COMPUTER VISION (2023-2024)

REPORT: **Task 1**

Group members:

Francesc Josep Castanyer Bibiloni, <frcastab37@alumnes.ub.edu>, Xisco354
Sergio Hernández Antón, <shernaan7@alumnes.ub.edu>, shernaan7
Ana Victoria Galindo, <avictoga7@alumnes.ub.edu>, Ana

1. MODEL DESCRIPTION

The model architecture that worked best for us was VGG19. Not only it presents a simple structure, as we will be highlighting later, but also achieves the best results in our tests for both speed and performance (as we will explain later). After the VGG19 backbone, we added some fully connected layers to adapt the network to our output format. Moreover, in these layers we applied L2 regularization and added dropout layers between them to avoid overfitting and achieve the highest possible performance. The changes we made to the starting-kit are the backbone model, the L2 regularization on the fully connected layers and the addition of the dropout ones.

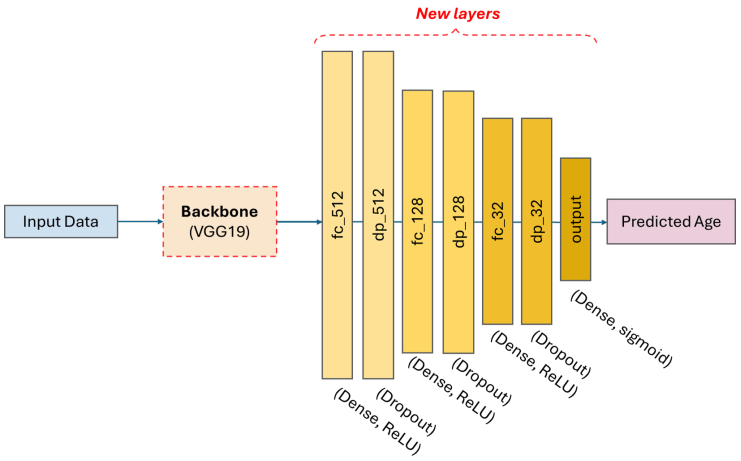

Figure 1: Illustration of the proposed final model.

2. BIAS MITIGATION STRATEGY

The dataset was clearly unbalanced and for this reason biases were high. Firstly, one could think that it is easy to do the data augmentation by just increasing the number of images of the categories that had less images, for example: age greater than 60, afroamerican ethnicity and other facial expressions. The problem with this method is that when you balance one attribute, another one can become unbalanced. To avoid this, we proposed to solve a linear problem with constraints. This linear problem consists in obtaining an integer number between n_{min} and n_{max} for each picture, such that the sum of imbalances is minimized. The problem is the following:

$$\text{Minimize} \quad \sum_{j=1}^{2} d_{\text{gender},j} + \sum_{j=1}^{3} d_{\text{ethnicity},j} + \sum_{k=1}^{4} d_{\text{emotion},k} + \sum_{k=1}^{4} d_{\text{age},k}$$

subject to:

- $\left|\sum_{i=1}^{2845} x_i M_{i,j} - \text{mean\_gender}\right| \leq d_{\text{gender},j}$, with j $\in \{1, 2\}$
- $\left|\sum_{i=1}^{2845} x_i M_{i,j+2} - \text{mean\_ethnicity}\right| \leq d_{\text{ethnicity},j}$, with j $\in \{1, 2, 3\}$
- $\left|\sum_{i=1}^{2845} x_i M_{i,j+5} - \text{mean\_emotion}\right| \leq d_{\text{emotion},j}$, with j $\in \{1, 2, 3, 4\}$
- $\left|\sum_{i=1}^{2845} x_i M_{i,j+9} - \text{mean\_age}\right| \leq d_{\text{age},j}$, with j $\in \{1, 2, 3, 4\}$

Solving the problem means obtaining a vector x which indicates how many times each picture should appear (modified) to balance the dataset. Our first idea was to force the images to appear between 5 and 10 times, which would augment the dataset while balancing it. It wasn't possible because it required too much memory. To deal with Codalab's memory limitations, we finally used $n_{min} = 1$ and $n_{max} = 5$.

We did not include each image just as in the original dataset the number of times the vector indicated. We created another function that, given an image and a natural number n, created n-1 transformations of the original image and appended them to the original dataset (with its corresponding Y label and Metadata attributes). To create the n-1 transformations (plus the original), the function chose randomly from the following list:

- adjust the brightness of the image.
- adjust the darkness of the image.
- apply a Gaussian blur to the image, smoothing it.
- apply a sharpening effect.
- apply a horizontal symmetry to the image, flipping it horizontally.
- combine horizontal symmetry and brightness adjustment.
- combine horizontal symmetry and darkness adjustment.
- combine horizontal symmetry and Gaussian blur.
- combine horizontal symmetry and sharpening.

Note that all images appear in their original form once. When n=1 we forced the function to avoid doing a transformation. With this from a dataset with 2845 images, we obtained a more balanced dataset with 4048 images.

We used the Gini coefficient to measure the imbalance in the dataset, obtaining the following results:

Original Gender G.C.: 0.0033
Augmented Gender G.C.: 0.0000

Original Ethnicity G.C.: 0.5509
Augmented Ethnicity G.C.: 0.4480

Original Emotion G.C.: 0.3430
Augmented Emotion G.C.: 0.2129

Original Age G.C.: 0.4173
Augmented Age G.C.: 0.3131

As we can see, the categories got balanced within the possible margins.

3. TRAINING STRATEGY

After some research, we concluded that doing the 2 stage training proposed in the starting-kit was the usual approach and also the one which achieves the highest performance. However, we have tried some variations of it, but without success. For instance, one of our tests was to divide the training in 3 stages, one for the regression head, another one for the second half of our backbone model and finally a third one where we trained all layers. Although interesting, this only led to overfitting or, in some cases, to a model which learned slowly and inconsistently.

For this reason, we decided to leave the 2 stage training as it was and focus on the rest of hyperparameters. We also tried training the model from scratch with random weights, but it wasn't successful. Hence, we used the pre-trained ImageNet model of VGG19 (transfer learning) and maintained 'Adam' as our optimizer, but switched the loss function to the mean average error, which is, along with the biases, what we are trying to minimize. As for the other settings, we went with a learning rate of 10^{-4}, 70 epochs and batch size of 32 for the first stage. Then, to perform a fine-tune of the model in the second stage, we decreased these values to 10^{-5}, 50 and 16, respectively.

Finally, for the augmented data training, we followed the same strategy, changing the batch size to 32 because the dataset was bigger, to avoid super high training costs.

4. EXPERIMENTS AND RESULTS

TABLE 1: Comparing the results of the final model using data augmentation vs. the baseline results **on the Test Set**. Better results are highlighted in bold.

	Data augmentation	**Gender (bias)**	**Expression (bias)**	**Ethnicity (bias)**	**Age (bias)**	**Avg bias**	**MAE**
Starting-kit	NO	0.154	**0.1023**	0.3441	3.1021	0.9256	4.8120
Starting-kit	YES	**0.0269**	0.1389	0.7588	**2.2669**	**0.7979**	**4.4403**
VGG19	YES	0.5895	0.2607	**0.0900**	4.3622	1.3251	5.4672

Once the data augmentation was done, the first experiment we did was using the starting-kit model. The results were better than without the data augmentation (see TABLE 1). It was clear that our data augmentation was successful.
The next experiment we performed was to try and get the best backbone we could. We used the list given in the presentation[1] to choose the architecture we wanted. We chose Xception because it had one of top-1 and top-5 highest accuracies on the ImageNet validation set and a reasonable time per inference step. Furthermore, we did several experiments maintaining the ImageNet weights and changing the number of dense layers and later changing the dropout value. However, nothing seemed to improve the starting-kit model, which had outstanding results just by doing the data augmentation.

[1] The list is from the webpage https://keras.io/api/applications/

TABLE 2: Results comparing different backbones trained in stage 2 (results applied on validation set).

	Data augmentation	**Gender (bias)**	**Expression (bias)**	**Ethnicity (bias)**	**Age (bias)**	**Avg bias**	**MAE**
VGG19 backbone	NO	0.1841	0.4345	0.6079	2.7594	**0.9965**	5.1153
Inception backbone	NO	0.0671	0.3807	1.1092	3.5371	1.2735	5.0469
Xception backbone	NO	0.2179	0.5564	0.8138	4.5899	1.5445	6.3345
ResNet50V2 backbone	NO	0.2647	0.4672	0.9256	3.6857	1.3358	5.8756

At this point we decided to try other models.
We tried to use InceptionResNetV2 because it had also top-1 and top-5 high accuracies. The problem with this model was that it required a huge amount of time to be trained (without data augmentation the first stage took more than one hour and a half) and the results were not that much better than the ones we previously had, so we decided to abandon this idea.
Then we tried with ResNet50V2. In this case the time was reasonable, but results were not better than the ones we had with the starting-kit model adding data augmentation.
Finally, we experimented with VGG19. At first, this model seemed to be better than Xception, InceptionResNetV2 and ResNet50V2. So we kept with the experiments. Firstly we tried with 50 epochs in the first training stage and 30 in the second, then we increased the epochs to 70 and 50, respectively, which resulted better. You can see the final results in TABLE 2 (dropout 0.3 and regularization in dense layers, same additional layers as the final model). In addition, this model deals nicely with Colab limitations since it does not take too much time nor memory to be trained.
Taking into account the times needed to do all the training in all the experiments, we decided to stick to these two principal experiments.

5. FINAL REMARKS

In conclusion, we have tried four different models with different hyperparameters but none seemed to improve the results from the starting-kit model. The best one was VGG19, which was close and dealt with Colab limitations. However, our data augmentation showed satisfactory results in all the models and experiments. When comparing the same model with and without data augmentation, always the model including data augmentation showed less biases and higher accuracy. If we had more time we could keep trying with other models to check if we can find another better than ResNet50.