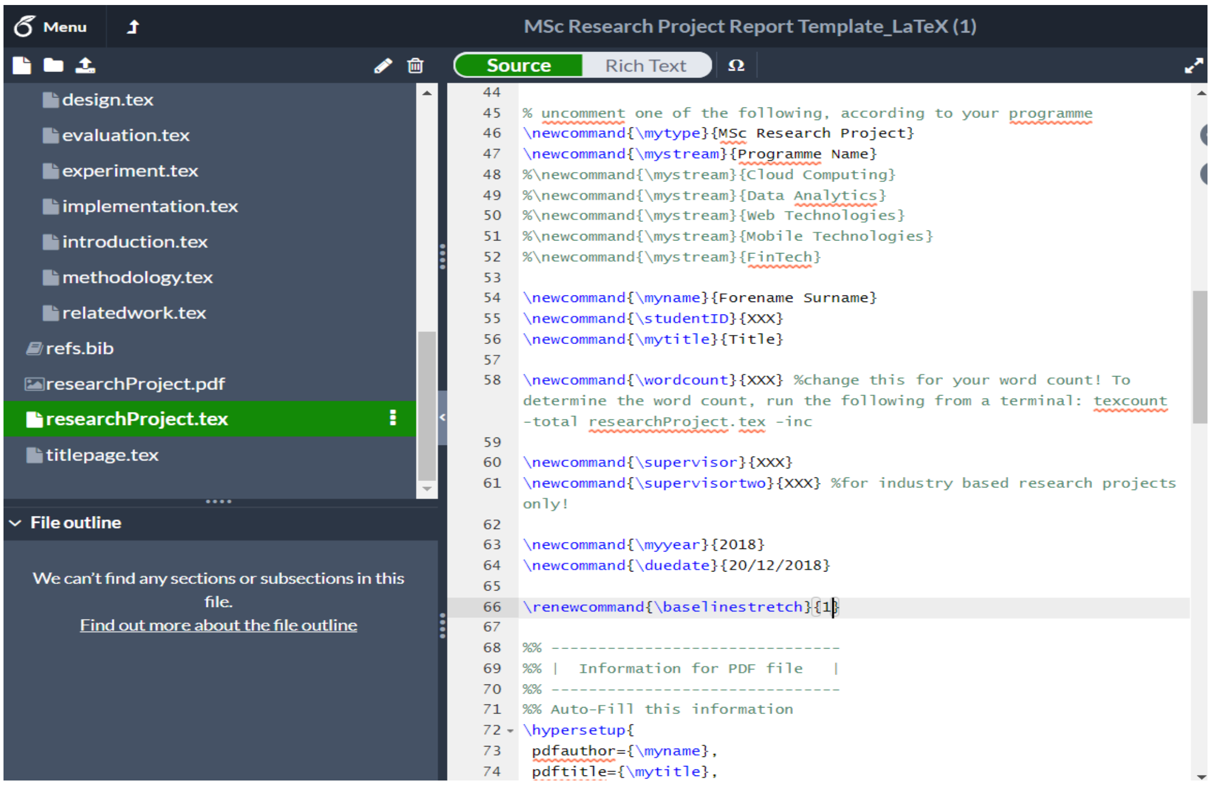
After the above command is executed , there will be a pop up message which will ask us to confirm the google drive account which we want to mount to the google colab notebook. We have give some permissions so that google drive can be mounted to our Colab notebook.

Colab GPU

We will enable Google colab GPU using the below command. Google usually provide low power GPU for less intensive task

2.2.2 Other Software Used

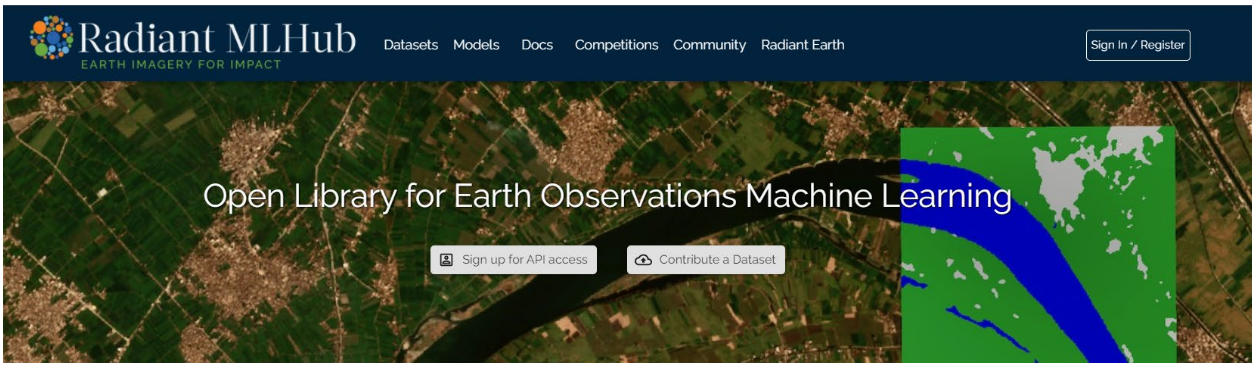
In order to interact with Google colab we have used google chrome web browser and for report writing we have used overleaf which handles LeTex style reporting with ease.



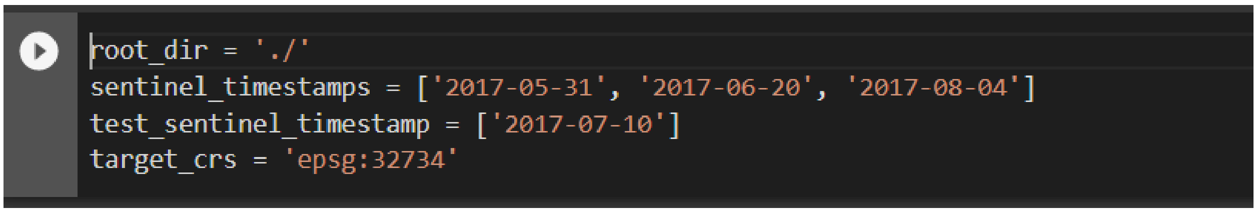
```

% uncomment one of the following, according to your programme
\newcommand{\mytype}{MSc Research Project}
\newcommand{\mystream}{Programme Name}
%\newcommand{\mystream}{Cloud Computing}
%\newcommand{\mystream}{Data Analytics}
%\newcommand{\mystream}{Web Technologies}
%\newcommand{\mystream}{Mobile Technologies}
%\newcommand{\mystream}{FinTech}

\newcommand{\myname}{Forename Surname}
\newcommand{\studentID}{XXX}
\newcommand{\mytitle}{Title}

\newcommand{\wordcount}{XXX} %change this for your word count! To determine the word count, run the following from a terminal: texcount -total researchProject.tex -inc

\newcommand{\supervisor}{XXX}
\newcommand{\supervisortwo}{XXX} %for industry based research projects only!

\newcommand{\myyear}{2018}
\newcommand{\duedate}{20/12/2018}

\renewcommand{\baselinestretch}{1}

%% -------------------------------
%% |  Information for PDF file   |
%% -------------------------------
%% Auto-Fill this information
\hypersetup{
 pdfauthor={\myname},
 pdftitle={\mytitle},
```

3 Data Preparation

The dataset used for this research work is collected from RadiantMLHub portal. Which is an open library for satellite images taken from multiple satellite of different different location.

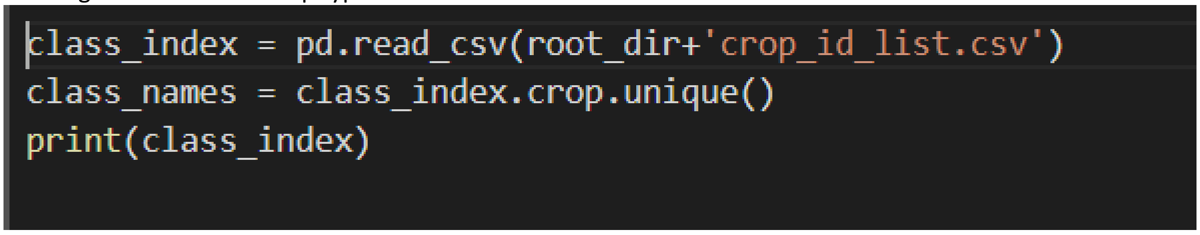


The images from the sentinel 2 satellite which belong from the same month are put into same folder having same time stamp. (March 22, May 31, June 20 and August 4)



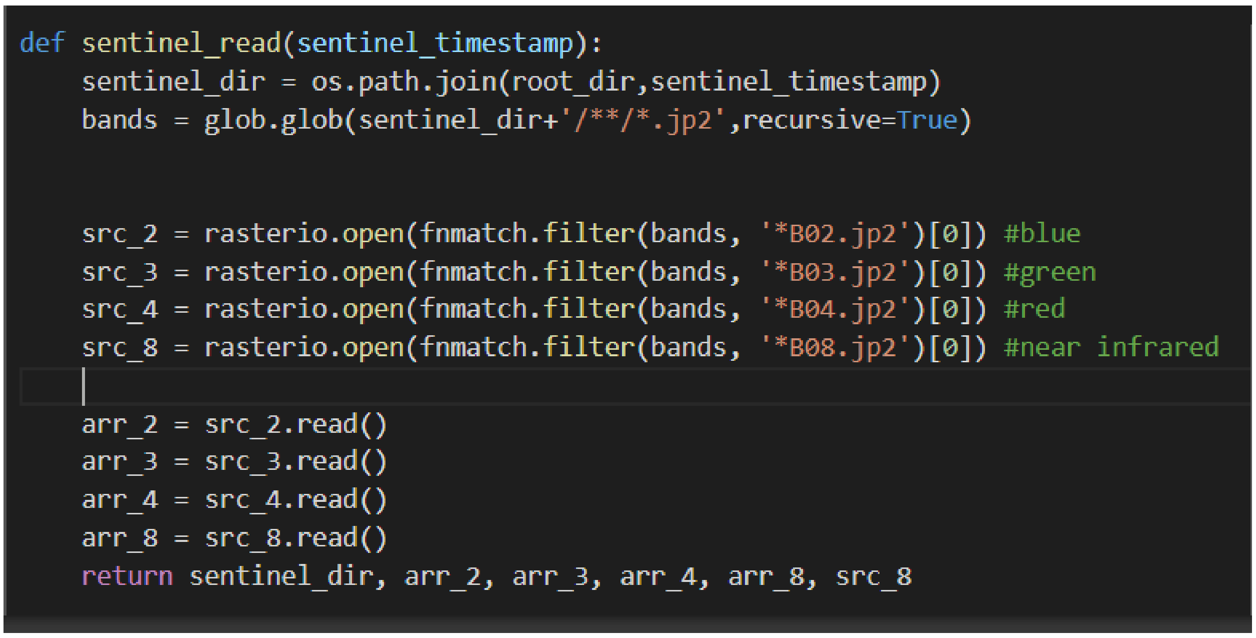
```
root_dir = './'
sentinel_timestamps = ['2017-05-31', '2017-06-20', '2017-08-04']
test_sentinel_timestamp = ['2017-07-10']
target_crs = 'epsg:32734'
```

Reading the class of the crop type



```
class_index = pd.read_csv(root_dir+'crop_id_list.csv')
class_names = class_index.crop.unique()
print(class_index)
```

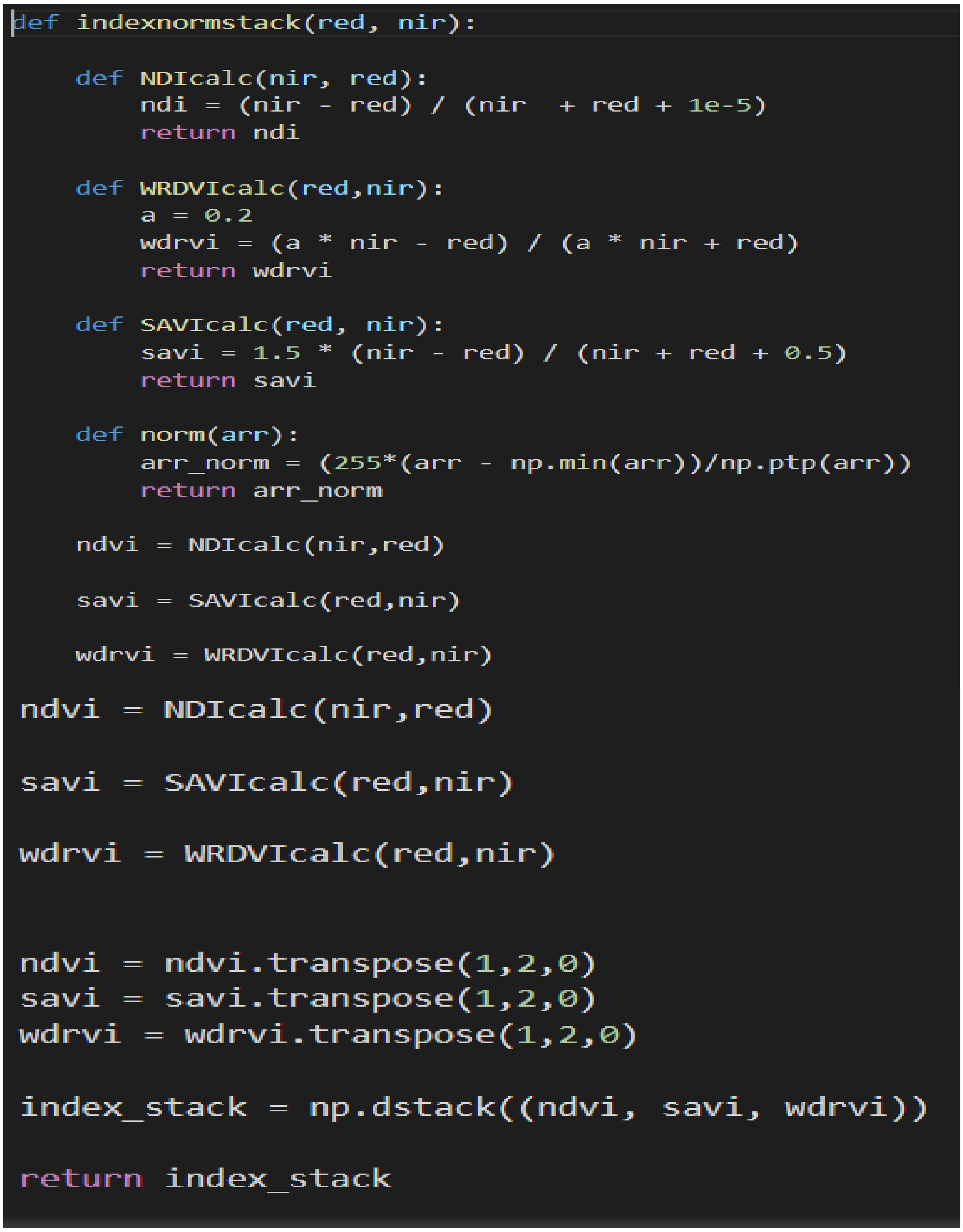
Calculating the spectral indexes using band 2, 3, 4 and 8 from the sentinel 2 images which are red, green , blue and near infrared wavelengths. First, we need to get the bands metadata and create an array from the images.



```
def sentinel_read(sentinel_timestamp):
    sentinel_dir = os.path.join(root_dir,sentinel_timestamp)
    bands = glob.glob(sentinel_dir+'/**/*.jp2',recursive=True)


    src_2 = rasterio.open(fnmatch.filter(bands, '*B02.jp2')[0]) #blue
    src_3 = rasterio.open(fnmatch.filter(bands, '*B03.jp2')[0]) #green
    src_4 = rasterio.open(fnmatch.filter(bands, '*B04.jp2')[0]) #red
    src_8 = rasterio.open(fnmatch.filter(bands, '*B08.jp2')[0]) #near infrared

    arr_2 = src_2.read()
    arr_3 = src_3.read()
    arr_4 = src_4.read()
    arr_8 = src_8.read()
    return sentinel_dir, arr_2, arr_3, arr_4, arr_8, src_8
```

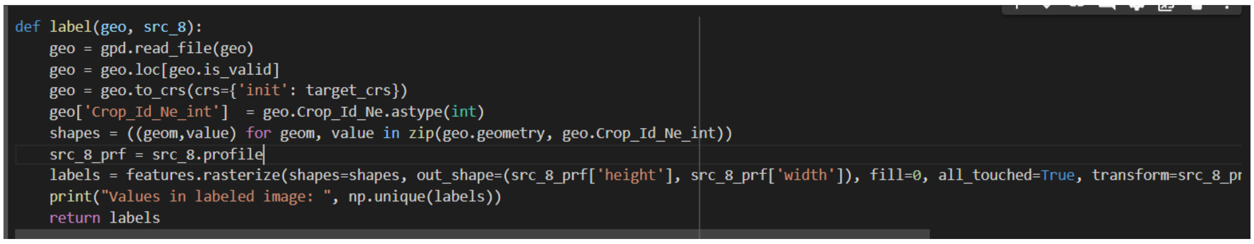
Calculating Vegetation Indexes from Images Bands

We will now calculate the spectral indexes and concatenate them into 1 index stack.



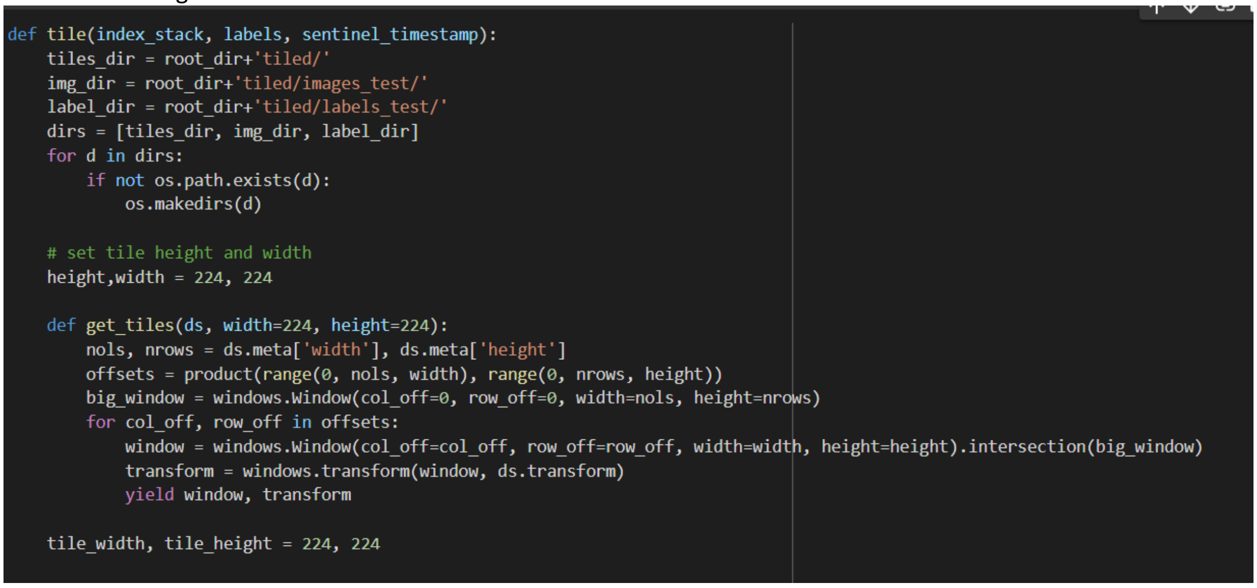
```
def indexnormstack(red, nir):

    def NDIcalc(nir, red):
        ndi = (nir - red) / (nir  + red + 1e-5)
        return ndi

    def WRDVIcalc(red,nir):
        a = 0.2
        wdrvi = (a * nir - red) / (a * nir + red)
        return wdrvi

    def SAVIcalc(red, nir):
        savi = 1.5 * (nir - red) / (nir + red + 0.5)
        return savi

    def norm(arr):
        arr_norm = (255*(arr - np.min(arr))/np.ptp(arr))
        return arr_norm

    ndvi = NDIcalc(nir,red)

    savi = SAVIcalc(red,nir)

    wdrvi = WRDVIcalc(red,nir)
ndvi = NDIcalc(nir,red)

savi = SAVIcalc(red,nir)

wdrvi = WRDVIcalc(red,nir)


ndvi = ndvi.transpose(1,2,0)
savi = savi.transpose(1,2,0)
wdrvi = wdrvi.transpose(1,2,0)

index_stack = np.dstack((ndvi, savi, wdrvi))

return index_stack
```

The shapefile having labels are read into a geopandas dataframe just to check the irregularities in the geometries and also the local CRS is set.
Information from the images having grayscale band are used to rasterize the polygons having labels.
Reprojecting training data into local coordinate system is also performed here.

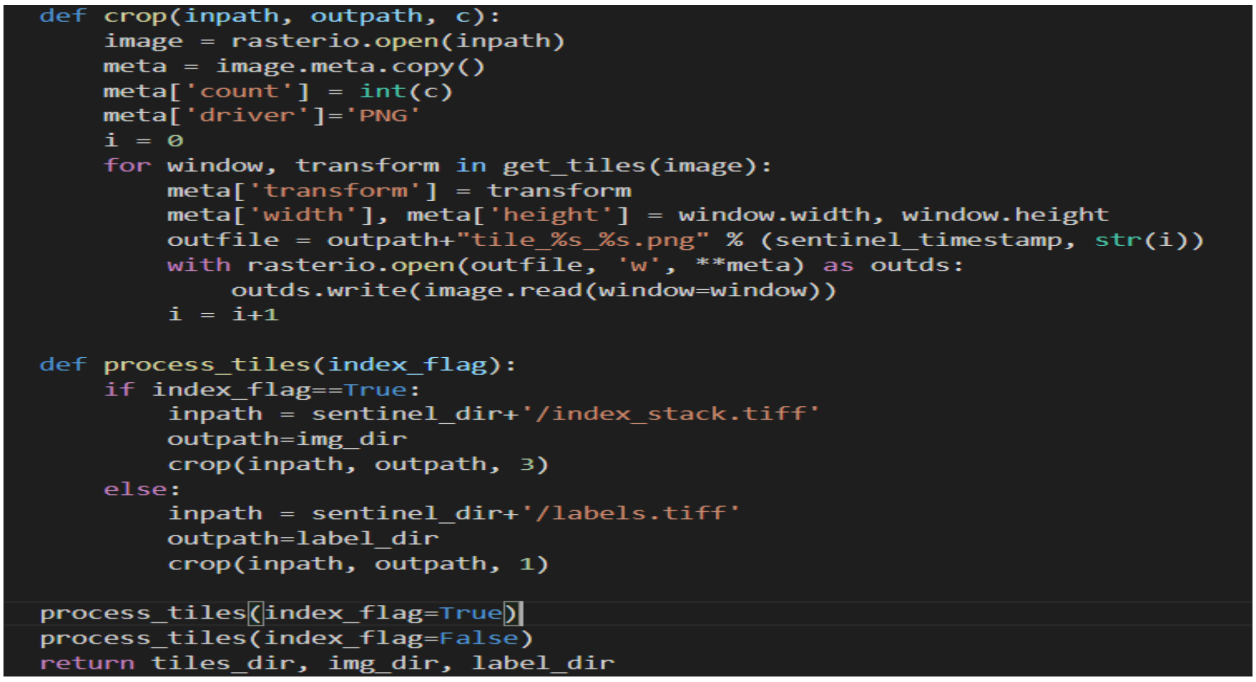


```
def label(geo, src_8):
    geo = gpd.read_file(geo)
    geo = geo.loc[geo.is_valid]
    geo = geo.to_crs(crs={'init': target_crs})
    geo['Crop_Id_Ne_int']  = geo.Crop_Id_Ne.astype(int)
    shapes = ((geom,value) for geom, value in zip(geo.geometry, geo.Crop_Id_Ne_int))
    src_8_prf = src_8.profile
    labels = features.rasterize(shapes=shapes, out_shape=(src_8_prf['height'], src_8_prf['width']), fill=0, all_touched=True, transform=src_8_pr
    print("Values in labeled image: ", np.unique(labels))
    return labels
```

The labeled images and index stack is converted into tiles of 224 x 224

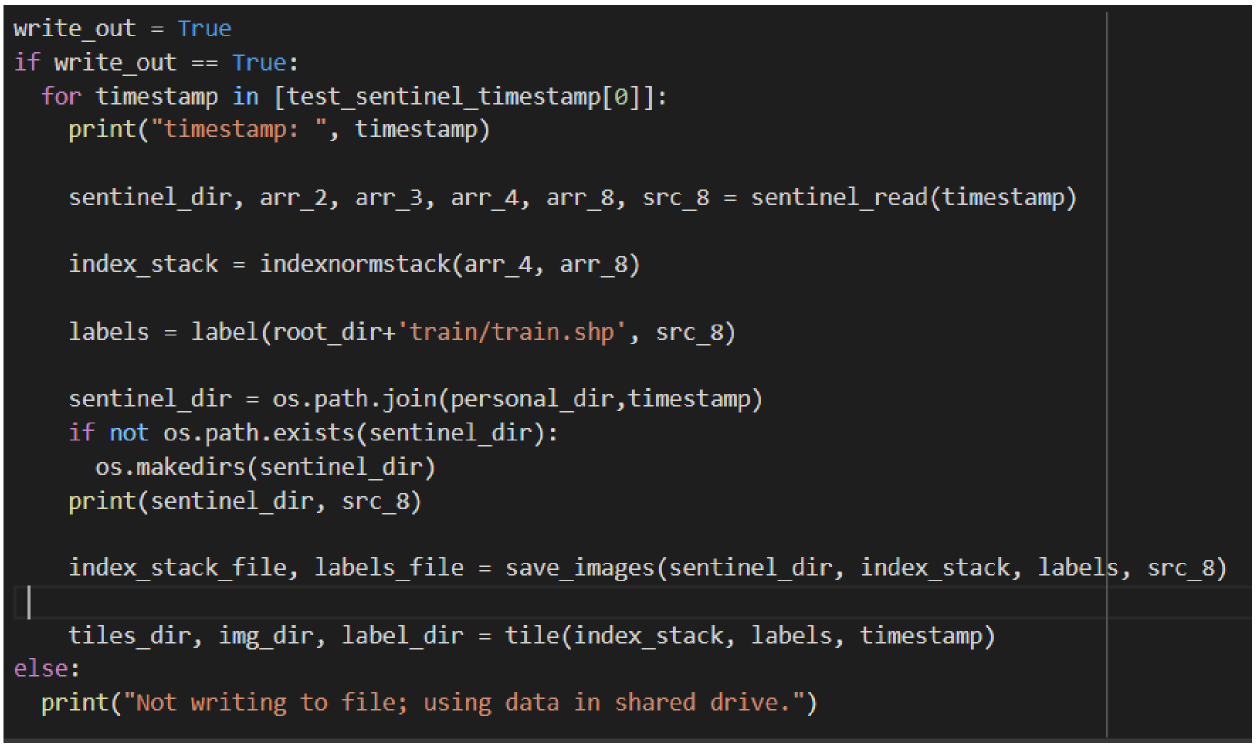


```
def tile(index_stack, labels, sentinel_timestamp):
    tiles_dir = root_dir+'tiled/'
    img_dir = root_dir+'tiled/images_test/'
    label_dir = root_dir+'tiled/labels_test/'
    dirs = [tiles_dir, img_dir, label_dir]
    for d in dirs:
        if not os.path.exists(d):
            os.makedirs(d)

    # set tile height and width
    height,width = 224, 224

    def get_tiles(ds, width=224, height=224):
        nols, nrows = ds.meta['width'], ds.meta['height']
        offsets = product(range(0, nols, width), range(0, nrows, height))
        big_window = windows.Window(col_off=0, row_off=0, width=nols, height=nrows)
        for col_off, row_off in offsets:
            window = windows.Window(col_off=col_off, row_off=row_off, width=width, height=height).intersection(big_window)
            transform = windows.transform(window, ds.transform)
            yield window, transform

    tile_width, tile_height = 224, 224
```



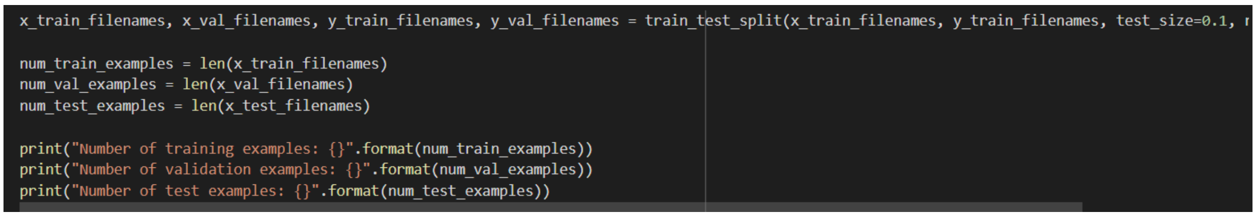
```
def crop(inpath, outpath, c):
    image = rasterio.open(inpath)
    meta = image.meta.copy()
    meta['count'] = int(c)
    meta['driver']='PNG'
    i = 0
    for window, transform in get_tiles(image):
        meta['transform'] = transform
        meta['width'], meta['height'] = window.width, window.height
        outfile = outpath+"tile_%s_%s.png" % (sentinel_timestamp, str(i))
        with rasterio.open(outfile, 'w', **meta) as outds:
            outds.write(image.read(window=window))
        i = i+1

def process_tiles(index_flag):
    if index_flag==True:
        inpath = sentinel_dir+'/index_stack.tiff'
        outpath=img_dir
        crop(inpath, outpath, 3)
    else:
        inpath = sentinel_dir+'/labels.tiff'
        outpath=label_dir
        crop(inpath, outpath, 1)

process_tiles(index_flag=True)
process_tiles(index_flag=False)
return tiles_dir, img_dir, label_dir
```

Once the data is prepared, we will run the image processing workflow which will stitch all the information. This function takes time to run.

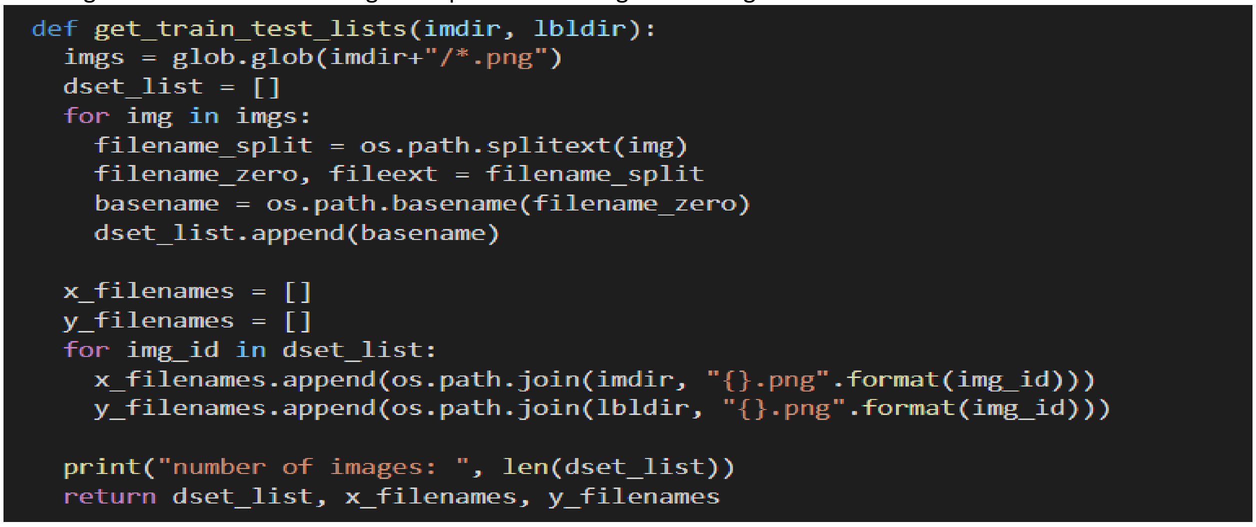


```
write_out = True
if write_out == True:
  for timestamp in [test_sentinel_timestamp[0]]:
    print("timestamp: ", timestamp)

    sentinel_dir, arr_2, arr_3, arr_4, arr_8, src_8 = sentinel_read(timestamp)

    index_stack = indexnormstack(arr_4, arr_8)

    labels = label(root_dir+'train/train.shp', src_8)

    sentinel_dir = os.path.join(personal_dir,timestamp)
    if not os.path.exists(sentinel_dir):
      os.makedirs(sentinel_dir)
    print(sentinel_dir, src_8)

    index_stack_file, labels_file = save_images(sentinel_dir, index_stack, labels, src_8)

    tiles_dir, img_dir, label_dir = tile(index_stack, labels, timestamp)
else:
  print("Not writing to file; using data in shared drive.")
```

Splitting the data into test and train set

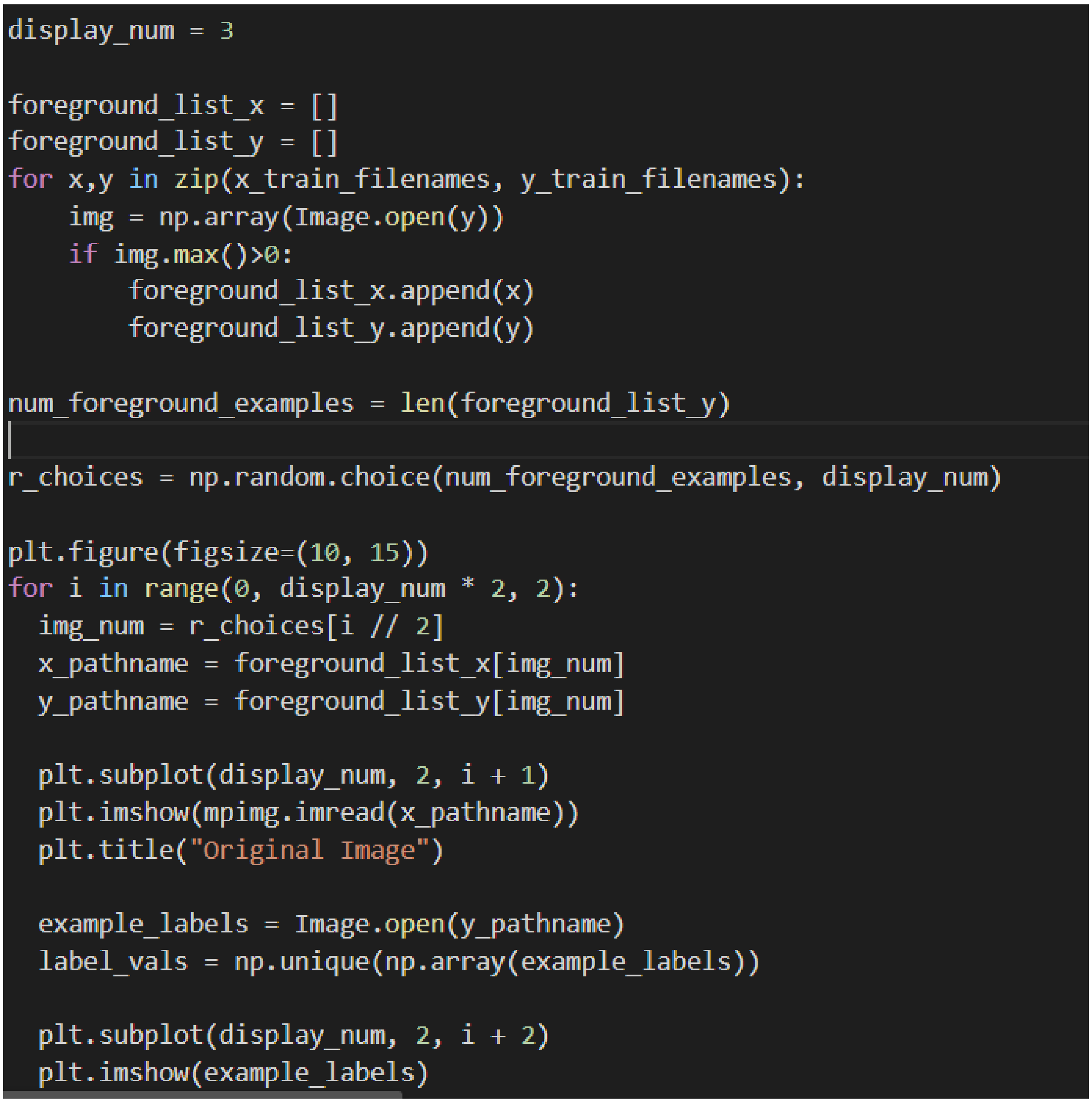

```
x_train_filenames, x_val_filenames, y_train_filenames, y_val_filenames = train_test_split(x_train_filenames, y_train_filenames, test_size=0.1, r

num_train_examples = len(x_train_filenames)
num_val_examples = len(x_val_filenames)
num_test_examples = len(x_test_filenames)

print("Number of training examples: {}".format(num_train_examples))
print("Number of validation examples: {}".format(num_val_examples))
print("Number of test examples: {}".format(num_test_examples))
```

Fetching the list of labels and image tiles pairs for training and testing


```
def get_train_test_lists(imdir, lbldir):
  imgs = glob.glob(imdir+"/*.png")
  dset_list = []
  for img in imgs:
    filename_split = os.path.splitext(img)
    filename_zero, fileext = filename_split
    basename = os.path.basename(filename_zero)
    dset_list.append(basename)

  x_filenames = []
  y_filenames = []
  for img_id in dset_list:
    x_filenames.append(os.path.join(imdir, "{}.png".format(img_id)))
    y_filenames.append(os.path.join(lbldir, "{}.png".format(img_id)))

  print("number of images: ", len(dset_list))
  return dset_list, x_filenames, y_filenames
```

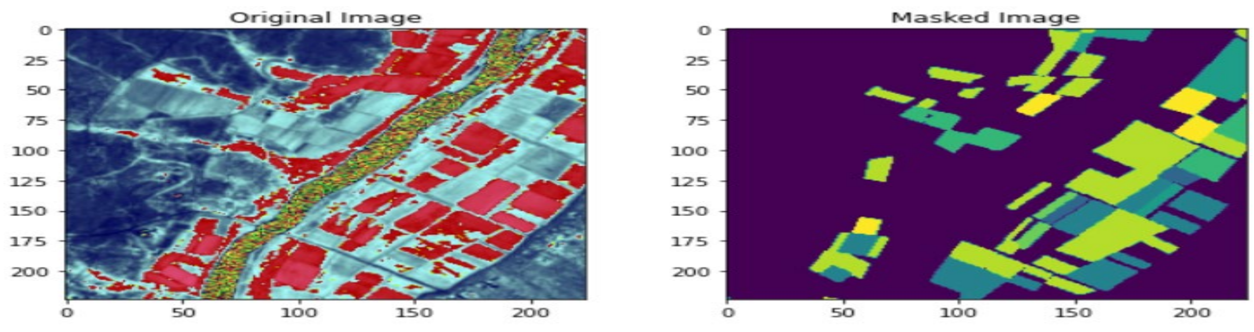
Visualizing the data

Visualizing the data to check if the original and mask images are loaded successfully



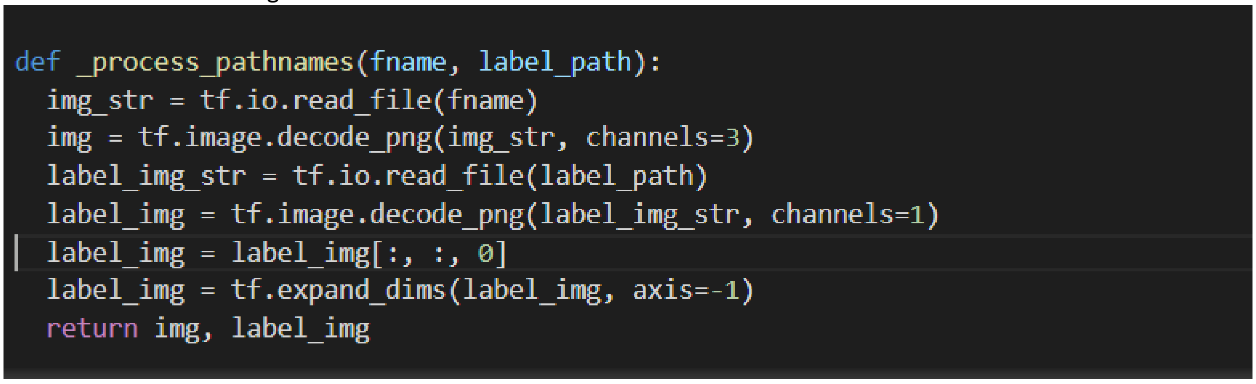
```
display_num = 3

foreground_list_x = []
foreground_list_y = []
for x,y in zip(x_train_filenames, y_train_filenames):
    img = np.array(Image.open(y))
    if img.max()>0:
        foreground_list_x.append(x)
        foreground_list_y.append(y)

num_foreground_examples = len(foreground_list_y)

r_choices = np.random.choice(num_foreground_examples, display_num)

plt.figure(figsize=(10, 15))
for i in range(0, display_num * 2, 2):
  img_num = r_choices[i // 2]
  x_pathname = foreground_list_x[img_num]
  y_pathname = foreground_list_y[img_num]

  plt.subplot(display_num, 2, i + 1)
  plt.imshow(mpimg.imread(x_pathname))
  plt.title("Original Image")

  example_labels = Image.open(y_pathname)
  label_vals = np.unique(np.array(example_labels))

  plt.subplot(display_num, 2, i + 2)
  plt.imshow(example_labels)
```

Output images



As we can see the images are loaded correctly and the masked images are also showing correctly

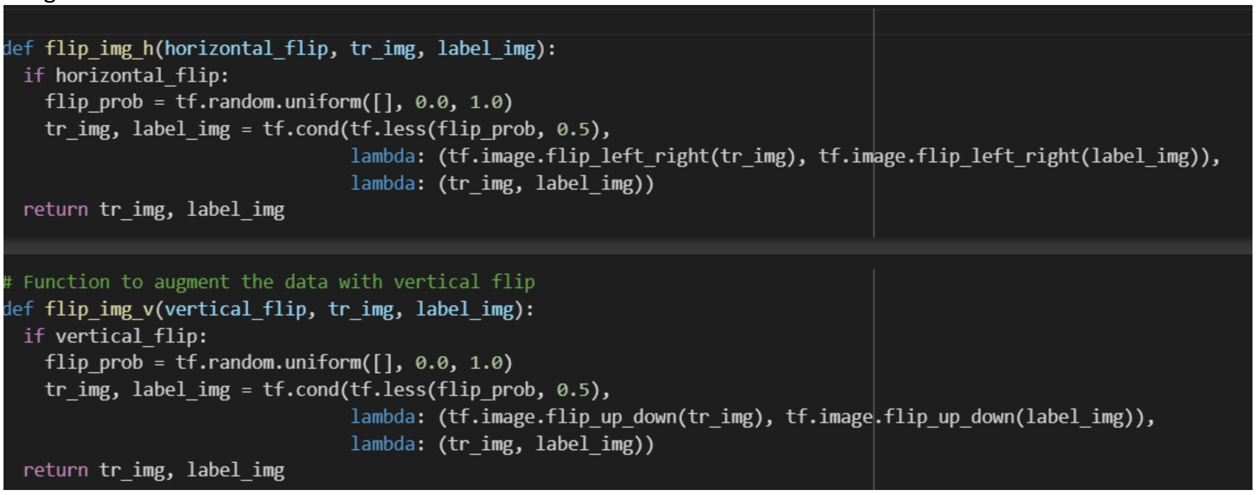
The next function is designed to read the tiles in tensor of tenserflow



```
def _process_pathnames(fname, label_path):
  img_str = tf.io.read_file(fname)
  img = tf.image.decode_png(img_str, channels=3)
  label_img_str = tf.io.read_file(label_path)
  label_img = tf.image.decode_png(label_img_str, channels=1)
  label_img = label_img[:, :, 0]
  label_img = tf.expand_dims(label_img, axis=-1)
  return img, label_img
```

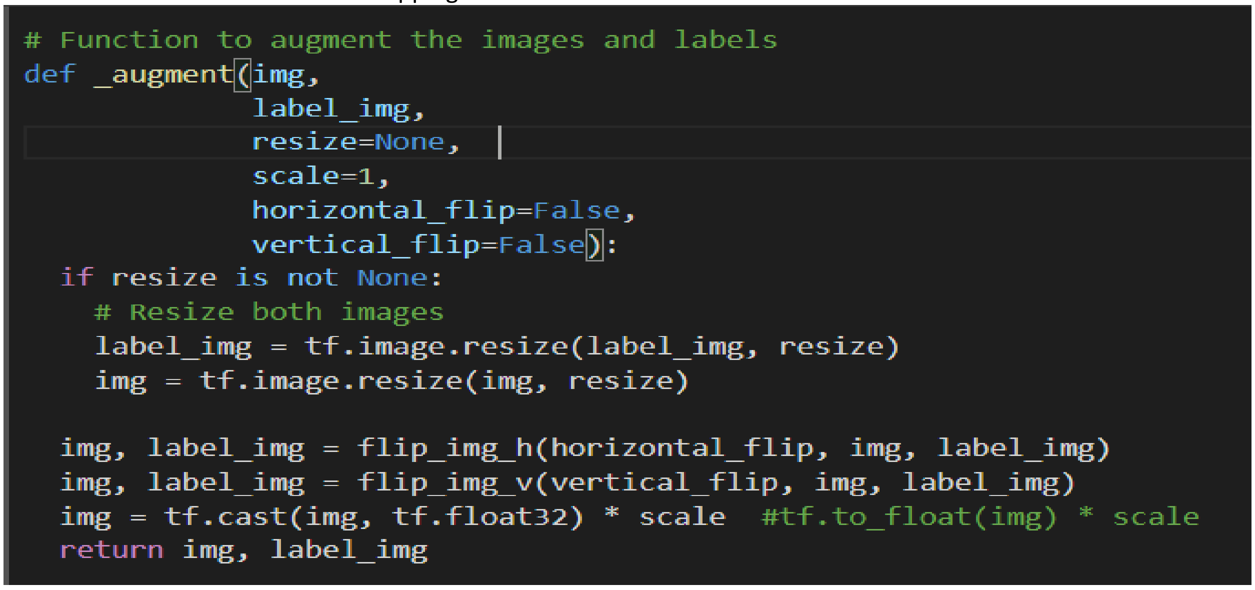
Data augmentation

Data augmentation is performed in the next few function so that we can increase the variance in the datasets images.



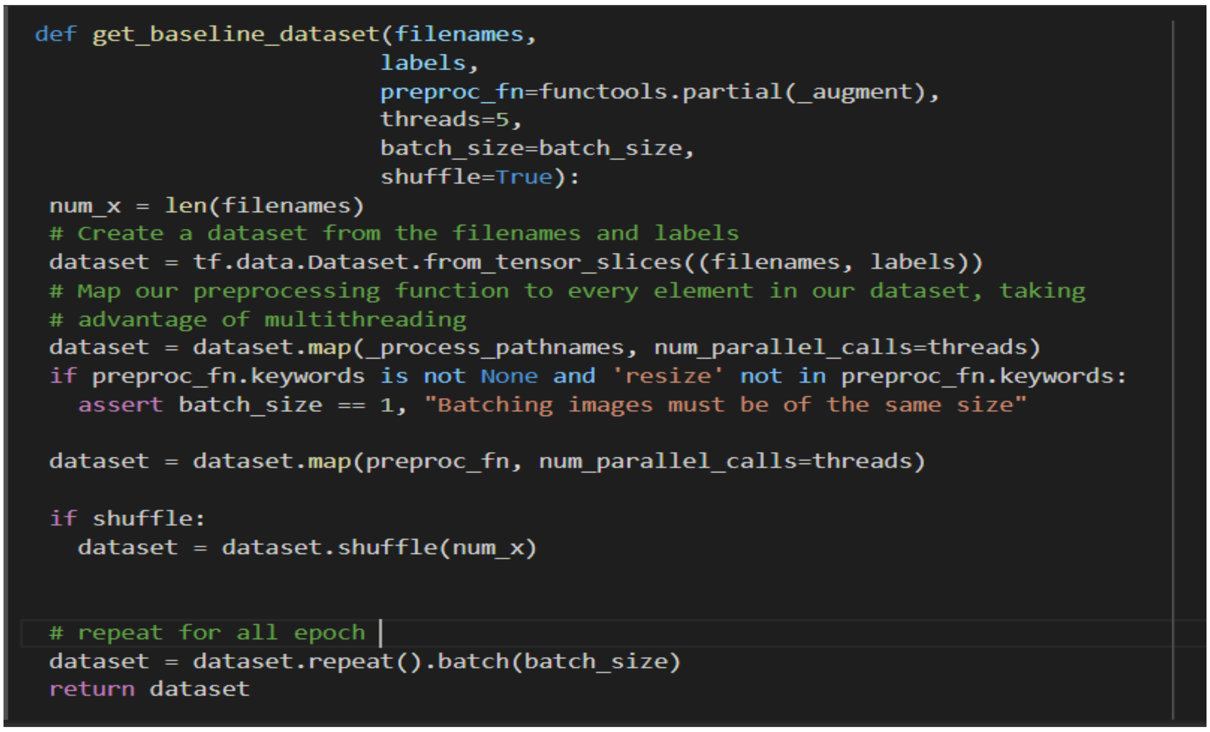
```
def flip_img_h(horizontal_flip, tr_img, label_img):
  if horizontal_flip:
    flip_prob = tf.random.uniform([], 0.0, 1.0)
    tr_img, label_img = tf.cond(tf.less(flip_prob, 0.5),
                                lambda: (tf.image.flip_left_right(tr_img), tf.image.flip_left_right(label_img)),
                                lambda: (tr_img, label_img))
  return tr_img, label_img

# Function to augment the data with vertical flip
def flip_img_v(vertical_flip, tr_img, label_img):
  if vertical_flip:
    flip_prob = tf.random.uniform([], 0.0, 1.0)
    tr_img, label_img = tf.cond(tf.less(flip_prob, 0.5),
                                lambda: (tf.image.flip_up_down(tr_img), tf.image.flip_up_down(label_img)),
                                lambda: (tr_img, label_img))
  return tr_img, label_img
```

Other than vertical and horizontal flipping the data is also scaled and resized . The next function does that.



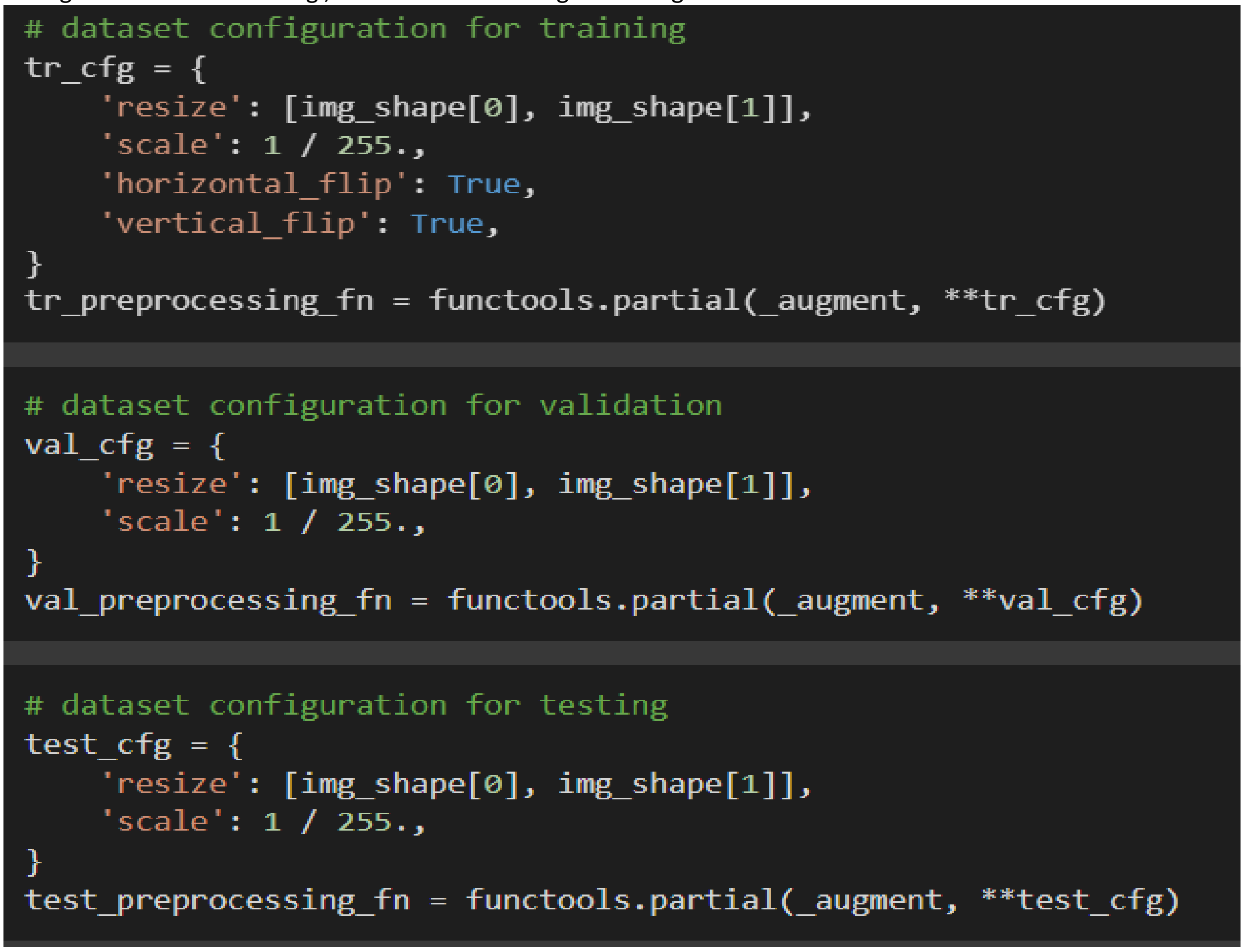
```
# Function to augment the images and labels
def _augment(img,
             label_img,
             resize=None,
             scale=1,
             horizontal_flip=False,
             vertical_flip=False):
  if resize is not None:
    # Resize both images
    label_img = tf.image.resize(label_img, resize)
    img = tf.image.resize(img, resize)

  img, label_img = flip_img_h(horizontal_flip, img, label_img)
  img, label_img = flip_img_v(vertical_flip, img, label_img)
  img = tf.cast(img, tf.float32) * scale  #tf.to_float(img) * scale
  return img, label_img
```

The next function helps to tie all the dataset processing function together



```
def get_baseline_dataset(filenames,
                         labels,
                         preproc_fn=functools.partial(_augment),
                         threads=5,
                         batch_size=batch_size,
                         shuffle=True):
  num_x = len(filenames)
  # Create a dataset from the filenames and labels
  dataset = tf.data.Dataset.from_tensor_slices((filenames, labels))
  # Map our preprocessing function to every element in our dataset, taking
  # advantage of multithreading
  dataset = dataset.map(_process_pathnames, num_parallel_calls=threads)
  if preproc_fn.keywords is not None and 'resize' not in preproc_fn.keywords:
    assert batch_size == 1, "Batching images must be of the same size"

  dataset = dataset.map(preproc_fn, num_parallel_calls=threads)

  if shuffle:
    dataset = dataset.shuffle(num_x)


  # repeat for all epoch
  dataset = dataset.repeat().batch(batch_size)
  return dataset
```

Configuration for the training , validation and testing of the augmented dataset is set below



```
# dataset configuration for training
tr_cfg = {
    'resize': [img_shape[0], img_shape[1]],
    'scale': 1 / 255.,
    'horizontal_flip': True,
    'vertical_flip': True,
}
tr_preprocessing_fn = functools.partial(_augment, **tr_cfg)
```

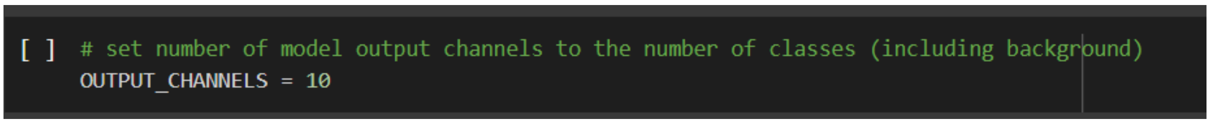
```
# dataset configuration for validation
val_cfg = {
    'resize': [img_shape[0], img_shape[1]],
    'scale': 1 / 255.,
}
val_preprocessing_fn = functools.partial(_augment, **val_cfg)
```

```
# dataset configuration for testing
test_cfg = {
    'resize': [img_shape[0], img_shape[1]],
    'scale': 1 / 255.,
}
test_preprocessing_fn = functools.partial(_augment, **test_cfg)
```

Defining the model

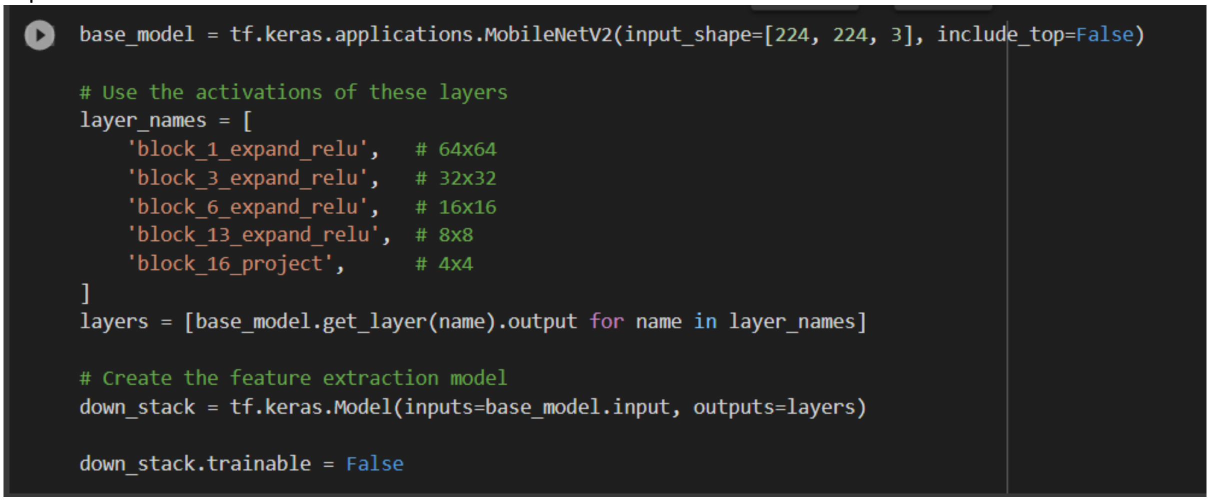
In this architecture, pretrained Mobilenetv2 is used as encoder instead of using encoder from the Unet rest of the architecture remains the same as Unet. Figure 8 shows a graphical depiction of the suggested model. The upsampling and downsampling parts are connected via a horizontal connector.

Because there are ten possible labels for each pixel, ten channels are delivered. Consider this multiple class classification, where every pixel is divided into ten categories.



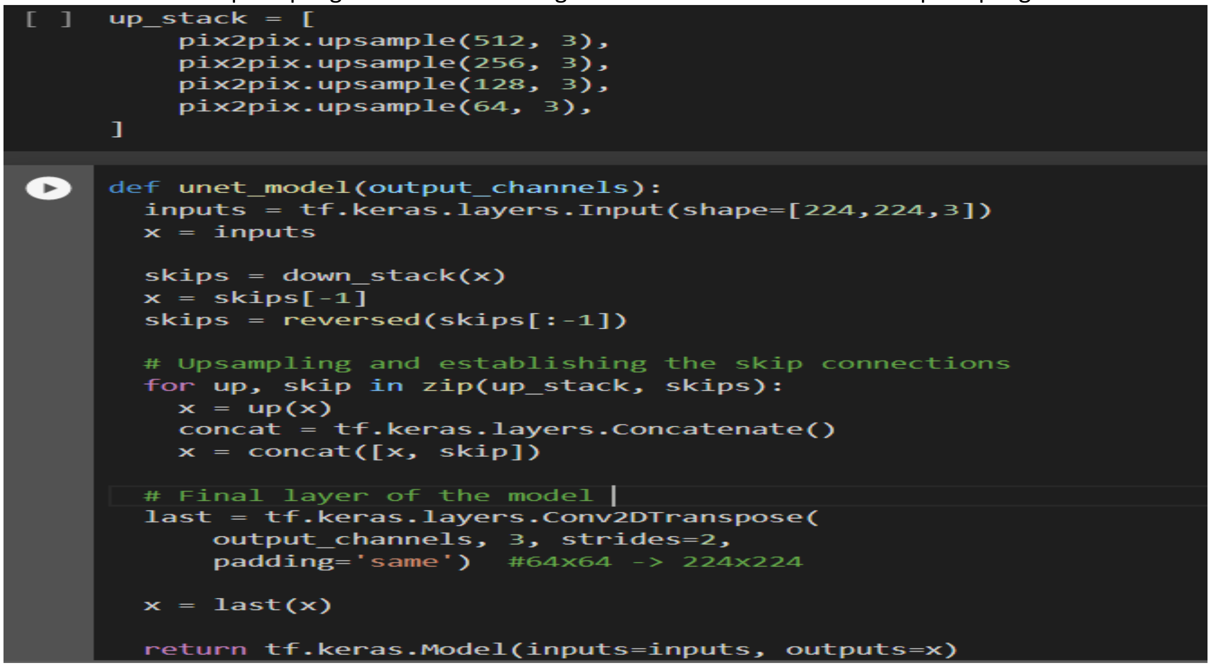
```
# set number of model output channels to the number of classes (including background)
OUTPUT_CHANNELS = 10
```

As previously stated, the downsampling section is a MobileNetV2 model that has been pre-trained and is alredy avaliable in the keras package tf.keras.applications. The encoder is made up of specified responses from the model's intermediate layers. During initial stage of training the encoder is not trained as a part of experiment



```
base_model = tf.keras.applications.MobileNetV2(input_shape=[224, 224, 3], include_top=False)

# Use the activations of these layers
layer_names = [
    'block_1_expand_relu',   # 64x64
    'block_3_expand_relu',   # 32x32
    'block_6_expand_relu',   # 16x16
    'block_13_expand_relu',  # 8x8
    'block_16_project',      # 4x4
]
layers = [base_model.get_layer(name).output for name in layer_names]

# Create the feature extraction model
down_stack = tf.keras.Model(inputs=base_model.input, outputs=layers)

down_stack.trainable = False
```

The encoder or the upsampling section is built using tensorflow which are series of upsampling blocks

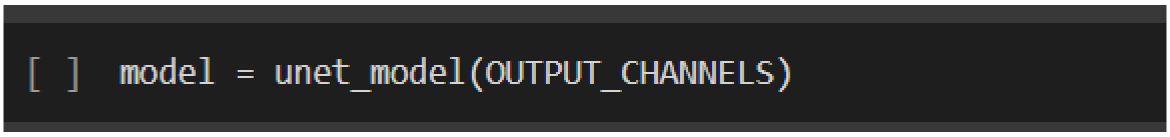


```
up_stack = [
    pix2pix.upsample(512, 3),
    pix2pix.upsample(256, 3),
    pix2pix.upsample(128, 3),
    pix2pix.upsample(64, 3),
]
```

```
def unet_model(output_channels):
  inputs = tf.keras.layers.Input(shape=[224,224,3])
  x = inputs

  skips = down_stack(x)
  x = skips[-1]
  skips = reversed(skips[:-1])

  # Upsampling and establishing the skip connections
  for up, skip in zip(up_stack, skips):
    x = up(x)
    concat = tf.keras.layers.Concatenate()
    x = concat([x, skip])

  # Final layer of the model
  last = tf.keras.layers.Conv2DTranspose(
      output_channels, 3, strides=2,
      padding='same')  #64x64 -> 224x224

  x = last(x)

  return tf.keras.Model(inputs=inputs, outputs=x)
```

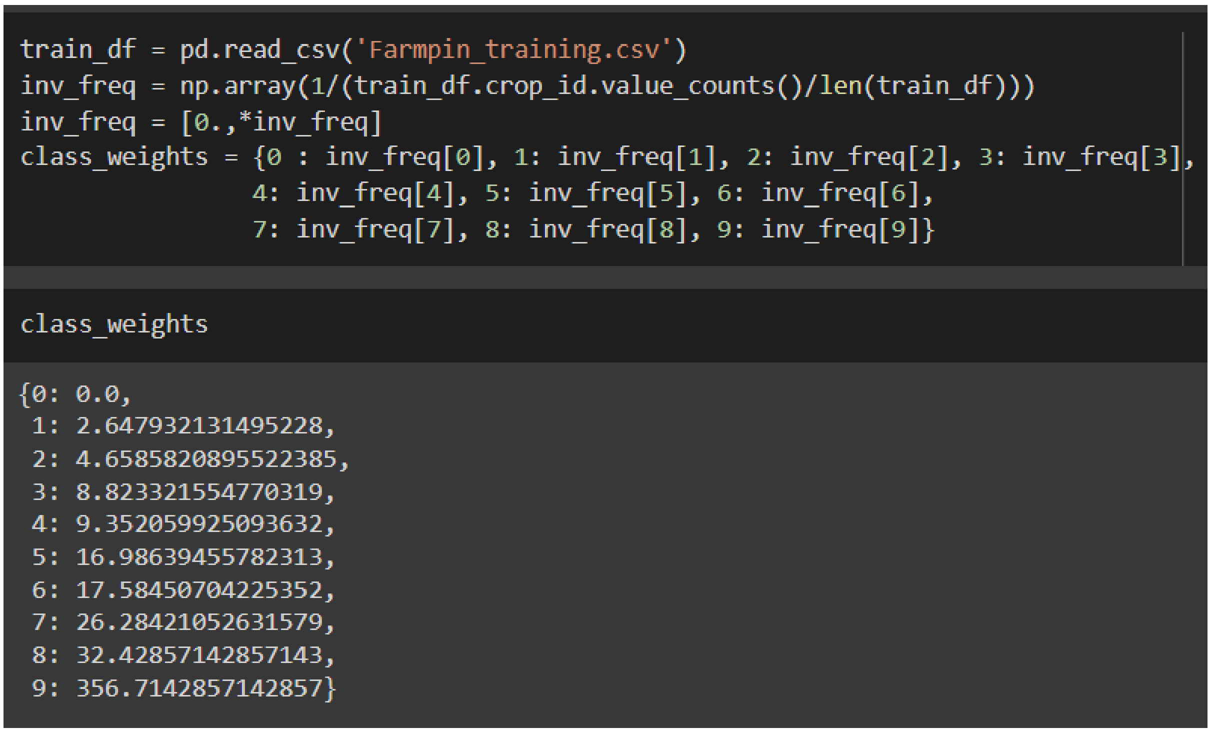
Traning the model without Data Augmentation

 During training of the model the choice of loss function is focal loss. The reason of choosing this function as the loss function is explained in the main report. The output of the network consists of 10 channels. The model is trying to predict 10 classes and hence we have kept 10 channels



```
[ ]  model = unet_model(OUTPUT_CHANNELS)
```

Checking for class imblance

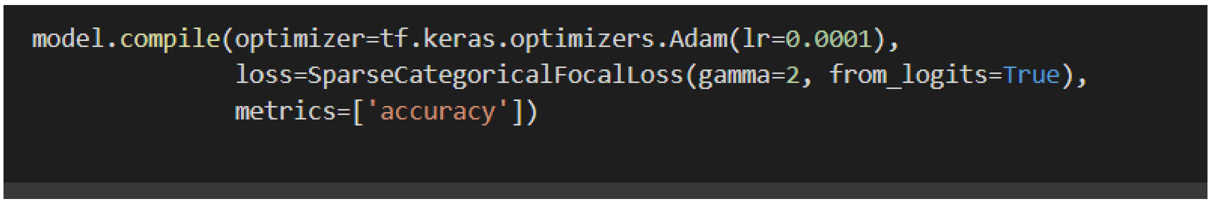


```
train_df = pd.read_csv('Farmpin_training.csv')
inv_freq = np.array(1/(train_df.crop_id.value_counts()/len(train_df)))
inv_freq = [0.,*inv_freq]
class_weights = {0 : inv_freq[0], 1: inv_freq[1], 2: inv_freq[2], 3: inv_freq[3],
                4: inv_freq[4], 5: inv_freq[5], 6: inv_freq[6],
                7: inv_freq[7], 8: inv_freq[8], 9: inv_freq[9]}
```

```
class_weights
```

```
{0: 0.0,
 1: 2.647932131495228,
 2: 4.6585820895522385,
 3: 8.823321554770319,
 4: 9.352059925093632,
 5: 16.98639455782313,
 6: 17.58450704225352,
 7: 26.28421052631579,
 8: 32.42857142857143,
 9: 356.7142857142857}
```

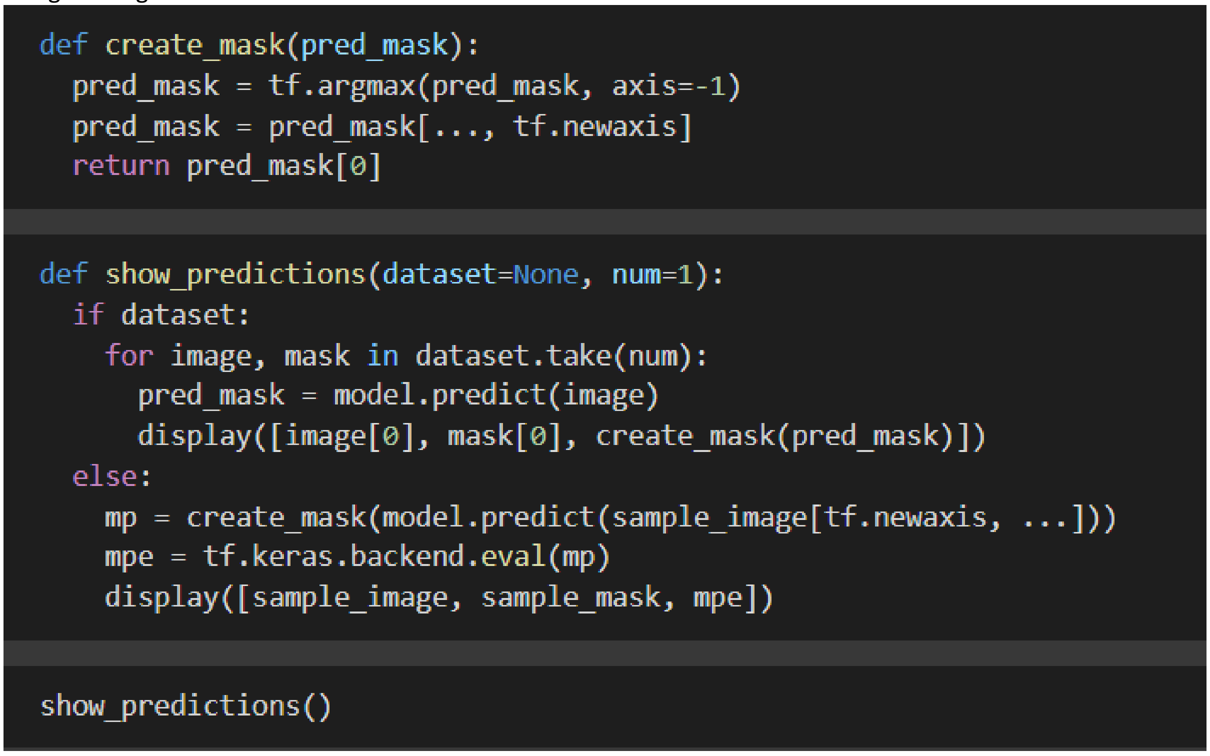
Per-pixel accuracy will be used to understance the performance of the model during traning of the model. Focal loss is implemented and we have also used Adam optimiser so that we don’t have to face the issue of vanising gradiant desent



```
model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.0001),
              loss=SparseCategoricalFocalLoss(gamma=2, from_logits=True),
              metrics=['accuracy'])
```

Making Prediction

Now we will try and use the model with out traning the model with the crop dataset and we will also not be using the augmented data

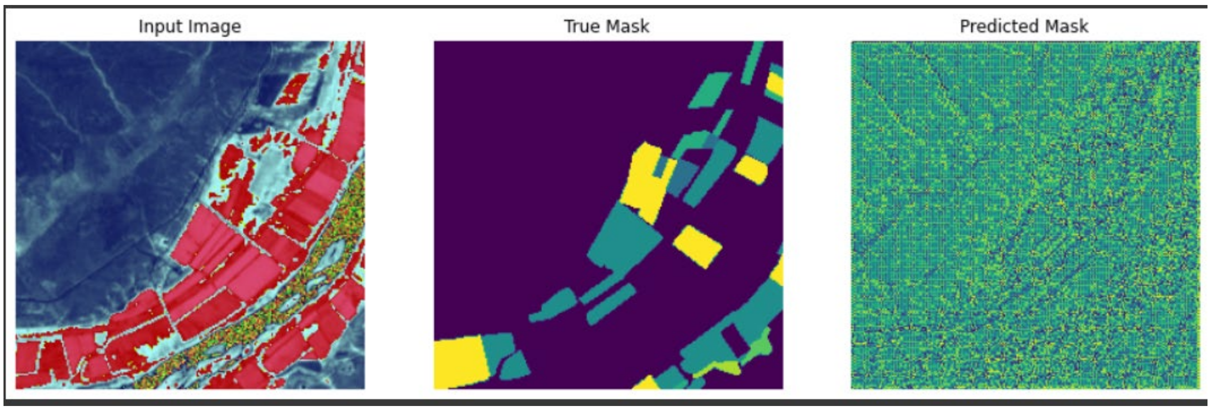


```
def create_mask(pred_mask):
  pred_mask = tf.argmax(pred_mask, axis=-1)
  pred_mask = pred_mask[..., tf.newaxis]
  return pred_mask[0]
```

```
def show_predictions(dataset=None, num=1):
  if dataset:
    for image, mask in dataset.take(num):
      pred_mask = model.predict(image)
      display([image[0], mask[0], create_mask(pred_mask)])
  else:
    mp = create_mask(model.predict(sample_image[tf.newaxis, ...]))
    mpe = tf.keras.backend.eval(mp)
    display([sample_image, sample_mask, mpe])
```

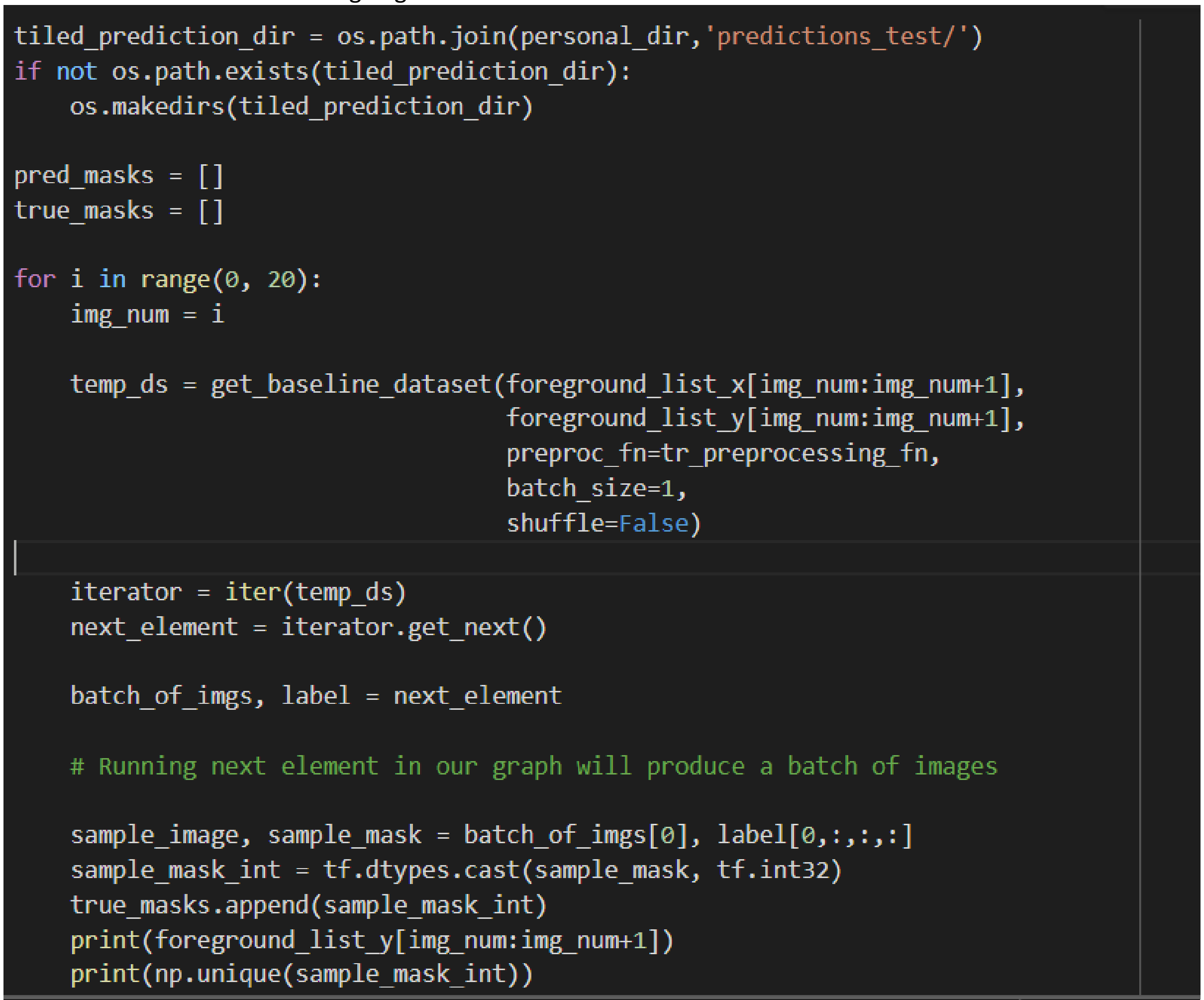
```
show_predictions()
```

Output of the model

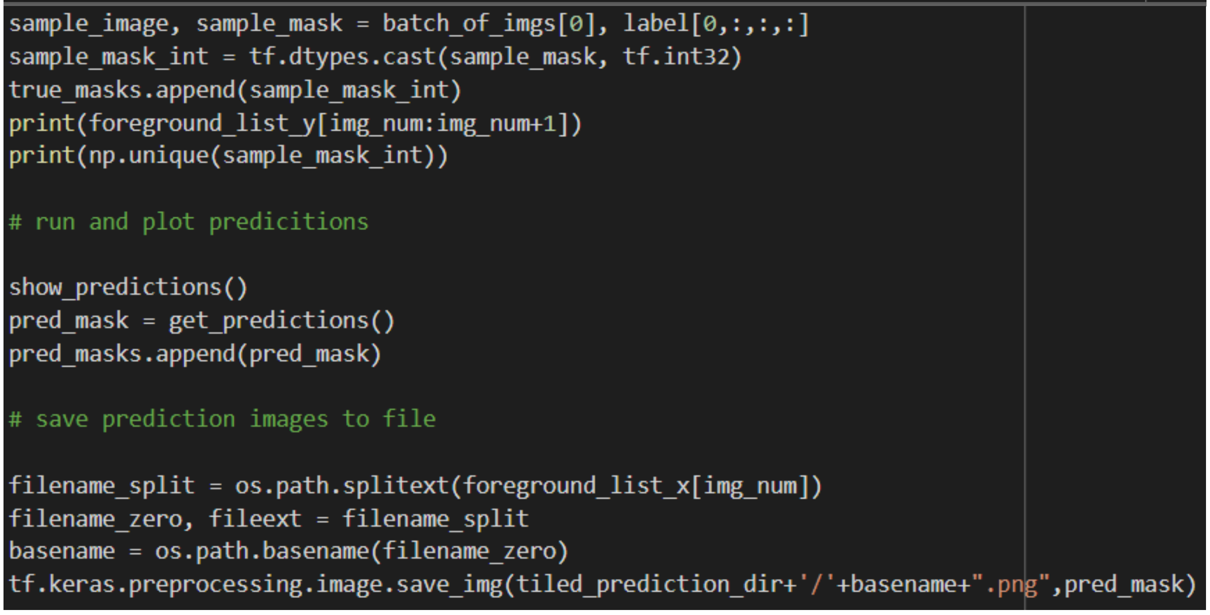


Training Model using Augmented data

As we can see that the predicted Mask has a lot of noise and the accuracy and F 1score is also quite low , now we will train the model and using augmented data.



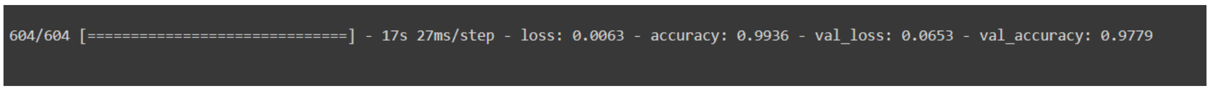
```python
tiled_prediction_dir = os.path.join(personal_dir,'predictions_test/')
if not os.path.exists(tiled_prediction_dir):
    os.makedirs(tiled_prediction_dir)

pred_masks = []
true_masks = []

for i in range(0, 20):
    img_num = i

    temp_ds = get_baseline_dataset(foreground_list_x[img_num:img_num+1],
                                   foreground_list_y[img_num:img_num+1],
                                   preproc_fn=tr_preprocessing_fn,
                                   batch_size=1,
                                   shuffle=False)

    iterator = iter(temp_ds)
    next_element = iterator.get_next()

    batch_of_imgs, label = next_element

    # Running next element in our graph will produce a batch of images

    sample_image, sample_mask = batch_of_imgs[0], label[0,:,:,:]
    sample_mask_int = tf.dtypes.cast(sample_mask, tf.int32)
    true_masks.append(sample_mask_int)
    print(foreground_list_y[img_num:img_num+1])
    print(np.unique(sample_mask_int))
```



```python
sample_image, sample_mask = batch_of_imgs[0], label[0,:,:,:]
sample_mask_int = tf.dtypes.cast(sample_mask, tf.int32)
true_masks.append(sample_mask_int)
print(foreground_list_y[img_num:img_num+1])
print(np.unique(sample_mask_int))
```

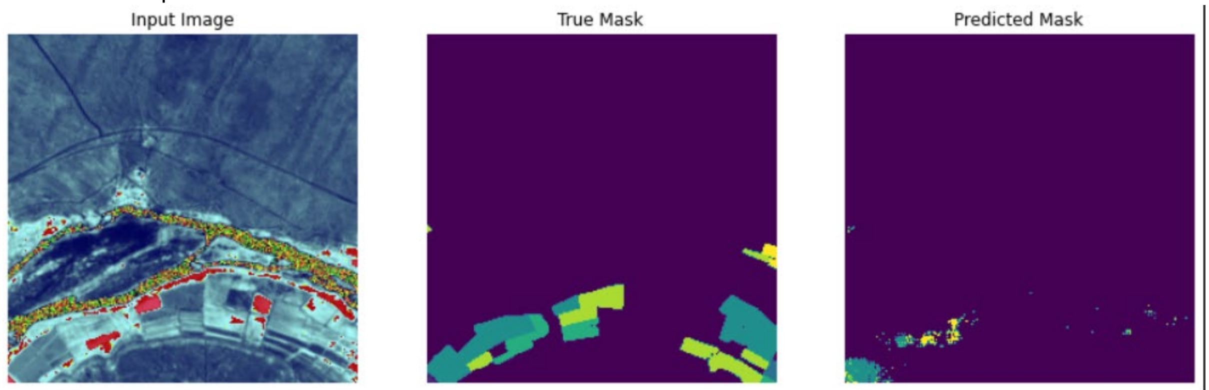
```python
# run and plot predicitions

show_predictions()
pred_mask = get_predictions()
pred_masks.append(pred_mask)

# save prediction images to file

filename_split = os.path.splitext(foreground_list_x[img_num])
filename_zero, fileext = filename_split
basename = os.path.basename(filename_zero)
tf.keras.preprocessing.image.save_img(tiled_prediction_dir+'/'+basename+".png",pred_mask)
```

Predction after 100 epoch

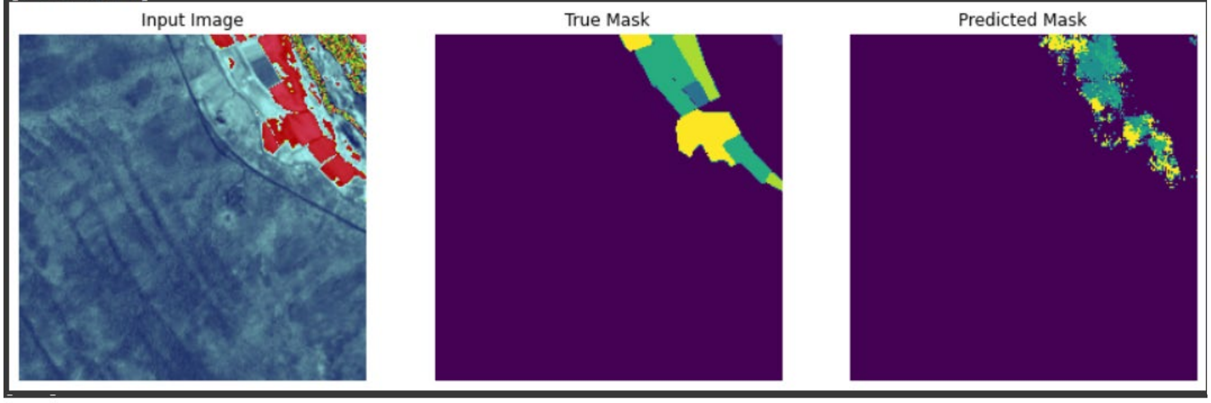


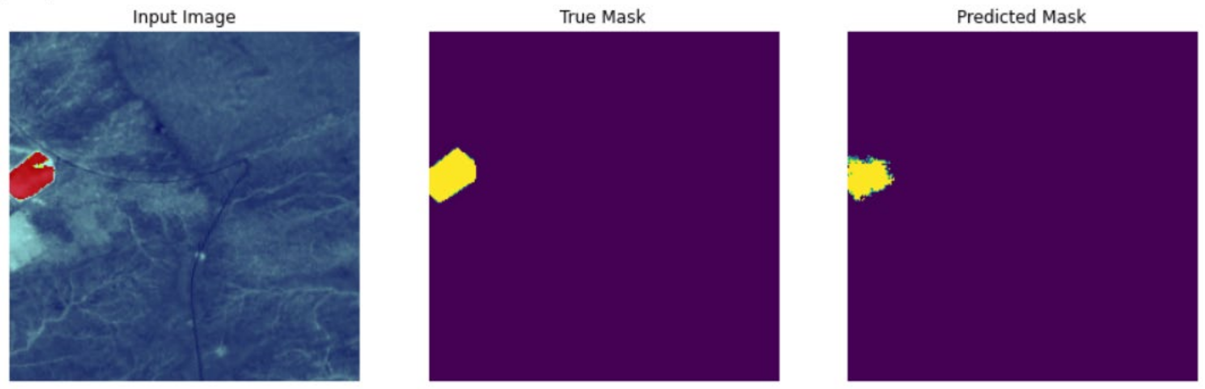
```
604/604 [==============================] - 17s 27ms/step - loss: 0.0063 - accuracy: 0.9936 - val_loss: 0.0653 - val_accuracy: 0.9779
```

Few of the output results are shown below



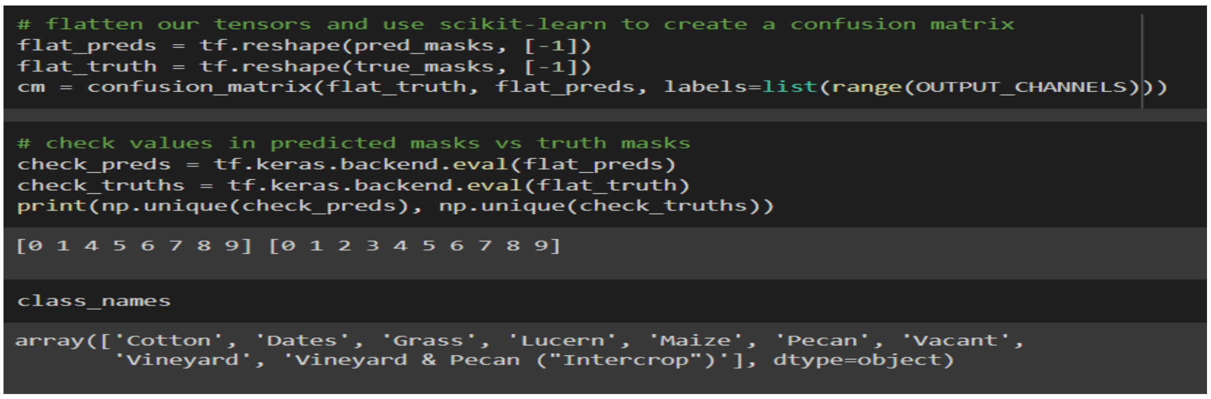
```
['./tiled/labels_test/tile_2017-07-10_2390.png']
[0 1 3 5 7 8]
```





Evaluation

Next we have performed the evaluation of the model using the below funtion . In this we can calculated the F 1 score and created a normalized confution matrix.



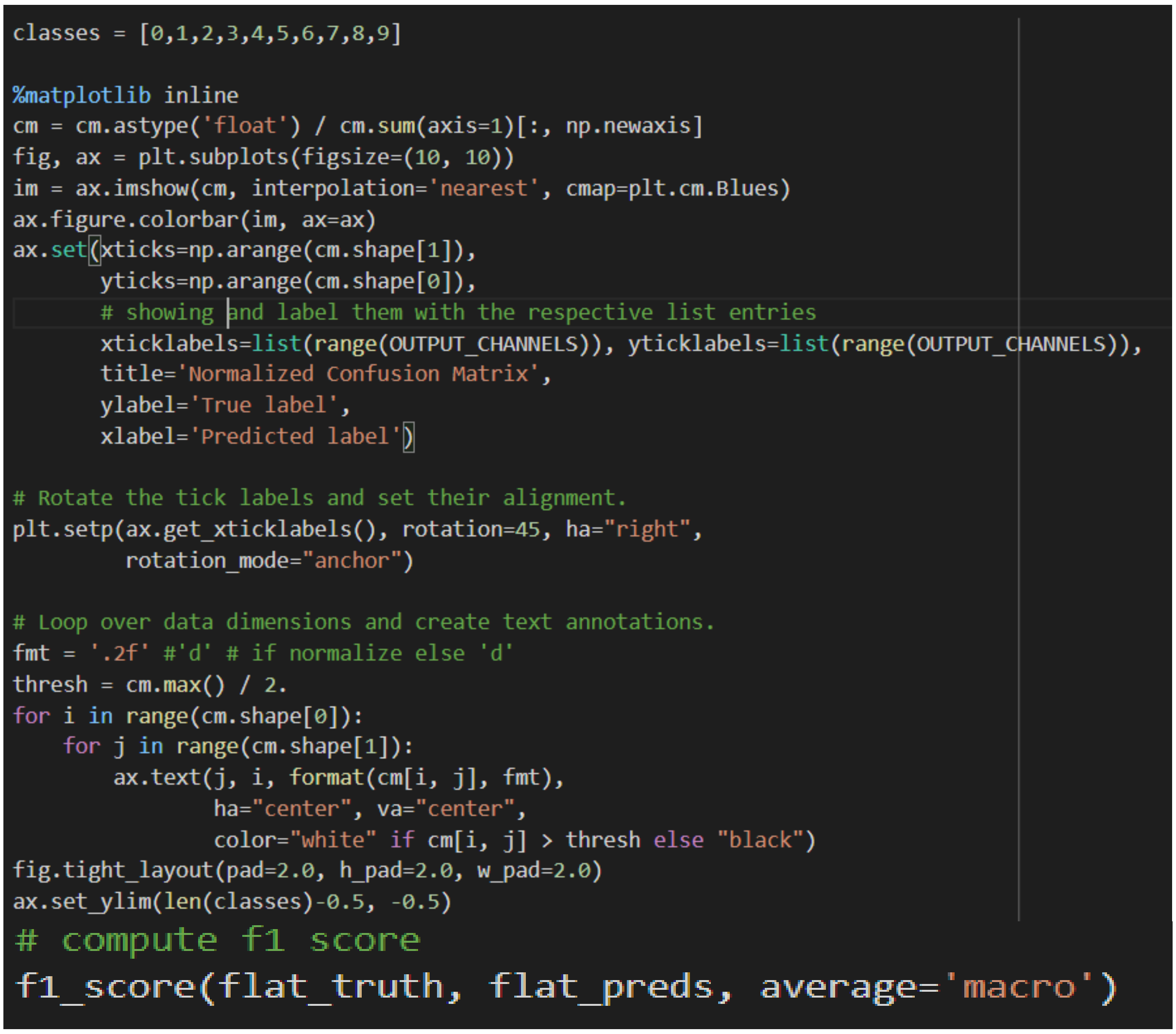
```
# flatten our tensors and use scikit-learn to create a confusion matrix
flat_preds = tf.reshape(pred_masks, [-1])
flat_truth = tf.reshape(true_masks, [-1])
cm = confusion_matrix(flat_truth, flat_preds, labels=list(range(OUTPUT_CHANNELS)))
```

```
# check values in predicted masks vs truth masks
check_preds = tf.keras.backend.eval(flat_preds)
check_truths = tf.keras.backend.eval(flat_truth)
print(np.unique(check_preds), np.unique(check_truths))
```

```
[0 1 4 5 6 7 8 9] [0 1 2 3 4 5 6 7 8 9]
```

```
class_names
```

```
array(['Cotton', 'Dates', 'Grass', 'Lucern', 'Maize', 'Pecan', 'Vacant',
       'Vineyard', 'Vineyard & Pecan ("Intercrop")'], dtype=object)
```



```
classes = [0,1,2,3,4,5,6,7,8,9]

%matplotlib inline
cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
fig, ax = plt.subplots(figsize=(10, 10))
im = ax.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
ax.figure.colorbar(im, ax=ax)
ax.set(xticks=np.arange(cm.shape[1]),
       yticks=np.arange(cm.shape[0]),
       # showing and label them with the respective list entries
       xticklabels=list(range(OUTPUT_CHANNELS)), yticklabels=list(range(OUTPUT_CHANNELS)),
       title='Normalized Confusion Matrix',
       ylabel='True label',
       xlabel='Predicted label')

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
         rotation_mode="anchor")

# Loop over data dimensions and create text annotations.
fmt = '.2f' #'d' # if normalize else 'd'
thresh = cm.max() / 2.
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        ax.text(j, i, format(cm[i, j], fmt),
                ha="center", va="center",
                color="white" if cm[i, j] > thresh else "black")
fig.tight_layout(pad=2.0, h_pad=2.0, w_pad=2.0)
ax.set_ylim(len(classes)-0.5, -0.5)
# compute f1 score
f1_score(flat_truth, flat_preds, average='macro')
```