**DATASET**

To perform this experiment we have collected images of four different breed of sheeps from different sources.

**MODULES TO BE USED**

1. Os- It offers portable way of using operating system dependent functionality.
2. Numpy- We can perform a wide variety of mathematical operations on arrays by using this.
3. Matplotlib- We can create good quality of plots and figures using this.
4. Tensorflow- It is used for building and training machine learning models.
5. Pandas- We can analyze, clean, explore, and manipulate data from datasets using this.
6. PIL- We can open, manipulate and save different kind of image formats using this.
7. Sklearn – This library mostly used for machine learning in python.

from linux terminal go to the python3 environment

**python3**

Python 3.10.9 | packaged by conda-forge | (main, Feb 2 2023, 20:20:04) [GCC 11.3.0] on linux

Type "help", "copyright", "credits" or "license" for more information.

#import modules to be used

**>>> import os**

**>>> import numpy as np**

**>>> import matplotlib.pyplot as plt**

**>>> from sklearn.model\_selection import train\_test\_split**

**>>> from sklearn.metrics import accuracy\_score**

**>>> import pandas as pd**

**>>> import tensorflow as tf**

**>>> import PIL**

**>>> import PIL.Image**

**>>> from tensorflow import keras**

**>>> import tensorflow\_datasets as tfds**

**>>> import matplotlib.pyplot as plt**

#train a model to classify among images of 4 breeds of sheep: Himalayan, Marino, Telengana, Odisha

#defines the model

**>>> model = tf.keras.models.Sequential([**

**... tf.keras.layers.Conv2D(16, (3,3), activation='relu', input\_shape=(150, 150, 3)),** #1st convolutional layer

**... tf.keras.layers.MaxPooling2D(2, 2),**

**... keras.layers.Dropout(rate=0.15),** #adding dropout regularization throughout the model to deal with overfitting

**... tf.keras.layers.Conv2D(32, (3,3), activation='relu'),** #2nd convolutional layer

**... tf.keras.layers.MaxPooling2D(2,2),**

**... keras.layers.Dropout(rate=0.1),**

**... tf.keras.layers.Conv2D(64, (3,3), activation='relu'),** #3rd convolutional layer

**... tf.keras.layers.MaxPooling2D(2,2),**

**... keras.layers.Dropout(rate=0.10),**

**... tf.keras.layers.Flatten(),** ##flatten the results to fit in a DNN

**... tf.keras.layers.Dense(512, activation='relu'),** ##512 neuron hidden layer

**... tf.keras.layers.Dense(4, activation='softmax')])**

#4 output neuron for the 4 classes of sheep image and softmax activation function used for multiclass classification

#the summary of the model

**>>> model.summary()**

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d (Conv2D) (None, 148, 148, 16) 448

max\_pooling2d (MaxPooling2D (None, 74, 74, 16) 0

)

dropout (Dropout) (None, 74, 74, 16) 0

conv2d\_1 (Conv2D) (None, 72, 72, 32) 4640

max\_pooling2d\_1 (MaxPooling (None, 36, 36, 32) 0

2D)

dropout\_1 (Dropout) (None, 36, 36, 32) 0

conv2d\_2 (Conv2D) (None, 34, 34, 64) 18496

max\_pooling2d\_2 (MaxPooling (None, 17, 17, 64) 0

2D)

dropout\_2 (Dropout) (None, 17, 17, 64) 0

flatten (Flatten) (None, 18496) 0

dense (Dense) (None, 512) 9470464

dense\_1 (Dense) (None, 4) 2052

=================================================================

Total params: 9,496,100

Trainable params: 9,496,100

Non-trainable params: 0

**>>> from tensorflow.keras.optimizers import RMSprop**

#compile the model

**>>> model.compile(loss='categorical\_crossentropy',**

**... optimizer="adam",**

**... metrics=['acc'])**

**>>> from tensorflow.keras.preprocessing.image import ImageDataGenerator**

#splits data into train and test set in a ratio of 80:20

**>>> train\_datagen =ImageDataGenerator(rescale=1./255, validation\_split=0.2)**

**>>> import matplotlib.pyplot as plt**

#loading the training data

**>>> train\_generator = train\_datagen.flow\_from\_directory('/home/….path', target\_size=(150, 150),batch\_size=15,class\_mode='categorical',subset = 'training')**

Found 1137 images belonging to 4 classes.

#set the epoch according to your dataset

**>>> epochs = 15**

#loading the test or validation data

**>>> validation\_generator = train\_datagen.flow\_from\_directory('/home/……path', target\_size=(150, 150),batch\_size=15,class\_mode='categorical',subset = 'validation')**

Found 283 images belonging to 4 classes.

**>>> history = model.fit\_generator(train\_generator,steps\_per\_epoch=75,epochs=epochs,validation\_data = validation\_generator,validation\_steps = 15,verbose=1)** #model fitting for a number of epoch

Epoch 1/15

75/75 [==============================] - 11s 136ms/step - loss: 1.1370 - acc: 0.5339 - val\_loss: 0.6113 - val\_acc: 0.7022

Epoch 2/15

75/75 [==============================] - 9s 118ms/step - loss: 0.5157 - acc: 0.7861 - val\_loss: 0.4559 - val\_acc: 0.7822

Epoch 3/15

75/75 [==============================] - 9s 119ms/step - loss: 0.3102 - acc: 0.8761 - val\_loss: 0.4609 - val\_acc: 0.7867

Epoch 4/15

75/75 [==============================] - 10s 130ms/step - loss: 0.2537 - acc: 0.8939 - val\_loss: 0.4295 - val\_acc: 0.8356

Epoch 5/15

75/75 [==============================] - 9s 123ms/step - loss: 0.1914 - acc: 0.9323 - val\_loss: 0.5081 - val\_acc: 0.7733

Epoch 6/15

75/75 [==============================] - 9s 120ms/step - loss: 0.1338 - acc: 0.9510 - val\_loss: 0.5360 - val\_acc: 0.8000

Epoch 7/15

75/75 [==============================] - 9s 122ms/step - loss: 0.1283 - acc: 0.9537 - val\_loss: 0.4813 - val\_acc: 0.8222

Epoch 8/15

75/75 [==============================] - 9s 114ms/step - loss: 0.0895 - acc: 0.9635 - val\_loss: 0.3141 - val\_acc: 0.8933

Epoch 9/15

75/75 [==============================] - 9s 116ms/step - loss: 0.1014 - acc: 0.9688 - val\_loss: 0.4526 - val\_acc: 0.8356

Epoch 10/15

75/75 [==============================] - 9s 121ms/step - loss: 0.0742 - acc: 0.9724 - val\_loss: 0.5160 - val\_acc: 0.8089

Epoch 11/15

75/75 [==============================] - 9s 124ms/step - loss: 0.0504 - acc: 0.9768 - val\_loss: 0.8205 - val\_acc: 0.8089

Epoch 12/15

75/75 [==============================] - 9s 119ms/step - loss: 0.0789 - acc: 0.9742 - val\_loss: 0.3231 - val\_acc: 0.8622

Epoch 13/15

75/75 [==============================] - 9s 119ms/step - loss: 0.0302 - acc: 0.9893 - val\_loss: 0.8252 - val\_acc: 0.7778

Epoch 14/15

75/75 [==============================] - 9s 122ms/step - loss: 0.0422 - acc: 0.9857 - val\_loss: 0.6717 - val\_acc: 0.8489

Epoch 15/15

75/75 [==============================] - 9s 117ms/step - loss: 0.0726 - acc: 0.9768 - val\_loss: 0.5916 - val\_acc: 0.8267

#print training and testing accuracy

**>>> print("Training Accuracy:"), print(history.history['acc'][-1])**

Training Accuracy:

0.976827085018158

(None, None)

**>>> print("Testing Accuracy:"), print (history.history['val\_acc'][-1])**

Testing Accuracy:

0.8266666531562805

(None, None)

#plot graph between training & validation accuracy and training & validation loss

**>>> acc = history.history['acc']**

**>>> val\_acc = history.history['val\_acc']**

**>>> loss = history.history['loss']**

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

**>>> epochs\_range = range(epochs)**

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

<Figure size 800x800 with 0 Axes>

**>>> plt.subplot(1, 2, 1)**

<Axes: >

**>>> plt.plot(epochs\_range, acc, label='Training Accuracy')**

[<matplotlib.lines.Line2D object at 0x7f019c394cd0>]

**>>> plt.plot(epochs\_range, val\_acc, label='Validation Accuracy')**

[<matplotlib.lines.Line2D object at 0x7f019c394f10>]

**>>> plt.legend(loc='lower right')**

<matplotlib.legend.Legend object at 0x7f019c4fd570>

**>>> plt.title('Training and Validation Accuracy')**

Text(0.5, 1.0, 'Training and Validation Accuracy')

**>>> plt.subplot(1, 2, 2)**

<Axes: >

**>>> plt.plot(epochs\_range, loss, label='Training Loss')**

[<matplotlib.lines.Line2D object at 0x7f019c3cada0>]

**>>> plt.plot(epochs\_range, val\_loss, label='Validation Loss')**

[<matplotlib.lines.Line2D object at 0x7f019c3c9de0>]

**>>> plt.legend(loc='upper right')**

<matplotlib.legend.Legend object at 0x7f019c396140>

**>>> plt.title('Training and Validation Loss')**

Text(0.5, 1.0, 'Training and Validation Loss')

#show the graph

**>>> plt.show()**

****

#save the json file of model

**>>> model\_json = model.to\_json()**

**>>> with open("./model.json","w") as json\_file:**

**... json\_file.write(model\_json)**

4030

#save the weights of model

**>>> model.save\_weights("./model.h5")**

**>>> print("saved model..! ready to go.")**

saved model..! ready to go.

**>>> from keras.models import model\_from\_json**

**>>> import cv2**

**>>> import numpy as np**

#load the model

**>>> json\_file = open('/home/………………/model.json', 'r')**

**>>> loaded\_model\_json = json\_file.read()**

**>>> json\_file.close()**

**>>> loaded\_model = model\_from\_json(loaded\_model\_json)**

**>>> loaded\_model.load\_weights("/home/…………………./model.h5")**

**>>> print("Loaded model from disk")**

Loaded model from disk

#compile the loaded model

**>>> '''loaded\_model.compile(loss=keras.losses.categorical\_crossentropy,**

**... optimizer=keras.optimizers.Adadelta(),**

**... metrics=['accuracy'])’’’**

"loaded\_model.compile(loss=keras.losses.categorical\_crossentropy,\n optimizer=keras.optimizers.Adadelta(),\n metrics=['accuracy'])\n"

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

#load the image you want to test

**>>> img = cv2.imread('/home/shaileshlab2/sheepbreed/output/test/himalayan/1.JPG')**

**>>> print(img)**

[[[ 81 100 121]

[ 82 101 122]

[ 84 103 124]

...

[ 85 109 121]

[ 83 107 119]

[ 80 106 118]]

[[ 80 99 120]

[ 79 98 119]

[ 80 99 120]

...

[ 88 112 124]

[ 80 104 116]

[ 73 99 111]]

[[ 88 107 128]

[ 80 99 120]

[ 75 94 115]

...

[ 94 118 130]

[ 84 108 120]

[ 76 102 114]]

...

[[ 99 128 143]

[ 95 124 139]

[ 90 118 135]

...

[ 37 42 43]

[ 38 43 44]

[ 38 43 44]]

[[100 129 144]

[ 96 125 140]

[ 91 119 136]

...

[ 27 32 33]

[ 28 33 34]

[ 29 34 35]]

[[ 96 125 140]

[ 93 122 137]

[ 89 117 134]

...

[ 17 22 23]

[ 19 24 25]

[ 20 25 26]]]

#resize the image according to the parameter used in the cnn

**>>> img = cv2.resize(img, (150,150))**

**>>> print(img.shape)**

(150, 150, 3)

**>>> img = img.reshape(1, 150, 150, 3)**

**>>> print(img.shape)**

(1, 150, 150, 3)

**>>> result = loaded\_model.predict(img)**

1/1 [==============================] - 0s 192ms/step

**>>> print(result)**

[[1. 0. 0. 0.]]

#confusion matrix building

**>>> y\_prediction = model.predict(train\_generator)**

76/76 [==============================] - 7s 91ms/step

**>>> y\_prediction = np.argmax (y\_prediction, axis = 1)**

**>>> y\_test = model.predict(train\_generator)**

76/76 [==============================] - 7s 94ms/step

**>>> y\_test = np.argmax (y\_test, axis = 1)**

**>>> result = confusion\_matrix(y\_test, y\_prediction)**

**>>> print(result)**

[[ 62 99 93 56]

[ 93 103 86 54]

[ 91 78 86 50]

[ 64 56 40 26]]

[[ 19 24 22 10]

[ 22 30 25 7]

[ 24 20 33 13]

[ 10 10 10 4]]