MACHINE LEARNING II LABORATORY

ASSIGNMENT NO. : 4

NAME : Sharvari P Deshpande

CLASS : B.E. CSE IS-1

ROLL NO. : 2173182

DATE : 28/11/2020

I. AIM :

Develop Face Recognition system using CNN. Create a dataset of minimum 20 students from your class. Check and validate the accuracy of the model.

- Apply Dimensionality Reduction on input image and plot the change in accuracy of the system.

II. OBJECTIVES :

1. To learn dataset creation

2. To learn data normalization

III. THEORY :

Dataset creation steps

Whenever we begin a machine learning project, the first thing that we need is a dataset. Dataset will be the pillar of your training model. You can build the dataset either automatically or manually. Image recognition is a common aspect of Deep Learning. Datasets can be created manually for the same.

Dataset is the collection of specific data for your ML project needs. The type of data depends on the kind of AI you need to train. Basically, you have two datasets:

- Training
- Testing

Step 1 : Data Acquisition

Download the images from Google or manually import the images into a folder. Name the folder as per your convenience. Rename the images if needed.

Step 2 : Data Pre-processing

Backup your original images before pre-processing. This is a good practice in case something goes wrong. Resize the images to a standard size (for eg. 224x224). Label all the images and manually separate all the images into different folders as per the need.

Step 3 : Create the zip-file

Zip all the folders together and finally the dataset is ready to be used and imported.

Image Augmentation

The performance of deep learning neural networks often improves with the amount of data available. Data augmentation is a technique to artificially create new training data from existing training data. This is done by applying domain-specific techniques to examples from the training data that create new and different training examples.

Image data augmentation is perhaps the most well-known type of data augmentation and involves creating transformed versions of images in the training dataset that belong to the same class as the original image. Transforms include a range of operations from the field of image manipulation, such as shifts, flips, zooms, and much more. The intent is to expand the training dataset with new, plausible examples. This means, variations of the training set images that are likely to be seen by the model. For example, a horizontal flip of a picture of a cat may make sense, because the photo could have been taken from the left or right. A vertical flip of the photo of a cat does not make sense and would probably not be appropriate given that the model is very unlikely to see a photo of an upside down cat. Image data augmentation is typically only applied to the training dataset, and not to the validation or test dataset. This is different from data preparation such as image resizing and pixel scaling; they must be performed consistently across all datasets that interact with the model.

Libraries for Image Augmentation

There are a lot of Image Augmentation packages. The popular ones are :

1. skimage – sci-kit image is an open source Python package that works with numpy arrays. It implements algorithms and utilities for use in research, education and industry applications. It is a fairly simple and straightforward library even for those who are new to Python's ecosystem. This code is of high-quality and peer-reviewed, written by an active community of volunteers.

2. OpenCV – OpenCV essentially stands for Open Source Computer Vision Library. Although it is written in optimized C/C++, it has interfaces for Python and Java along with C++. OpenCV-Python is the python API for OpenCV. You can think of it as a python wrapper around the C++ implementation of OpenCV. OpenCV-Python is not only fast (since the background consists of code written in C/C++) but is also easy to code and deploy(due to the Python wrapper in foreground). This makes it a great choice to perform computationally

intensive programs. Python wrapper in foreground). This makes it a great choice to perform computationally intensive programs.

3. imgaug – imgaug is a library for image augmentation in machine learning experiments. It supports a wide range of augmentation techniques, allows to easily combine these and to execute them in random order or on multiple CPU cores, has a simple yet powerful stochastic interface and can not only augment images, but also keypoints/landmarks, bounding boxes, heatmaps and segmentation maps.

4. Albumentations – Albumentations is a fast image augmentation library and easy to use wrapper around other libraries. It is based on numpy, OpenCV, imgaug picking the best from each of them. It is written by Kagglers and was used to get top results in many DL competitions at Kaggle, topcoder, CVPR, MICCAI.

5. Keras Image Data Generator - The Keras library has a built in class created just for the purpose of adding transformations to images. This class is called ImageDataGenerator and it generates batches of tensor image data with real-time data augmentations.

6. SOLT : Streaming Over Lightweight Data Transformations – SOLT is a fast data augmentation library, supporting arbitrary amount of images, segmentation masks, keypoints and data labels. It has OpenCV in its back-end, thus it works very fast.

Fit_generator

Fit_generator() in Python is a deep learning library which can be used to train our machine learning and deep learning models.

Syntax :

```
fit_generator(object, generator, steps_per_epoch, epochs = 1,
  verbose = getOption("keras.fit_verbose", default = 1),
  callbacks = NULL, view_metrics = getOption("keras.view_metrics",
  default = "auto"), validation_data = NULL, validation_steps = NULL,
  class_weight = NULL, max_queue_size = 10, workers = 1,
  initial_epoch = 0)
```

How to use Keras fit_generator() :

```
# performing data argumentation by training image generator
dataAugmentaion = ImageDataGenerator(rotation_range = 30, zoom_range =
0.20,
fill_mode = "nearest", shear_range = 0.20, horizontal_flip = True,
width_shift_range = 0.1, height_shift_range = 0.1)
```

```
# training the model
model.fit_generator(dataAugmentaion.flow(trainX, trainY, batch_size = 32),
 validation_data = (testX, testY), steps_per_epoch = len(trainX) // 32,
 epochs = 10)
```

Data Augmentation is a method of artificially creating a new dataset for training from the existing training dataset to improve the performance of deep learning neural networks with the amount of data available. It is a form of regularization which makes our model generalize better than before.
Here we have used a Keras ImageDataGenerator object to apply data augmentation for randomly translating, resizing, rotating, etc the images. Each new batch of our data is randomly adjusting according to the parameters supplied to ImageDataGenerator**.**

Validation_generator()

The validation generator works exactly like the training generator. You define how many batches it will wield per epoch.

1. The training generator will yield steps_per_epoch batches.

2. When the epoch ends, the validation generator will yield validation_steps batches.

Predict_generator()

Generates predictions for the input samples from a data generator. The generator should return the same kind of data as accepted by predict_on_batch. It returns numpy array of predictions.

IV. DATASET USED AND ITS ATTRIBUTES :

The dataset was made by collecting 9 photos of 20 people each. Different folders were made by the name of the people. The dataset was split in such a way that, total 108 images were used for training and 72 images were used for testing. The images were pre-processed by cropping them and normalizing it later. This dataset was then used for training purpose and the model accuracy was analysed.

V. IMPLEMENTATION/PROGRAM/CODE :

```
import numpy as np
import pandas as pd
import glob
import tensorflow as tf
import matplotlib.pyplot as plt
import dlib
import cv2
from PIL import Image
import os
```

```
path = 'C:\\Users\\HP\\Desktop\\Dataset'
```

```
print("Total no. of Photos :"+str(len(glob.glob(path+"/*/*"))))
```

```
Total no. of Photos :180
```

```
categories = os.listdir(path)
print(categories)
```

```
['Abhishek', 'Aditya', 'Divya', 'Faguni', 'Karan', 'Mayank', 'Neha', 'Rahul', 'Rutuja', 'Sagarika',
```

```
os.makedirs("Faces",exist_ok=True)
```

```
faces ='C:\\Users\\HP\\Faces'
```

```
face_detector = dlib.get_frontal_face_detector()
```

```
for c in categories:
  os.makedirs(faces+f"/{c}",exist_ok=True)
  i=0
  for f in glob.glob(path+f"/{c}/*"):

    img = Image.open(f).convert('L')
    detected_face = face_detector(np.array(img),1)
    if len(detected_face)==1:
      crop_area = (detected_face[0].left(), detected_face[0].top(), detected_face[0].right(), detected_face[0].bottom())
      cropped_image = img.crop(crop_area)

      name = c+str(i)
      cropped_image.save(faces+f"/{c}/{name}.jpg")
      i=i+1
```

```
df = pd.DataFrame()
for c in categories:
    for f in glob.glob(faces+f"/{c}/*"):
        df = df.append({"filename":f,"class":c},ignore_index=True)
```

```
df
```

	class	filename
0	Abhishek	C:\Users\HP\Faces/Abhishek\Abhishek0.jpg
1	Abhishek	C:\Users\HP\Faces/Abhishek\Abhishek1.jpg
2	Abhishek	C:\Users\HP\Faces/Abhishek\Abhishek2.jpg
3	Abhishek	C:\Users\HP\Faces/Abhishek\Abhishek3.jpg
4	Abhishek	C:\Users\HP\Faces/Abhishek\Abhishek4.jpg
...	...	...
175	Unzela	C:\Users\HP\Faces/Unzela\Unzela4.jpg
176	Unzela	C:\Users\HP\Faces/Unzela\Unzela5.jpg
177	Unzela	C:\Users\HP\Faces/Unzela\Unzela6.jpg
178	Unzela	C:\Users\HP\Faces/Unzela\Unzela7.jpg
179	Unzela	C:\Users\HP\Faces/Unzela\Unzela8.jpg

180 rows × 2 columns

```
[ ] test_ratio = 0.4
    test_df = df.sample(frac=test_ratio,random_state=0)
    train_df = df.drop(test_df.index, axis=0)
```

```
[ ] train = train_df.reset_index(drop=True)
    test = test_df.reset_index(drop=True)
```

```
train_data_gen = tf.keras.preprocessing.image.ImageDataGenerator(
    rotation_range=15,
    rescale=1./255,
    zoom_range=0.2,
    )
train_gen = train_data_gen.flow_from_dataframe(dataframe=train,xcol='filename',ycol='class',target_size=(224,224),batch_size=8,class_mode='categorical')

valid_data_gen =  tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
valid_gen = valid_data_gen.flow_from_dataframe(dataframe=test,xcol='filename',ycol='class',target_size=(224,224),batch_size=8,class_mode='categorical',random=False)
```

```
Found 108 validated image filenames belonging to 20 classes.
Found 72 validated image filenames belonging to 20 classes.
```

```
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow import keras
from tensorflow.keras import Model
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.models import Sequential
```

```
model_vgg16 = VGG16(weights = 'imagenet', include_top = False)
x = model_vgg16.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation = 'relu')(x)
predictions = Dense(20, activation = 'softmax')(x)
model = Model(inputs = model_vgg16.input, outputs = predictions)

for layer in model_vgg16.layers:
    layer.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 12s 0us/step
```

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

+ Code + Text

```
history = model.fit(train_gen, epochs=50, validation_data=valid_gen)
```

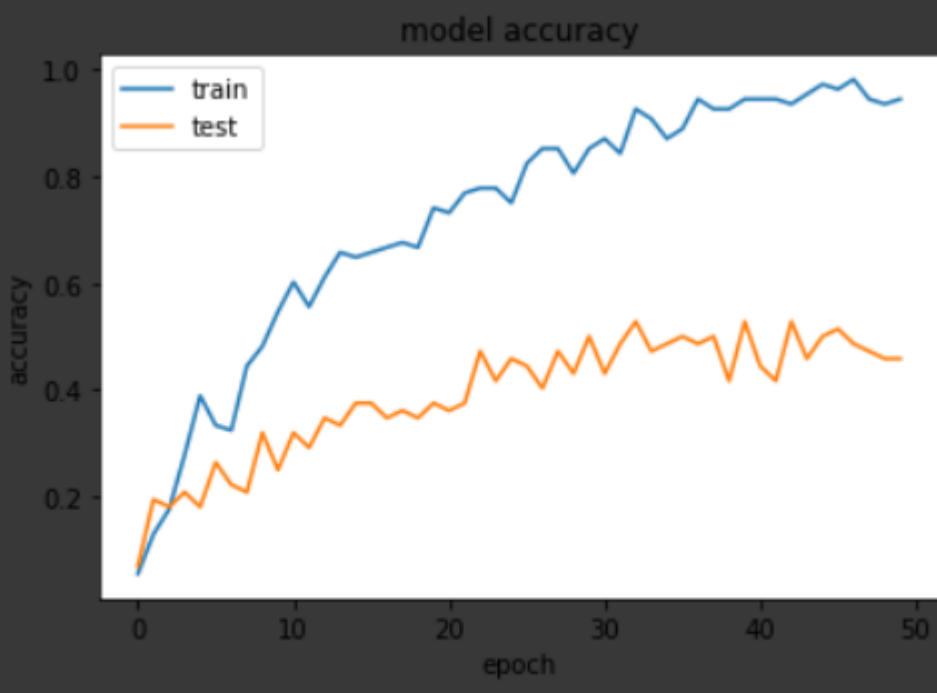
```
Epoch 1/50
14/14 [==============================] - 48s 3s/step - loss: 3.2120 - accuracy: 0.0556 - val_loss: 3.2647 - val_accuracy: 0.0694
Epoch 2/50
14/14 [==============================] - 45s 3s/step - loss: 2.9392 - accuracy: 0.1296 - val_loss: 3.1044 - val_accuracy: 0.1944
Epoch 3/50
14/14 [==============================] - 45s 3s/step - loss: 2.7869 - accuracy: 0.1759 - val_loss: 2.9804 - val_accuracy: 0.1806
Epoch 4/50
14/14 [==============================] - 45s 3s/step - loss: 2.6064 - accuracy: 0.2778 - val_loss: 2.9168 - val_accuracy: 0.2083
Epoch 5/50
14/14 [==============================] - 45s 3s/step - loss: 2.3695 - accuracy: 0.3889 - val_loss: 2.8115 - val_accuracy: 0.1806
Epoch 6/50
14/14 [==============================] - 45s 3s/step - loss: 2.3426 - accuracy: 0.3333 - val_loss: 2.7140 - val_accuracy: 0.2639
Epoch 7/50
14/14 [==============================] - 31s 2s/step - loss: 2.2327 - accuracy: 0.3241 - val_loss: 2.6811 - val_accuracy: 0.2222
Epoch 8/50
14/14 [==============================] - 32s 2s/step - loss: 2.0615 - accuracy: 0.4444 - val_loss: 2.6121 - val_accuracy: 0.2083
Epoch 9/50
14/14 [==============================] - 32s 2s/step - loss: 1.9343 - accuracy: 0.4815 - val_loss: 2.5614 - val_accuracy: 0.3194
Epoch 10/50
14/14 [==============================] - 32s 2s/step - loss: 1.8866 - accuracy: 0.5463 - val_loss: 2.5705 - val_accuracy: 0.2500
Epoch 11/50
14/14 [==============================] - 32s 2s/step - loss: 1.7516 - accuracy: 0.6019 - val_loss: 2.4713 - val_accuracy: 0.3194
Epoch 12/50
14/14 [==============================] - 34s 2s/step - loss: 1.6928 - accuracy: 0.5556 - val_loss: 2.4623 - val_accuracy: 0.2917
Epoch 13/50
14/14 [==============================] - 32s 2s/step - loss: 1.5636 - accuracy: 0.6111 - val_loss: 2.2879 - val_accuracy: 0.3472
```

```
[ ]  model.evaluate(train_gen)

     14/14 [==============================] - 15s 1s/step - loss: 0.3399 - accuracy: 0.9259
     [0.3398509919643402, 0.9259259104728699]

     score=model.evaluate(valid_gen)

     9/9 [==============================] - 11s 1s/step - loss: 2.2646 - accuracy: 0.4583
```

VI. RESULTS/VISUALIZATION GRAPHS :

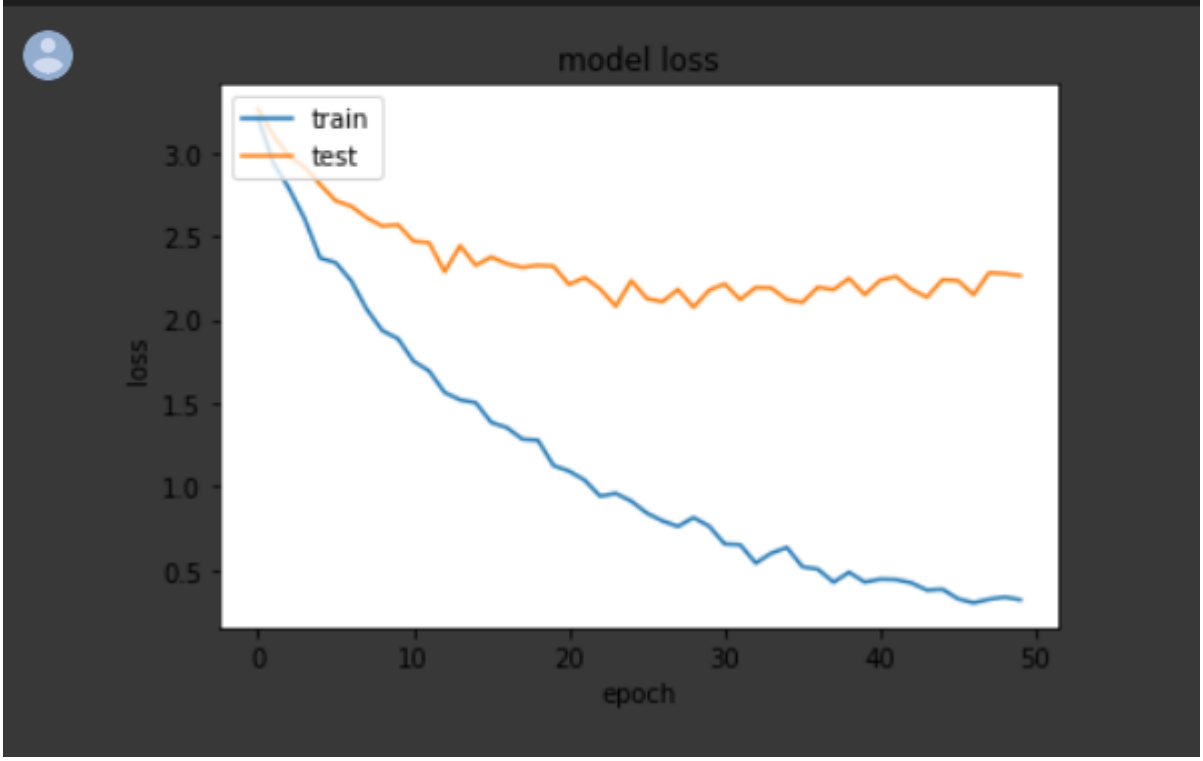
- Accuracy Graph :

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



- Loss Graph :

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



VII. ANALYSIS OF RESULTS :

Training Accuracy : 92.59%

Testing Accuracy : 45.83%

Here, it can be observed that the training accuracy is much more higher as compared to the testing accuracy. The reason for this is that the images we collected for the dataset are very less in number, for validation purpose. Also, the quality of pictures affects the training of the model, and the images in the dataset which were collected were not up to the mark in quality. More amount of images and better quality pictures can give higher accuracy and can generalise the model well.

VIII. CONCLUSION :

In this way, we developed a face recognition system using CNN. A dataset of 20 students was created. The accuracy of the model was checked and validated after training the model.