Neural Networks
Project Submission
Manvitha Kanmantha Reddy
UIN: 427007573

1. **Topic:**
   I have built a CNN network that when inputted with an human image tries to classify it to one of the 8 age categories: (0-2. 4-6, 8-13, 15-20, 25-32, 38-43, 48-53, 60+).
   Reference Paper:
   https://talhassner.github.io/home/projects/cnn_agegender/CVPR2015_CNN_AgeGenderEstimation.pdf

2. **Dataset:**
   I used Aligned images of Adience Dataset. It had total 17423 images with labels. All these images are colored and I have reshaped them to 227 X 227 while inputting to my CNN. But the dataset is very noisy i.e., there are many blurred images, images captured under very bad lighting conditions and some pictures even had more than 1 person in them. Also the small dataset was one of the bottlenecks for not building a deeper neural network which leads to overfitting.
   http://www.cslab.openu.ac.il/download/

   2.a Data-Preprocessing:
   I created one more dataset by running face detection algorithm on all the images and extracting only facial portion from images. If any extracted image had 2 faces, I have chosen the face that is most dominant in that image and then resized all the images to 227 X 227.If there is an image which is so blurred or tilted in such a way that face detection is difficult then I have kept the original image as it is.

3. **Implementation Details:**
   I trained the network from scratch using the CNN model as stated below. After 40 epochs with 0.001 learning rate and decay rate (10^-5) with SGD optimizer it has attained an accuracy of 42.8 % ± 3.

   Now I loaded this trained model and fine tuned it using the cropped face images (i.e., which are obtained by running face detection algorithms on original images) with all other parameters being kept same. After 40 epochs this model has attained an accuracy 43.2 ± 4 with highest observed 47.12 %.

   This increment can be attributed to using facial crop images as face plays a major role in determining a person's age and also most of these facial images get rid of disturbing background thus leading to better learning of our neural network. One notable point here is even the cropped face images still had these blurred Images.

3.1 Model Architecture:

It is a sequential model with 3 convd_blocks. There are 3 convolution layers each followed by max pool layer and then batch normalization layers. After the convolution part, I have added 2 dense layers with a 0.5% dropout and then final output layer. For all layers I have used “relu” activation except for the output layer where softmax activation is being used.

```
#MODEL ARCHITECTURE
def baseModel():
        model = Sequential()
        model.add(layers.Conv2D(96, (7, 7), activation='relu', input_shape=(227, 227, 3), kernel_initializer=initializers.random_normal(stddev=0.01)))
        model.add(layers.MaxPooling2D((3, 3), strides=(2,2)))
        model.add(BatchNormalization())

        model.add(layers.Conv2D(256, (5, 5), activation='relu', kernel_initializer=initializers.random_normal(stddev=0.01)))
        model.add(layers.MaxPooling2D((3, 3), strides=(2,2) ))
        model.add(BatchNormalization())

        model.add(layers.Conv2D(384, (3, 3), activation='relu', kernel_initializer=initializers.random_normal(stddev=0.01)))
        model.add(layers.MaxPooling2D((3, 3)))

        model.add(layers.Flatten())
        model.add(layers.Dense(512, activation='relu', kernel_initializer=initializers.random_normal(stddev=0.01)))
        model.add(Dropout(0.5))

        model.add(layers.Dense(256, activation='relu', kernel_initializer=initializers.random_normal(stddev=0.01)))
        model.add(Dropout(0.5))
        model.add(layers.Dense(8, activation='softmax'))
        return model
```

All the layer weights are random initialized with zero mean Gaussian distribution and standard deviation of 0.01.

First Convolution layer used 96 filters of size 7 X 7 and stride (2,2)
Second Convolution layer has 256 filters of size 5 X 5 with stride (2,2)
Third Convolution layer has 384 filters of size 3 X 3

Model.summary:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 221, 221, 96)      14208
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 110, 110, 96)      0
_________________________________________________________________
batch_normalization_1 (Batch (None, 110, 110, 96)      384
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 106, 106, 256)     614656
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 52, 52, 256)       0
_________________________________________________________________
batch_normalization_2 (Batch (None, 52, 52, 256)       1024
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 50, 50, 384)       885120
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 16, 16, 384)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 98304)             0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               50332160
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 8)                 2056
=================================================================
Total params: 51,980,936
Trainable params: 51,980,232
Non-trainable params: 704
_________________________________________________________________
None
```

My total data is tentatively divided into 11000 train images, 2500 test Images and 3500 Validation Images

3.2 Input: Shape of Tensor

X_Train shape (11000, 227, 227)
X_Test shape (2500, 227, 227)

3.3 Output: Shape of Tensor

Y_Train shape (11000,)
Y_Test shape (2500,)

3.4 Shape of Output tensor for each layer:

First Convolution layer: (96, 221, 221)
Max_pooling layer: (96, 110, 110)
Batch_normalization(96,110,110)

First Convolution layer: (256, 106, 106)
Max_pooling layer: (256, 52, 52)
Batch_normalization(256,52,52)

First Convolution layer: (384, 50, 50)
Max_pooling layer: (384, 16, 16)

Flatten Layer: 98304
First Dense Layer: 512
Second Dense Layer: 256

4. HyperParameters

4.1 Various HyperParameters Tried:

Batch Size 32, 50, 64
Epochs 10 - 40
Learning Rates 0.01, 0.001, 0.0001

4.2 Optimal Parameters Found:

Batch Size: 50
Epochs 40
Learning rate: 0.001
Optimizer = Stochastic Gradient Descent
Learning rate = started with 0.001 with decay of (10 ^ -5)

5. Code:

It has 2 parts. One is Neural network code and other is for face detection and creation of cropped face images from a given Image.

crop+align+15.txt ×

*8class_model.py

```python
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential, Model, load_model
from keras import layers
from keras.layers import Dropout, Flatten, Dense, Input
from keras import applications
from keras.applications.vgg16 import VGG16
from keras.layers import Conv2D, Convolution2D, MaxPooling2D, ZeroPadding2D
from keras.optimizers import RMSprop, SGD, Adam
import numpy as np
import os
from keras.layers.normalization import BatchNormalization
from keras.callbacks import ModelCheckpoint
from keras import initializers

#MODEL ARCHITECTURE
def baseModel():
        model = Sequential()
        model.add(layers.Conv2D(96, (7, 7), activation='relu', input_shape=(227, 227, 3), kernel_initializer=initializers.random_normal(stddev=0.01)))
        model.add(layers.MaxPooling2D((3, 3), strides=(2,2)))
        model.add(BatchNormalization())

        model.add(layers.Conv2D(256, (5, 5), activation='relu', kernel_initializer=initializers.random_normal(stddev=0.01)))
        model.add(layers.MaxPooling2D((3, 3), strides=(2,2) ))
        model.add(BatchNormalization())

        model.add(layers.Conv2D(384, (3, 3), activation='relu', kernel_initializer=initializers.random_normal(stddev=0.01)))
        model.add(layers.MaxPooling2D((3, 3)))

        model.add(layers.Flatten())
        model.add(layers.Dense(512, activation='relu', kernel_initializer=initializers.random_normal(stddev=0.01)))
        model.add(Dropout(0.5))

        model.add(layers.Dense(256, activation='relu', kernel_initializer=initializers.random_normal(stddev=0.01)))
        model.add(Dropout(0.5))
        model.add(layers.Dense(8, activation='softmax'))
        return model


#GENERATING IMAGE DATA GENERATORS
def baseModelDataGen(trainData, validData, testData):
        batchSize = 50
        train_datagen = ImageDataGenerator(rescale=1./255, width_shift_range=0.2,height_shift_range=0.2,shear_range=0.2,zoom_range=0.2,horizontal_flip=True,fill_mode='nearest')
        train_generator = train_datagen.flow_from_directory(trainData, target_size = (227, 227), batch_size = batchSize, class_mode = "categorical", shuffle = True)

        valid_datagen = ImageDataGenerator(rescale=1./255, width_shift_range=0.2,height_shift_range=0.2,shear_range=0.2,zoom_range=0.2,horizontal_flip=True,fill_mode='nearest')
        valid_generator = valid_datagen.flow_from_directory(validData, target_size = (227, 227), batch_size = batchSize, class_mode = "categorical", shuffle = True)

        test_datagen = ImageDataGenerator(rescale=1./255, width_shift_range=0.2,height_shift_range=0.2,shear_range=0.2,zoom_range=0.2,horizontal_flip=True,fill_mode='nearest')
        test_generator = test_datagen.flow_from_directory(testData,target_size = (227, 227), batch_size = 1, class_mode = "categorical", shuffle = True)

        return (train_generator, valid_generator, test_generator)



#TRAINING AND TESTING THE MODEL
def baseTrain(trainData, validData, testData):
                cwd = os.getcwd()
```

Python ▾ Tab Width: 8

crop+align+15.txt ×   *8class_model.py

```python
        model.add(layers.Dense(256, activation='relu', kernel_initializer=initializers.random_normal(stddev=0.01)))
        model.add(Dropout(0.5))
        model.add(layers.Dense(8, activation='softmax'))
        return model


#GENERATING IMAGE DATA GENERATORS
def baseModelDataGen(trainData, validData, testData):
        batchSize = 50
        train_datagen = ImageDataGenerator(rescale=1./255, width_shift_range=0.2,height_shift_range=0.2,shear_range=0.2,zoom_range=0.2,horizontal_flip=True,fill_mode='nearest')
        train_generator = train_datagen.flow_from_directory(trainData, target_size = (227, 227), batch_size = batchSize, class_mode = "categorical", shuffle = True)

        valid_datagen = ImageDataGenerator(rescale=1./255, width_shift_range=0.2,height_shift_range=0.2,shear_range=0.2,zoom_range=0.2,horizontal_flip=True,fill_mode='nearest')
        valid_generator = valid_datagen.flow_from_directory(validData, target_size = (227, 227), batch_size = batchSize, class_mode = "categorical", shuffle = True)

        test_datagen = ImageDataGenerator(rescale=1./255, width_shift_range=0.2,height_shift_range=0.2,shear_range=0.2,zoom_range=0.2,horizontal_flip=True,fill_mode='nearest')
        test_generator = test_datagen.flow_from_directory(testData,target_size = (227, 227), batch_size = 1, class_mode = "categorical", shuffle = True)

        return (train_generator, valid_generator, test_generator)



#TRAINING AND TESTING THE MODEL
def baseTrain(trainData, validData, testData):
                cwd = os.getcwd()
                trainLen = sum(len(files) for _, _, files in os.walk(trainData))
                validLen = sum(len(files) for _, _, files in os.walk(validData))
                testLen =  sum(len(files) for _, _, files in os.walk(testData))
                batch_size = 50
                myModel = load_model(cwd+'/hdf/cropped_aligned_40.h5')
                train_generator, valid_generator, test_generator = baseModelDataGen(trainData, validData, testData)
                test_acc = myModel.evaluate_generator(test_generator, int(np.ceil(testLen/batch_size)))
                print("test", test_acc[0], test_acc[1])
                myModel.compile(optimizer=SGD(lr=0.001, decay = 1e-5),loss='categorical_crossentropy',metrics=['accuracy'])
                callbacks = [ModelCheckpoint(filepath=cwd + '/call' + '/saved-model-{epoch:02d}-{val_acc:.2f}.hdf5', verbose=1, save_best_only=False, save_weights_only=False, period=3)
                history = myModel.fit_generator(train_generator,
                    steps_per_epoch = int(np.ceil(trainLen/batch_size)),
                    epochs=40,
                    validation_data=valid_generator,
                    validation_steps= int(np.ceil(validLen/batch_size)),
                    verbose=1)
                myModel.save(cwd + '/hdf' +'/cropped_aligned_41.h5')
                test_acc = myModel.evaluate_generator(test_generator, int(np.ceil(testLen/batch_size)))
                print("test details", test_acc[0], test_acc[1])


if __name__ == '__main__':
        cwd = os.getcwd()
        trainData = cwd + '/data/age_train'
        validData = cwd + '/data/age_valid'
        testData = cwd + '/data/age_test'
        baseTrain(trainData, validData, testData)
```

In [1]:

```python
import bz2
import os

from urllib.request import urlopen

def download_landmarks(dst_file):
    url = 'http://dlib.net/files/shape_predictor_68_face_landmarks.dat.bz2'
    decompressor = bz2.BZ2Decompressor()

    with urlopen(url) as src, open(dst_file, 'wb') as dst:
        data = src.read(1024)
        while len(data) > 0:
            dst.write(decompressor.decompress(data))
            data = src.read(1024)

dst_dir = 'models'
dst_file = os.path.join(dst_dir, 'landmarks.dat')

if not os.path.exists(dst_file):
    os.makedirs(dst_dir)
    download_landmarks(dst_file)
```

In [80]:

```python
import cv2
import dlib
import numpy as np

TEMPLATE = np.float32([
    (0.0792396913815, 0.339223741112), (0.0829219487236, 0.456955367943),
    (0.0967927109165, 0.575648016728), (0.122141515615, 0.691921601066),
    (0.168687863544, 0.800341263616), (0.239789390707, 0.895732504778),
    (0.325662452515, 0.977068762493), (0.422318282013, 1.04329000149),
    (0.531777802068, 1.06080371126), (0.641296298053, 1.03981924107),
    (0.738105872266, 0.972268833998), (0.824444363295, 0.889624082279),
    (0.894792677532, 0.792494155836), (0.939395486253, 0.681546643421),
    (0.96111933829, 0.562238253072), (0.970579841181, 0.441758925744),
    (0.971193274221, 0.322118743967), (0.163846223133, 0.249151738053),
    (0.21780354657, 0.204255863861), (0.291299351124, 0.192367318323),
    (0.367460241458, 0.203582210627), (0.4392945113, 0.233135599851),
    (0.586445962425, 0.228141644834), (0.660152671635, 0.195923841854),
    (0.737466449096, 0.182360984545), (0.813236546239, 0.192828009114),
    (0.8707571886, 0.235293377042), (0.51534533827, 0.31863546193),
    (0.516221448289, 0.396200446263), (0.517118861835, 0.473797687758),
    (0.51816430343, 0.553157797772), (0.433701156035, 0.604054457668),
    (0.475501237769, 0.62076344024), (0.520712933176, 0.634268222208),
    (0.565874114041, 0.618796581487), (0.607054002672, 0.60157671656),
    (0.252418718401, 0.331052263829), (0.298663015648, 0.302646354002),
    (0.355749724218, 0.303020650651), (0.403718978315, 0.33867711083),
    (0.352507175597, 0.349987615384), (0.296791759886, 0.350478978225),
```

```
            return None


    def findLandmarks(self, rgbImg, bb):

        assert rgbImg is not None
        assert bb is not None

        points = self.predictor(rgbImg, bb)
        return list(map(lambda p: (p.x, p.y), points.parts()))

    def align(self, imgDim, rgbImg, bb=None,
              landmarks=None, landmarkIndices=INNER_EYES_AND_BOTTOM_LIP,
              skipMulti=False):

        assert imgDim is not None
        assert rgbImg is not None
        assert landmarkIndices is not None

        if bb is None:
            bb = self.getLargestFaceBoundingBox(rgbImg, skipMulti)
            if bb is None:
                return

        if landmarks is None:
            landmarks = self.findLandmarks(rgbImg, bb)

        npLandmarks = np.float32(landmarks)
        npLandmarkIndices = np.array(landmarkIndices)

        H = cv2.getAffineTransform(npLandmarks[npLandmarkIndices],
                                   imgDim * MINMAX_TEMPLATE[npLandmarkIndices])
        thumbnail = cv2.warpAffine(rgbImg, H, (imgDim, imgDim))

        return thumbnail
```

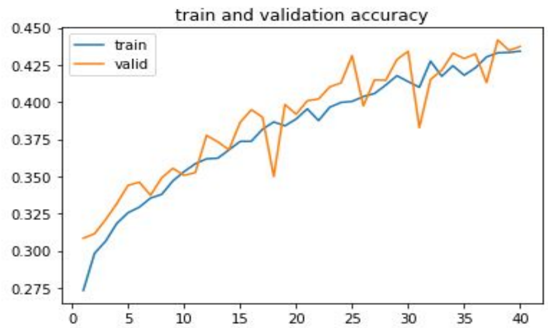
2]:

```
import cv2
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import glob
import shutil

#from AlignDlib import *

#from align import AlignDlib

%matplotlib inline
```

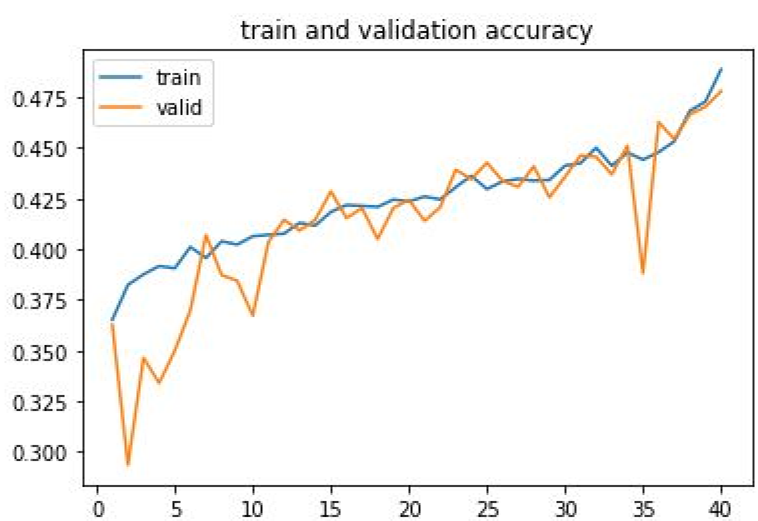
```
import matplotlib.patches as patches
import glob
import shutil

#from AlignDlib import *

#from align import AlignDlib

%matplotlib inline

def load_image(path):
    img = cv2.imread(path, 1)
    # OpenCV loads images with color channels
    # in BGR order. So we need to reverse them
    return img[...,::-1]

# Initialize the OpenFace face alignment utility
#print("hello")
alignment = AlignDlib('models/landmarks.dat')

# Load an image of Jacques Chirac
#jc_orig = load_image(metadata[2].image_path())

cwd = os.getcwd()
#imageFolders = [x[0] for x in os.walk(cwd + '/aligned')]
imageFolders =  os.listdir(cwd + '/aligned')
newDir = cwd+'/cropped'
if os.path.exists(newDir):
    shutil.rmtree(newDir)
os.mkdir(newDir)

#print(imageFolders)

for folder in imageFolders:
    #subDir = folder.replace('aligned','cropped')\
    newDir = cwd + '/cropped/' + folder
    oldDir = cwd + '/aligned/' + folder
    os.mkdir(newDir)
    pa = oldDir + '/*'
    for filename in glob.glob(pa):
        #print(filename)
        img = cv2.imread(filename)
        bb = alignment.getLargestFaceBoundingBox(img)
        cropped = alignment.align(227, img, bb, landmarkIndices=AlignDlib.OUTER_EYES_AND_NOSE)
        imgName = filename.replace('aligned', 'cropped')
        if bb == None:
            cropped = img
        #print(imgName)
        cv2.imwrite(imgName, cropped)
```

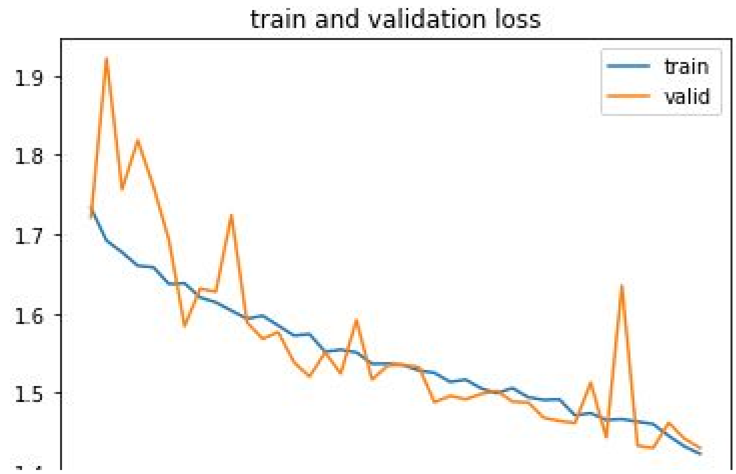
Accuracy Plots for validation and train data:

1. Base Model



2. Model preloaded with base and trained on cropped Images






```
Epoch 1/40 - 179s 516ms/step - loss: 1.7331 - acc: 0.3651 - val_loss: 1.7211 - val_acc: 0.3626
Epoch 2/40 - 176s 507ms/step - loss: 1.6918 - acc: 0.3823 - val_loss: 1.9214 - val_acc: 0.2935
Epoch 3/40 - 176s 506ms/step - loss: 1.6771 - acc: 0.3875 - val_loss: 1.7562 - val_acc: 0.3463
Epoch 4/40 - 176s 507ms/step - loss: 1.6601 - acc: 0.3916 - val_loss: 1.8184 - val_acc: 0.3338
Epoch 5/40 - 173s 500ms/step - loss: 1.6583 - acc: 0.3905 - val_loss: 1.7603 - val_acc: 0.3498
Epoch 6/40 - 174s 502ms/step - loss: 1.6371 - acc: 0.4010 - val_loss: 1.6924 - val_acc: 0.3698
Epoch 7/40 - 174s 501ms/step - loss: 1.6380 - acc: 0.3956 - val_loss: 1.5839 - val_acc: 0.4069
Epoch 8/40 - 174s 502ms/step - loss: 1.6201 - acc: 0.4038 - val_loss: 1.6312 - val_acc: 0.3872
Epoch 9/40 - 175s 504ms/step - loss: 1.6140 - acc: 0.4022 - val_loss: 1.6273 - val_acc: 0.3843
Epoch 10/40 - 174s 502ms/step - loss: 1.6035 - acc: 0.4063 - val_loss: 1.7241 - val_acc: 0.3672
Epoch 11/40 - 171s 492ms/step - loss: 1.5933 - acc: 0.4070 - val_loss: 1.5885 - val_acc: 0.4034
Epoch 12/40 - 174s 500ms/step - loss: 1.5971 - acc: 0.4076 - val_loss: 1.5680 - val_acc: 0.4144
Epoch 13/40 - 173s 499ms/step - loss: 1.5846 - acc: 0.4129 - val_loss: 1.5765 - val_acc: 0.4092
Epoch 14/40 - 175s 503ms/step - loss: 1.5722 - acc: 0.4116 - val_loss: 1.5385 - val_acc: 0.4144
Epoch 15/40 - 178s 514ms/step - loss: 1.5739 - acc: 0.4183 - val_loss: 1.5203 - val_acc: 0.4284
Epoch 16/40 - 175s 505ms/step - loss: 1.5518 - acc: 0.4217 - val_loss: 1.5509 - val_acc: 0.4153
Epoch 17/40 - 175s 505ms/step - loss: 1.5542 - acc: 0.4213 - val_loss: 1.5240 - val_acc: 0.4202
Epoch 18/40 - 175s 503ms/step - loss: 1.5509 - acc: 0.4208 - val_loss: 1.5918 - val_acc: 0.4049
Epoch 19/40 - 176s 508ms/step - loss: 1.5365 - acc: 0.4244 - val_loss: 1.5165 - val_acc: 0.4202
Epoch 20/40 - 175s 503ms/step - loss: 1.5366 - acc: 0.4235 - val_loss: 1.5343 - val_acc: 0.4243
Epoch 21/40 - 174s 502ms/step - loss: 1.5350 - acc: 0.4259 - val_loss: 1.5350 - val_acc: 0.4139
Epoch 22/40 - 175s 503ms/step - loss: 1.5280 - acc: 0.4245 - val_loss: 1.5323 - val_acc: 0.4205
Epoch 23/40 - 174s 502ms/step - loss: 1.5251 - acc: 0.4305 - val_loss: 1.4880 - val_acc: 0.4391
Epoch 24/40 - 176s 507ms/step - loss: 1.5137 - acc: 0.4361 - val_loss: 1.4961 - val_acc: 0.4345
Epoch 25/40 - 174s 501ms/step - loss: 1.5165 - acc: 0.4296 - val_loss: 1.4918 - val_acc: 0.4426
Epoch 26/40 - 176s 508ms/step - loss: 1.5054 - acc: 0.4333 - val_loss: 1.4985 - val_acc: 0.4336
Epoch 27/40 - 175s 504ms/step - loss: 1.4994 - acc: 0.4346 - val_loss: 1.5023 - val_acc: 0.4307
Epoch 28/40 - 175s 505ms/step - loss: 1.5059 - acc: 0.4337 - val_loss: 1.4887 - val_acc: 0.4408
Epoch 29/40 - 176s 507ms/step - loss: 1.4943 - acc: 0.4341 - val_loss: 1.4878 - val_acc: 0.4255
Epoch 30/40 - 176s 507ms/step - loss: 1.4910 - acc: 0.4411 - val_loss: 1.4686 - val_acc: 0.4356
Epoch 31/40 - 177s 509ms/step - loss: 1.4916 - acc: 0.4424 - val_loss: 1.4644 - val_acc: 0.4461
Epoch 32/40 - 174s 501ms/step - loss: 1.4714 - acc: 0.4500 - val_loss: 1.4619 - val_acc: 0.4455
Epoch 33/40 - 175s 506ms/step - loss: 1.4743 - acc: 0.4411 - val_loss: 1.5129 - val_acc: 0.4368
Epoch 34/40 - 173s 497ms/step - loss: 1.4657 - acc: 0.4476 - val_loss: 1.4436 - val_acc: 0.4510
Epoch 35/40 - 176s 507ms/step - loss: 1.4668 - acc: 0.4441 - val_loss: 1.6351 - val_acc: 0.3881
Epoch 36/40 - 175s 505ms/step - loss: 1.4635 - acc: 0.4478 - val_loss: 1.4332 - val_acc: 0.4626
Epoch 37/40 - 175s 505ms/step - loss: 1.4605 - acc: 0.4531 - val_loss: 1.4304 - val_acc: 0.4542
Epoch 38/40 - 174s 501ms/step - loss: 1.4456 - acc: 0.4682 - val_loss: 1.4622 - val_acc: 0.4664
Epoch 39/40 - 175s 505ms/step - loss: 1.4324 - acc: 0.4728 - val_loss: 1.4422 - val_acc: 0.4702
Epoch 40/40 - 173s 498ms/step - loss: 1.4230 - acc: 0.4886 - val_loss: 1.4305 - val_acc: 0.4779
('test details', 1.5034706606262032, 0.47128735632183906)
```

6. Training and Testing performance:

Base Model:

Train Data: 48.86

Valid Data: 47.79

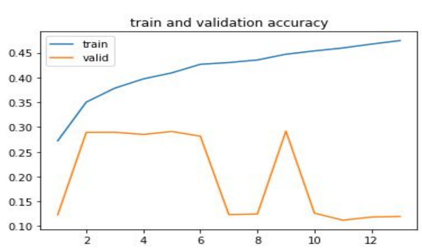
Test Data: 47.1

Model	**Train**	**Valid**	**Test**
Base Model	45.42	43.73	44.9
PreTrained with Facial Images	**48.86**	**47.79**	**47.1**

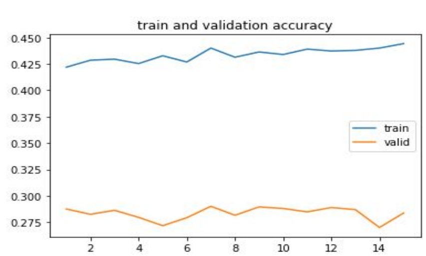
7. Other Tried Approaches:

I even tried resnet-50 where I first freezed resnet layers and trained only top two dense layers with 512 and 256 nodes and it started overfitting around 7th epoch. Later I unfreezed all the layers and tried to finetune it using with adience dataset, train accuracy kept on increasing but validation accuracy was hovering around 28%. One of the main reasons that we can't use more deeper networks is smaller dataset. I even experimented with VGG16 for 4 classes during previous submission but even that didn't work out.

a. Resnet Model with lower layers freezed:



b. Resnet Model with all layers Unfreezed



Instructions for Running:
Dependencies:
Python 3: Running CNN
Python 2.7: Running Tkinter
Tkinter
numpy
Keras
Pillow
Numpy
Pathlib
cv2
dlib
gensim
cmake

Run:
After cloning the repository from github (assuming you maintained same relative folder structure)

1. Python Base.py: For running base model
2. Python preTrained.py: For running pretrained model
3. GUI.py : For running GUI
4. faceRecog.ipynb: For creating cropped face images

Demo link:
https://drive.google.com/file/d/12ZEh4d3bdHRrlWx4McgDqdc7HXPAFDpB/view

Github Link:
Entire code Base
https://github.com/Manvitha-K/Age-Prediction-Neural-Project

GDrive Link:
For loading the trained model: These models should be downloaded to /hdf folders.
https://drive.google.com/open?id=1M-H2zI1u6GkV8ozDYNb3_GdErs9utqav

Updated Report:

As per suggestion, I tried heatmaps(class activation maps). I have randomly picked images from various classes and tried heatmaps on top of those and observation is my current CNN model is triggered by face, especially eyes part. It is not focussing on any background or some disturbing ailment.

What I have written in this report is not my baseline model. My baseline model was only categorizing these images into 4 types which I have submitted in project4 submission. My improved model had 2 parts. Face detection and 8 classifier CNN.

First Part: Implementing the face detection algorithm for finding faces and then using these detected faces for training CNN.

Second part: CNN (3 Convolution blocks) for 8 class classification (0-2. 4-6, 8-13, 15-20, 25-32, 38-43, 48-53, 60+)**.** I trained this model from scratch and after 40 epochs with 0.001 learning rate and decay rate (10^-5) with SGD optimizer it has attained an accuracy of **42.8 % ± 3.** Now I loaded this trained model and fine tuned it using the cropped face images (i.e., which are obtained by running face detection algorithms on original images) with all other parameters being kept same. After 40 epochs this model has attained an accuracy **43.2 ± 4** with highest observed **47.12 %.**

The lower accuracy mentioned in the above section can be attributed to the difficulty of age classification problem unlike imagenet classification where we have many objects and different features are used for distinguishing them whereas here entire classification is based on the facial features.The publication which I took as baseline had only accuracy of 44% whereas my model was able to achieve 47%. My model is also focussing on the right parameters which is clearly evident from the heatmaps.

Attached heatmap images and corresponding code has been pushed to github.

Neural Network Architecture :

(Loss function : SGD Optimizer)