TASK: PROJECT DOCUMENTATION

**PROJECT TITLE**

IMAGE AUGMENTATION IN LAND USE LAND COVER USING ROTATION TECHNIQUE

**ANDREW KIBIWOTT TU01-IC211-0383/2017**

**import numpy as np** *<<library for mathematical computation for image array>>*

**import matplotlib.pyplot as plt** <<library for interactive visualization>>

**import os** *<< module for retrieving the DATADIR>>*

**import cv2** *<< library for solving computer vision problems; cv2.IMREAD\_GRAYSCALE>>*

**DATADIR= 'C:\\Users\\HP\\Desktop\\Datasets\\rivers** *<<location path for our file directory>>*

**CATEGORIES = ["original", "rotated"]** *<< object categories of our dataset>>*

**for category in CATEGORIES:**  *<< object is the category, CATEGORIES is the class>>*

**path = os.path.join(DATADIR, category)** *<<directory path>>*

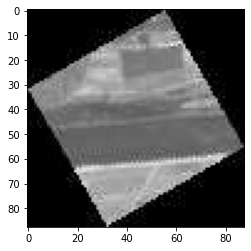
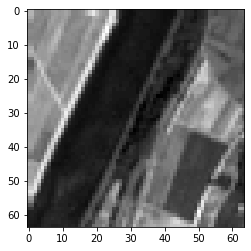
**for img in os.listdir(path):*<<****for loop ,iterate through the datasets>>*

**img\_array = cv2.imread(os.path.join(path,img), cv2.IMREAD\_GRAYSCALE)** <<a string to represent the path of image to be read>>

**plt.imshow(img\_array, cmap="gray") plt.show()** <<graph the image and display the output>>

**break** *<<bringing the program control out of the loop; original>>*

**break** *<< bringing the program control out of the loop; rotated>>*

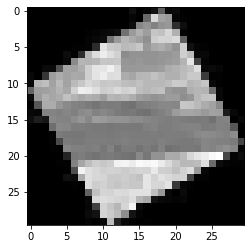


**print (img\_array.shape)** *<< displaying the arrays of the image>>*

**IMG\_SIZE =30** <<image size>>

**new\_array = cv2.resize(img\_array, (IMG\_SIZE, IMG\_SIZE))** *<<resizing the image array to IMG\_SIZE; 30>>*

**plt.imshow(new\_array,cmap='gray') plt.show()** *<<visualizes the grayscale image>>*



**training data = []**

**def create\_training\_data():** *<< defining a function to create training data>>*

**for category in CATEGORIES:** *<< object is the category, CATEGORIES is the class>>*

**path = os.path.join(DATADIR, category)**

**class\_num =CATEGORIES.index(category)**

**for img in os.listdir(path):**

**try:** *<<tests for error in a block of codes>>*

**img\_array = cv2.imread(os.path.join(path,img), cv2.IMREAD\_GRAYSCALE) )** *<<a string to represent the path of image to be read>>*

**new\_array = cv2.resize(img\_array, (IMG\_SIZE, IMG\_SIZE))** << resizing the image array to IMG\_SIZE; 30>>

**training\_data.append([new\_array, class\_num])** *<< adding a single item in the training data list>>*

**except Exception as e:** *<<handle errors detected in a block of code>>*

**pass** *<<null statement avoiding getting an error in the loop>>*

**create\_training\_data()** <<calling the function for create\_training data>>

**import random**

**random.shuffle(training data)** *<<shuffling the images in a nice way>>*

**for sample in training\_data[:10]:** *<<iterates through the sampled images >>*

**print (sample[1])** *<<prints the output>>*

**X=[]** *<<Define X as original image>>*

**y=[]** *<<Define y as the rotated image>>*

**for features, label in training\_data: *<<****a for loop, for getting features and labels in the training\_data>>*

**X.append(features)** *<< function which has parameter of features>>*

**y.append(label)** *<< function of y, rotated image of parameter label >>*

**X = np.array(X). reshape(-1, IMG\_SIZE, IMG\_SIZE, 1)**

**import pickle** *<< module for importing library for converting objects into bytes >>*

**pickle\_out= open("X.pickle","wb")** *<< function with parameter of X.pickles to write bytes>>*

**pickle.dump(X, pickle\_out)** *<< function for putting images in X open file >>*

**pickle\_out.close()** *<< closing the X file >>*

**pickle\_out= open("y.pickle","wb")** *<< function with parameter of y.pickles to write bytes>>*

**pickle.dump(y, pickle\_out)** *<< function for putting images in open y file >>*

**pickle\_out.close()** *<< closing the y file >>*

**pickle\_in = open("X.pickle", "rb") ")** *<< function with parameter of X.pickles to read bytes>>*

**X = pickle.load(pickle\_in)** *<< function to load the pickle file in VarChar>>*

**import tensorflow as tf**

**from tensorflow.keras.models import Sequential**

**from tensorflow.keras.layers import Dense,Dropout,Activation,Flatten,Conv2D,MaxPooling2D**

**from tensorflow.keras.callbacks import TensorBoard**

*#dense layer perform vector matrix multiplication*

*#flatten it convert 3d inputs into 1d*

*#dropout it help to train the model better*

*#activation increases non linearity in the output*

*#tensorboard helps in saving the model*

*#optimizer calculate weight updates*

*#conv2d creates a convolutional kernel that is is convolved with layer input to produce a tensors of output*

**import pickle**

**import numpy as np**

**import time**

**tensorboard = TensorBoard(log\_dir="logs/{}".format(NAME))**

**pickle\_in = open("X.pickle", "rb")** *<< function with parameter of X.pickles to read bytes>>*

**X= pickle.load(pickle\_in)** *<< function to load the pickle file in VarChar>>*

**pickle\_in = open("y.pickle", "rb")** *<< function with parameter of y.pickles to read bytes>>*

**y= pickle.load(pickle\_in)** *<< function to load the pickle file in VarChar>>*

**X = X/255.0** *<<divide by maximum number of pixel in an image>>*

**dense\_layers = [0]** *<< dense layer perform vector matrix multiplication>>*

**layer\_sizes = [64]**

**conv\_layers = [3]**

**for dense\_layer in dense\_layers:** << looping through dense layers>>

**for layer\_size in layer\_sizes:** << looping through layer\_size>>

**for conv\_layer in conv\_layers**: << looping through conv\_layers>>

**NAME = "{}-conv-{}-nodes-{}-dense-{}".format(conv\_layer, layer\_size, dense\_layer, int(time.time()))**

**print(NAME)**

**model = Sequential( )** *<<build a model layer by layer in orderly manner>>*

**model.add(Conv2D(layer\_size,(3,3), input\_shape = X.shape[1:] ))** *<< creates a convolutional kernel that is convolved with layer input to produce a tensors of output >>*

**model.add(Activation("relu"))** << activation increases non linearity in the output>>

**model.add(MaxPooling2D(pool\_size=(2,2)))** *<<takes maximum value by pool size 2>>*

**for l in range(conv\_layer-1):**

**model.add(Conv2D(layer\_size,(3,3) ))**

**model.add(Activation("relu"))**

**model.add(MaxPooling2D(pool\_size=(2,2)))**

**model.add(Flatten())** *<<convert 3d inputs into 1d>>*

**for \_ in range(dense\_layer):**

**model.add(Dense(layer\_size))**

**model.add(Activation('relu'))**

**model.add(Dense(1))**

**model.add(Activation('sigmoid'))**

**model.compile(loss='binary\_crossentropy', optimizer='adam', metrics= ['accuracy'],)**

**X = np.asarray(X)**

**y = np.asarray(y)**

**model.save('64 x 3- CNN.model')**

**model.fit(X, y, batch\_size=32, epochs=3, validation\_split =0.1, callbacks=[tensorboard] )**