CS 335 : Facial Recognition

Introduction:

—----------

This project has been done as a part of the CS335 Course by Sartaj, Margav, Dhvanit and Harsha. To train the different models we used GPU provided by google colab. To train a model it would be better to use GPU on google Colab, but if we just want to predict outcomes using our models it can be done locally also using predict.ipynb..

We have developed the facial recognition module in 4 steps:

a) Collecting the Dataset
b) Architecture.py
c) utils.py
d) train.ipynb
e) predict.ipynb

We also developed an alternate model where we used transfer learning using pretrained ResNet152 and fine tuned its last layer for classification.Let us begin with the description of our main model

Collecting the Dataset:

—----------

Dataset used for this module is extracted from Kaggle.This Dataset contains 2507 images . Each folder is named after the person in the picture for convenience.We are using only those persons whose dataset has more than 20 images for training purposes.These datasets are preprocessed in the “train.ipynb” by first converting them from rgb to grayscale images
.
Advantages: Using grayscale images helps reduce the number of model parameters and also helps to train it faster

Split the dataset into training and test with 80:20 ratio

Train Data size : 1427
Validation Data size : 357

Defined a Custom class MyDataset with a custom function such as __get_item__() to get transformed images from original images and applied it on the entire dataset.

Advantage of GPU:

High Data Throughput: a GPU consists of hundreds of cores performing the same operation on multiple data items in parallel. Because of that, **a GPU can push vast volumes of processed data through a workload, speeding up specific tasks beyond what a CPU can handle**.

Architecture.py:

—----------

In this file we have described the entire architecture on which the model is built.
The architecture is neatly divided into 4 parts

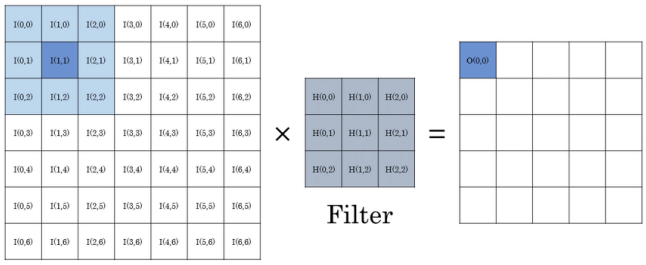
a) Layer 1
b) Layer 2
c) Layer 3
d) Class Architecture

A bit of information about the inbuilt functions used.

Conv2d Layer:

—----------

There is a filter which passes through the Image and creates a new image of reduced dimension. This basically convolutes on the image wrt to the filter such as taking the weighted average of all pixels with respect to the filter and storing it in the center pixel of the filter.Image provided for more information



Layer 1:

—---------

This layer is made up of different sequential components which are ordered in the following way:
Conv2d —---> ReLU —---> BatchNorm2d —--->Conv2d —---> ReLU —---->BatchNorm2d —----> Conv2d —---> ReLU —---> BatchNorm2d —---->MaxPool2d—---->Dropout

Layer 2:

—----------

This layer is made up of different sequential components which are ordered in the following way:

Conv2d ——-----> ReLU ——-----> BatchNorm2d ——-----> MaxPool2d ——-----> Dropout

Layer 3:

——----------

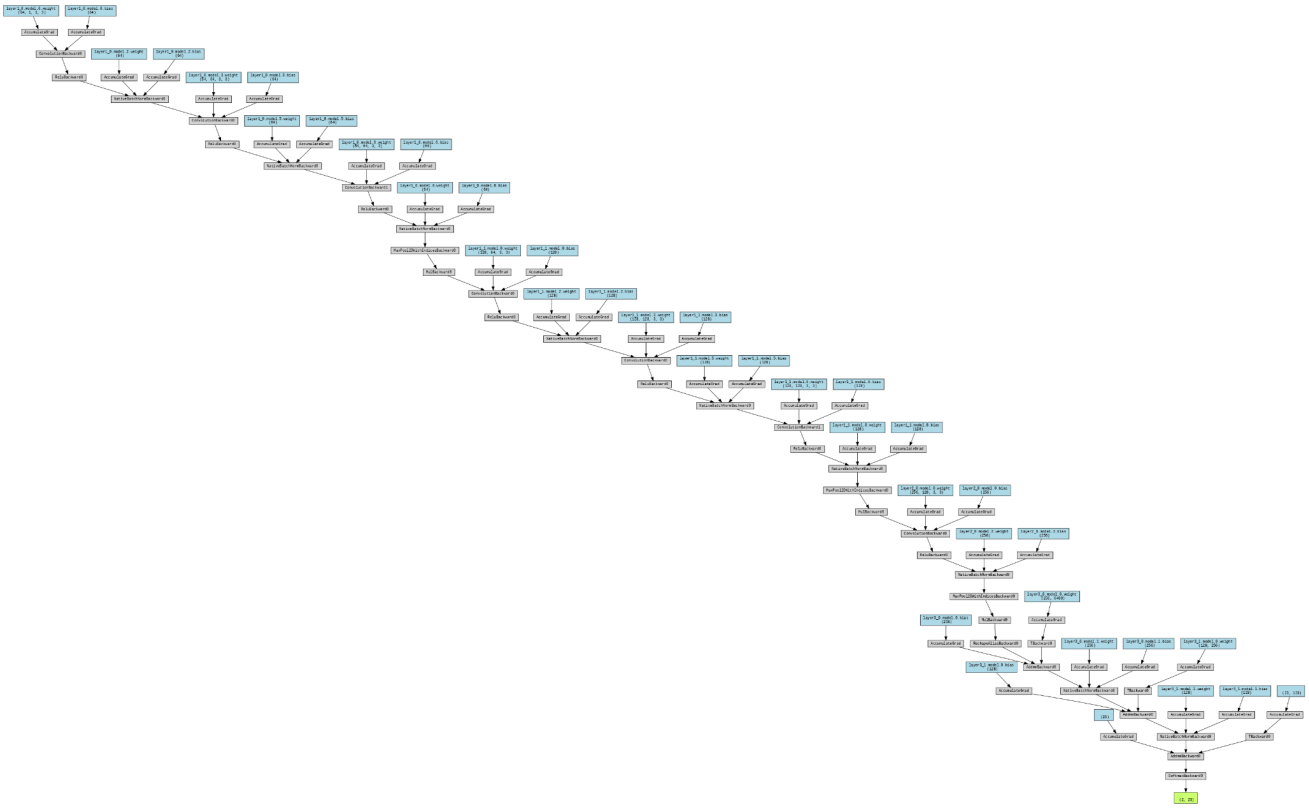
This layer is made up of different sequential components which are ordered in the following way:

Linear ——-----> BatchNorm1d

Architecture:

——----------

Defined a custom function addLayer which adds a module into the model class with corresponding correct input and output parameters.Next we have arranged the architecture as given below in the Block Diagram



These are the hyperparameters we used. To train a different model you can change them and train the model using corresponding train.ipynb file.

```
{
        'image_shape': (2, 1, 100, 100),
         'in_ch1': 1,
         'out_ch1': 64,
         'in_ch2': 256,
         'out_ch2': 22,
         'layer1': 2,
         'layer2': 1,
         'layer3': 2
}
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
             Conv2d-1          [-1, 64, 62, 62]             640
               ReLU-2          [-1, 64, 62, 62]               0
        BatchNorm2d-3          [-1, 64, 62, 62]             128
             Conv2d-4          [-1, 64, 60, 60]          36,928
               ReLU-5          [-1, 64, 60, 60]               0
        BatchNorm2d-6          [-1, 64, 60, 60]             128
             Conv2d-7          [-1, 64, 60, 60]          36,928
               ReLU-8          [-1, 64, 60, 60]               0
        BatchNorm2d-9          [-1, 64, 60, 60]             128
         MaxPool2d-10          [-1, 64, 30, 30]               0
           Dropout-11          [-1, 64, 30, 30]               0
            Layer1-12          [-1, 64, 30, 30]               0
            Conv2d-13         [-1, 128, 28, 28]          73,856
              ReLU-14         [-1, 128, 28, 28]               0
       BatchNorm2d-15         [-1, 128, 28, 28]             256
            Conv2d-16         [-1, 128, 26, 26]         147,584
              ReLU-17         [-1, 128, 26, 26]               0
       BatchNorm2d-18         [-1, 128, 26, 26]             256
            Conv2d-19         [-1, 128, 26, 26]         147,584
              ReLU-20         [-1, 128, 26, 26]               0
       BatchNorm2d-21         [-1, 128, 26, 26]             256
         MaxPool2d-22         [-1, 128, 13, 13]               0
           Dropout-23         [-1, 128, 13, 13]               0
            Layer1-24         [-1, 128, 13, 13]               0
            Conv2d-25         [-1, 256, 11, 11]         295,168
              ReLU-26         [-1, 256, 11, 11]               0
       BatchNorm2d-27         [-1, 256, 11, 11]             512
         MaxPool2d-28           [-1, 256, 5, 5]               0
           Dropout-29           [-1, 256, 5, 5]               0
            Layer2-30           [-1, 256, 5, 5]               0
           Flatten-31                [-1, 6400]               0
            Linear-32                 [-1, 256]       1,638,656
       BatchNorm1d-33                 [-1, 256]             512
            Layer3-34                 [-1, 256]               0
            Linear-35                 [-1, 128]          32,896
       BatchNorm1d-36                 [-1, 128]             256
            Layer3-37                 [-1, 128]               0
            Linear-38                  [-1, 23]           2,967
           Softmax-39                  [-1, 23]               0
================================================================
Total params: 2,415,639
Trainable params: 2,415,639
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.02
Forward/backward pass size (MB): 25.16
Params size (MB): 9.21
Estimated Total Size (MB): 34.39
----------------------------------------------------------------
```

Summary of the layers used

utils.py

—----------

We trained the model with respect to the architecture of the neural network provided in the architecture.py with utilizing the predefined binary cross entropy loss function. We used the SGD (Stochastic Gradient descent) as the optimizer with corresponding learning rate parameters and weight decay to reduce overfitting. We have used 20 epochs to train the model.

train.ipynb:

—----------

This file basically uses all the defined functions in the above three files sequentially to create a model and finally save it after training. For model1 we obtained validation accuracy 86.48 whereas for model2 we obtained 84.69 .

predict.ipynb:

—----------

This file basically loads the models which we saved. Using this model it predicts the output for images which we give in the predict images list and prints the results obtained in an easily interpretable way.

resnet152.ipynb:

—----------

This file basically trains and saves the model for transfer learning. We have used RGB images and got an encoding of length 2048 from the resnet model. Then using a linear layer whose parameters we tuned predicted our outputs. We got a validation accuracy of 76.69 for this model.