Human Race Detection from Facial Images

Interim Report

COMP3359 – Artificial Intelligence Applications

Deni Susanto (3035361635)

1 Background

Human species can be divided into several groups based on their visual traits, e.g. skin tone, facial features, hair type, etc [1]. These groups are called the "*racial groups*". The terms and criteria used to identify someone belonging to a group have always been changing from time to time [2] [3]. Even the existence of *"race"* itself is still a debatable topic as some researchers even claimed that race does not actually exist and has no scientific basis [4]. Regardless of the true existence of race, most of our society still accept the concept of it, put themselves into a racial group, and able to visually distinguish human into a major racial group. With the advent of artificial intelligence (AI), especially in the field of computer vision, it is definitely compelling to teach a machine classifying us, human species. This application can be useful in public demographic study.

2 Objectives

There are three objectives in this project:

2.1 Build a Model to Classify Images into Racial Groups

The main objective of this project is to build a model that can accurately classify human into 5 major racial groups, namely White race (Caucasian), Black race (Ethiopian), Asian race (Mongolian), and Indian race (Caucasoid-Australoid), and others.

2.2 Infer the Relation between Facial Features and Race

To investigate the impact of race to human facial features, e.g. facial structure, ratios, and skin tone. Since deep learning models are likely to act like a black-box, i.e. hard to infer features relation, this project will also explore shallow learning algorithms which, hopefully, will be able to measure facial features significance in determining race.

2.3 Exploring, Experimenting, and Learning

This project will also serve as a tool to explore, experiment, and learn the various technology in artificial intelligence, especially in the field of computer vision. Different approaches will be used to build the model. With the challenges faced in implementing various methods, this project should help overcoming the steep learning curve of understanding AI techniques and algorithms.

3 About the Data

The dataset is retrieved from *UTKFace*, a large-scale face dataset with labelled information such as race, age, and gender [5]. The data consists of both male and female faces with age range from 0 to 116. The race information, which the project is interested in, is split into 5 groups, i.e. *White, Black, Asian, Indian,* and *Others* (including Hispanic, Latino, Middle Eastern, etc.). In total, the dataset consists of 24,106 images with different dimensions which contain faces in it. Unfortunately, there is data imbalanced between the *race* labels, with *White:* ~10.2k, *Black:* ~4.6k, *Asian:* ~3.6k, *Indian:* ~4k, and *Others:* ~1.7k. Additionally, there appear to be some mislabelled data as well, i.e. missing information or wrong label formatting.

4 Proposed Methodology

This section will discuss the planned methodology to carry out the project.

4.1 Data Pre-processing

The images retrieved from the *UTKFace* dataset is far from clean. Almost all the images have noise from the background or unnecessary objects for the model to train on. Additionally, the images are in different sizes and imbalanced. The below subsections will discuss the planned steps for data preprocessing.

4.1.1 Face Detection

For every image in the dataset, it will go through face detection to check if the face is visible on the image, and the number of faces in the picture. The purpose is to filter out any images that does not have face in it or in a poor quality such that the face is not detected by the face detection algorithm. This project will utilize the *Dlib*'s pretrained face detection algorithm which will return bounding boxes indicating the location of the faces in the image (refer to Figure 4.1).

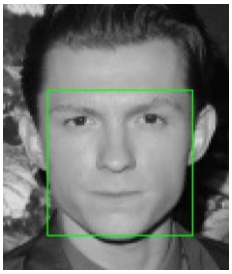


Figure 4.1 Dlib's Face Detector

4.1.2 Facial Landmark Detection

This step is done after the bounding boxes, representing face location, has been obtained. The bounding box will serve as the region of interest (RoI) for the facial landmark detection algorithm to work on. This project will utilize the pretrained 68-points facial landmark detection [6] to detect 68 coordinates of critical points on the face (refer to Figure 4.2). These points will be used later to extract facial structure features and for face alignment.

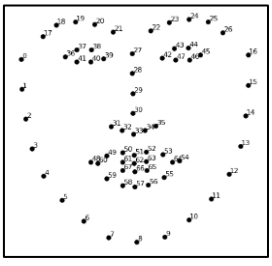


Figure 4.2 Facial Landmark Detection 68 Points

4.1.3 Face Alignment

Aligning the face for all the images is intuitively a good approach since it will help reducing the model capturing unnecessary features due to different faces orientation. Using the coordinates from the *facial landmark detector*, the orientation of the face (slope) can be calculated. For example, the coordinates of the 2 eyes can be used to calculate the slope between them. Then, using image processing techniques, the image can be rotated to such that the face will be aligned perpendicularly to the horizontal axis. See Figure 4.3 as an illustration.

Figure 4.3 Original image (left) and aligned image (right)

4.1.4 Crop to Face-only Images

The aim of this step is to clear out any noise, e.g. background, cloth, or other objects, so that only the face appears on the image. This is because the project aims to train a model that could predict race based on facial image only, so by reducing the image to face-only, it should help reducing the chance of the model learning unnecessary features.

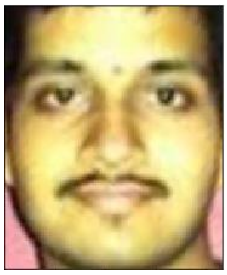


Figure 4.4 Expected result of face-only

To crop the image, the facial landmark coordinates will be used to determine where to crop. However, the facial landmark detection does not give the location of the forehead. Therefore, the golden ratio of human face could be useful in this situation, i.e. the average height of human forehead is 0.5 of the distance from eyebrow to chin. See Figure 4.4 for expected result.

4.1.5 Data Up-sampling and Augmentation

To address the imbalanced in the dataset, data up-sampling and augmentation will be used on the training set before feeding it for model training.

4.2 Modelling

As mentioned in the objective, this project will explore and experiment on different approaches. The next subsections will list the approaches that are planned to be used.

4.2.1 Convolutional Neural Network (CNN) Model

The first approach is to train a CNN model. Currently, the exact model architecture is still being considered. It is likely that the architecture will be based on some other popular architecture that have been proven to work well, e.g. VGG-16, AlexNet, ResNet50, etc. However, since the computational resource is limited (architecture like VGG-16 has over 140 million parameters

[7], which is too complex), this project might build a customized version of the mentioned architecture instead, while still applying the unique essence of the architecture.

4.2.2 Transfer Learning

Another approach is transfer learning. The transfer learning will use a pretrained model to extract features from the images and the only model being trained is multi-layer perceptron (MLP) model to classify the image from these extracted features. This approach will require less computational resources. And finally, its performance can be compared with the fully trained CNN model.

4.2.3 Shallow Learning Algorithms

Since both CNN model and transfer learning approach can be considered deep learning models, it is hard to infer which features are significant in determining one's race. Therefore, less complex models will also be considered, e.g. SVM, Random Forest, Multinomial Logistic Regression, etc. The features for the model input will be extracted from the facial landmark coordinates (refer to section 4.1.2). One concern is that the landmark information might not be sufficient to predict one's race.

4.3 Evaluation and Analysis

Evaluation will be done for every model built in this project. If there are some issues with the model, e.g. overfitting or underfitting, then counter measures will be taken to fix the issue. However, there is indeed some concerns for CNN models evaluation. Since training CNN model from scratch can be time consuming, there might be a limited trial-and-error attempts in remodifying and retraining the model, e.g. change data input, adding layers, or even completely change the model architecture.

5 Task Achieved, Results, and Difficulties

This section will discuss the current status of the project, task achieved, results of the experiments carried out, and difficulties encountered.

5.1 Literatures Review

As the implementation of image processing and computer vision is still unfamiliar for me, I spent some time to learn about the core libraries, recommended techniques, and available tools that might help during implementation. This phase has proven to help me identifying available solutions when facing difficulties.

5.2 Platform, Dependencies, and Environment Setup

As mentioned in previous sections, this project will be developed in HKU GPUFarm server due to its high computational resource requirement. After applying for the server access, I spent some times to install the libraries, download the raw data, pre-trained models, etc.

5.3 Data Cleaning and Preprocessing

Data cleaning and preprocessing have been carried out. The code for data cleaning and pre-processing based on the proposed methodology with improvements due to unseen difficulties. After data cleaning and preprocessing, the total number of images for modeling are around 20k images. Figure 5 illustrates the results after image processing.

Figure 5 Before and After Image Processing

5.3.1 Difficulties: Cleaning Time and Lost Progress

The first difficulty faced is that the face detection and facial landmark predictor takes too much time. To clean 1 image, it takes around 6-10 seconds, and in some cases, it takes up to 50 seconds when the image is large. Since there are more than 24,000 data to process, this could take more than 50 hours just to clean the data. The improvement I did to make the cleaning faster is to reduce the size of every image to have width of 400 pixels and greyscale them before doing facial and landmark prediction. This improves the algorithm significantly with only 1-3 seconds to process each image. However, it requires more effort since the landmarks and bounding box of the resized image needs to be retranslated and rescaled to its original size.

The second difficulty is that the progress of data cleaning often lost due to connection interuption with the GPUFarm server. The data cleaning should normally takes around 10 hours without error, so losing progress of data cleaning and restarting from the beginning is very time consuming. The issue is solved by applying checkpoint and auto-backup mechanishm due to corrupted file can happen when connection is down when pickeling a file. However, due to prior progress loses the data cleaning took much more time than estimated.

5.3.2 Improved Algorithm

i. Read the image
ii. Read image labels and proceed only if the age between 8 and 100 years old. This is because babies don't have fully developed facial features and the images of people older than 100 years old is not too many and often duplicated.
iii. Confirm that image have not been processed (checkpoint mechanism)
iv. Resize and greyscale the image (for time improvement)
v. Apply face detector and facial landmark predictor.
vi. Confirm that there exist a face on the image and that the face is not covered/cropped.
vii. Rotate the image to align the face

viii. Rotate and rescale the facial landmark to match the aligned face and the original image size. Then, save the landmark information together with the image file name and its label for input data of the shallow learning model.
ix. Save the rotated image and rescaled image to a new directory.
x. Marked the image file as *processed* (checkpoint mechanism)
xi. Backup the file that carry landmark information and processed image (backup mechanism for the corrupted file issue)

5.4 Data Splitting, Augmentation, and Generator

The data is split into training, validating, and testing set. The *train:test* set ratio is 0.8:0.2 and *validation:train* set ratio is also 0.8:0.2. I decided not to use Training-Dev set because the data is for sure from the same distribution, therefore in the error analysis, there should be no error contribution from data mismatch. For the training set, I balanced the dataset by random sampling and mirror the image (augmentation) since a mirrored face is still a face and one's race should persist. This should also help introducing more variance to the model.

Due to the large data input size, a data generator is required [8]. Since this project will try different models, the required input size might be different as well. Therefore, a *custom* data generator is needed where it can pre-process the image first. So, the custom data generator will read the image from directory, mirror it if necessary, then resize the image to the required input size (using bilinear interpolation). At first, I was not aware that a data generator is required until I received the exhausted resource error, hence this step took more time than expected to research and implement the custom generator.

5.5 CNN Modelling, Training, and Evaluation

The first model that was trained is based on the VGG architecture characteristic, i.e. using only (3x3) filter size in all the convolutional layers [7]. The model uses ADAM as optimizer, a very popular optimizer for deep learning models, due to its adaptive learning rate ability which could make the model training faster (refer to Figure 6 for the model details).

Then, I trained another model inspired by the AlexNet architecture, with filter size gradually decreasing in latter convolutional layer, using ReLU activation, max pooling after every convolutional layer, and 3 fully connected layers with 4096, 4096, and 1000 nodes respectively. I tried to add batch normalization layer which is said to help the model converge faster [9] (refer to Figure 6 for the model details).

After the model evaluation (which will be discussed in the next subsection), the models undergo tuning based on the training result. The training is done in using multiple GPU, i.e. multiple model was trained at the same time, since training 1 model takes a very long time to converge (4-16 hours). It is also worth noting that the model's evaluation couldn't afford to use cross validation due to computational resource and time limitation of this project.

VGG Custom Net

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 256, 256, 32)	896
dropout (Dropout)	(None, 256, 256, 32)	0
conv2d_1 (Conv2D)	(None, 256, 256, 32)	9248
dropout_1 (Dropout)	(None, 256, 256, 32)	0
max_pooling2d (MaxPooling2D)	(None, 128, 128, 32)	0
conv2d_2 (Conv2D)	(None, 128, 128, 64)	18496
dropout_2 (Dropout)	(None, 128, 128, 64)	0
conv2d_3 (Conv2D)	(None, 128, 128, 64)	36928
dropout_3 (Dropout)	(None, 128, 128, 64)	0
max_pooling2d_1 (MaxPooling2	(None, 64, 64, 64)	0
conv2d_4 (Conv2D)	(None, 64, 64, 128)	73856
dropout_4 (Dropout)	(None, 64, 64, 128)	0
conv2d_5 (Conv2D)	(None, 64, 64, 128)	147584
dropout_5 (Dropout)	(None, 64, 64, 128)	0
max_pooling2d_2 (MaxPooling2	(None, 32, 32, 128)	0
flatten (Flatten)	(None, 131072)	0
dense (Dense)	(None, 5)	655365

Total params: 942,373
Trainable params: 942,373
Non-trainable params: 0

AlexNet Optimized

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 256, 256, 64)	23296
batch_normalization (BatchNo	(None, 256, 256, 64)	256
max_pooling2d (MaxPooling2D)	(None, 85, 85, 64)	0
conv2d_1 (Conv2D)	(None, 85, 85, 128)	401536
batch_normalization_1 (Batch	(None, 85, 85, 128)	512
max_pooling2d_1 (MaxPooling2	(None, 28, 28, 128)	0
conv2d_2 (Conv2D)	(None, 28, 28, 192)	221376
batch_normalization_2 (Batch	(None, 28, 28, 192)	768
max_pooling2d_2 (MaxPooling2	(None, 9, 9, 192)	0
conv2d_3 (Conv2D)	(None, 9, 9, 256)	442624
batch_normalization_3 (Batch	(None, 9, 9, 256)	1024
max_pooling2d_3 (MaxPooling2	(None, 3, 3, 256)	0
flatten (Flatten)	(None, 2304)	0
dense (Dense)	(None, 4096)	9441280
batch_normalization_4 (Batch	(None, 4096)	16384
dense_1 (Dense)	(None, 4096)	16781312
batch_normalization_5 (Batch	(None, 4096)	16384
dense_2 (Dense)	(None, 1000)	4097000
batch_normalization_6 (Batch	(None, 1000)	4000
dense_3 (Dense)	(None, 5)	5005

Total params: 31,452,757
Trainable params: 31,433,093
Non-trainable params: 19,664

Figure 6 Model details

5.5.1 Model Result & Evaluation: VGG Custom Net

The first attempt on the custom VGG model resulted in overfitting where the model remembers too much of the training data. The accuracy difference between training and validation set is too far (refer to Figure 7). Since the data from validation and training set comes from the same source, therefore, the error should not come from data mismatch. Hence, the error occurred due to the high variance of the model, i.e. different performance on unseen data. Also, note that the validation and test accuracy displayed in Figure 7 is based on the model's lowest loss, which is in epoch 20 (lowest loss is recorded every 10 epochs).

An attempt to fix the issue is to make the model less complex by removing some convolutional layers. Another solution is that might work is to add regularization to the model. However, since the model already has dropout layer to begin with 0.4 rate, then reducing the model architectural complexity seems more reasonable.

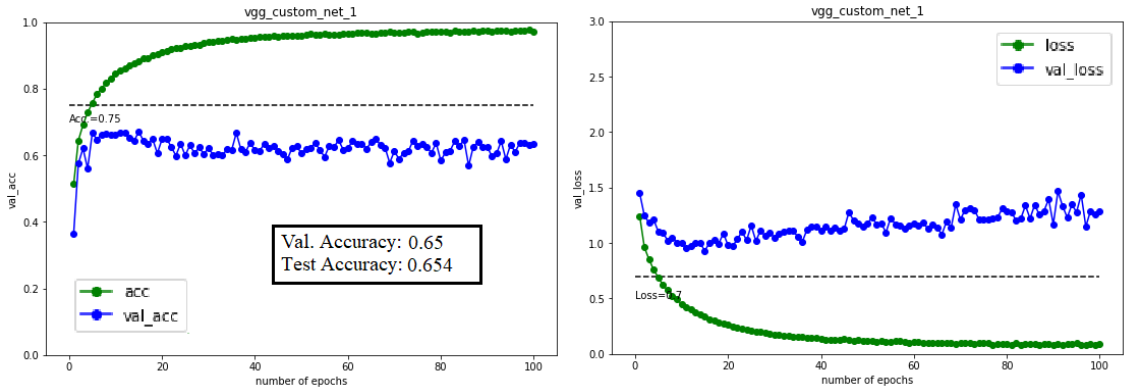


Figure 7 VGG Custom Net v.1

After multiple iterations of retraining and re-evaluation (over 20 version of VGG custom net was made), I finally arrived with a model that has a good fit to the data. It was achieved by dropping some convolutional layers and reduce the number of filters in each convolutional layer drastically. Additionally, the optimizer still uses ADAM with learning rate initialized to be very small, i.e. 0.00001. The dropout layers are still applied after each convolutional layer to reduce the risk of overfitting (Figure 8 for more details). To avoid overfitting, I also applied early stopping in the model when the validation accuracy gets saturated and the gap with the training accuracy gets wider. The validation and test accuracy are slightly better than the original model. After trying over 20 models with different hyperparameters settings, most of the model has accuracy of 0.65 – 0.7, which does not vary much. After outputting some of the misclassified test images, I found out that some images are mislabelled or indeed hard for even human to classify, e.g. picture is too small and blurry, faces that resemble more than one ethnicity. I believe that most of the source of error is from the unavoidable bias.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 256, 256, 8)	224
dropout (Dropout)	(None, 256, 256, 8)	0
max_pooling2d (MaxPooling2D)	(None, 128, 128, 8)	0
conv2d_1 (Conv2D)	(None, 128, 128, 8)	584
dropout_1 (Dropout)	(None, 128, 128, 8)	0
max_pooling2d_1 (MaxPooling2	(None, 64, 64, 8)	0
flatten (Flatten)	(None, 32768)	0
dense (Dense)	(None, 16)	524304
dense_1 (Dense)	(None, 5)	85

Total params: 525,197
Trainable params: 525,197
Non-trainable params: 0

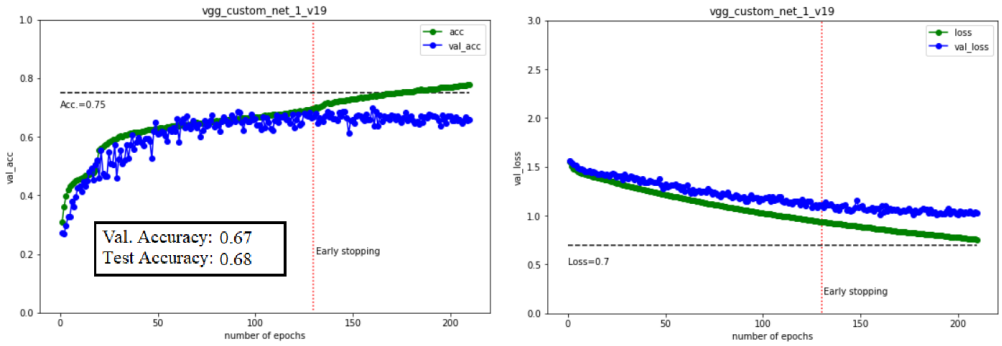


Figure 8 VGG Net v.19

5.5.2 Model Result & Evaluation: Optimized AlexNet

With much more trainable parameters and complex architecture, it is not surprising that the first attempt on the optimized AlexNet-based model suffers from overfitting just like the custom VGG net *v.1* model (refer to Figure 9). However, the accuracy from AlexNet seems to be better than the VGG Custom net, i.e. validation accuracy of 0.74 and test accuracy of 0.76.

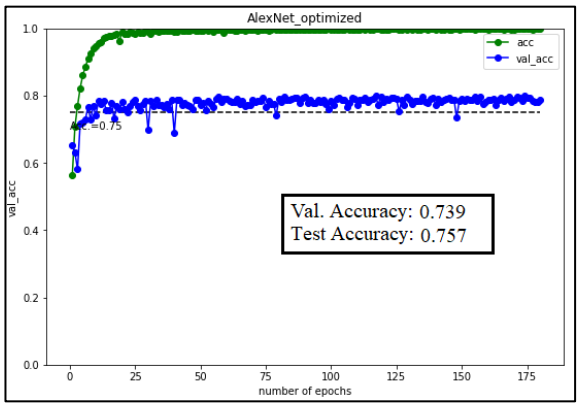
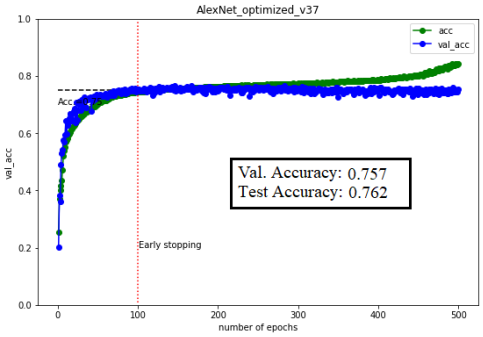


Figure 9 AlexNet Optimized v.1

After more than 40 attempts on constructing a more suitable model that avoids overfitting but still maintaining predictive performance of the model, i.e. not too simple that it has low bias, I finally managed to construct such model. Such model was achieved by removing reducing the number of filters on each convolutional layer, reducing the number of layers and the number of nodes in the fully connected layer. Additionally, the model was early stopped when the validation loss and accuracy get saturated to avoid overfitting. See Figure 10 for more details about the final model. The model resulted in both validation and test accuracy of roughly 0.76, which is much better than the custom VGG net model.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 256, 256, 64)	23296
max_pooling2d (MaxPooling2D)	(None, 85, 85, 64)	0
conv2d_1 (Conv2D)	(None, 85, 85, 128)	401536
max_pooling2d_1 (MaxPooling2	(None, 28, 28, 128)	0
conv2d_2 (Conv2D)	(None, 28, 28, 128)	147584
max_pooling2d_2 (MaxPooling2	(None, 9, 9, 128)	0
conv2d_3 (Conv2D)	(None, 9, 9, 192)	221376
max_pooling2d_3 (MaxPooling2	(None, 3, 3, 192)	0
flatten (Flatten)	(None, 1728)	0
dense (Dense)	(None, 16)	27664
dense_1 (Dense)	(None, 5)	85

Total params: 821,541
Trainable params: 821,541
Non-trainable params: 0



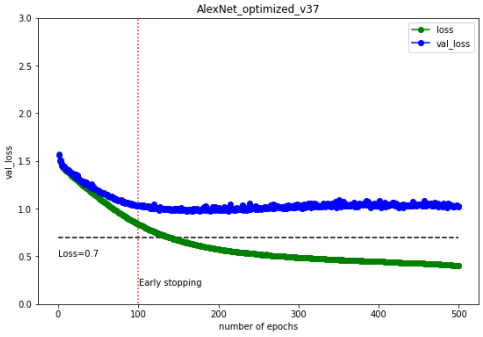


Figure 10 AlexNet V.37 Model Details

5.5.3 Difficulties

There are some difficulties faced during this phase. The first difficulty comes from TensorFlow's memory leaking bug. Due to limited callback functions on Keras training, I decided to create my own implementation of callback mechanism that allows to save model progress and training history with more flexibility. However, it appears that there is memory leak bug in Keras TensorFlow that will eventually cause Out of Memory (OOM) error. After extensive research, I finally managed to find the source of memory leakage and solve the issue by manually destroying certain variables, calling the garbage collector, and clearing the Keras session.

The second difficulty is regarding the resource and time limitation to carry out multiple experiments. Each model training takes a long time and the GPUFarm only allows to utilize a maximum of 4 GPUs at the same time. This limits the experiments needed as an inexperience CNN user, which undergo plenty of trial and error because of, for example, vanishing gradient problem, experimenting on model complexity, etc.

6 Future Works

The future work will mainly focus on the transfer learning implementation in order to stick with the project timeline. However, if there are time for some more experiments on the CNN models, implementing these might increase the performance of the model:

1. Further filter the data based on minimum source image dimension and B&W images. This is because after inspecting the misclassified images on the test set, I realized that plenty of the misclassified images are having a really small pixels size (below 70 pixels), making it really blurry, or the picture is in black and white which could be considered as an outlier of the image pool (since majority of the images are in color).
2. To expand the data size by doing data augmentation, i.e. mirroring every face image and some resampling. The aim is to introduce more variance to the model.
3. To omit the "*Others*" class from the dataset. This is because the *Others* class often have entries that is even challenging for human to determine the ethnicity, e.g. Middle Eastern (*Others)* often look like *Indian* ethnicity, or some *Hispanic* ethnicity looks like *Asian* in the image.

7 References

[1] M. J. Bamshad and S. E. Olson, “Does Race Exist?,” Scientific American, 2004.

[2] J. F. Blumenbach, T. Bendyshe, P. Flourens, R. Wagner, J. Hunter and K. F. H. Marx, The anthropological treatises of Johann Friedrich Blumenbach, London: Longman, Green, Roberts, & Green, 1865.

[3] C. S. Coon, The Races of Europe, New York: Macmillan Co., 1939.

[4] S. Worrall, “Why Race Is Not a Thing, According to Genetics,” National Geographic, 14 10 2017. [Online]. Available: https://www.nationalgeographic.com/news/2017/10/genetics-history-race-neanderthal-rutherford/.

[5] Y. Song and Z. Zhang, “UTKFace - Large Scale Face Dataset,” GitHub Pages, 2017. [Online]. Available: https://susanqq.github.io/UTKFace/.

[6] Z. Zhang, P. Luo, C. C. Loy and X. Tang, “Facial Landmark Detection by Deep Multi-task Learning,” ECCV, 2004.

[7] S. K. Basaveswara, “CNN Architectures, a Deep-dive,” Toward Data Science, 27 08 2019. [Online]. Available: https://towardsdatascience.com/cnn-architectures-a-deep-dive-a99441d18049.

[8] A. Amidi and S. Amidi, “A detailed example of how to use data generators with Keras,” Stanford University, [Online]. Available: https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly.

[9] S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” JMLR.org, Lille, France, 2015.

[10] E. Kolbert, “There’s No Scientific Basis for Race—It's a Made-Up Label,” National Geographic, 2017. [Online]. Available: https://www.nationalgeographic.com/magazine/2018/04/race-genetics-science-africa/.

[11] Arunava, “Convolutional Neural Network,” Towards Data Science, 25 12 2018. [Online]. Available: https://towardsdatascience.com/convolutional-neural-network-17fb77e76c05.

[12] A. Desphande, “A Beginner's Guide To Understanding Convolutional Neural Networks,” 2017. [Online]. Available: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/.

[13] A. Rosebrock, “Keras Conv2D and Convolutional Layers,” pyimagesearch, 31 12 2018. [Online]. Available: https://watch.lolesports.com/home.

[14] J. Brownlee, “Your First Deep Learning Project in Python with Keras Step-By-Step,” Machine Learning Mastery, 24 07 2019. [Online]. Available: https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/.

[15] A. Dennanni, “How to deal with image resizing in Deep Learning,” Neuronio.ai, 28 09 2018. [Online]. Available: https://medium.com/neuronio/how-to-deal-with-image-resizing-in-deep-learning-e5177fad7d89.

[16] I. Jose, “Saving your weights for each epoch — Keras callbacks,” Medium, 22 08 2019. [Online]. Available: https://medium.com/@italojs/saving-your-weights-for-each-epoch-keras-callbacks-b494d9648202.

[17] T. Boyle, “Dealing with Imbalanced Data,” Towards Data Science, 04 02 2019. [Online]. Available: https://towardsdatascience.com/methods-for-dealing-with-imbalanced-data-5b761be45a18.

[18] J. Brownlee, “How to Check-Point Deep Learning Models in Keras,” Machine Learning Mastery, 15 06 2016. [Online]. Available: https://machinelearningmastery.com/check-point-deep-learning-models-keras/.

[19] J. Brownlee, “Use Early Stopping to Halt the Training of Neural Networks At the Right Time,” Machine Learning Mastery, 10 12 2018. [Online]. Available: https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/.

[20] M. Desphande, “Understanding Advanced Convolutional Neural Networks,” Zenva, 22 11 2017. [Online]. Available: https://pythonmachinelearning.pro/understanding-advanced-convolutional-neural-networks/.