* Selecting Datasets name
* Collecting image datasets
* Data cleaning

Code : # Import libraries

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow import keras

from keras.layers import \*

from keras.models import \*

from keras.preprocessing import image

from keras.preprocessing.image import ImageDataGenerator

import os, shutil

import warnings

warnings.filterwarnings('ignore')

# Let's plot a few images

train\_path = "/content/drive/MyDrive/Datasets/Train"

validation\_path = "/content/drive/MyDrive/Datasets/Validation"

test\_path = "/content/drive/MyDrive/Datasets/Test"

image\_categories = os.listdir('/content/drive/MyDrive/Datasets/Train')

import keras.utils as image

def plot\_images(image\_categories):

    # Create a figure

    plt.figure(figsize=(12, 12))

    for i, cat in enumerate(image\_categories):

        # Load images for the ith category

        image\_path = train\_path + '/' + cat

        images\_in\_folder = os.listdir(image\_path)

        first\_image\_of\_folder = images\_in\_folder[0]

        first\_image\_path = image\_path + '/' + first\_image\_of\_folder

        img = image.load\_img(first\_image\_path)

        img\_arr = image.img\_to\_array(img)/255.0

        # Create Subplot and plot the images

        plt.subplot(4, 4, i+1)

        plt.imshow(img\_arr)

        plt.title(cat)

        plt.axis('off')

    plt.show()

# Call the function

plot\_images(image\_categories)

# Creating Image Data Generator for train, validation and test set

# 1. Train Set

train\_gen = ImageDataGenerator(rescale = 1.0/255.0) # Normalise the data

train\_image\_generator = train\_gen.flow\_from\_directory(

                                            train\_path,

                                            target\_size=(150, 150),

                                            batch\_size=32,

                                            class\_mode='binary')

# 2. Validation Set

val\_gen = ImageDataGenerator(rescale = 1.0/255.0) # Normalise the data

val\_image\_generator = train\_gen.flow\_from\_directory(

                                            validation\_path,

                                            target\_size=(150, 150),

                                            batch\_size=32,

                                            class\_mode='binary')

# 3. Test Set

test\_gen = ImageDataGenerator(rescale = 1.0/255.0) # Normalise the data

test\_image\_generator = train\_gen.flow\_from\_directory(

                                            test\_path,

                                            target\_size=(150, 150),

                                            batch\_size=32,

                                            class\_mode='binary')

# Print the class encodings done by the generators

class\_map = dict([(v, k) for k, v in train\_image\_generator.class\_indices.items()])

print(class\_map)

# Build a custom sequential CNN model

model = Sequential() # model object

# Add Layers

model.add(Conv2D(filters=32, kernel\_size=3, strides=1, padding='same', activation='relu', input\_shape=[150, 150, 3]))

model.add(MaxPooling2D(2, ))

model.add(Conv2D(filters=64, kernel\_size=3, strides=1, padding='same', activation='relu'))

model.add(MaxPooling2D(2))

# Flatten the feature map

model.add(Flatten())

# Add the fully connected layers

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.25))

model.add(Dense(128, activation='relu'))

model.add(Dense(1, activation='sigmoid'))

# print the model summary

model.summary()

# Compile and fit the model

early\_stopping = keras.callbacks.EarlyStopping(patience=5) # Set up callbacks

model.compile(optimizer='Adam', loss='binary\_crossentropy', metrics='accuracy')

hist = model.fit(train\_image\_generator,

                 epochs=100,

                 verbose=1,

                 validation\_data=val\_image\_generator,

                 callbacks=early\_stopping)

# Plot the error and accuracy

h = hist.history

plt.style.use('ggplot')

plt.figure(figsize=(10, 5))

plt.plot(h['loss'], c='red', label='Training Loss')

plt.plot(h['val\_loss'], c='red', linestyle='--', label='Validation Loss')

plt.plot(h['accuracy'], c='blue', label='Training Accuracy')

plt.plot(h['val\_accuracy'], c='blue', linestyle='--', label='Validation Accuracy')

plt.xlabel("Number of Epochs")

plt.legend(loc='best')

plt.show()

# Testing the Model

test\_image\_path = '/content/drive/MyDrive/Datasets/Test/Boats/images39.jpg'

def generate\_predictions(test\_image\_path, actual\_label):

    # 1. Load and preprocess the image

    test\_img = image.load\_img(test\_image\_path, target\_size=(150, 150))

    test\_img\_arr = image.img\_to\_array(test\_img)/255.0

    test\_img\_input = test\_img\_arr.reshape((1, test\_img\_arr.shape[0], test\_img\_arr.shape[1], test\_img\_arr.shape[2]))

    # 2. Make Predictions

    predicted\_label = np.argmax(model.predict(test\_img\_input))

    predicted\_vegetable = class\_map[predicted\_label]

    plt.figure(figsize=(4, 4))

    plt.imshow(test\_img\_arr)

    plt.title("Predicted Label: {}, Actual Label: {}".format(predicted\_vegetable, actual\_label))

    plt.grid()

    plt.axis('off')

    plt.show()

# call the function

generate\_predictions(test\_image\_path, actual\_label='Boats')

* Dataset uploaded in drive for easy retrival
* from google.colab import drive
* drive.mount('/content/drive')