

PROFESSIONAL TRAINING REPORT
At
SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Submitted in partial fulfillment of the requirements for the
award of Bachelor of Engineering Degree in

Information Technology

By

Sharon Swetha. X
(Reg. No. 39120109)



DEPARTMENT OF INFORMATION TECHNOLOGY
SCHOOL OF COMPUTING

**SATHYABAMA INSTITUTE OF
SCIENCE AND TECHNOLOGY**
(DEEMED TO BE UNIVERSITY)

(Established under Section 3 of UGC Act, 1956)
Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai - 600119
www.sathyabamauniversity.ac.in



APRIL 2022



SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited with Grade “A” by NAAC
(Established under Section 3 of UGC Act, 1956)
JEPPIAAR NAGAR, RAJIV GANDHI SALAI
CHENNAI– 600119
www.sathyabama.ac.in



DEPARTMENT OF INFORMATION TECHNOLOGY

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **SHARON SWETHA.X (Reg. No: 39120109)** who carried out the project entitled “**AGE DETECTION THROUGH IMAGE PROCESSING**” under my supervision from March 2022 to April 2022.

Internal Guide

Dr. JEBERSON RETNA RAJ, M.E., Ph.D.,

Head of the Department

Dr. R. SUBHASHINI, M.E., Ph.D.,

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I, SHARON SWETHA X (Reg. No. 39120109) hereby declare that the Project report entitled "**AGE DETECTION THROUGH IMAGE PROCESSING** " done by me under the guidance of **Dr. JEBERSON RETNA RAJ, M.E., Ph.D.**, is submitted in partial fulfillment of the requirements for the award of Bachelor of degree in Information Technology.

Sharon Swetha

DATE:

PLACE:

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala, M.E., Ph.D., Dean, School of Computing, Dr. Subhashini, M.E., Ph.D., Head of the Department of Information Technology** for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. JEBERSON RETNA RAJ, M.E., Ph.D.**, for his valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Information Technology** who were helpful in many ways for the completion of the project

TRAINING CERTIFICATE

ABSTRACT

Age detection is the process of automatically discerning the age of a person solely from a photo of their face. This involves two stages, importing the images and then age recognition by mapping pixel images to its corresponding age and giving it as a feed to age detector algorithm. The human face has biometric properties that are important for providing age information because of the aging process of the face. Automatic Age estimation is a difficult problem because the relationship between facial images and age is not very linear.

So, we use FGNET dataset for detecting the age. The FG-NET aging database was released in 2004 in an attempt to support research activities related to facial aging. Since then, a number of researchers used the database for carrying out research in various disciplines related to facial aging. So, I have used ResNet-34 pretrained model for the age detection. It is used to denote the variant that can work with 34 neural network layers.

The main scope of the project is to build a robust age detection model by using ResNeT-34 (short form for Residual Network) convolutional neural network model and used it to predict the age of the person in an image and compared the accuracy with sequential model. Then found out the mean absolute error too. This research work will use images feed as input to get live results.

TABLE OF CONTENTS

CHAPTER No.	TITLE	PAGE No
	ABSTRACT	i
	LIST OF FIGURES	ii
	LIST OF TABLES	iii
	LIST OF ABBREVIATIONS	iv
1	INTRODUCTION	1
	1.1 WHAT IS DEEP LEARNING?	1
	1.2 CONVOLUTIONAL NEURAL NETWORK	1
	1.3 ResNet-34 MODEL	1
	1.4 AGE ESTIMATION	3
2.	AIM AND SCOPE OF THE PRESENT INVESTIGATION	4
	2.1 AIM	4
	2.2 SCOPE	4
3.	EXPERIMENTAL OR MATERIALS AND METHODS, ALGORITHMS USED	5
	3.1 OVERVIEW	5
	3.2 REQUIREMENTS	5
	3.3 DATASET INFORMATION	5
	3.4 METHODOLOGY	6
4.	RESULTS AND DISCUSSION, PERFORMANCE ANALYSIS	14
	4.1 RESULT	14

	4.2 ANALYSIS	14
5	SUMMARY AND CONCLUSIONS	16
	REFERENCES	16
	APPENDIX	16
	A. SCREENSHOTS	16
	B. SOURCE CODE	32

LIST OF FIGURES

FIGURE No.	TITLE	PAGE No.
1.1	ResNet-34 ARCHITECTURE	2
1.2	AGE ESTIMATION	3
3.1	BAR PLOT	7
3.2	PLOT	7
3.3	MISSING AGES	8
3.4	READING	8
3.5	DATASET	9
3.6	BASIC BLOCK	11
3.7	TRAINING METRICS	13
4.1	RESULT	14
4.2	TRAINING ACCURACY	14
4.3	TEST ACCURACY	14
4.4	SEQ TRAIN ACCURACY	15
4.5	SEQ TEST ACCURACY	15

LIST OF TABLES

TABLE No.	TITLE	PAGE No.
3.1	STRUCTURE	11

LIST OF ABBREVIATIONS

CNN	-	Convolutional Neural Network
ReLU	-	Rectified Linear Unit
ResNet	-	Residual Network
VGG	-	Visual Geometry Group
GPU	-	Graphical Processing Unit
BN	-	Batch Normalizations
Conv	-	Convolutional Layers

CHAPTER - 1

INTRODUCTION

What is deep learning?

Deep learning is a subset of machine learning, which is essentially a neural network with three or more layers. These neural networks attempt to simulate the behavior of the human brain. While a neural network with a single layer can still make approximate predictions, additional hidden layers can help to optimize and refine for accuracy.

Convolutional Neural Network

Computer vision technique introduces Convolutional Neural Network (CNN) as a development of Neural Networks. CNN achieves great successfully end-to-end classification for various tasks related to images such as image classification, objects for detection / localization, face recognition and image segmentation. CNN with deep architecture can recognize objects in large-scale drawing data sets. Some components are used to build a more effective CNN model: the activation unit called rectified linear unit (ReLU) helps accelerate convergence during training and dropout regularize prevents over fitting by setting some activation units to zero in certain layers.

ResNet-34 MODEL

Classical The first ResNet architecture was the Resnet-34 which involved the insertion of shortcut connections in turning a plain network into its residual network counterpart. In this case, the plain network was inspired by VGG neural networks (VGG-16, VGG-19), with the convolutional networks having 3×3 filters. However, compared to VGGNets, ResNets have fewer filters and lower complexity. The 34-layer ResNet achieves a performance of 3.6 bn FLOPs, compared to 1.8bn FLOPs of smaller 18-layer ResNets.

It also followed two simple design rules – the layers had the same number of filters for the same output feature map size, and the number of filters doubled in case the feature map size was halved in order to preserve the time complexity per layer. It consisted of 34 weighted layers

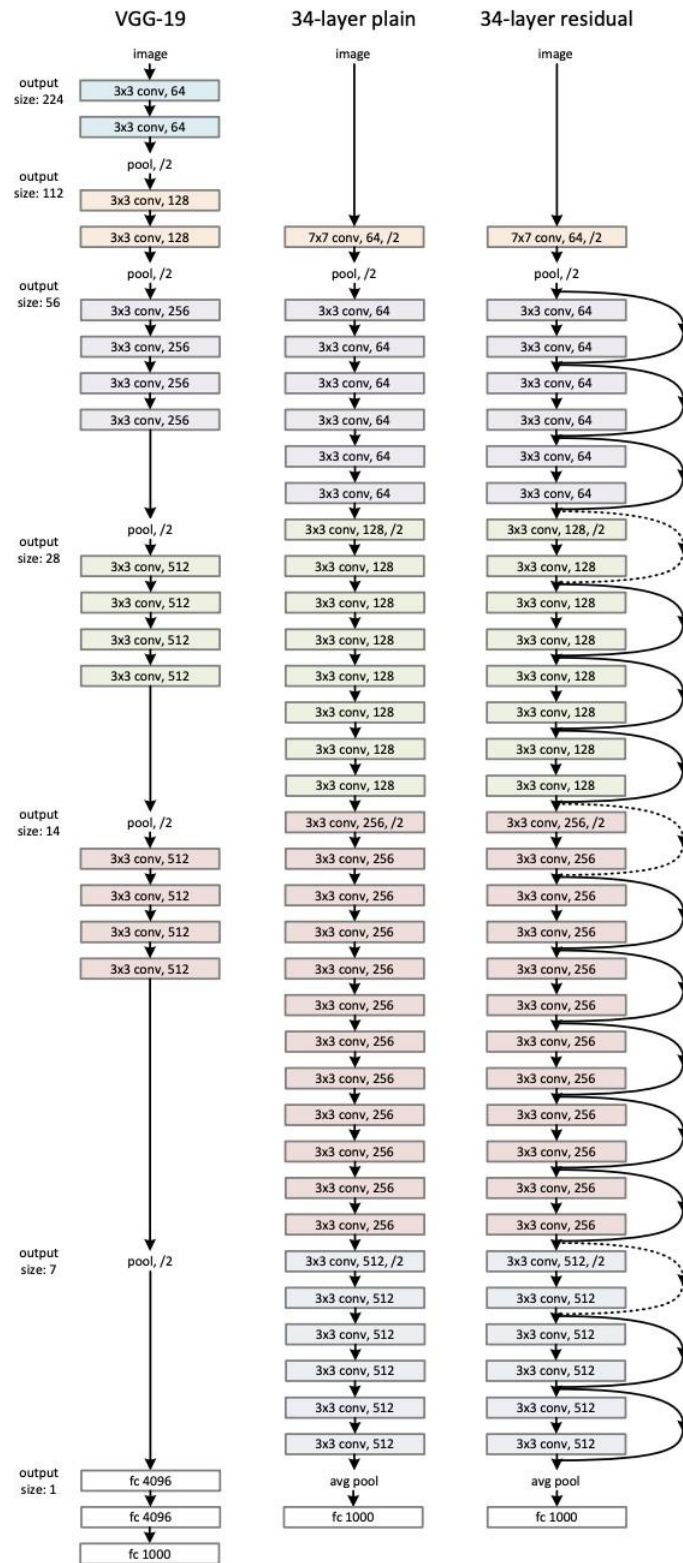


Fig 1.1: ResNet-34 ARCHITECTURE

AGE ESTIMATION

Age estimation plays a significant role for individual identification in law enforcement, security control, and human social interaction. Recently, the use of the CNN method for age estimation has been widely adopted because of its superior performance compared to the statistical regression and classification. This involves two stages, importing the images and then age recognition by giving it as a feed to age detector algorithm. Age estimation of human faces is a difficult task because of the highly non-linear relationship between facial images and chronological age. Thus, mapping pixel images to its corresponding age requires robust and accurate functions.

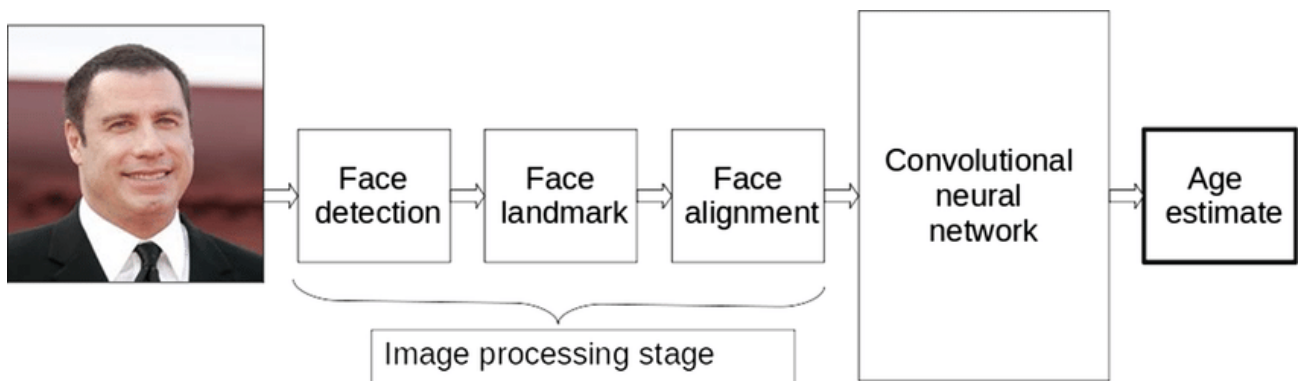


Fig 1.2: AGE ESTIMATION

CHAPTER - 2

AIM AND SCOPE OF THE PRESENT INVESTIGATION

AIM:

The aim of the project is to detect the age through image processing by using convolutional neural network model.

SCOPE:

- The main scope of the project is to detect the age of a person
- Then to build a robust age detection model by using ResNeT-34 (short form for Residual Network) convolutional neural network model and use it to find the train and test accuracy of the prediction.
- I have found out custom loss function and mean absolute error for both train and test data.

CHAPTER - 3

EXPERIMENTAL OR MATERIALS AND METHODS, ALGORITHMS USED

AGE DETECTION THROUGH IMAGE PROCESSING

Overview:

- The aim of the project is to detect the age through image processing by using convolutional neural network model.
- Then to build a robust age detection model by using ResNeT-34 (short form for Residual Network) convolutional neural network model and use it to find the train and test accuracy of the prediction.
- I have found out custom loss function and mean absolute error for both train and test data.

Requirements:

To build the convolutional neural network model, the following components are listed below:

- ***DATASET INFORMATION:***

The FG-NET aging database was released in 2004 in an attempt to support research activities related to facial aging. Since then, a number of researchers used the database for carrying out research in various disciplines related to facial aging. The FG-NET-AD contains 1002 images from 82 different subjects with ages ranging between newborns to 69 years old subjects. However, ages between zero to 40 years are the most populated in the database.

With the exception of images showing individuals at more recent ages, for which digital images were available, in most cases FG-NET-AD images were collected by scanning photographs of subjects found in personal collections. As a consequence, the quality of images depends on the photographic skills of the photographer, the quality of the imaging equipment used, the quality of photographic paper and printing and the condition of

photographs. As a result, face images in the FG-NET-AD display considerable variability in resolution, quality, illumination, viewpoint and expression. Occlusions in the form of spectacles, facial hair and hats are also present in a number of images. Each image in the dataset was annotated with 68 landmark points located at key positions and also a semantic description of each image was recorded. In particular, information about the age, gender, expression, pose, image quality and appearance of occlusions (i.e., moustaches, beards, hats or spectacles) was recorded

- **GPU:**

GPU (Graphical Processing Unit)- It is a specialized electronic circuit designed to render 2D and 3D graphics together with a CPU. GPU is also known as Graphics Card in the Gaming culture.

Now GPUs are being harnessed more broadly to accelerate computational workloads in areas such as financial modeling, cutting-edge scientific research, deep learning, analytics, etc. GPU can handle tens of thousands of operations per cycle.

- **GOOGLE COLAB:**

Colaboratory, or “Colab” for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education.

METHODOLOGY:

- **IMPORTING ALL THE NECESSARY LIBRARIES:**

1. NUMPY
2. MATPLOTLIB
3. TENSORFLOW
4. PANDAS
5. SKLEARN
6. KERAS
7. PROGRESS BAR

- **DATA ANALYSIS:**

Data analysis is the first step to do the project because data exploration is very important to know the inputs. Data Analysis is the process of inspecting, cleansing, transforming, and modelling data with the goal of discovering useful information by informing conclusions and supporting decision making.

- 1. **GRAPHS:**

They are used to represent, find, analyze, and optimize connections between elements. Ex: Bar plot, Heat map

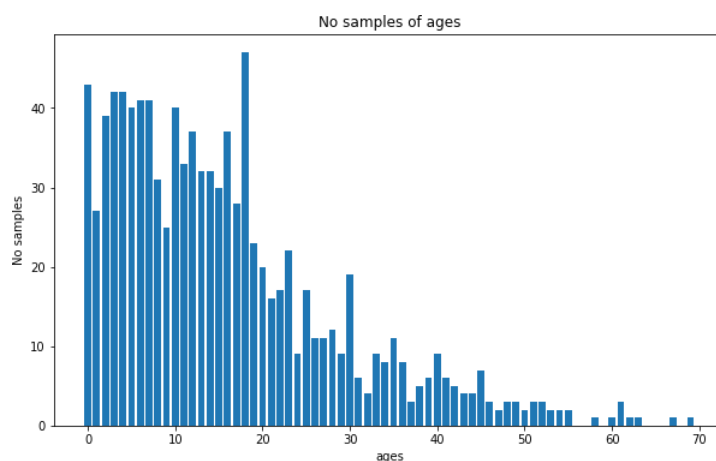


Fig 3.1: BAR PLOT

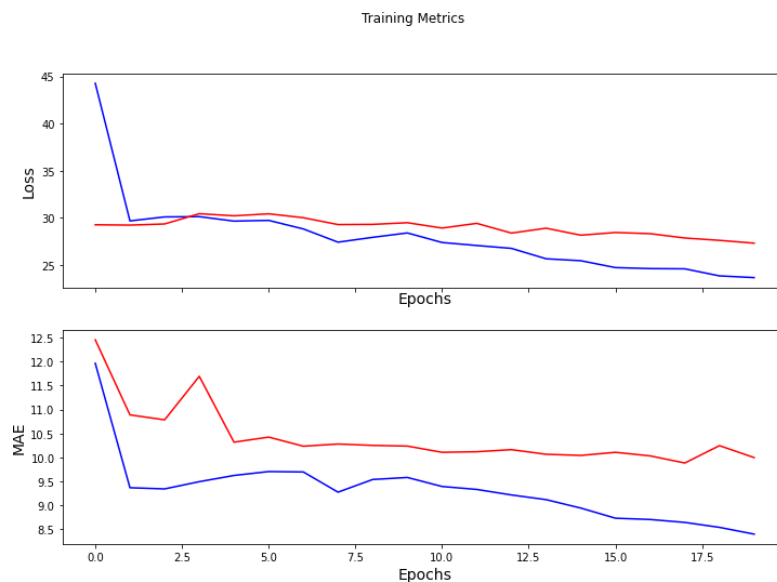


Fig 3.2: PLOT

2. Finding Missing Ages:

```
[ ] #Find missing ages
miss_ages = []
for age in range(70):
    if age not in age_lst:
        miss_ages.append(age)

miss_ages

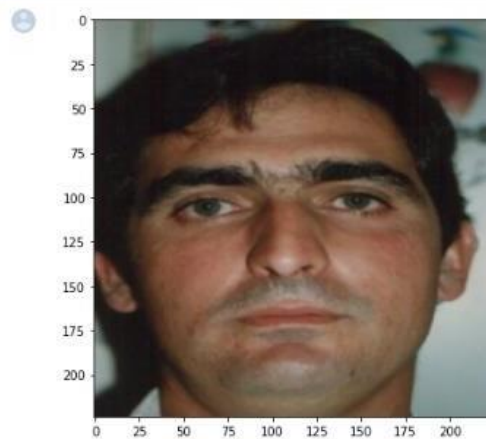
[56, 57, 59, 64, 65, 66, 68]
```

Fig 3.3: MISSING AGES

3. READING THE PIXELS:

```
from matplotlib import image

plt.figure(figsize = (6, 6))
plt.imshow(images[10])
plt.show()
```



```
[ ]

print(ages[10])

34
```

Fig 3.4: READING

- **DATA PREPROCESSING:**

1. ***Splitting the Dataset into the Training set and Test set***

In machine learning data preprocessing, we divide our dataset into a training set and test set. This is one of the crucial steps of data preprocessing as by doing this, we can enhance the performance of our machine learning model.

Suppose, if we have given training to our machine learning model by a dataset and we test it by a completely different dataset. Then, it will create difficulties for our model to understand the correlations between the models.

If we train our model very well and its training accuracy is also very high, but we provide a new dataset to it, then it will decrease the performance. So, we always try to make a machine learning model which performs well with the training set and also with the test dataset. Here, we can define these datasets as:

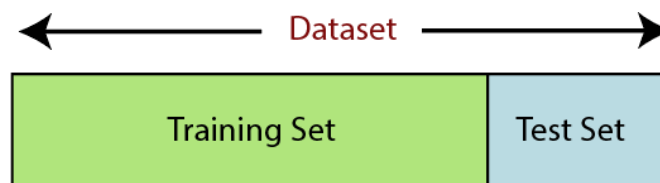


FIG 3.5: DATASET

Training Set: A subset of dataset to train the machine learning model, and we already know the output.

Test set: A subset of dataset to test the machine learning model, and by using the test set, model predicts the output.

For splitting the dataset, we will use the below lines of code:

1. `from sklearn.model_selection import train_test_split`
2. `(trainData, testData, trainLabels, testLabels) = train_test_split(images, ages, test_size = 0.4, random_state = 42, shuffle =True)`

Explanation:

In the above code, the first line is used for splitting arrays of the dataset into random train and test subsets.

- In the second line, we have used four variables for our output that are
- **trainData:** features for the training data
- **testData:** features for testing data
- **trainLabels:** Dependent variables for training data
- **testLabels:** Independent variable for testing data
- In **train_test_split() function**, we have passed four parameters in which first two are for arrays of data, and **test_size** is for specifying the size of the test set. The **test_size** maybe .4, .3, or .2, which tells the dividing ratio of training and testing sets.

2. DATA AUGMENTATION:

Data augmentation is a set of techniques to artificially increase the amount of data by generating new data points from existing data. This includes making small changes to data or using deep learning models to generate new data points.

3. BUILDING MODEL BASE ON RESNET 34 ARCHITECTURE:

Residual building block is the infrastructure of the ResNet-34 network, and the whole network is mainly composed of it. A shortcut-connection was used by the residual building block to skip the convolutional layers, which effectively alleviates the problem of gradient disappearance or gradient explosion caused by increasing depth in the neural networks, and it helped us to construct CNN structures more flexibly and improve the recognition rate.

The structure of the basic-block is shown in Figure 3.6. It is used for the 34 layers of ResNet. The residual building block is composed of several convolutional layers (Conv), batch normalizations (BN), a rectified linear unit (ReLU) activation function, and a shortcut. The output of the residual building block can be formulated as follows:

$$y = F(x) + x$$

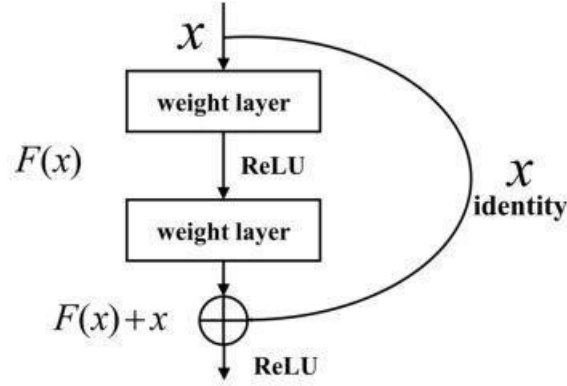


FIG 3.6: BASIC BLOCK

The ResNet-34 contains 33 convolutional layers, a max-pooling layer of size 3×3 , an average pool layer, followed by a fully connected layer. A classical ResNet-34 model involves 63.5 million parameters, where rectification nonlinearity (ReLU) activation and batch normalization (BN) is applied to the back of all convolution layers in the “BasicBlock” block and the softmax function is applied in the final layer. The architecture of ResNet-34 is shown in Table 3.1.

Layer Name	Output Size	34-Layer
Conv1	112×112	$7 \times 7, 64$, stride 2 3×3 max pool, stride 2
Conv2_x	56×56	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$
Conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$
Conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$
Conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax

TABLE 3.1: STRUCTURE

STAGES IN BUILDING THE MODEL:

The first step on the ResNet before entering the common layer behavior is a block — called here Conv1 — consisting on a convolution + batch normalization + max pooling operation. A convolutional layer is the main building block of a CNN. It contains a set of filters (or kernels), parameters of which are to be learned throughout the training.

The next step is the batch normalization, which is an element-wise operation and therefore, it does not change the size of our volume. Finally, we have the (3x3) Max Pooling operation with a stride of 2. We can also infer that they first pad the input volume, so the final volume has the desired dimensions.

Every layer of a ResNet is composed of several blocks. This is because when ResNets go deeper, they normally do it by increasing the number of operations within a block, but the number of total layers remains the same. An operation here refers to a convolution a batch normalization and a ReLU activation to an input, except the last operation of a block, that does not have the ReLU. Each convolutional layer is followed by a batch normalization layer and a ReLU activation function. A convolution layer **transforms the input image in order to extract features from it**

ACTIVATION FUNCTION:

The activation function is the nonlinear transformation that we do over the input signal. This transformed output is then sent to the next layer of neurons as input.

So, in this model we use two activation function that are:

- *ReLU*- The rectified linear activation function overcomes the vanishing gradient problem, allowing models to learn faster and perform better.
 - *Softmax*- This allows the output to be interpreted directly as a probability.
- we use ReLU in hidden layer to avoid vanishing gradient problem and better computation performance, and Softmax function use in last output layer

4. CREATE CUSTOM LOSS FUNCTION

A custom loss function can be created by defining a function that takes the true values and predicted values as required parameters. The function should return an array of losses. The function can then be passed at the compile stage. Then we have to find out mean absolute error.

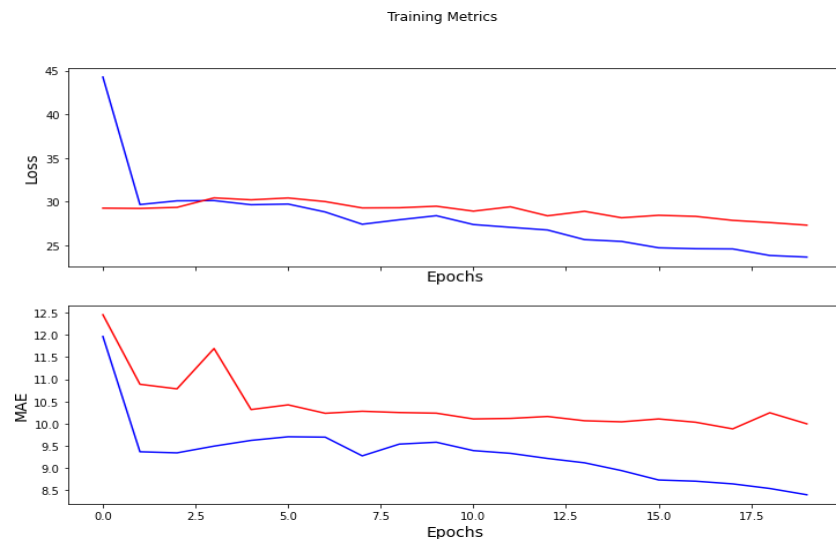


FIG 3.7: TRAINING METRICS

- **FINDING ACCURACY:**

- After building the build the model we have to fit the model
- Then we have to compile and evaluate the model
- At last, find out the train and test accuracy
- And then compare the model with sequential model.

CHAPTER - 4

RESULTS AND DISCUSSION, PERFORMANCE ANALYSIS

Result:

I have predicted the age of the person through image processing using ResNet-34 convolutional neural network model



FIG 4.1: RESULT

ANALYSIS:

By building the ResNet-34 model, I have found out the training and test accuracy of my prediction and compared the accuracy with sequential model.

ResNet-34 Model

```
Epoch 35/35
6/6 [=====] - 7s 1s/step - loss: 0.2308 - accuracy: 0.9429
<keras.callbacks.History at 0x7fa7c003c490>
```

FIG 4.2: TRAINING ACCURACY

```
[ ] (loss, accuracy) = model.evaluate(testData, testLabels, batch_size = 128, verbose = 1)
print("accuracy: {:.2f}%".format(accuracy * 100))

/usr/local/lib/python3.7/dist-packages/tensorflow/python/util/dispatch.py:1082: UserWarning: ``sparse_categor:
return dispatch_target(*args, **kwargs)
3/3 [=====] - 3s 641ms/step - loss: 6.4520 - accuracy: 0.0465
accuracy: 4.65%
```

FIG 4.3: TESTING ACCURACY

Sequential Model

```
Epoch 35/35  
25/25 [=====] - 1s 49ms/step - loss: 1.2479 - accuracy: 0.5000
```

FIG 4.4: SEQ TRAIN ACCURACY

```
(loss, accuracy) = model.evaluate(testData, testLabels, batch_size = 128, verbose = 1)  
print("accuracy: {:.2f}%".format(accuracy * 100)) #testing accuracy  
  
3/3 [=====] - 2s 679ms/step - loss: nan - accuracy: 0.0266  
accuracy: 2.66%
```

FIG 4.5: SEQ TEST ACCURACY

After comparing both the models ResNet-34 has the highest accuracy rate in the age detection. So ResNet-34 is the better model for image classification.

CHAPTER - 5

SUMMARY AND CONCLUSION

So, by using FG-NET dataset I have detected the age of a person solely by their faces and used ResNet-34 CNN model for predicting the training and testing accuracy of my detection.

REFERENCES:

<https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8>

<https://cnvrg.io/keras-custom-loss-functions/>

https://www.researchgate.net/publication/284839126_An_Overview_of_Research_Activities_in_Facial_Age_Estimation_Using_the_FG-NET_Aging_Database

APPENDIX

A. SCREENSHOTS



```
from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive

[2] %cd /content/drive/MyDrive/
/content/drive/MyDrive

[3] import os
import zipfile
zip_ref=zipfile.ZipFile("/content/drive/MyDrive/FGNET.zip", 'r')
zip_ref.extractall('/content/drive/MyDrive/FG_NET')
zip_ref.close()

[4] !ls "/content/drive/MyDrive/FG_NET/FGNET"

Data_files  images  points

[5] import os
data_dir="/content/drive/MyDrive/FG_NET/FGNET/images"
files=os.listdir(data_dir)
print(len(files))

1002

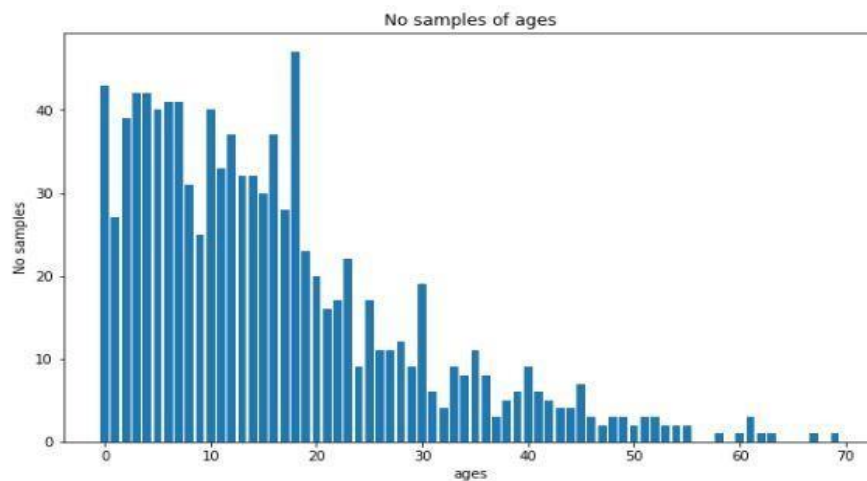
[6] import numpy as np
ages = []
for age in files:
    age = age[age.find('A') + 1: age.find('A') + 1 + 2]
    # print(age)
    ages.append(int(age))
ages = np.array(ages)

[7] len(ages)

1002
```

▼ Perform Data Exploration

```
✓ [8] import matplotlib.pyplot as plt  
1s  
  
def _plot_bar(varname):  
    labels, values = np.unique(ages, return_counts=True)  
    plt.figure(figsize=(10, 6))  
    ax = plt.subplot()  
    ax.bar(x=labels, height=values)  
    ax.set_title('No samples of ' + varname)  
    ax.set_xlabel(varname)  
    ax.set_ylabel('No samples')  
    plt.show()  
  
_plot_bar('ages')
```



```

import pandas as pd

labels, values = np.unique(ages, return_counts=True)
age_df = pd.DataFrame(data = values, index = labels, columns = ['Freq'])
age_df

```

	Freq
0	43
1	27
2	39
3	42
4	42
...	...
61	3
62	1
63	1
67	1
69	1

63 rows x 1 columns

```

[10] age_df.describe()

```

	Freq
count	63.000000
mean	15.904782
std	14.785803
min	1.000000
25%	3.000000
50%	9.000000
75%	29.000000
max	47.000000

```

[11]
age_df.loc[age_df.idxmax()]

```

	Freq
18	47

```

[12] age_df.loc[age_df.idxmin()]

```

	Freq
58	1

```

[61] age_lst = list(age_df.index.values)

#Find missing ages
miss_ages = []
for age in range(70):
    if age not in age_lst:
        miss_ages.append(age)

miss_ages

```

[56, 57, 59, 64, 65, 66, 68]

```

[54] import numpy as np
from PIL import Image
import progressbar

widgets = ['Reading images:', progressbar.Percentage(), " ", progressbar.Bar(), " ", progressbar.ETA()]
pbar = progressbar.ProgressBar(maxval = len(files), widgets = widgets).start()

images = []
for i, imageData in enumerate(files):
    image = np.array(Image.open(data_dir + '/' + imageData).convert('RGB').resize((224, 224)))
    images.append(image)
    pbar.update(i)

pbar.finish()

images = np.array(images)

```

Reading images:100% [#####] Time: 0:00:09

```
from matplotlib import image

plt.figure(figsize = (6, 6))
plt.imshow(images[10])
plt.show()
```



```
[ ]

print(ages[10])

34
```

```
images[0].shape
(224, 224, 3)
```

```
[23] import matplotlib
def displayImage(startIdx, numImages, imagePath, showAge = True):
    endIdx = startIdx + numImages
    size = (5,5)
    fig = plt.figure(figsize = (3*numImages, 2*numImages))
    listImage = os.listdir(imagePath)
    for i, imagename in enumerate(listImage[startIdx : endIdx]):
        ax = fig.add_subplot(size[0], size[1], i + 1, xticks = [], yticks = [])
        img = matplotlib.image.imread(imagePath + '/' + imagename)
        ax.imshow(img)
        if showAge:
            ax.set_title('Imagenname: {}, Age: {}'.format(imagename, ages[startIdx + i]), color = 'green')
```

```
[24] displayImage(20, 10, '/content/drive/MyDrive/FG_NET/FGNET/images')
```

Imagenname: 054A17.JPG, Age: 17



Imagenname: 054A16.JPG, Age: 16



Imagenname: 054A18.JPG, Age: 18



Imagenname: 054A19.JPG, Age: 19



Imagenname: 055A04.JPG, Age: 4



Imagenname: 055A05.JPG, Age: 5



Imagenname: 055A06.JPG, Age: 6



Imagenname: 055A08.JPG, Age: 8



Imagenname: 055A14.JPG, Age: 14



Imagenname: 055A12.JPG, Age: 12



Split data in training set and validation set

```
from sklearn.model_selection import train_test_split

images = images.astype('float')/255

(trainData, testData, trainLabels, testLabels) = train_test_split(images, ages, test_size = 0.3, random_state = 42, shuffle =True)
```

Create Dataset with tf.data.Dataset

```
[27] import tensorflow as tf
train_dataset = tf.data.Dataset.from_tensor_slices((trainData, trainLabels))
test_dataset = tf.data.Dataset.from_tensor_slices((testData, testLabels))
```

Perform Data Augmentation

```
[28] def augment(image,label, mode):
    if mode == 'train':
        # image = tf.image.resize_with_crop_or_pad(image, 230, 230) # Add 6 pixels of padding
        image = tf.image.random_crop(image, size=[218, 218, 3])
        image = tf.image.resize(image, size=[224, 224])
        image = tf.image.random_saturation(image, 2, 3)
        image = tf.image.random_brightness(image, max_delta=0.5) # Random brightness
        image = tf.image.random_flip_left_right(image)
        return image,label
    else:
        image_cropped = tf.image.crop_to_bounding_box(image, offset_height=20, offset_width=0, target_height=200, target_width=224)
        image_resized = tf.image.resize(image_cropped, size= [224,224])
        return image_resized, label
```

```
[29] BATCH_SIZE_TRAIN = 64
BATCH_SIZE_TEST = 16
SHUFFLE_BUFFER_SIZE = 100
```

```
train_dataset = train_dataset.shuffle(SHUFFLE_BUFFER_SIZE).map(lambda image, label: augment(image, label, mode = 'train'), num_parallel_calls = tf.data.experimental.AUTOTUNE).batch(BATCH_SIZE_TRAIN)
test_dataset = test_dataset.map(lambda image, label: augment(image, label, mode = 'eval'), num_parallel_calls = tf.data.experimental.AUTOTUNE).batch(BATCH_SIZE_TEST)
```

BUILDING MODEL BASE ON RESNET 34 ARCHITECTURE

```
[30] from tensorflow.keras.models import Sequential
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Activation, Flatten, Dense, BatchNormalization, ZeroPadding2D, Add, Input, AveragePooling2D, GlobalMaxPooling2D, Lambda
from tensorflow.keras.initializers import glorot_uniform
import tensorflow as tf
```

```
[31] def identity_block(X, f, filters, stage, block):
    conv_name_base = 'res' + str(stage) + block + '_branch'
    bn_name_base = 'bn' + str(stage) + block + '_branch'

    F1, F2 = filters

    X_shortcut = X

    X = Conv2D(filters = F1, kernel_size=(f,f), strides= (1,1), padding='same', name = conv_name_base + '2a', kernel_initializer=glorot_uniform(seed = 0))(X)
    X = BatchNormalization(axis = -1, name = bn_name_base + '2a')(X)
    X = Activation('relu')(X)

    X = Conv2D(filters = F2, kernel_size= (f,f), strides= (1,1), padding='same', name = conv_name_base + '2b', kernel_initializer=glorot_uniform(seed = 0))(X)
    X = BatchNormalization(axis = -1, name = bn_name_base + '2b')(X)

    X = Add()([X_shortcut, X])
    X = Activation('relu')(X)

    return X
```

```
[32] def convolutional_block(X, f, filters, stage, block, s = 2):
    conv_name_base = 'res' + str(stage) + block + '_branch'
    bn_name_base = 'bn' + str(stage) + block + '_branch'

    F1, F2 = filters

    X_shortcut = X

    X = Conv2D(filters = F1, kernel_size= (f,f), strides= (s,s), padding='same', name = conv_name_base + '2a', kernel_initializer=glorot_uniform(seed = 0))(X)
    X = BatchNormalization(axis = -1, name = bn_name_base + '2a')(X)
    X = Activation('relu')(X)

    X = Conv2D(filters = F2, kernel_size= (f,f), strides= (1,1), padding='same', name = conv_name_base + '2b', kernel_initializer=glorot_uniform(seed = 0))(X)
    X = BatchNormalization(axis = -1, name = bn_name_base + '2b')(X)
```

```

X_shortcut = Conv2D(filters = F2, kernel_size=(1,1), strides = (s,s), name = conv_name_base + '1', kernel_initializer=glorot_uniform(seed=0))(X)
X_shortcut = BatchNormalization(axis = -1, name = bn_name_base + '1')(X_shortcut)

X = Add()([X_shortcut, X])
X = Activation('relu')(X)

return X

```

```

[33] def ResNet_34Base(input_shape, classes, activation = 'softmax'):
    X_input = Input(input_shape)

    X = ZeroPadding2D((3,3))(X_input)

    #Stage 1
    X = Conv2D(64, (7, 7), strides = (2, 2), name = 'conv1', kernel_initializer = glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis = 3, name = 'bn_conv1')(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((3, 3), strides=(2, 2))(X)

    #Stage 2
    X = convolutional_block(X, f = 3, filters = [64, 64], stage = 2, block = 'a', s = 1)
    X = identity_block(X, f = 3, filters= [64, 64], stage = 2, block = 'b')
    X = identity_block(X, f = 3, filters= [64, 64], stage = 2, block = 'c')

    #Stage 3
    X = convolutional_block(X, f = 3, filters = [128, 128], stage = 3, block = 'a', s = 2)
    X = identity_block(X, f = 3, filters= [128, 128], stage = 3, block = 'b')
    X = identity_block(X, f = 3, filters= [128, 128], stage = 3, block = 'c')
    X = identity_block(X, f = 3, filters= [128, 128], stage = 3, block = 'd')

    #Stage 4
    X = convolutional_block(X, f = 3, filters = [256, 256], stage = 4, block = 'a', s = 2)
    X = identity_block(X, f = 3, filters= [256, 256], stage = 4, block = 'b')
    X = identity_block(X, f = 3, filters= [256, 256], stage = 4, block = 'c')
    X = identity_block(X, f = 3, filters= [256, 256], stage = 4, block = 'd')
    X = identity_block(X, f = 3, filters= [256, 256], stage = 4, block = 'e')
    X = identity_block(X, f = 3, filters= [256, 256], stage = 4, block = 'f')

    #Stage 5
    X = convolutional_block(X, f = 3, filters = [512, 512], stage = 5, block = 'a', s = 2)
    X = identity_block(X, f = 3, filters= [512, 512], stage = 5, block = 'b')
    X = identity_block(X, f = 3, filters= [512, 512], stage = 5, block = 'c')

    X = AveragePooling2D(pool_size=(2,2), name = 'avg_pool')(X)

    X = Flatten()(X)
    X = Dense(classes, activation= 'softmax', name = 'fc' + str(classes), kernel_initializer=glorot_uniform(seed = 0))(X)

    model = Model(inputs = X_input, outputs = X, name = 'age_resnet_34')

```

✓ 0s completed

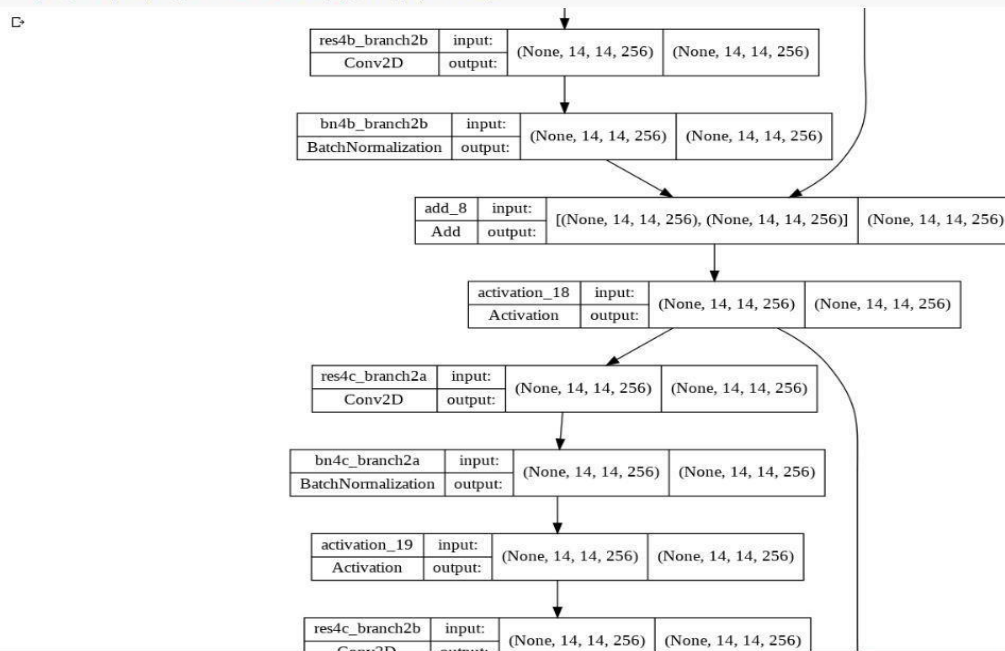
```
[34] model = ResNet_34Base((224, 224, 3), 70)

model.summary()

res5a_branch2b (Conv2D)      (None, 7, 7, 512) 2359808 ['activation_27[0][0]']
bn5a_branch1 (BatchNormalizat (None, 7, 7, 512) 2048 ['res5a_branch1[0][0]']
ion)
bn5a_branch2b (BatchNormalizat (None, 7, 7, 512) 2048 ['res5a_branch2b[0][0]']
ion)
add_13 (Add)                 (None, 7, 7, 512) 0 ['bn5a_branch1[0][0]',
                        'bn5a_branch2b[0][0]']
activation_28 (Activation)    (None, 7, 7, 512) 0 ['add_13[0][0]']
res5b_branch2a (Conv2D)      (None, 7, 7, 512) 2359808 ['activation_28[0][0]']
bn5b_branch2a (BatchNormalizat (None, 7, 7, 512) 2048 ['res5b_branch2a[0][0]']
ion)
activation_29 (Activation)    (None, 7, 7, 512) 0 ['bn5b_branch2a[0][0]']
res5b_branch2b (Conv2D)      (None, 7, 7, 512) 2359808 ['activation_29[0][0]']
bn5b_branch2b (BatchNormalizat (None, 7, 7, 512) 2048 ['res5b_branch2b[0][0]']
ion)
add_14 (Add)                 (None, 7, 7, 512) 0 ['activation_28[0][0]',
                        'bn5b_branch2b[0][0]']
activation_30 (Activation)    (None, 7, 7, 512) 0 ['add_14[0][0]']
res5c_branch2a (Conv2D)      (None, 7, 7, 512) 2359808 ['activation_30[0][0]']
bn5c_branch2a (BatchNormalizat (None, 7, 7, 512) 2048 ['res5c_branch2a[0][0]']
ion)
activation_31 (Activation)    (None, 7, 7, 512) 0 ['bn5c_branch2a[0][0]']
res5c_branch2b (Conv2D)      (None, 7, 7, 512) 2359808 ['activation_31[0][0]']
bn5c_branch2b (BatchNormalizat (None, 7, 7, 512) 2048 ['res5c_branch2b[0][0]']
ion)
add_15 (Add)                 (None, 7, 7, 512) 0 ['activation_30[0][0]',
                        'bn5c_branch2b[0][0]']
activation_32 (Activation)    (None, 7, 7, 512) 0 ['add_15[0][0]']
avg_pool (AveragePooling2D)   (None, 3, 3, 512) 0 ['activation_32[0][0]']
flatten (Flatten)            (None, 4608) 0 ['avg_pool[0][0]']
fc70 (Dense)                 (None, 70) 322630 ['flatten[0][0]']

Total params: 21,637,254
Trainable params: 21,620,102
Non-trainable params: 17,152
```

```
from tensorflow.keras.utils import plot_model
plot_model(model, to_file = 'Resnet34Base.png', show_shapes = True)
```



```

def mean(p_i):
    mean = 0
    # print('y_pred_i', y_pred_i)
    # print('Shape', tf.shape(y_pred_i))
    # for j in range(tf.shape(y_pred_i).numpy().shape[1]):
    for j in range(len(p_i)):
        mean += p_i[j]*(j+1)
    return mean

[38] def variance(p_i):
    variance = 0
    # print('y_pred_i', y_pred_i)
    # for j in range(tf.shape(y_pred_i).numpy().shape[1]):
    for j in range(len(p_i)):
        variance += p_i[j]*((j+1) - mean(p_i))**2
    return variance

[39] def mean_loss(y_true, p_pred):
    # print('y_pred', y_pred)
    # print('y_true', y_true)
    # print('Shape', p_pred.numpy().shape)
    m_tensor = tf.constant([mean(i) for i in p_pred.numpy()], dtype = 'float32')
    squared_difference = tf.square(tf.cast(y_true, dtype = 'float32') - m_tensor)
    return (1/2)*tf.reduce_mean(squared_difference, axis=-1)

[40] def variance_loss(y_true, p_pred):
    total_variance = 0
    # for i in range(tf.shape(y_pred_i).numpy().shape[1]):
    # print(p_pred.numpy().shape)
    for i in range(p_pred.numpy().shape[0]):
        total_variance += variance(p_pred.numpy()[i])
    return (1/p_pred.numpy().shape[0])*total_variance

[41] def loss(model, x, y_true, lambda1, lambda2, training):
    p_pred = model(x, training=training)
    # y_pred = tf.constant([mean(i) for i in p_pred.numpy()], dtype = 'float32')
    # print(y_pred)
    softmax_loss = tf.keras.losses.SparseCategoricalCrossentropy()
    return softmax_loss(y_true, p_pred) + lambda1*mean_loss(y_true, p_pred) + lambda2*variance_loss(y_true, p_pred)

[63] INIT_LEARNING_RATE = 0.001
    EPOCHS = 15
    LAMBDA_1 = 0.2
    LAMBDA_2 = 0.05

```

```

0s Ir_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate=INIT_LEARNING_RATE,
    decay_steps=200,
    decay_rate=0.1,
    staircase = True
)

[44] optimizer = tf.keras.optimizers.SGD(learning_rate=lr_schedule, momentum=0.9, nesterov=True)

[45] def grad(model, inputs, targets, lambda1, lambda2):
    with tf.GradientTape() as tape:
        loss_value = loss(model, inputs, targets, lambda1, lambda2, training=True)
    return loss_value, tape.gradient(loss_value, model.trainable_variables)

[46] import progressbar

def train(train_set, test_set, model, h_para = {}):
    train_loss_results = []
    train_mae_results = []
    test_loss_results = []
    test_mae_results = []

    num_epochs, lambda1, lambda2 = h_para['epochs'], h_para['l_1'], h_para['l_2']

    # widgets = ['Epochs: ', progressbar.Percentage(), " ", progressbar.Bar(), " ", progressbar.ETA()]
    # pbar = progressbar.ProgressBar(maxval = 1000, widgets = widgets).start()

    for epoch in range(num_epochs):
        epoch_train_loss_avg = tf.keras.metrics.Mean()
        epoch_train_mae = tf.keras.metrics.MeanAbsoluteError()
        epoch_test_loss_avg = tf.keras.metrics.Mean()
        epoch_test_mae = tf.keras.metrics.MeanAbsoluteError()

        # Training loop - using batches of 64
        for count, (x, y) in enumerate(train_set):
            #Calculate loss value and update weight
            loss_value, grads = grad(model, x, y, lambda1, lambda2)
            optimizer.apply_gradients(zip(grads, model.trainable_variables))

        # Track progress
        epoch_train_loss_avg.update_state(loss_value) # Add current batch loss
        # Compare predicted label to actual label
        # training=True is needed only if there are layers with different
        # behavior during training versus inference (e.g. Dropout).
        probs = model(x, training=True)
        # what = tf.nn.softmax(probs, axis=-1, output_dtype=tf.float32)

```

```

probs = model(x, training=True)
# y_hat = tf.argmax(probs, axis=1, output_type=tf.int32)
y_hat = tf.constant([mean(i) for i in probs.numpy()], dtype = 'float32')
y_hat = tf.round(y_hat)
epoch_train_mae.update_state(y, y_hat)

# pbar.update(count)

train_loss_results.append(epoch_train_loss_avg.result())
train_mae_results.append(epoch_train_mae.result())
# Validation loop batch size 16
for test_x, test_y in test_set:
    # test_y_hat = tf.argmax(test_probs, axis=1, output_type=tf.int32)
    test_loss_value, _ = grad(model, test_x, test_y, lambda1, lambda2)
    epoch_test_loss_avg.update_state(test_loss_value)

    test_probs = model(test_x, training=False)
    test_y_hat = tf.constant([mean(i) for i in test_probs.numpy()], dtype = 'float32')
    test_y_hat = tf.round(test_y_hat)
    epoch_test_mae.update_state(test_y, test_y_hat)

test_loss_results.append(epoch_test_loss_avg.result())
test_mae_results.append(epoch_test_mae.result())

if epoch % 5 == 0:
    print("\nEpoch {:03d}: Loss: {:.3f}, MAE: {:.3f}, Val_loss : {:.3f}, Val_MAE: {:.3f}".format(epoch,
                                                                                               epoch_train_loss_avg.result(),
                                                                                               epoch_train_mae.result(),
                                                                                               epoch_test_loss_avg.result(),
                                                                                               epoch_test_mae.result()))

# pbar.finish()

history = {'Loss': train_loss_results, 'MAE': train_mae_results, 'Val_loss': test_loss_results, 'Val_MAE': test_mae_results}
return history

```

```

[ ] h_para = {'epochs': EPOCHS, 'l_1': LAMBDA_1, 'l_2': LAMBDA_2}

history = train(train_dataset, test_dataset, model, h_para)

```

```

Epoch 000: Loss: 44.279, MAE: 11.963812, Val_loss : 29.265, Val_MAE: 12.454327
Epoch 005: Loss: 29.731, MAE: 9.704000, Val_loss : 30.428, Val_MAE: 10.423077
Epoch 010: Loss: 27.385, MAE: 9.392250, Val_loss : 28.918, Val_MAE: 10.105769
Epoch 015: Loss: 24.722, MAE: 8.730312, Val_loss : 28.445, Val_MAE: 10.105769

```

```
✓ [47] model.compile(optimizer='adam', loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])
```

```
✓ ▶ model.fit(trainData, trainLabels, batch_size=128, epochs=35)
```

```
Epoch 0/35  
6/6 [=====] - 7s 1s/step - loss: 3.6039 - accuracy: 0.0884  
Epoch 9/35  
6/6 [=====] - 7s 1s/step - loss: 3.7099 - accuracy: 0.1056  
Epoch 10/35  
6/6 [=====] - 7s 1s/step - loss: 3.4922 - accuracy: 0.1312  
Epoch 11/35  
6/6 [=====] - 7s 1s/step - loss: 3.3738 - accuracy: 0.1469  
Epoch 12/35  
6/6 [=====] - 7s 1s/step - loss: 3.2754 - accuracy: 0.1484  
Epoch 13/35  
6/6 [=====] - 7s 1s/step - loss: 3.1439 - accuracy: 0.1854  
Epoch 14/35  
6/6 [=====] - 7s 1s/step - loss: 3.2076 - accuracy: 0.1655  
Epoch 15/35  
6/6 [=====] - 7s 1s/step - loss: 2.9179 - accuracy: 0.2297  
Epoch 16/35  
6/6 [=====] - 7s 1s/step - loss: 2.9724 - accuracy: 0.2511  
Epoch 17/35  
6/6 [=====] - 7s 1s/step - loss: 2.7152 - accuracy: 0.2967  
Epoch 18/35  
6/6 [=====] - 7s 1s/step - loss: 2.4383 - accuracy: 0.3766  
Epoch 19/35  
6/6 [=====] - 7s 1s/step - loss: 2.4129 - accuracy: 0.3723  
Epoch 20/35  
6/6 [=====] - 7s 1s/step - loss: 2.4310 - accuracy: 0.3852  
Epoch 21/35  
6/6 [=====] - 7s 1s/step - loss: 2.1212 - accuracy: 0.4650  
Epoch 22/35  
6/6 [=====] - 7s 1s/step - loss: 1.9316 - accuracy: 0.5036  
Epoch 23/35  
6/6 [=====] - 7s 1s/step - loss: 1.6164 - accuracy: 0.5749  
Epoch 24/35  
6/6 [=====] - 7s 1s/step - loss: 1.2710 - accuracy: 0.6833  
Epoch 25/35  
6/6 [=====] - 7s 1s/step - loss: 0.9118 - accuracy: 0.7689  
Epoch 26/35  
6/6 [=====] - 7s 1s/step - loss: 0.7004 - accuracy: 0.8288  
Epoch 27/35  
6/6 [=====] - 7s 1s/step - loss: 0.5729 - accuracy: 0.8573  
Epoch 28/35  
6/6 [=====] - 7s 1s/step - loss: 0.5043 - accuracy: 0.8702  
Epoch 29/35  
6/6 [=====] - 7s 1s/step - loss: 0.3864 - accuracy: 0.8973  
Epoch 30/35  
6/6 [=====] - 7s 1s/step - loss: 0.3694 - accuracy: 0.8987
```

```
Epoch 30/35
6/6 [=====] - 7s 1s/step - loss: 0.4818 - accuracy: 0.8745
Epoch 31/35
6/6 [=====] - 7s 1s/step - loss: 0.4637 - accuracy: 0.8973
Epoch 32/35
6/6 [=====] - 7s 1s/step - loss: 0.5527 - accuracy: 0.8887
Epoch 33/35
6/6 [=====] - 7s 1s/step - loss: 0.4455 - accuracy: 0.9173
Epoch 34/35
6/6 [=====] - 7s 1s/step - loss: 0.3214 - accuracy: 0.9258
Epoch 35/35
6/6 [=====] - 7s 1s/step - loss: 0.2308 - accuracy: 0.9429
<keras.callbacks.History at 0x7fa7c003c490>
```

```
[ ] [(loss, accuracy)] = model.evaluate(testData, testLabels, batch_size = 128, verbose = 1)
print("accuracy: {:.2f}%".format(accuracy * 100)) #Testing Accuracy max=5.71
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/util/dispatch.py:1082: UserWarning: "`sparse_categorical_crossentropy` rec
return dispatch_target(*args, **kwargs)
3/3 [=====] - 3s 641ms/step - loss: 6.4520 - accuracy: 0.0465
accuracy: 4.65%
```

```
import matplotlib.pyplot as plt

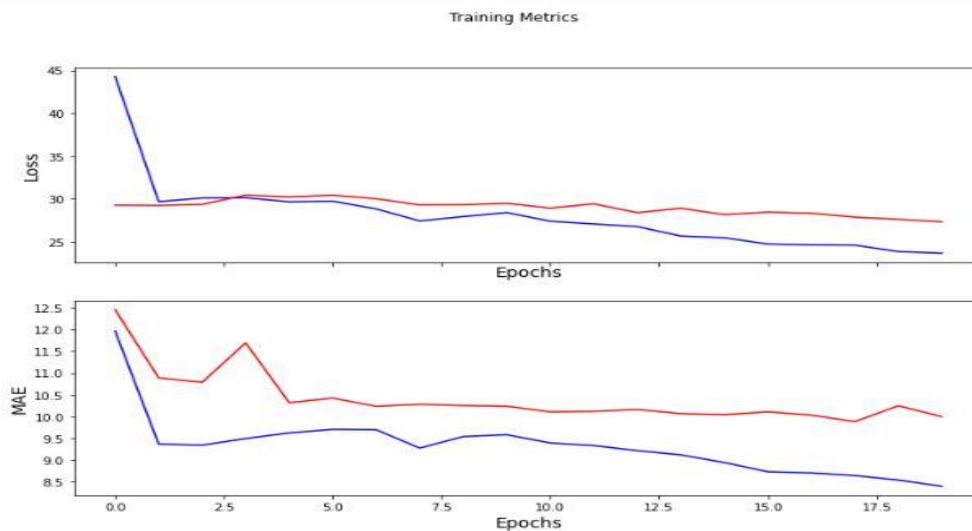
def plot_history(history):

    fig, axes = plt.subplots(2, sharex=True, figsize=(12, 8))
    fig.suptitle('Training Metrics')
    epochs = 20

    axes[0].set_xlabel("Epochs", fontsize=14)
    axes[0].set_ylabel("Loss", fontsize=14)
    axes[0].plot(history['Loss'], 'b')
    axes[0].plot(history['Val_loss'], 'r')

    axes[1].set_xlabel("Epochs", fontsize=14)
    axes[1].set_ylabel("MAE", fontsize=14)
    axes[1].plot(history['MAE'], 'b')
    axes[1].plot(history['Val_MAE'], 'r')
    plt.show()
```

```
[ ] plot_history(history)
```



TESTING

```
[47] test_image = testData[90]
test_image = tf.constant(test_image, dtype = 'float32')
test_image = tf.expand_dims(test_image, axis = 0)
prediction = model(test_image)
```

```
[48] prediction

<tf.Tensor: shape=(1, 70), dtype=float32, numpy=
array([[ 3.43913078e-01,  1.66526728e-03,  4.13009301e-02,  3.02118212e-02,
         1.14029966e-01,  6.39221910e-03,  1.22170732e-03,  9.52320918e-03,
         2.30569057e-02,  7.11284170e-04,  1.91412622e-03,  2.81124301e-02,
         8.17919243e-03,  4.10219058e-02,  3.08347098e-03,  1.29599990e-02,
         1.33815373e-03,  1.43407106e-01,  1.63424611e-01,  4.48539481e-03,
         2.40834386e-04,  3.93372757e-05,  2.96453596e-04,  5.22164349e-03,
         2.75978709e-05,  3.91154643e-03,  2.00181408e-03,  2.02596048e-03,
         1.98350332e-04,  1.23924983e-04,  3.91435251e-03,  3.80730198e-04,
         3.02795932e-04,  1.45127551e-05,  1.61010334e-08,  3.17014928e-05,
         4.20687895e-04,  6.77102179e-08,  8.52160920e-09,  1.57865943e-05,
         2.69935554e-04,  1.26065133e-04,  1.33556823e-05,  3.73881790e-06,
         1.15196983e-07,  5.00801107e-05,  1.62215588e-06,  2.72187259e-04,
         1.70685311e-07,  3.26639451e-07,  1.49595871e-06,  1.40543621e-09,
         2.11495399e-06,  6.94541161e-11,  3.69520848e-07,  7.45481930e-06,
         9.09597786e-10,  2.04700167e-12,  1.34155360e-11,  1.87195551e-10,
         4.69726047e-07,  1.18482649e-05,  6.55540935e-05,  1.26608973e-10,
         1.48338286e-10,  9.05788430e-11,  1.02027956e-10,  5.22194387e-05,
         7.29408114e-12,  4.91173380e-10]], dtype=float32)>
```

```
[49] prediction = tf.argmax(prediction, axis = 1, output_type=tf.int32)
prediction

<tf.Tensor: shape=(1,), dtype=int32, numpy=array([0], dtype=int32)>
```

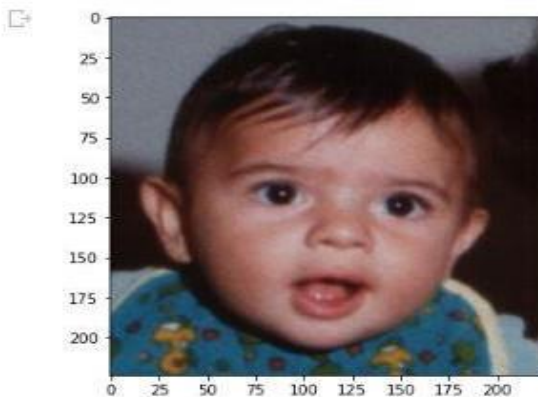
```
[50] age_pred = int(prediction.numpy())
print('Prediction', age_pred)

Prediction 0
```

```
[51] age_true = int(testLabels[90])
print('Ground true', age_true)
```

Ground true 0

```
[52] plt.figure(figsize=(5,5))
plt.imshow(testData[90])
plt.show()
```



▼ Comparing with Sequential Model

```
[ ] from sklearn.model_selection import train_test_split

images = images.astype('float')/255

(trainData, testData, trainLabels, testLabels) = train_test_split(images, ages, test_size = 0.3, random_state = 42, shuffle =True)
```

```
[ ] import tensorflow as tf
train_dataset = tf.data.Dataset.from_tensor_slices((trainData, trainLabels))
test_dataset = tf.data.Dataset.from_tensor_slices((testData, testLabels))
```

```
▶ import tensorflow as tf

model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(filters=16, kernel_size=(3,3), activation='relu',input_shape=(224, 224, 3)),
    tf.keras.layers.Conv2D(filters=32, kernel_size=(3,3), activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=(2,2)),
    tf.keras.layers.BatchNormalization(axis=-1),

    tf.keras.layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu'),
    tf.keras.layers.Conv2D(filters=128, kernel_size=(3,3), activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=(2,2)),
    tf.keras.layers.BatchNormalization(axis=-1),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512,activation='relu'),
    tf.keras.layers.BatchNormalization() ,
    tf.keras.layers.Dropout(rate=0.5),

    tf.keras.layers.Dense(3,activation='softmax')
])

learning_rate = 0.001
epochs=35
opt= tf.keras.optimizers.Adam(learning_rate=learning_rate , decay=learning_rate/(epochs*0.5))
model.compile(loss='sparse_categorical_crossentropy',optimizer=opt,metrics=['accuracy'])

aug = tf.keras.preprocessing.image.ImageDataGenerator(
    rotation_range=10,
```

```

width_shift_range=0.1,
height_shift_range=0.1,
shear_range=0.15,
horizontal_flip= False,
vertical_flip= False,
fill_mode="nearest"
)

trainData = tf.random.normal((50, 224, 224, 3))
trainLabels = tf.random.uniform((50, ), maxval=3, dtype=tf.int32)
history = model.fit(aug.flow(trainData, trainLabels, batch_size=2), epochs=epochs)#training accuracy

```

```

Epoch 1/35
25/25 [=====] - 3s 58ms/step - loss: 2.8038 - accuracy: 0.3400
Epoch 2/35
25/25 [=====] - 1s 48ms/step - loss: 2.7423 - accuracy: 0.4200
Epoch 3/35
25/25 [=====] - 1s 49ms/step - loss: 2.6322 - accuracy: 0.3600
Epoch 4/35
25/25 [=====] - 1s 49ms/step - loss: 2.2750 - accuracy: 0.2800
Epoch 5/35
25/25 [=====] - 1s 49ms/step - loss: 1.4792 - accuracy: 0.4000
Epoch 6/35
25/25 [=====] - 1s 48ms/step - loss: 1.4582 - accuracy: 0.4600
Epoch 7/35
25/25 [=====] - 1s 49ms/step - loss: 1.6196 - accuracy: 0.3800
Epoch 8/35
25/25 [=====] - 1s 48ms/step - loss: 1.6187 - accuracy: 0.4400
Epoch 9/35
25/25 [=====] - 1s 48ms/step - loss: 1.4319 - accuracy: 0.4400
Epoch 10/35
25/25 [=====] - 1s 48ms/step - loss: 1.8414 - accuracy: 0.3000
Epoch 11/35
25/25 [=====] - 1s 48ms/step - loss: 1.7221 - accuracy: 0.4000
Epoch 12/35
25/25 [=====] - 1s 49ms/step - loss: 1.5497 - accuracy: 0.3200
Epoch 13/35
25/25 [=====] - 1s 48ms/step - loss: 1.4706 - accuracy: 0.3800
Epoch 14/35
25/25 [=====] - 1s 48ms/step - loss: 1.7160 - accuracy: 0.3400
Epoch 15/35
25/25 [=====] - 1s 49ms/step - loss: 1.3519 - accuracy: 0.4600
Epoch 16/35
25/25 [=====] - 1s 48ms/step - loss: 1.4218 - accuracy: 0.4000
Epoch 17/35
25/25 [=====] - 1s 49ms/step - loss: 1.3944 - accuracy: 0.4000
Epoch 18/35
25/25 [=====] - 1s 49ms/step - loss: 1.4734 - accuracy: 0.3400
Epoch 19/35
25/25 [=====] - 1s 50ms/step - loss: 1.2949 - accuracy: 0.4200
Epoch 20/35
25/25 [=====] - 1s 48ms/step - loss: 1.5459 - accuracy: 0.3400

```

```

Epoch 21/35
25/25 [=====] - 1s 48ms/step - loss: 1.4382 - accuracy: 0.4000
Epoch 22/35
25/25 [=====] - 1s 48ms/step - loss: 1.6594 - accuracy: 0.2400
Epoch 23/35
25/25 [=====] - 1s 48ms/step - loss: 1.2617 - accuracy: 0.4600
Epoch 24/35
25/25 [=====] - 1s 49ms/step - loss: 1.0294 - accuracy: 0.5800
Epoch 25/35
25/25 [=====] - 1s 49ms/step - loss: 1.4643 - accuracy: 0.4600
Epoch 26/35
25/25 [=====] - 1s 48ms/step - loss: 1.5502 - accuracy: 0.2800
Epoch 27/35
25/25 [=====] - 1s 49ms/step - loss: 1.7699 - accuracy: 0.2800
Epoch 28/35
25/25 [=====] - 1s 49ms/step - loss: 1.5374 - accuracy: 0.3400
Epoch 29/35
25/25 [=====] - 1s 49ms/step - loss: 1.3033 - accuracy: 0.5200
Epoch 30/35
25/25 [=====] - 1s 48ms/step - loss: 1.5423 - accuracy: 0.3400
Epoch 31/35
25/25 [=====] - 1s 48ms/step - loss: 1.4449 - accuracy: 0.3600
Epoch 32/35
25/25 [=====] - 1s 49ms/step - loss: 1.5026 - accuracy: 0.3600
Epoch 33/35
25/25 [=====] - 1s 49ms/step - loss: 1.4067 - accuracy: 0.2800
Epoch 34/35
25/25 [=====] - 1s 48ms/step - loss: 1.7168 - accuracy: 0.2600
Epoch 35/35
25/25 [=====] - 1s 49ms/step - loss: 1.2479 - accuracy: 0.5000

```

```

[loss, accuracy] = model.evaluate(testData, testLabels, batch_size = 128, verbose = 1)
print("accuracy: {:.2f}%".format(accuracy * 100)) #testing accuracy

```

```

3/3 [=====] - 2s 679ms/step - loss: nan - accuracy: 0.0266
accuracy: 2.66%

```

model.summary()

Model: "sequential_3"

Layer (type)	Output Shape	Param #
conv2d_12 (Conv2D)	(None, 222, 222, 16)	448
conv2d_13 (Conv2D)	(None, 220, 220, 32)	4640
max_pooling2d_7 (MaxPooling 2D)	(None, 110, 110, 32)	0
batch_normalization_9 (Batch Normalization)	(None, 110, 110, 32)	128
conv2d_14 (Conv2D)	(None, 108, 108, 64)	18496
conv2d_15 (Conv2D)	(None, 106, 106, 128)	73856
max_pooling2d_8 (MaxPooling 2D)	(None, 53, 53, 128)	0
batch_normalization_10 (Batch Normalization)	(None, 53, 53, 128)	512
flatten_4 (Flatten)	(None, 359552)	0
dense_6 (Dense)	(None, 512)	184091136
batch_normalization_11 (Batch Normalization)	(None, 512)	2048
dropout_3 (Dropout)	(None, 512)	0
dense_7 (Dense)	(None, 3)	1539
Total params: 184,192,803		
Trainable params: 184,191,459		
Non-trainable params: 1,344		

B. SOURCE CODE

Importing the dataset

```
from google.colab import drive
drive.mount('/content/drive')
%cd /content/drive/MyDrive/
import os
import zipfile
zip_ref=zipfile.ZipFile("/content/drive/MyDrive/FGNET.zip",'r')
zip_ref.extractall('/content/drive/MyDrive/FG_NET')
zip_ref.close()
!ls "/content/drive/MyDrive/FG_NET/FGNET"
import os
data_dir="/content/drive/MyDrive/FG_NET/FGNET/images"
files=os.listdir(data_dir)
print(len(files))
import numpy as np
ages = []
for age in files:
    age = age[age.find('A') + 1: age.find('A') + 1 + 2]
    # print(age)
    ages.append(int(age))

ages = np.array(ages)
len(ages)
```

Perform Data Exploration

```
import matplotlib.pyplot as plt

def _plot_bar(varname):
    labels, values = np.unique(ages, return_counts=True)
    plt.figure(figsize=(10, 6))
    ax = plt.subplot()
    ax.bar(x=labels, height=values)
    ax.set_title('No samples of ' + varname)
    ax.set_xlabel(varname)
    ax.set_ylabel('No samples')
    plt.show()
_plot_bar('ages')
import pandas as pd

labels, values = np.unique(ages, return_counts=True)
age_df = pd.DataFrame(data = values, index = labels, columns = ['Freq'])

age_df
age_df.describe()
age_df.loc[age_df.idxmax()]
age_df.loc[age_df.idxmin()]
```

```

age_lst = list(age_df.index.values)

#Find missing ages
miss_ages = []
for age in range(70):
    if age not in age_lst:
        miss_ages.append(age)
miss_ages

import numpy as np
from PIL import Image
import progressbar

widgets = ['Reading images:', progressbar.Percentage(), " ", progressbar.Bar(),
" ", progressbar.ETA()]
pbar = progressbar.ProgressBar(maxval = len(files), widgets = widgets).start()

images = []
for i, imageData in enumerate(files):
    image = np.array(Image.open(data_dir + '/' + imageData).convert('RGB').resize(
((224, 224))))
    images.append(image)
    pbar.update(i)

pbar.finish()
images = np.array(images)
from matplotlib import image
plt.figure(figsize = (6, 6))
plt.imshow(images[10])
plt.show()
print(ages[10])
import matplotlib
def displayImage(startIdx, numImages, imagePath, showAge = True):
    endIdx = startIdx + numImages
    size = (5,5)
    fig = plt.figure(figsize = (3*numImages, 2*numImages))
    listImage = os.listdir(imagePath)
    for i, imagename in enumerate(listImage[startIdx : endIdx]):
        ax = fig.add_subplot(size[0], size[1], i + 1, xticks = [], yticks = [])
        img = matplotlib.image.imread(imagePath + '/' + imagename)
        ax.imshow(img)
        if showAge:
            ax.set_title('Imagename: {}, Age: {}'.format(imagename, ages[startIdx +
i]), color = 'green')
displayImage(20, 10, '/content/drive/MyDrive/FG_NET/FGNET/images')

```

Split data in training set and validation set

```

from sklearn.model_selection import train_test_split
images = images.astype('float')/255
(trainData, testData, trainLabels, testLabels) = train_test_split(images, age
s, test_size = 0.3, random_state = 42, shuffle =True)

```

Create Dataset with tf.data.Dataset

```
import tensorflow as tf
train_dataset = tf.data.Dataset.from_tensor_slices((trainData, trainLabels))
test_dataset = tf.data.Dataset.from_tensor_slices((testData, testLabels))
```

Perform Data Augmentation

```
def augment(image, label, mode):
    if mode == 'train':
        # image = tf.image.resize_with_crop_or_pad(image, 230, 230) # Add 6 pixels of padding
        image = tf.image.random_crop(image, size=[218, 218, 3])
        image = tf.image.resize(image, size=[224, 224])
        image = tf.image.random_saturation(image, 2, 3)
        image = tf.image.random_brightness(image, max_delta=0.5) # Random brightness
        image = tf.image.random_flip_left_right(image)
        return image, label
    else:
        image_cropped = tf.image.crop_to_bounding_box(image, offset_height=20, offset_width=0, target_height=200, target_width=224)
        image_resized = tf.image.resize(image_cropped, size=[224, 224])
        return image_resized, label

BATCH_SIZE_TRAIN = 64
BATCH_SIZE_TEST = 16
SHUFFLE_BUFFER_SIZE = 100

train_dataset = train_dataset.shuffle(SHUFFLE_BUFFER_SIZE).map(lambda image, label: augment(image, label, mode = 'train'), num_parallel_calls = tf.data.experimental.AUTOTUNE).batch(BATCH_SIZE_TRAIN)
test_dataset = test_dataset.map(lambda image, label: augment(image, label, mode = 'eval'), num_parallel_calls = tf.data.experimental.AUTOTUNE).batch(BATCH_SIZE_TEST)
```

BUILDING MODEL BASE ON RESNET 34 ARCHITECTURE

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Activation, Flatten,
    Dense, BatchNormalization, ZeroPadding2D, Add, Input, AveragePooling2D, GlobalMaxPooling2D, Lambda
from tensorflow.keras.initializers import glorot_uniform
import tensorflow as tf

def identity_block(X, f, filters, stage, block):
    conv_name_base = 'res' + str(stage) + block + '_branch'
    bn_name_base = 'bn' + str(stage) + block + '_branch'

    F1, F2 = filters

    X_shortcut = X
    X = Conv2D(filters = F1, kernel_size= (f,f), strides= (1,1), padding= 'same'
, name = conv_name_base + '2a', kernel_initializer=glorot_uniform(seed = 0))(X
)
    X = BatchNormalization(axis = -1, name = bn_name_base + '2a')(X)
    X = Activation('relu')(X)

    X = Conv2D(filters = F2, kernel_size= (f,f), strides= (1,1), padding= 'same'
, name = conv_name_base + '2b', kernel_initializer=glorot_uniform(seed = 0))(X
)
    X = BatchNormalization(axis = -1, name = bn_name_base + '2b')(X)
    X = Add()([X_shortcut, X])
    X = Activation('relu')(X)
    return X

def convolutional_block(X, f, filters, stage, block, s = 2):
    conv_name_base = 'res' + str(stage) + block + '_branch'
    bn_name_base = 'bn' + str(stage) + block + '_branch'

    F1, F2 = filters

    X_shortcut = X

    X = Conv2D(filters = F1, kernel_size= (f,f), strides= (s,s), padding= 'same'
, name = conv_name_base + '2a', kernel_initializer=glorot_uniform(seed = 0))(X
)
    X = BatchNormalization(axis = -1, name = bn_name_base + '2a')(X)
    X = Activation('relu')(X)
```

```

    X = Conv2D(filters = F2, kernel_size=(f,f), strides=(1,1), padding='same'
, name = conv_name_base + '2b', kernel_initializer=glorot_uniform(seed = 0))(X
)
    X = BatchNormalization(axis = -1, name = bn_name_base + '2b')(X)

    X_shortcut = Conv2D(filters = F2, kernel_size=(1,1), strides = (s,s), name =
conv_name_base + '1', kernel_initializer=glorot_uniform(seed = 0))(X_shortcut
)
    X_shortcut = BatchNormalization(axis = -
1, name = bn_name_base + '1')(X_shortcut)

    X = Add()([X_shortcut, X])
    X = Activation('relu')(X)
    return X
def ResNet_34Base(input_shape, classes, activation = 'softmax'):
    X_input = Input(input_shape)

    X = ZeroPadding2D((3,3))(X_input)
    #Stage 1
    X = Conv2D(64, (7, 7), strides = (2, 2), name = 'conv1', kernel_initializer
= glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis = 3, name = 'bn_conv1')(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((3, 3), strides=(2, 2))(X)
    #Stage 2
    X = convolutional_block(X, f = 3, filters = [64, 64], stage = 2, block = 'a'
, s = 1)
    X = identity_block(X, f = 3, filters= [64, 64], stage = 2, block = 'b')
    X = identity_block(X, f = 3, filters= [64, 64], stage = 2, block = 'c')
    #Stage 3
    X = convolutional_block(X, f = 3, filters = [128, 128], stage = 3, block = '
a', s = 2)
    X = identity_block(X, f = 3, filters= [128, 128], stage = 3, block = 'b')
    X = identity_block(X, f = 3, filters= [128, 128], stage = 3, block = 'c')
    X = identity_block(X, f = 3, filters= [128, 128], stage = 3, block = 'd')
    #Stage 4
    X = convolutional_block(X, f = 3, filters = [256, 256], stage = 4, block = 'a
', s = 2)
    X = identity_block(X, f = 3, filters= [256, 256], stage = 4, block = 'b')
    X = identity_block(X, f = 3, filters= [256, 256], stage = 4, block = 'c')
    X = identity_block(X, f = 3, filters= [256, 256], stage = 4, block = 'd')
    X = identity_block(X, f = 3, filters= [256, 256], stage = 4, block = 'e')
    X = identity_block(X, f = 3, filters= [256, 256], stage = 4, block = 'f')

```

```

#Stage 5
X = convolutional_block(X, f = 3, filters = [512, 512], stage = 5, block = 'a', s = 2)
X = identity_block(X, f = 3, filters= [512, 512], stage = 5, block = 'b')
X = identity_block(X, f = 3, filters= [512, 512], stage = 5, block = 'c')
X = AveragePooling2D(pool_size=(2,2), name = 'avg_pool')(X)
X = Flatten()(X)
X = Dense(classes, activation= 'softmax', name = 'fc' + str(classes), kernel_initializer=glorot_uniform(seed = 0))(X)
model = Model(inputs = X_input, outputs = X, name = 'age_resnet_34')
return model
model = ResNet_34Base((224, 224, 3), 70)
model.summary()

```

Model Architecture

```

from tensorflow.keras.utils import plot_model
plot_model(model, to_file = 'Resnet34Base.png', show_shapes = True)
def mean(p_i):
    mean = 0
    for j in range(len(p_i)):
        mean += p_i[j]*(j+1)
    return mean
def variance(p_i):
    variance = 0
    for j in range(len(p_i)):
        variance += p_i[j]*((j+1) - mean(p_i))**2
    return variance
def mean_loss(y_true, p_pred):
    # print('y_pred', y_pred)
    # print('y_true', y_true)
    # print('Shape', p_pred.numpy().shape)
    m_tensor = tf.constant([mean(i) for i in p_pred.numpy()], dtype = 'float32')
    squared_difference = tf.square(tf.cast(y_true, dtype = 'float32') - m_tensor)
    return (1/2)*tf.reduce_mean(squared_difference, axis=-1)
def variance_loss(y_true, p_pred):
    total_variance = 0
    # for i in range(tf.shape(y_pred_i).numpy().shape[1]):
    # print(p_pred.numpy().shape)
    for i in range(p_pred.numpy().shape[0]):
        total_variance += variance(p_pred.numpy()[i])
    return (1/p_pred.numpy().shape[0])*total_variance
def loss(model, x, y_true, lambda1, lambda2, training):
    p_pred = model(x, training=training)

```

```

    # y_pred = tf.constant([mean(i) for i in p_pred.numpy()], dtype = 'float32')
    # print(y_pred)
    softmax_Loss = tf.keras.losses.SparseCategoricalCrossentropy()
    return softmax_Loss(y_true, p_pred) + lambda1*mean_loss(y_true, p_pred) + lambda2*variance_loss(y_true, p_pred)
INIT_LEARNING_RATE = 0.001
EPOCHS = 15
LAMBDA_1 = 0.2
LAMBDA_2 = 0.05
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate=INIT_LEARNING_RATE,
    decay_steps=200,
    decay_rate=0.1,
    staircase = True
)
optimizer = tf.keras.optimizers.SGD(learning_rate=lr_schedule, momentum=0.9, nesterov=True)
def grad(model, inputs, targets, lambda1, lambda2):
    with tf.GradientTape() as tape:
        loss_value = loss(model, inputs, targets, lambda1, lambda2, training=True)
    return loss_value, tape.gradient(loss_value, model.trainable_variables)
import progressbar

def train(train_set, test_set, model, h_para = {}):
    train_loss_results = []
    train_mae_results = []
    test_loss_results = []
    test_mae_results = []

    num_epochs, lambda1, lambda2 = h_para['epochs'], h_para['l_1'], h_para['l_2']

    for epoch in range(num_epochs):
        epoch_train_loss_avg = tf.keras.metrics.Mean()
        epoch_train_mae = tf.keras.metrics.MeanAbsoluteError()
        epoch_test_loss_avg = tf.keras.metrics.Mean()
        epoch_test_mae = tf.keras.metrics.MeanAbsoluteError()

        # Training loop - using batches of 64
        for count, (x, y) in enumerate(train_set):
            #Calculate loss value and update weight
            loss_value, grads = grad(model, x, y, lambda1, lambda2)
            optimizer.apply_gradients(zip(grads, model.trainable_variables))

```

```

# Track progress
epoch_train_loss_avg.update_state(loss_value) # Add current batch loss
# Compare predicted label to actual label
# training=True is needed only if there are layers with different
# behavior during training versus inference (e.g. Dropout).
probs = model(x, training=True)
# y_hat = tf.argmax(probs, axis=1, output_type=tf.int32)
y_hat = tf.constant([mean(i) for i in probs.numpy()], dtype = 'float32')
y_hat = tf.round(y_hat)
epoch_train_mae.update_state(y, y_hat)
# pbar.update(count)
train_loss_results.append(epoch_train_loss_avg.result())
train_mae_results.append(epoch_train_mae.result())
#Validation loop batch size 16
for test_x, test_y in test_set:
    # test_y_hat = tf.argmax(test_probs, axis=1, output_type=tf.int32)
    test_loss_value, _ = grad(model, test_x, test_y, lambda1, lambda2)
    epoch_test_loss_avg.update_state(test_loss_value)

    test_probs = model(test_x, training=False)
    test_y_hat = tf.constant([mean(i) for i in test_probs.numpy()], dtype =
'float32')
    test_y_hat = tf.round(test_y_hat)
    epoch_test_mae.update_state(test_y, test_y_hat)

    test_loss_results.append(epoch_test_loss_avg.result())
    test_mae_results.append(epoch_test_mae.result())

    if epoch % 5 == 0:
        print("\nEpoch {:03d}: Loss: {:.3f}, MAE: {:.3f}, Val_loss : {:.3f}, Va
l_MAE: {:.3f}".format(epoch,
                                                                    epoch_train
n_loss_avg.result(),
                                                                    epoch_train
n_mae.result(),
                                                                    epoch_test
_loss_avg.result(),
                                                                    epoch_test
_mae.result()))

    history = {'Loss':train_loss_results, 'MAE': train_mae_results, 'Val_loss'
:test_loss_results, 'Val_MAE':test_mae_results}
    return history

```

```

h_para = {'epochs':EPOCHS, 'l_1': LAMBDA_1, 'l_2' : LAMBDA_2}

history = train(train_dataset, test_dataset, model, h_para)
model.compile(optimizer='adam',loss = tf.keras.losses.SparseCategoricalCross
entropy(from_logits=True), metrics=['accuracy'])
model.fit(trainData, trainLabels,batch_size=128,epochs=35)
(loss, accuracy) = model.evaluate(testData, testLabels, batch_size = 128, ve
rbose = 1)
print("accuracy: {:.2f}%".format(accuracy * 100))
import matplotlib.pyplot as plt

def plot_history(history):

    fig, axes = plt.subplots(2, sharex=True, figsize=(12, 8))
    fig.suptitle('Training Metrics')
    epochs = 20
    axes[0].set_xlabel("Epochs", fontsize=14)
    axes[0].set_ylabel("Loss", fontsize=14)
    axes[0].plot(history['Loss'], 'b')
    axes[0].plot(history['Val_loss'], 'r')

    axes[1].set_xlabel("Epochs", fontsize=14)
    axes[1].set_ylabel("MAE", fontsize=14)
    axes[1].plot(history['MAE'], 'b')
    axes[1].plot(history['Val_MAE'], 'r')
    plt.show()
plot_history(history)

```

TESTING

```

test_image = testData[90]
test_image = tf.constant(test_image, dtype = 'float32')
test_image = tf.expand_dims(test_image, axis = 0)
prediction = model(test_image)
prediction
prediction = tf.argmax(prediction, axis = 1, output_type=tf.int32)
prediction
age_pred = int(prediction.numpy())
print('Prediction', age_pred)
age_true = int(testLabels[90])
print('Ground true', age_true)
plt.figure(figsize=(5,5))
plt.imshow(testData[90])
plt.show()

```

Comparing with Sequential Model

```
from sklearn.model_selection import train_test_split

images = images.astype('float')/255

(trainData, testData, trainLabels, testLabels) = train_test_split(images, ages,
    test_size = 0.3, random_state = 42, shuffle =True)
import tensorflow as tf
train_dataset = tf.data.Dataset.from_tensor_slices((trainData, trainLabels))
test_dataset = tf.data.Dataset.from_tensor_slices((testData, testLabels))
import tensorflow as tf

model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(filters=16, kernel_size=(3,3), activation='relu',inp
ut_shape=(224, 224, 3)),
    tf.keras.layers.Conv2D(filters=32, kernel_size=(3,3), activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=(2,2)),
    tf.keras.layers.BatchNormalization(axis=-1),

    tf.keras.layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu'),
    tf.keras.layers.Conv2D(filters=128, kernel_size=(3,3), activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=(2,2)),
    tf.keras.layers.BatchNormalization(axis=-1),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512,activation='relu'),
    tf.keras.layers.BatchNormalization() ,
    tf.keras.layers.Dropout(rate=0.5),

    tf.keras.layers.Dense(3,activation='softmax')

])

learning_rate = 0.001
epochs=35
opt= tf.keras.optimizers.Adam(learning_rate=learning_rate , decay=learning_rate
/(epochs*0.5))
model.compile(loss='sparse_categorical_crossentropy',optimizer=opt,metrics=['ac
curacy'])

aug = tf.keras.preprocessing.image.ImageDataGenerator(
    rotation_range=10,
```

```

zoom_range=0.15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.15,
    horizontal_flip= False,
    vertical_flip= False,
    fill_mode="nearest"
)

trainData = tf.random.normal((50, 224, 224, 3))
trainLabels = tf.random.uniform((50, ), maxval=3, dtype=tf.int32)
history = model.fit(aug.flow(trainData, trainLabels, batch_size=2), epochs=epochs) #training accuracy
(loss, accuracy) = model.evaluate(testData, testLabels, batch_size = 128, verbose = 1)
print("accuracy: {:.2f}%".format(accuracy * 100)) #testing accuracy
model.summary()

```