# Prepare Notebook for Image Classification

* In the collateral directory from the github project copy the extracted contents of the VisualRecognition.zip to the notebooks directory
* In a browser open the Juypiter app, this is likely at <http://localhost:8888/>
* Enter the notebook directory
* Click new “Python 3” to create a new notebook and save it as “ClassificationModel”
* Add the following imports

import tensorflow as tf

from tensorflow import keras

import numpy as np

import matplotlib.pyplot as plt

from os import walk

# Get the Files to Train With

* Add in the following to walk the directory and get the names of all the files in the training and scoring directory

IMAGE\_SIZE = 96

trainingFilesPath = 'VisualRecognitionImages/training/'

scoringFilesPath = 'VisualRecognitionImages/test/'

train\_images = []

scoring\_images = []

train\_labels = [];

scoring\_labels = [];

val\_data = [];

def get\_model\_classes(targetPath):

for (dirpath, classes, filenames) in walk(targetPath):

return classes

def get\_file\_names(targetPath):

for (dirpath, classes, filenames) in walk(targetPath):

return [fi for fi in filenames if fi.endswith(".jpg")]

# Read in and Reformat Data

* To train our model it needs numeric classes instead of strings. We will add a method to allow for that

def get\_class\_index(className):

return {

'Maple Tree': 1,

'Palm Tree': 2,

'Podocarpus Tree': 3,

'Unknown': 4

}.get(className, 4)

* We will also create a function to read in the data

def load\_image(imagePath):

image\_string = tf.io.read\_file(imagePath)

image\_decoded = tf.image.decode\_jpeg(image\_string)

image\_normalized = (tf.cast(image\_decoded, tf.float32)/127.5) - 1

image\_resized = tf.image.resize(image\_normalized, (IMAGE\_SIZE, IMAGE\_SIZE))

return image\_resized.numpy()

The images are reformatted to a float32 for the pixel information and the images are resized to a common resolution and aspect ratio

* Read in the contents of the files for the training data

classes = get\_model\_classes(trainingFilesPath)

for trainingClass in classes:

files = get\_file\_names(trainingFilesPath + trainingClass)

for fileName in files:

train\_labels.append(get\_class\_index(trainingClass))

train\_images.append(load\_image(trainingFilesPath + trainingClass + "/" + fileName))

This is reading the data into two a arrays, one for the labels and one for the images

* Also read in the scoring data

classes = get\_model\_classes(scoringFilesPath)

for scoringClass in classes:

files = get\_file\_names(scoringFilesPath + scoringClass)

for fileName in files:

scoring\_labels.append(get\_class\_index(scoringClass))

scoring\_images.append(load\_image(scoringFilesPath + scoringClass + "/" + fileName))

val\_data.append([get\_class\_index(scoringClass), load\_image(scoringFilesPath + scoringClass + "/" + fileName)])

# View an Image

* We can view some of the images that we have read in and pre-processed for training. Command will show us the image in the training data at position 30.

plt.figure()

plt.imshow(train\_images[30])

plt.colorbar()

plt.grid(False)

plt.show()

# Create the Model

Create our model and the training layers. First, we flatten the input data and then run it though two other processes,

Note: In input shape should only be specified on the first layer

model = keras.Sequential([

keras.layers.Flatten(input\_shape=(IMAGE\_SIZE, IMAGE\_SIZE, 3)),

keras.layers.Dense(512, activation=tf.nn.relu),

keras.layers.Dense(10, activation='sigmoid')

])

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

model.summary()

The Summary() method of the model gives information on the layers we setup for training

# Train the model

* Enter in the following information to train the model..

epochs = 300

train\_images = np.array(train\_images)

train\_labels = np.array(train\_labels)

scoring\_images = np.array(scoring\_images)

scoring\_labels = np.array(scoring\_labels)

history = model.fit(train\_images, train\_labels, epochs=epochs, validation\_data=(scoring\_images, scoring\_labels))

The number of epochs is the number of times we run through the data

# View the Training Results

* We can see the accuracy of the model by showing a graph of the training history. #Save the model

acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy']

loss = history.history['loss']

val\_loss = history.history['val\_loss']

plt.figure(figsize=(8, 8))

plt.subplot(2, 1, 1)

plt.plot(acc, label='Training Accuracy')

plt.plot(val\_acc, label='Validation Accuracy')

plt.legend(loc='lower right')

plt.ylabel('Accuracy')

plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)

plt.plot(loss, label='Training Loss')

plt.plot(val\_loss, label='Validation Loss')

plt.legend(loc='upper right')

plt.ylabel('Cross Entropy')

plt.title('Training and Validation Loss')

plt.xlabel('epoch')

plt.show()