Capstone Report - Age Estimator[1]

Khalid A. Al-Abbad

5 December 2018

[1] Image produced using a collage from 1200 images from the UTKFace database

I. DEFINITION

Project Overview

This section contains some high level information about the project and how the different components work at a basic level. The actual details are provided in later sections.

The purpose of this project is to build a machine learning based age estimator. The estimator will look at an image that contains a human face, and try to predict the age of the person. This is done using two main parts:

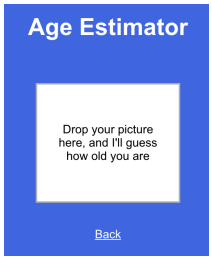
1) An application that interacts with the user, allowing the user to supply an image, takes some steps to process the image, and then passes the processed image to the actual machine learning model.

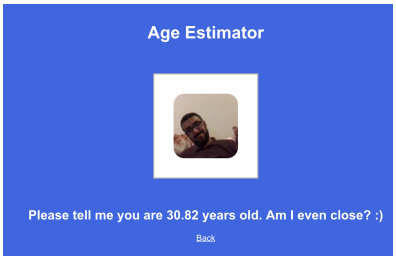
2) A machine learning model, based on Convolutional Neural Networks (CNNs), that takes the processed image and outputs an estimated age. This model does this based on training it has received on real life human images.

In order to better understand how this works, we will go into a bit more details.

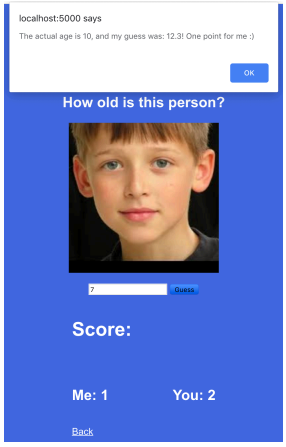
* **The Application:**

The application has a very simple interface. You are given the option to select between two activities. The first activity is the estimator: You simply take a photo of yourself, or use one of somebody you know, and drop it into the browser. As soon as you do that, you will receive feedback.

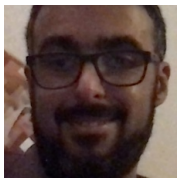




The other activity is a competition between you and the model. It will show you a photo picked at random, and ask you to predict the age of the person, and it will make its own prediction and hide it from you. If your prediction is closer to actual age, you get a point, otherwise, the model gets a point.



The application takes your image and passes it to a face detector. This face detector[2] has been previously trained to detect what are called landmarks in the human face. We use this information to first align the face (to make it easier to for our model), and then crop it to the main facial features (The model works best if you only provide it with the information that would best help it find the desired result). For example, the sample photo in the screenshot, will look like this after it is cropped and aligned:



[2] The face detector we are using is part of the Dlib library, using the default pre-trained facial shape predictor (http://dlib.net)

* **The Model:**

The model accepts images of a given size, and uses what it has learned previously estimate the age. This is the main part of the project, and the most interesting.

The technology the model uses is referred to as Convolutional Neural Networks (CNN). It's a special form of Neural Networks. Neural networks are structures modeled loosely after our brain. A Neural Network is a set of nodes connected to each other structured in layers. Each node receives input from the previous layer, calculates a weighted average of the values received, and passes it into what is referred to as an activation function. The activation function usually does a simple calculation with the data (in our case the activation functions, referred to as rectified linear unit drops values below zero). Activation functions are important because they allow the neural network to be complex, otherwise it would not be able to handle complex tasks.

Neural networks work with numbers, so we have to first convert our image into numbers, each pixel in the image is mapped to 3 numbers, representing red, green, and blue respectively. These numbers are what the neural network uses to make its calculations.

However, the problem with converting our image to a stream of numbers is that we lose the spatial information that is very critical to making our predictions accurate, and this is where CNNs come in. At a very high level, CNNs modify the neural network so a group of numbers are passed together between layers rather than a single number, thus allowing it to pass (as an example) a 3 by 3 set of numbers representing a small slice of the image that is 3 by 3 pixels. It then using a sliding window approach across the image and passes values to the next layer. These sliding windows can be thought of as filters searching for certain information in the image. In order to reduce the complexity of the model (since we are dealing with images that contain a lot of information), CNNs often contain layers that summarize the data (pooling layers), and these layers are distributed around the convolutional layers. Combining many of these layers after each other results in models that can visualize a given image, and even almost "understand" its contents.

In order to achieve these results, a neural network needs to be trained. Neural networks are trained by giving them sample data (images in this case), and allowing them to make any prediction. At first, these predictions will be very wrong. However, after each prediction or set of prediction, we adjust the network (through what is referred to as Backpropagation), and the different weights of the nodes, to make the prediction somewhat closer to our target (how much closer depends on a parameter can adjust, called the learning rate). After a number of repetitions, and given sufficient data, the model's performance starts improving and it makes better predictions.

In this project, I did not start from scratch. There are some CNN models available for use that have been trained on very large datasets, such as the ImageNet database[3]. I use ResNet50, which won the 2015 Large Scale Visual Recognition Challenge. The ResNet50 architecture was developed by a team at Microsoft and introduces innovative concepts such as residual learning. I take part of the model already designed and trained by the team, and pass the images through it before doing my own training. This is possible even though the model was designed for a different purpose (object recognition and classification in images), and this can be done through what is referred to as transfer learning.

[3] http://image-net.org

The output taken from the model is a set of numbers, which is passed to another neural network designed as part of this project. This is a traditional neural network with interconnected nodes, and finally a single node at the end, which outputs a number representing the age of the face detected in the image.

In order to train the model, I use a dataset of images that contains the age of the person in each image. The dataset is the UKTFace[4] dataset. For the purposes of this project, only 2500 images from the dataset are used. We split them into 3 parts, the first part, 1000 images, is used to train the model as described earlier. The second, 500, is used as 'validation' data, which the program checks against to ensure the model is actually learning something, and not just memorizing the faces and their ages. The final 1000 images are not shown to the model during training, and are only used to test the model after the training is done.

I also use another dataset in this project, the OUI-Adience[5] dataset. I use a portion of this dataset as a benchmark, since I am comparing my results to a previous age estimation project. Additionally, it is good to test the model against completely different data, just to ensure that it's not learning something specific to that dataset.

Problem Statement

The goal behind this project is to build a model that predicts the age of a person given a photo that includes the face of that person. Additionally, a proof of concept application will be built to use this model, allowing a user to upload an image to a web interface to estimate the age, and a challenge mode where the user competes with the model to see who would better estimate the age of random images from the UKTFace dataset.

The following steps will be taken to achieve this goal:

* Organize and setup a workspace to enable working on potentially large number of images with high performance. The ideal setup for this is using GPUs. Google Colab[6] has been selected as the primary workspace, and Google Drive is used to store the processed data.
* Download, prepare, and analyze the required images. The UKTFace dataset is used as the primary dataset, and the OUI-Adience dataset is used as a benchmark.
* Use a pre-trained face detector to detect and align the target face in the image (Using Dlib).
* Use a pre-trained network such as ResNet50[7] or InceptionV3[8] to extract bottleneck features.
* Develop and test various Deep Neural Network architectures to come up with a model that provides best performance on our test dataset.

---

[4] https://susanqq.github.io/UTKFace/

[5] https://talhassner.github.io/home/projects/Adience/Adience-data.html

[6] https://colab.research.google.com

[7] https://arxiv.org/pdf/1512.03385.pdf

[8] https://arxiv.org/pdf/1512.00567.pdf

* Further enhance the performance of the models with good performance by using data augmentation.
* Develop an application that provides a proof of concept for the usage of the model, and that consists of two activities: Age estimation for uploaded images, and challenging the user to compete with the model.

Metrics

The most common evaluation metric used by the different studies and applications is the mean absolute error (MAE). The formula for the MAE is as follows[9]:

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

Mean Absolute Error will be the primary metric for this project. Table 1 from Age Synthesis and Estimation via Faces survey[10] shows different values form MAE for different studies. They range - in case of regression - from 3 to 10. The target set for this project is to get below 10 mean absolute error.

An additional metric that will be used is what I refer to as “soft accuracy.” Rather than categorizing the age into specific age groups, which has the disadvantage of high error rate in cases at the edges of the categories (unless 1-off-accuracy is used). I chose this metric that simply considers the estimate accurate if it falls within X years from the real value. I propose to use two soft accuracies, the first is within 5 years, and the second is within 10 years.

Additionally, quantile loss[11] has been considered as an alternate loss function (In order to punish underestimation more than over estimation of the age, since older individuals tend to be underestimated). However, MAE proved to provide better results.

III. ANALYSIS

Data Exploration

We will provide the analysis of the two datasets used in the project separately:

* **UKTFace:**

The original dataset contains over 20,000 images. The dataset is provided in two flavors: Full in-the-wild images, and cropped and aligned faces. The images are cropped and aligned using Dlib. For each image age, gender, and race are provided, and are conveniently included as part of the file name for the image.

The ground truth for the age is actually an estimate reviewed by a human reviewer. Only the age will be used to train the model. This dataset is provided for non-commercial research purposes only.

---

[9] https://en.wikipedia.org/wiki/Mean_absolute_error

[10] http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=72B9E841852E2C33921B8F00049550692?doi=10.1.1.221.213&rep=rep1&type=pdf page 1970.

[11] https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0

The quality of the data is quite high, however, there are some clear errors. As an example, the photo on the side is marked as 57 years old, however, this lady is clearly younger than that.

Additionally, many of the photos contain watermarks on top of them. Which may cause our model to learn to associate these watermarks with certain ages.

Moreover, there are images, especially in the unprocessed version, which contain more than one face. This issue was not noticed in the cropped and aligned faces.

The age in the dataset is normally distributed with the mean at 33.3 years old and a standard deviation of 19.89 years, with an exception of a spike near zero. I believe this spike is to our advantage since it is arguably difficult to accurately specify the age of a young baby.

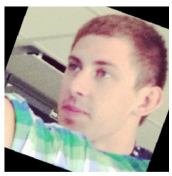
For our purposes, we have used 1000 photos for the training dataset, 500 for validation, and 1000 for testing. These sets have similar distributions to the main dataset.

* **OUI-Adience:**

The OUI-Adience (Face Image Project) is a dataset that includes over 26,000 photos obtained from Flickr uploads (Creative Commons). The dataset includes age-group (8 classifications), gender, and subject (There are 2000+ subjects). The dataset is provided for research purposes only.

This dataset is less clean than the previous dataset, some clean up was required to remove data with no proper age-group classification. Interestingly, the 8 age groups (0-2, 4-6, 8-13, 15-20, 25-32, 38-43, 48-53, 60-) leave gaps between them, the reason behind that is not clear. How should a person aged 35 be classified?

Additionally, many of the photos in the dataset are grainy, have black areas in the corners due to rotation, or contain more than one person. See samples below:



Since this dataset has age classification, while our target is regression, there is no perfect way to use this dataset for training (one trick would be to take the middle of the category, or a random value in the category). However, we could use as a benchmark, and to test the accuracy of our model by converting the output of our model to the appropriate category (and extending the categories to cater for the gaps).

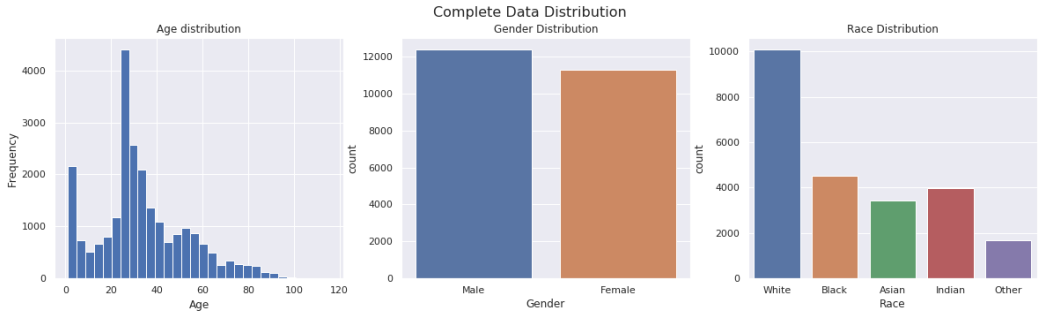
I have used fold 0 of the dataset as my benchmark. It consists of 4432 images after some clean up. The age categories are not normally or evenly distributed (see the below exploratory visualization), but I didn't want to change the data a lot since it is a benchmark. The only modification made was removing the images with no

age categorization, and cleaning up the groups (there were some images that had an actual age, rather than a group, and a single item was misclassified in a non-existent group by fault).

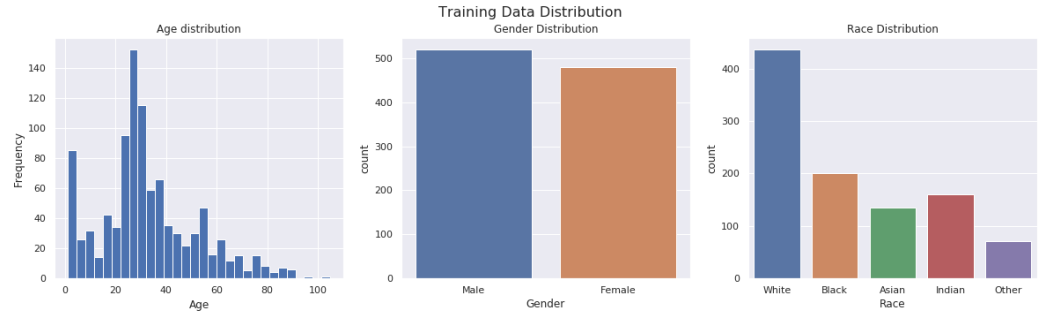
Exploratory Visualization

* **UKTFace:**

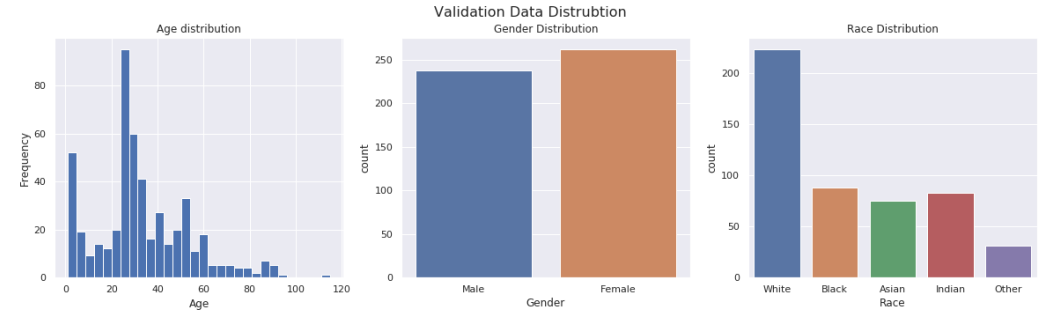
The following shows the data distribution for the complete UKTFace dataset:



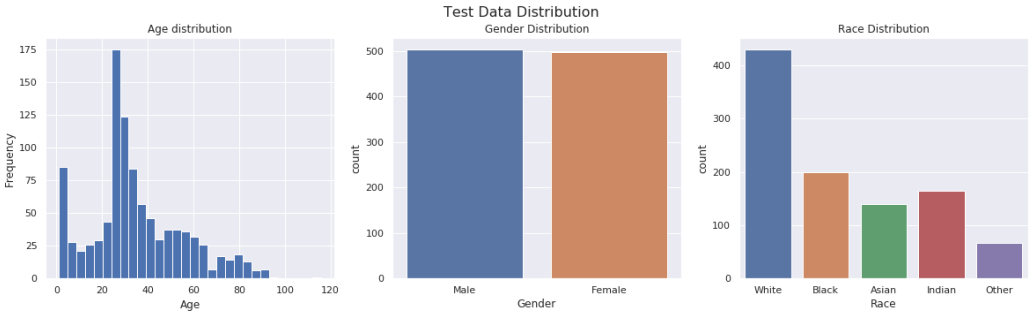
This is the distribution of the data for training:



This is the distribution of the data for validation:



This is the distribution of the data for testing:

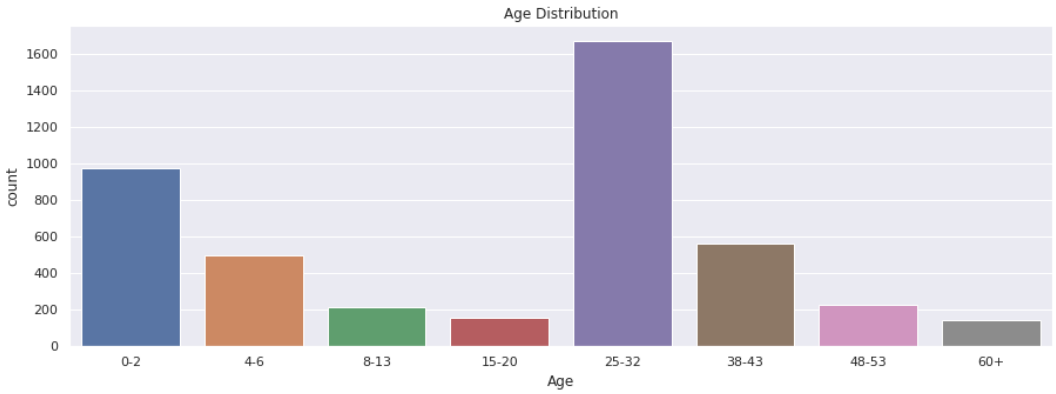


As can be observed from the above visualizations, the distribution is consistent between the different datasets. Additionally, our most important feature, age, is normally distributed with the exception of a spike near zero. Race and gender are not used at all during training (Because we do not want to require them as inputs later when using the model). However, it would be interesting to see if the model would perform differently for different Race/Gender combinations.

It is also worth noting that the dataset consists of white individuals more than any other race (twice the next race). It is interesting to see if this has any effect on our results.

* **OUI-Adience:**

The main thing we focus in the Adience dataset is the age categories. The following shows the distribution of age categories for the dataset:



As mentioned previously, the data is not very well-distributed. However, this is not a major issue since this dataset is simply used as a benchmark.

Algorithms and Techniques

1) Face Detection:

Face detection is used to crop and align input images prior to passing them to the model. The dataset is already cropped and aligned, but photos received by the model during actual predictions need to be cropped and aligned for best results. For that purpose, Dlib was used for face detection.

Dlib provides pre-trained face shape predictors that can be used for face alignment[12]. Additionally, according to the comments in the source code, Dlib uses "Histogram of Oriented Gradients (HOG) feature combined with a linear classifier[13]."

2) Transfer Learning:

Faces are very complex, and starting from scratch would require a huge amount of data and massive computing requirements. In order to focus at the task at hand, which is age estimation, I will use a pre-trained model to extract bottleneck features. Both ResNet50 and InceptionV3 will be evaluated, and the standard size for each will be used:

* ResNet50: 224 by 224 pixels
* InceptionV3: 299 by 299 pixels

Additionally, the top fully connected layers will be removed, since I will have to replace them with suitable layers for age estimation. The convolutional and pooling layers will be kept, and the weights will be based on ImageNet weights.

Since the dataset is very small relative to the ImageNet database, the weights will be used as is and not fine-tuned.

3) Deep Neural Network (DNN):

The extracted bottleneck features will be passed to a deep neural network. There are many variables to decide on when designing a neural networks, relating to both the training process and the DNN architecture. The following variables will be considered for the training process:

* *Batch Size:* The batch size has an effect on both the speed of the training, and in some cases the performance of the resulting model, several batch sizes have been investigated, and I have decided to go with 32.
* *Number of epochs:* The training model improves over time, but after a while, this improvement slows down. For the given amount of data used, I found that 100 epochs is more than sufficient, and kept as a standard to be able to compare between the different versions of the model.
* *Optimizer:* Different optimizers arrive at the result through different paths, and some are better than other, I have experimented with Adadelta and Nesterov Adam optimizers, and variations of the Adam optimizer.

---

[12] The one used by the project is http://dlib.net/files/shape_predictor_5_face_landmarks.dat.bz2

[13] http://dlib.net/face_landmark_detection.py.html

However, except for a few architectures that considered other optimizers, I have decided to stick with the default parameters for the Adam optimizer, since its seemed to produce the best results. The parameters used are:

    * Learning Rate: 0.001
    * Beta 1: 0.9
    * Beta 2: 0.999
    * Decay: 0

* *Loss Function:* I have considered two loss functions during the training. A Quantile Loss function, which may work better in case of biased data (using a q value of .7), and the mean absolute error.

With regards to the network architecture, variations of the following are considered:

* Number of dense layers.
* Number of units in each layer.
* A layer to flatten the data received from the pre-trained CNN in the form of bottleneck features.
* Batch Normalization layers: These layers promise to protect against overfitting, speed up training, and balance the network. They do so by scaling the values of the previous layer.
* Dropout layers: Dropout layers allow the network to train more evenly by randomly disabling a percentage of the units in a given training run. This will again protect against overfitting and make the network more robust.
* Other variables were considered such as kernel initializers.

4) Data Augmentation:

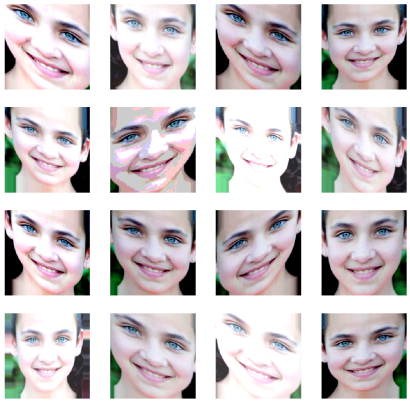
Data Augmentation makes our data more robust to different types of inputs. For this project, we will use the following forms of data augmentation:

* *Brightness:* Input pictures vary in brightness, so manually augmenting that would improve the performance of the model.
* *Shear:* While shear does make our images less natural, there are some photos in the dataset that have shear (e.g. taking a photo of a photo). It's good to have some form of shear in our training data.
* *Zoom:* A little bit of zoom could make the data more robust.
* *Channel shift:* This would modify the colors slightly, thus making our model more versatile when it comes to varying image qualities.
* *Horizontal flip:* A face changes slightly when you flip it horizontally, having that would add value to our data.

The following have been ruled out:

* *Rotation:* We are already aligning our photos, so rotating them back is counter productive.
* *Shifts:* Shifts produce undesirable effects in the photo and in some cases would show only part of the face. We don't want that.
* *Vertical flip:* This can be argued either way, but most photos are upright, and vertical flip can be considered a form of 180 degree rotation.

As an example, this below is are some samples to augmentations to the same image:



We will use 8 additional augmented images per image in our training dataset, and 4 additional augmented images per image in our validation dataset. All augmented images will be passed to our feature extractor before being used to train the model.

5) Workspace Continuity:

One of the biggest challenges when dealing with image data is that each step takes considerable time, and the resulting data is large. In order to be able to work on the project for several days and ensure smooth processing, all the middle values are saved. A "processed data" folder is created by the Jupyter notebook. Whenever a step is completed, the resulting data is saved in the appropriate place in the processed data folder. If the kernel disconnects for any reason, I have written functions in the notebook that would reload the data from the saved state. The saved data includes:

* Extracted bottleneck features (1GB for ResNet50, and 1.3GB for InceptionV3).
* Extracted bottleneck features (5.4GB for ResNet50).
* Images converted to tensors.
* Processed datasets.
* The best model for each version of the architecture, in hdf5 format.

Some of these files were removed from the Google Drive due to the 15GB limitation, but are backed up locally.

Benchmark

We will use two benchmarks:

* **External Benchmark:**

We will benchmark our results against "Deep Convolutional Neural Network for Age Estimation based on VGG-Face Model[14]".

This model is a classification model that uses transfer learning. It is based on the VGG-Face model[15], which is a CNN model specifically designed to extract facial features. This model has 11 layers, 8 of them are convolutional, and the last 3 are dense layers, each of the convolutional layers is followed by a ReLU activation, and max pooling, dropout and normalization layers are also included after convolutions. The team has replaced the last 3 dense layers with 4 dense layers, the first dense layer is 4096 units, followed by 2 layers of 5000 units, and finally an 8 unit layer that outputs a class related to the age group.

Images are scaled to 256 by 256 pixels, and as a form of augmentation, cropped into patches of 224 by 224 pixels (Same approach is followed for prediction, where 3 patches are extracted from the image from specified locations and the resulting probabilities are averaged to come up with the final class).

The team has opted not to change the weights for the pre-trained layers, and only modify the weights for the dense layers during the training.

The model was trained and tested using the OUI-Adience dataset and obtained 59.9% overall accuracy, and 90.57% one-off accuracy.

Since this study is a classification study, and groups ages into 8 age groups, we will transform the age predicted by the model into these 8 categories, and evaluate our performance using the OUI-Adience dataset against the performance of this model.

Please note that the our model will not be trained using OUI-Adience or any portions of it, since OUI-Adience does not contain actual ages, but rather age ranges.

* **Naive Benchmark:**

In addition to the external benchmark, I have developed a simple naive benchmark, which consists of a neural network with a single hidden layer (containing 10 units). I pass the image as an array of tensors, and train the model using our training/validation dataset, and tested against the same test data I test the other models against (from UKTFace dataset). The naive benchmark achieved a mean absolute error of 32.644 years, and 13% accurate within 10 years, and 10% accurate within 5 years.

---

[14] https://arxiv.org/pdf/1709.01664.pdf

[15] http://www.robots.ox.ac.uk/~vgg/software/vgg_face/

VI. METHODOLOGY

Data Preprocessing

All the processing steps done in this project have been included in the Jupyter Notebooks provided. The primary Jupyter Notebook is titled Age_Estimator[16], which uses ResNet50 for the transfer learning portion. A secondary Jupyter Notebook, named InceptionV3_Age_Estimator, was included to test the initial models against features extracted using InceptionV3. ResNet50 outperformed InceptionV3 consistently, and as such, no further progress was done on that notebook.

The description here will describe the data processing and preparation steps included in the notebook.

* **Setup:** This section imports all the requirements used by the rest of the notebook. Additionally, since Google colab was used, this step performed some activities specific to colab (such as installing some libraries, and connecting the notebook to google drive storage.
* **Data Preparation:** This section:
  - Sets up some variables that are used across the notebook, such as folder location, and some constants.
  - Loads the feature extractor model.
  - Processes the UTKFace cropped faces dataset. It takes the file names, and creates a data frame that contains the features extracted from the file name, namely age, race, and gender. There are very few malformed filenames, which this step clears from the dataset.
  - Creates the training, validation, and testing datasets, which are consistently used across the project.
  - Saves the prepared data so that we do not have to repeat this process again.
* **Data Analysis:** The data analysis section produces the exploratory visualizations showing the distribution of age, race and gender in the whole dataset and each of our training, validation, and test datasets. It also visualizes sample images from the dataset.
* **Feature Extraction:** This section extracts the bottleneck features using the feature extractor created, and saves the extracted features so that we do not need to repeat the process.
* **Load Saved Results:** Provides different functions that allow reloading saved data. These functions are to be executed whenever the notebook kernel is restarted. Which functions to be executed depends on what you want to do (work on augmented data, or the regular data set, or even the image tensors).

Additionally, towards the end of the notebook, I have kept two additional sections that relate to data preprocessing (they were moved to the bottom in order not to clutter the notebook).

[16] The notebook and all source code is available in the repository: https://github.com/abbadka/age_estimator

* **Image Augmentation:** This section evaluates the different image augmentation possibilities provided by keras, visualizes them, and selects some of them. It then extracts bottleneck features from each augmented image, and saves the augmented training / validation data for future use.
* **Adience Benchmark Data Preparation & Analysis:** This section performs the necessary preprocessing (and visualization) for the adience benchmark data:
    - It specifies some constants and directory locations to be used by this section.
    - It provides loaders (Similar to the Load Saved Results section). They are not included along with the other loaders because they depend on variables specific to the adience dataset.
    - It loads fold 0 of the adience dataset.
    - The adience dataset contains some data with no age information. This data is removed from our analysis.
    - It loads the images, detects faces in each image, and extracts the bottleneck features using the face image. If no face is detected, it uses the whole image as is (The record is flagged to allow further analysis if necessary).
    - The categories specified in the adience dataset have some inconsistencies. For instance, some records have an actual age, rather than a group, and one record has a group that is not part of the specified groups. A cleanup code passes the age category through a function that provides consistent result. This same function is later used to convert our estimated age into an age category for the benchmark.

Implementation

The implementation has two sections, the model implementation, and the application implementation.

With regards to the model implementation, it was done using the same Jupyter notebook specified above. The following section relate to implementation:

* **Custom Metrics:** This section contains the same custom metrics described in the metrics section above (Soft Accuracy within 5/10 years, and Quantile Loss or Tilted Loss). These metrics are passed to the model when compiling it, and when loading it. Quantile Loss was only used in version 6 of the model as a loss function, and mean absolute error was used elsewhere.
* **Model Support Tools:** This section contains a method to fit the model (run_model) and another to evaluate the model (evaluate_model). These are included to avoid repetition when testing different variations of the model.
* **Naive Benchmark:** This is the section where the native benchmark described in the benchmark section was built and evaluated.
* **Model Refinement:** This section contains the different versions of the model that were built (To be further discussed in the refinement section). Each model has a block to build the model, and one or more blocks to fit the model, each is followed by a block to evaluate the model. In order to best visualize the process, I've used a library called livelossplot to show the metrics live in a chart as the model is training.

As for the web application, the implementation consisted of two main parts:

* **Backend:** The backend was developed using python flask. The main application backend consists of one file (app.py). This application has two main end points:
    - The predict end point takes an image from the UI, detects the faces included in the image using Dlib, and then extracts the bottleneck features, and passes them to the trained model. It returns the age estimated by the model. If no face is detected, it returns NO_FACE.
    - The challenge_image end point takes an image at random from the the list of challenge images, predicts its age using the same mechanism described in the previous bullet, and sends the filename, actual age, and predicted age to the UI, allowing the UI to perform the necessary activities.
* **User Interface:** The user interface has been kept as simple as possible. It consists of 3 pages. An index page which has two links to the other pages:
    - The age estimation page, which uses a dropzone[17] component to allow the user to drag in an image and estimate the age of the person in the image. This page passes the image file to the predict end point, and displays the result back to the user.
    - The challenge page, which asks the the backend to supply an image, with its age. It displays the image to the user and using javascript keeps both the user score and model score. Once the user makes a guess, it loads another image.

The biggest challenge faced during the implementation stage was working around the limitations of the environment, since storage and processing power are limited. The augmented dataset size used goes very close to the limit of what Google colab allows, and the storage limit barely allows all the data required to be stored in Google Drive.

Another challenge was keeping track of the best model for each architecture. On hindsight, I should have established a better mechanism for keeping track of different runs of each version, so that if a given architecture produces a model that performs very well, I do not lose it by mistake.

Refinement

The initial model used took the bottleneck features extracted using ResNet50 and passed it to a DNN network with the following structure:

* A flatten layer that accepts input matching the bottleneck feature shape.
* A single hidden layer with 20 units.
* A Rectified Linear Unit (ReLU) activation.
* A final layer with a single unit.

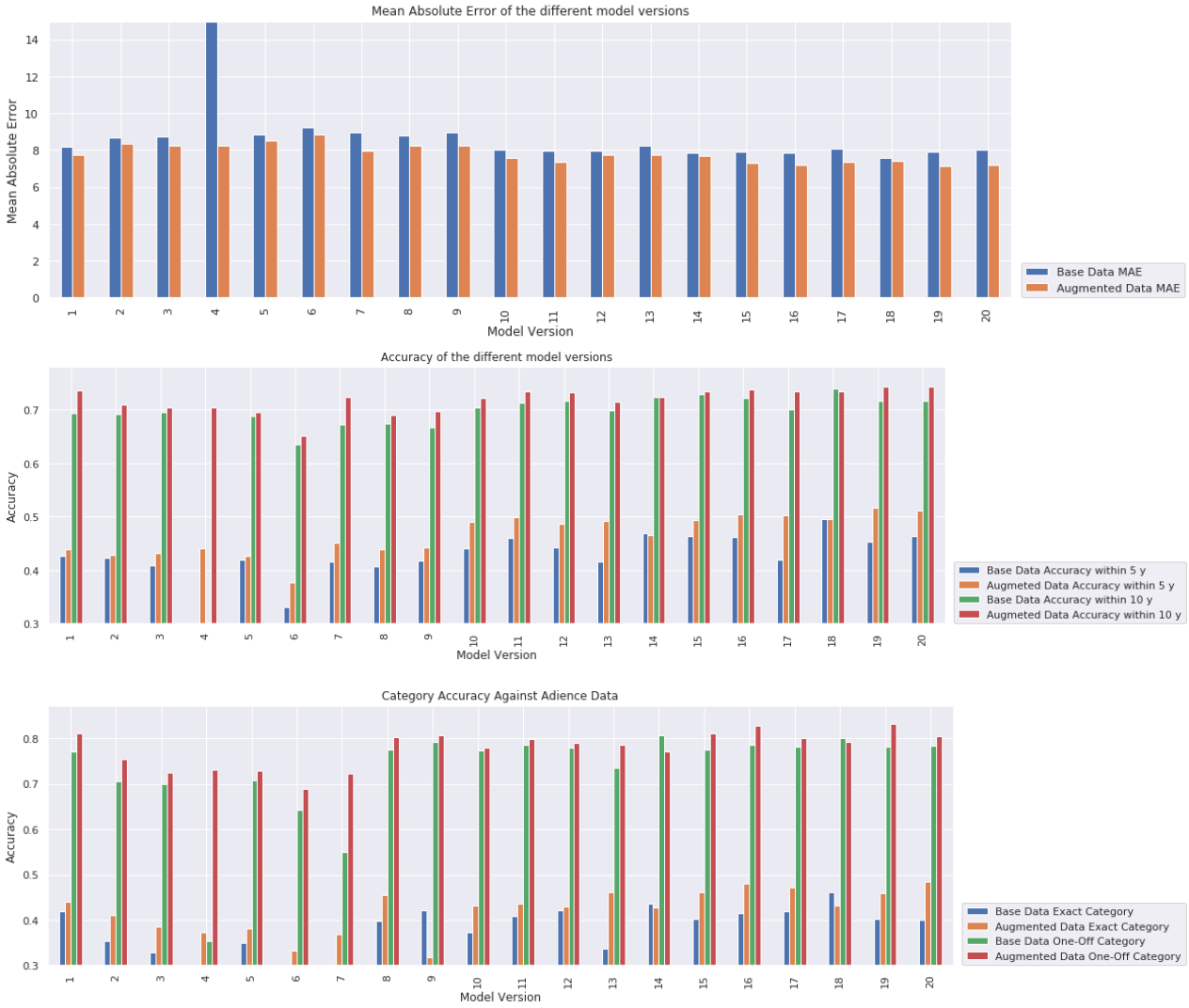
[17] http://dropzonejs.com

The model was trained with Mean Absolute Error as a loss function, and the Adam optimizer. The model achieved very good results from the get go. The loss was 8.2 on the base dataset and 7.8 on the augmented dataset. It achieved 70.7% accuracy with 10 years, and 43.4% on the base dataset, and 73.5% and 44.5% respectively for the augmented dataset. This result was not easy to improve on immediately.

Following that, I have gone through the process of modifying the model slightly and observing the result. The main versions recorded are available under the Model Refinement section in the Jupyter Notebook. Please note, however, that I have tried many variations that did not get recorded. Some large networks simply took too long to train and did not produce any good results that I had to cut them prematurely. Additionally, some variations to the optimizer produced chaotic results, or very slow learning, that I did not include them.

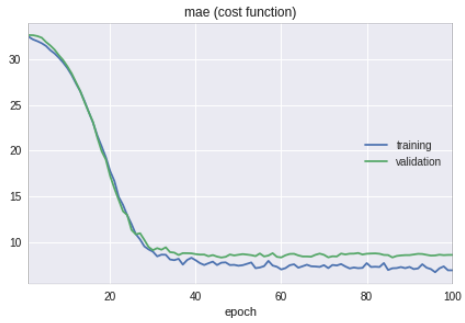
In order to keep the results comparable, I chose to standardize on a batch size of 32 (I have tried 8, and 64 as alternatives, but then elected to go for 32 as it balances out performance and speed). Additionally, I chose to standardize on 100 epochs, as most models seemed to stabilize by that time.

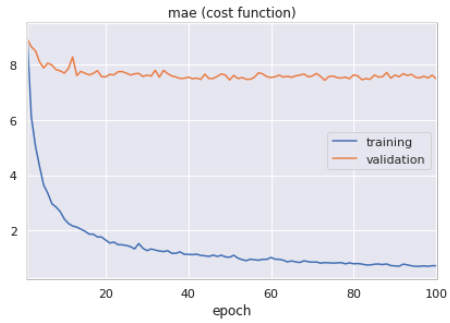
For the details of the different versions, you can refer to the Jupyter notebook, a summary is outputted for each of the tested models. Here's a summary of the performances of the different versions against different metrics:[18]



Some observations I have noticed during the refinement process:

* Dropout layers and batch normalization layers did not improve the final result. They did bring in the training loss and validation loss closer together, however, the final model performance was actually reduced. As an example, the figure on the left comes from version 2, which had both dropout and batch normalization, and the figure on the right is from version 19, which had none. Models similar to the one in the right outperformed those on the left:





* Version 4 had a slow learning rate, and performed extremely poorly (literally off the chart) when using the base dataset. However, it performed much better with the augmented dataset. I believe having a low learning rate, a large number of epochs, and a much bigger dataset would lead to a much better model.

* The default setup for the Adam optimizer is very good, my attempts at changing its parameters (version 4, version 5, version 6) produced less performance.

* Other optimizers such as Nesterov Adam, and Adadelta, produce less stable learning curves, and do not outperform Adam optimizer. Additionally, Adadelta optimizer diverged when training on the InceptionV3 extracted features. Base don this, default Adam optimizer seemed like a good choice.

* I have considered the Quantile Loss function as a loss function (version 6), it did not offer any improvement over the mean absolute error.

* Adding more hidden (up to 4) layers seemed to improve performance when the hidden layers are the same number of units. Varying the units did not improve the model (and in some cases reduced it).

The best results were obtained when 4 hidden layers were included. Version 15 produced excellent results with 4 layers of 50 units each, reaching almost 75% accuracy within 10 years. However, this result was not saved, and retraining the model again to try to achieve it was not successful.

Version 19 also performed very well, with 4 layers of 100 units each, and I have selected it as the best model.

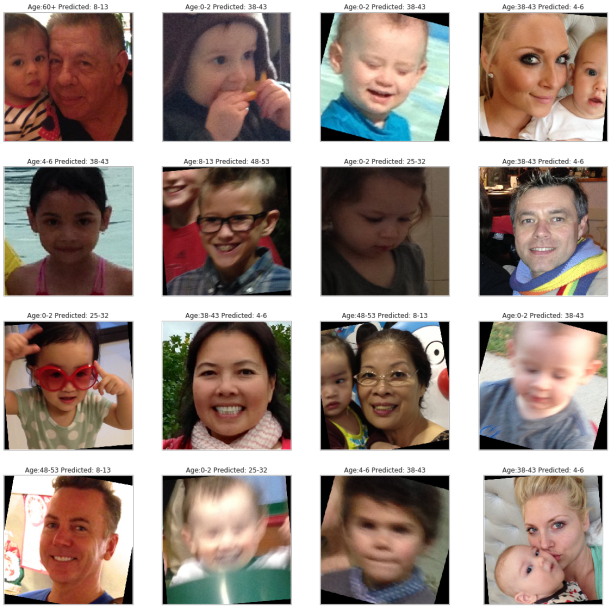
VII. RESULTS

Model Evaluation and Validation

The selected model is version 19 (ResNet50) as it has the best overall performance. It has the full following properties:

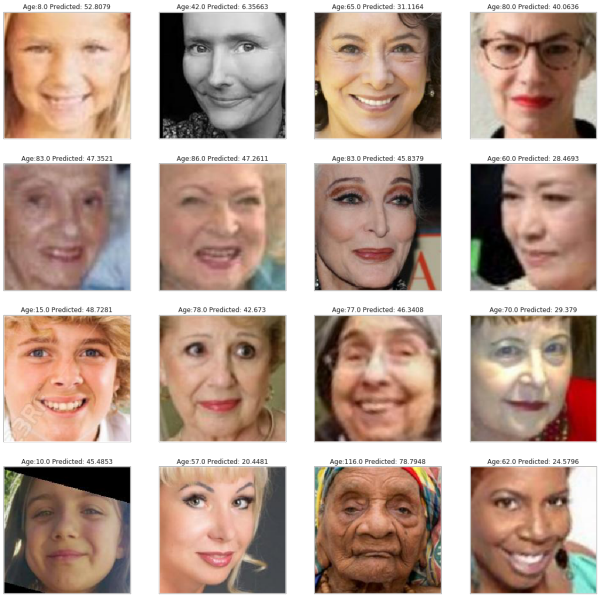
* It accepts an image and resized to 224 by 224 pixels.
* It passes the image through pre-trained ResNet50 model (ImageNet weights), and outputs a (7, 7, 2048) tensor.
* It passes it a flatten layer that flattens the tensor into 100,352 features.
* It passes through 4 hidden layers, each layer is 100 units followed by a ReLU activation.
* The total number of parameters trained by the model is 10,065,701.
* The model uses an Adam optimizer with standard parameters, MAE as a loss function, and was trained on the augmented dataset (9000 images total for test, and 4500 images for validation).

This is a random sample of adience images that were extremely misclassified by our model:



* Some of the images contain more than one face or part of face (image 1, 4, 6, 11, 16).
* Some of the images are very grainy, unclear, or have poor light conditions (5, 7, 12, 14, 15).
* Some contains unexpected objects in front of the face (2, 9).
* Some contain face painting (Not included in this sample).
* Some contain blacks on the corners due to rotation (3, 4, 6, 9, 11, 12, 13, 14, 15, 16).
* However, some are normal photos that should have been classified properly, but were missed by the model (8, 10). These cases are uncommon though.

Additionally, here is a random sample of UKTFace images that were misclassified by over 30 years:



* Some of the cases had very different lighting condition such as the grayscale image (2). Or a very grainy picture (1, 11)

* In some cases, I would argue that the model prediction is more accurate than the ground truth data, such as image # 14, which is clearly not 57 years old! Same goes for # 16, I believe she is younger than 62 years old.

* Aging affects different people in different ways, so we notice that the model fails to predict ages effectively for people over a certain age. Often, an 80 years old would look like someone in their 60, and a 60 year old would look like someone in their 80s.

Justification

The model has performed consistently well on training data, validation data, test data, and our adience benchmark data. Below is a summary of the performance of the model:

Metric	Performance	Target / Benchmark
Mean Absolute Error	7.1	Naive Benchmark: 32.6 / Proposal Target: < 10.0
Accuracy Within 10 Years	74.3%	Naive Benchmark: 13.4%
Accuracy Within 5 Years	51.7%	Naive Benchmark: 10.1%
Adience Exact Category	45.8%	Benchmark Study: 59.9%
Adience 1-Off Category	83.2%	Benchmark Study: 90.6%

This performance is very good given the amount of actual input images used (Only 1,500 original images are used for training/validation), and with regards to the adience benchmark, the dataset is considered a challenging dataset. Our model has, however, performed extremely well considering that it has not seen this data during training at all. Additionally, observing some of the images that were misclassified shows that part of the challenge is the quality of the dataset itself.

In fact, despite the limitations, this outcome exceeds that of several studies[19] observed during the research phase when preparing for this project.

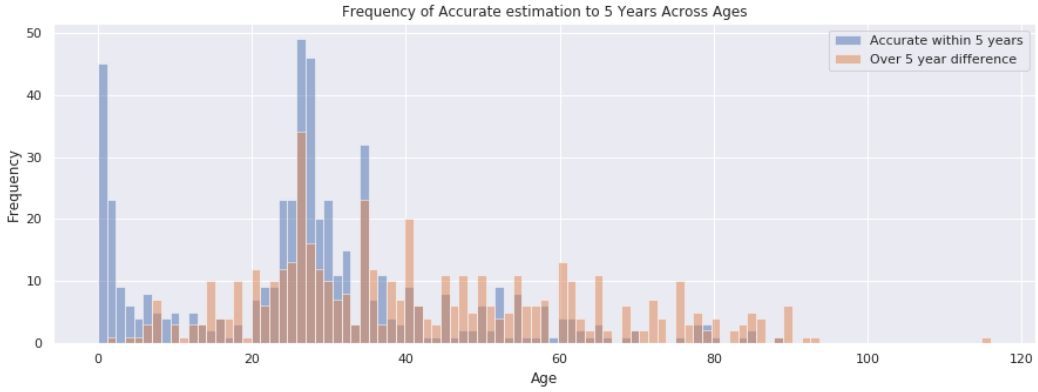
[19] http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=72B9E841852E2C33921B8F0004955069?doi=10.1.1.221.213&rep=rep1&type=pdf ; https://www.sighthound.com/technology/age-gender-emotion/benchmarks

V. CONCLUSION

Free-Form Visualization

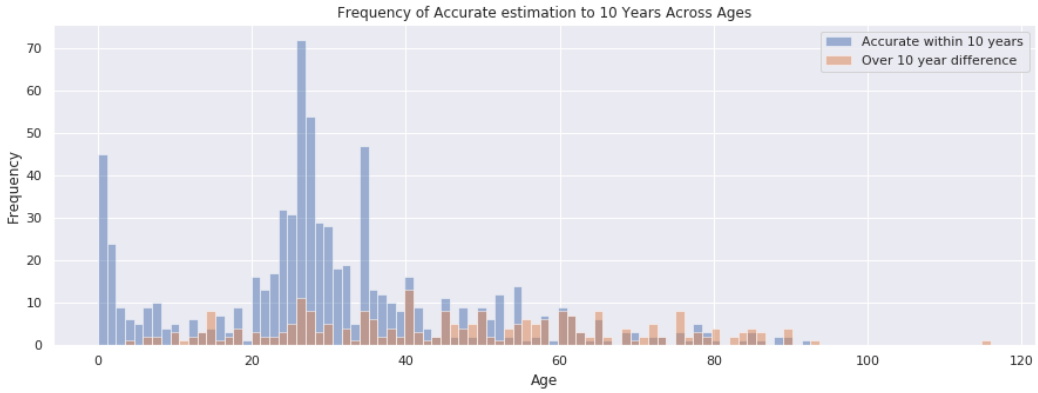
In this section, I would like to visualize two aspects of our model performance. First, how accuracy was different across the different ages. Does our model perform the same way evenly, or does it perform better in some cases than others?

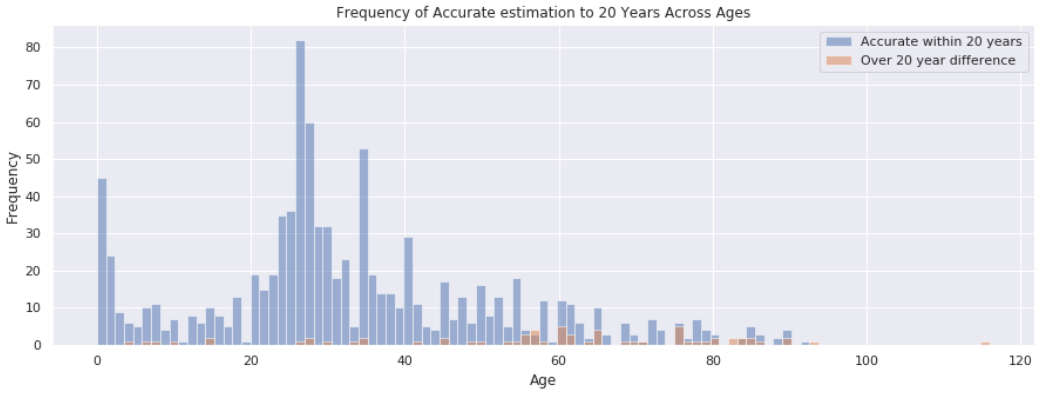
Let us first look at this histogram:



This histogram shows ages that were accurately estimated within 5 years. As we notice, the model performs much worse for ages above 40 years old than below (where it performs reasonably well). Additionally, we can see that there is a very high correlation between the number of photos available for that age, and the accuracy of prediction. This suggests that using a more evenly distributed dataset (rather than normally distributed), would probably produce better overall results.

Let us look at the same diagram, but now with 10 and 20 year accuracy respectively:





We can see that even though much more ages fall within the target boundary, we still see less accuracy for older ages. Could be due to what I have mentioned earlier, that aging is less consistent between people? Or is it just that our model has not seen enough photos of old people to know how to distinguish them well? I believe both aspects are partially to blame.

The other visualization I wanted to look at relates to the Race and Gender data that we have. Are we estimating gender and race evenly? Let's look at the data:

I find it very curious that white females are the most misclassified category. Even though (if you look at the general data distribution), whites are the most represented race in the data. Additionally, Asians are the most accurately estimated. Does this have to do with the quality of photos received from each age group? Or do white ladies age better? I believe if we passed the age as part of the data for the model to estimate the age, we may arrive at interesting results. However, this would require us to pass it every time we make a prediction, thus making our model weaker as a result.

Reflection

I can confidently say that I have learned a lot while working on this project. Throughout the nanodegree, a lot of the grunt work was already prepared for us, and we had to only apply a small portion of the process, which is most likely not even the most challenging. During this project, I had to learn to do this from end-to-end. These are the steps I have followed during the project:

* Reviewing past work that was performed on the field.
* Choosing appropriate datasets.
* Choosing a benchmark model, and developing a naive benchmark.
* Choosing a suitable workspace and a reasonable workflow.

* Analyzing and preprocessing the data.
* Understanding different aspects of face detection, and tools and techniques related to that.
* Understanding the different possible parameters.
* Designing different neural net architectures and trying them out, trying to achieve the best result.
* Setting up a process to visualize the data, both the base data and the model predictions.
* Improving the dataset through augmentation.
* Developing a base proof of concept application.
* Ensuring the preprocessing used in the application is consistent with that expected by the model.
* Evaluating the different model versions against the test data, and the benchmark data.
* Attempting to find interesting patterns in the data.

Dealing with images is especially challenging, since you are dealing with huge datasets, and pushing the limits of whatever computing environment you use. When I first started, I was not working in an organized way, and ended up wasting a huge amount of time running the same things again and again, until I decided to organize myself, and ensure continuity in the workspace. Saving any middle data, and not having to keep my kernel "alive" continuously in order to get results.

Playing around with parameters is a challenging thing, because neural networks are complex, and the best way to really evaluate the architecture of a neural network is to retrain it many times (over 30+ times according some recommendations). This is time consuming when you want to test modifying parameters, and if you make several modifications at once, you do not know the change is due to which modification. Thus, more experimentation gives you a better sense to select better parameters.

I believe the result I arrived at is quite good. It is definitely not perfect, and there is a lot of room for improvement, but it can be used already to make age predictions. A few years back I remember a gadget that described your emotions and estimated your age as you walked by it, and I was so impressed by it. Today, I believe my model can produce comparable, if not better, results.

Building a flask application was also a different experience for me. I am a software developer, and use other technologies often, but learning the basics of flask was also a valuable experience for the future.

Improvement

There are several areas of improvement I believe can be taken to enhance the results of this project:

* The most obvious improvement is: use more data, and with that, more complex DNN architectures.
* The application was developed as a very basic proof of concept. This can be enhanced by using a nicer user interface and better user experience. I did not want to dwell on this, and preferred to focus on the machine learning aspects.

* I have lost a great model version that performed very well by saving over it. I believe a better workflow, like having a “model runner function” that would generate a new file name every time the model is run would avoid this, rather than having to modify the file name manually and risk losing data.
* Face detection using Dlib is quite good, but there are many other alternatives. I would consider using a more complex face predictor than the one I have selected, or a combination of face detectors in case the first one fails.
* I have found a data augmentation library specific to faces, that may improve the face augmentation capabilities: https://github.com/iacopomasi/face_specific_augm
* I have considered using an ensemble method that takes the result of two different neural nets and averages them. Unfortunately though, since InceptionV3 did not perform comparably, I dropped this idea. It would still be good to consider that.
* The datasets are not perfect. In some cases, I’ve found mistakes in the data throughout the process. I do not believe it’s a good idea to manipulate the dataset since this would not make the results comparable, but at least noting down all of these cases and sending them to the dataset maintainers would be good.