Assignment 7
Generative Models

1 Introduction

In this lab, we'll build a facial detection model that learns the latent variables underlying face image datasets and uses this to adaptively re-sample the training data, thus mitigating any biases that may be present in order to train a debiased model.

2 Datasets

We'll be using three datasets in this lab. In order to train our facial detection models, we'll need a dataset of positive examples (i.e., of faces) and a dataset of negative examples (i.e., of things that are not faces). We'll use these data to train our models to classify images as either faces or not faces. Finally, we'll need a test dataset of face images. Since we're concerned about the potential bias of our learned models against certain demographics, it's important that the test dataset we use has equal representation across the demographics or features of interest. In this lab, we'll consider skin tone and gender.

1. **Positive training data** CelebA Dataset A large-scale (over 200K images) of celebrity faces.

2. **Negative training data** ImageNet Many images across many different categories. We'll take negative examples from a variety of non-human categories.

3. **Test data** Pilot Parliaments Benchmark(PPB) The PPB dataset consists of images of 1270 male and female parliamentarians from various African and European countries and exhibits parity in both skin tone and gender. The gender of each face is annotated with the sex-based "Male" and "Female" labels. Skin tone annotations are based on the Fitzpatrick skin type classification system, with each image labeled as "Lighter" or "Darker".

You can play around with displaying images to get a sense of what the training data actually looks like.
You can also create a "PPBFaceEvaluator" instance for the PPB dataset and display some example images. We'll use this dataset later on in the evaluation step.
Our goal is to build a model that trains on CelebA and achieves high classification accuracy on PPB across all demographics, and to thus show that this model does not suffer from any hidden bias.
What exactly do we mean when we say a classifier is biased? In order to formalize this, we'll need to think about latent variables, variables that define a dataset but are not strictly observed.

We'll use the term latent space to refer to the probability distributions of the aforementioned latent variables. Putting these ideas together, we consider a classifier biased if its classification decision changes after it sees some additional latent features. This notion of bias may be helpful to keep in mind throughout the rest of the lab.

3 CNN for facial detection

First, we'll define and train a CNN on the facial classification task, and evaluate its accuracy on the PPB dataset. Later, we'll evaluate the performance of our debiased models against this baseline CNN. The CNN model has a relatively standard architecture consisting of a series of convolutional layers with batch normalization followed by two fully connected layers to flatten the convolution output and generate a class prediction.

3.1 Define and train the CNN model

We'll define our CNN model, and then train it on the CelebA and ImageNet datasets using the tf.GradientTape class and the tf.GradientTape.gradient method.
You will find the architecture of the network explained in the starter code.

3.2 Evaluate performance of the standard CNN

Next, let's evaluate the classification performance of our CelebA-trained standard CNN on the training dataset and the PPB dataset. For the PPB data, we'll look at the classification accuracy across four different demographics defined in PPB: dark-skinned male, dark-skinned female, light-skinned male, and light-skinned female.
Take a look at the accuracies for this first model across these four groups. What do you observe? Would you consider this model biased or unbiased, and why?

4 Variational autoencoder (VAE) for learning latent structure

As you saw, the accuracy of the CNN varies across the four demographics we looked at. To think about why this may be the case, consider the dataset the model was trained on, CelebA. If certain features, such as dark skin or hats, are rare in CelebA, the model may end up biased against these as a result of training with a biased dataset. That is to say, its classification accuracy will be worse on faces that have under-represented features, such as dark-skinned faces or faces with hats, relative to faces have features that are well-represented in the training data! This is a problem.

Our goal is to train a debiased version of this classifier – one that accounts for potential disparities in feature representation within the training data. Specifically, to build a debiased facial classifier, we'll train a model that learns a representation of the underlying latent space to the face training data. The model then uses this information to mitigate unwanted biases by sampling faces with rare features, like dark skin or hats, more frequently during training. The key design requirement for our model is that it can learn an encoding of the latent features in the face data in an entirely unsupervised way. To achieve this, we'll turn to variational autoencoders (VAEs).

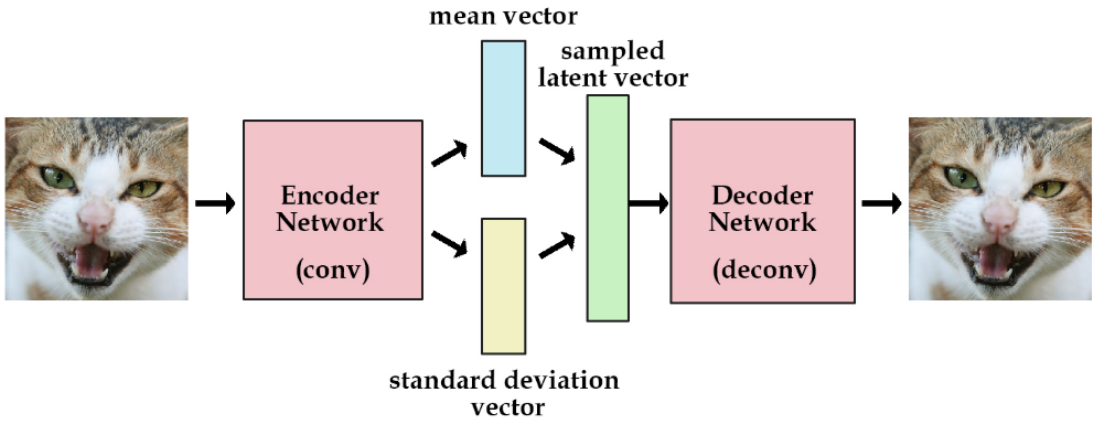

Figure 1

As shown in the schematic above, VAEs rely on an encoder-decoder structure to learn a latent representation of the input data. The encoder network takes in input images, encodes them into a series of variables defined by a mean and standard deviation, and then draws from the distributions defined by these parameters to generate a set of sampled latent variables. The decoder network then "decodes" these variables to generate a reconstruction of the original image, which is used during training to help the model identify which latent variables are important to learn.

Let's formalize two key aspects of the VAE model and define relevant functions for each.

4.1 Loss function

In practice, how can we train a VAE? In doing the reparameterization above, we constrain the means and standard deviations to approximately follow a unit Gaussian. Recall that these are learned parameters, and therefore must factor into the loss computation, and that the decoder portion of the VAE is using these parameters to output a reconstruction that should closely match the input image, which also must factor into the loss. What this means is that we'll have two terms in our VAE loss function:

1. Latent loss (L_{KL}): measures how closely the learned latent variables match a unit Gaussian and is defined by the Kullback-Leibler (KL) divergence. Note that the reparameterization trick is what makes this loss function differentiable!

2. Reconstruction loss ($L_x(x, \hat{x})$): measures how accurately the reconstructed outputs match the input and is given by the L2 norm of the input image and its reconstructed output.

The equations for both of these losses are provided below:

$$L_{KL}(\mu, \sigma) = \frac{1}{2} \sum_{j=0}^{k-1} (\sigma_j + \mu_j^2 - 1 - log\sigma_j)$$
$$L_x(x, \hat{x}) = ||x\hat{x}||_2$$

Thus for the VAE loss we have:

$$L_{VAE} = c.L_{KL} + L_x(x, \hat{x})$$

where c is a weighting coefficient used for regularization.

4.2 Reparameterization

VAEs use a "reparameterization trick" for sampling learned latent variables. Instead of the VAE encoder generating a single vector of real numbers for each latent variable, it generates a vector of means and a vector of standard deviations that are constrained to roughly follow Gaussian distributions. We then sample from the standard deviations and add back the mean to output this as our sampled latent vector. Formalizing this for a latent variable z where we sample $\epsilon \sim$ N(0,(I)) we have:

$$z = \mu + e^{(\frac{1}{2} log \sum)}.\epsilon$$

where μ is the mean and $\sum$ is the covariance matrix. This is useful because it will let us neatly define the loss function for the VAE, generate randomly sampled latent variables, achieve improved network generalization, and make our complete VAE network differentiable so that it can be trained via backpropagation. Quite powerful!

5 Debiasing variational autoencoder (DB-VAE) for facial detection

Now, we'll use the general idea behind the VAE architecture to build a model, termed a debiasing variational autoencoder or DB-VAE, to mitigate (potentially) unknown biases present within the training idea. We'll train our DB-VAE model on the facial detection task, run the debiasing operation during training, evaluate on the PPB dataset, and compare its accuracy to our original, biased CNN model.

5.1 The DB-VAE model

The key idea behind this debiasing approach is to use the latent variables learned via a VAE to adaptively re-sample the CelebA data during training. Specifically, we will alter the probability that a given image is used during training based on how often its latent features appear in the dataset. So, faces with rarer features (like dark skin, sunglasses, or hats) should become more likely to be sampled during training, while the sampling probability for faces with features that are over-represented in the training dataset should decrease (relative to uniform random sampling across the training data).

A general schematic of the DB-VAE approach is shown here:

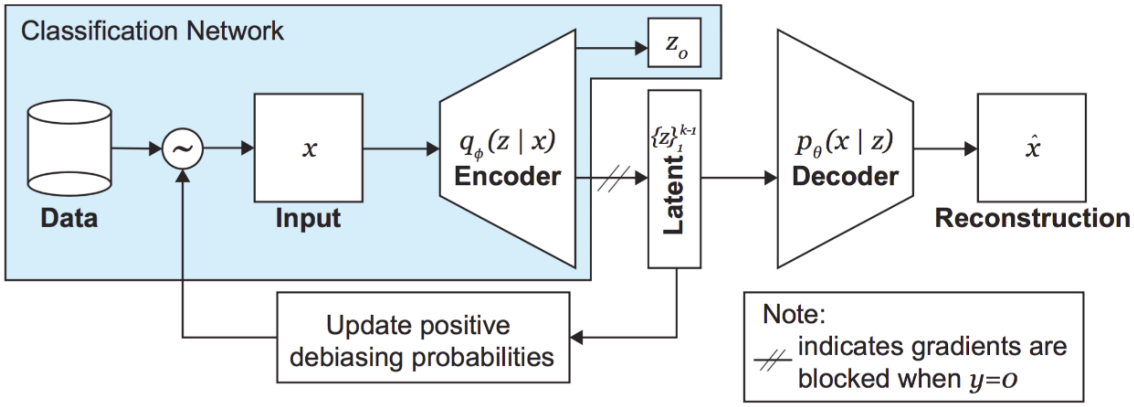

Figure 2

Recall that we want to apply our DB-VAE to a supervised classification problem – the facial detection task. Importantly, note how the encoder portion in the DB-VAE architecture also outputs a single supervised variable, z_o , corresponding to the class prediction – face or not face. Usually, VAEs are not trained to output any supervised variables (such as a class prediction)! This is another key distinction between the DB-VAE and a traditional VAE.

Keep in mind that we only want to learn the latent representation of faces, as that's what we're ultimately debiasing against, even though we are training a model on a binary classification problem. We'll need to ensure that, for faces, our DB-VAE model both learns a representation of the unsupervised latent variables, captured by the distribution $q_\phi(z|x)$, and outputs a supervised class prediction z_o, but that, for negative examples, it only outputs a class prediction z_o

5.2 Defining the DB-VAE loss function

. This means we'll need to be a bit clever about the loss function for the DB-VAE. The form of the loss will depend on whether it's a face image or a non-face image that's being considered.

For face images, our loss function will have two components:

1. **VAE loss**(L_{VAE}): consists of the latent loss and the reconstruction loss.

2. **Classification loss** ($L_y(y, \hat{y})$): standard cross-entropy loss for a binary classification problem.

In contrast, for images of non-faces, our loss function is solely the classification loss.

We can write a single expression for the loss by defining an indicator variable $\mathcal{I}_f$which reflects which training data are images of faces ($\mathcal{I}_f(x) = 1$) and which are images of non-faces ($\mathcal{I}_f(x) = 0$). Using this, we obtain:

$$L_{total} = L_y(y, \hat{y}) + \mathcal{I}_f(x)\Big[L_{VAE}\Big]$$

5.3 DB-VAE architecture

Now we're ready to define the DB-VAE architecture. First, let's define some key parameters for our model: the number of latent variables, the number of supervised outputs, and the starting number of filters for the first convolutional layer in the encoder.

To build the DB-VAE, we'll define each of the encoder and decoder networks separately, create and initialize the two models, and then construct the end-to-end VAE. We'll go through each of these steps in turn.

5.4 Adaptive resampling for automated debiasing with DB-VAE

So, how can we actually use DB-VAE to train a debiased facial detection classifier? Recall the DB-VAE architecture. As the input images are fed through the network, the encoder learns an estimate $\mathcal{Q}(z|X)$ of the latent space. We want to increase the relative frequency of rare data by increased sampling of under-represented regions of the latent space. We can approximate $\mathcal{Q}(z|X)$ using the frequency distributions of each of the learned latent variables, and then define the probability distribution of selecting a given datapoint x based on this approximation. These probability distributions will be used during training to re-sample the data.

You'll write a function to execute this update of the sampling probabilities, and then call this function within the DB-VAE training loop to actually debias the model.

First, you will find a short helper function get_latent_mu that returns the latent variable means returned by the encoder after a batch of images is inputted to the network.
Then, let's define the actual resampling algorithm 'get_training_sample_probabilities'. Importantly note the argument 'smoothing_fac'.
This parameter tunes the degree of debiasing: for 'smoothing_fac=0', the re-sampled training set will tend towards falling uniformly over the latent space.

Now that we've defined the resampling update, we can train our DB-VAE model on the CelebA/ImageNet training data, and run the above operation to re-weight the importance of particular data points as we train the model.
Remember again that we only want to debias for features relevant to faces, not the set of negative examples.

5.5 Evaluation on Pilot Parliaments Benchmark (PPB) Dataset

Finally let's test our DB-VAE model on the PPB dataset.

We'll evaluate both the overall accuracy of the DB-VAE as well as its accuracy on each the "Dark Male", "Dark Female", "Light Male", and "Light Female" demographics, and compare the performance of this debiased model against the biased CNN from earlier in the lab.

Here are some example images from the PPB dataset.

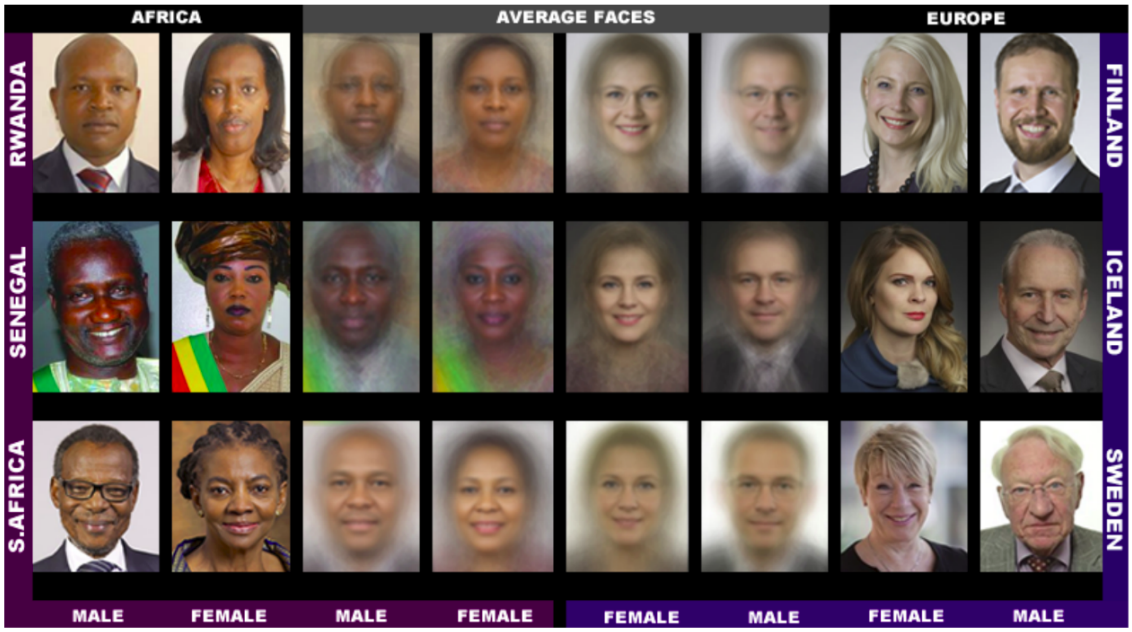

Figure 3

To assess performance, we'll measure the classification accuracy of each model, which we define as the fraction of PPB faces detected. By comparing the accuracy of a model without debiasing and our DB-VAE model, we can get a sense of how effectively we were able to debias against features like skin tone and gender.

Let's evaluate our debiased model on the PPB test dataset.

We can calculate the accuracies of our model on the whole PPB dataset as well as across the four demographics proposed and visualize our results comparing to the standard, biased

CNN.

6 Requirements

- Complete the "make_standard_classifier" function in order to build the CNN model mentioned in Section 3.1.
- Answer the question in Section 3.2.
- Complete the "vae_loss_function" function to follow the description in Section 4.1.
- Complete the "sampling" function to follow the description in Section 4.2.
- Complete the "debiasing_loss_function" function to follow the description in Section 5.2.
- Use the pr-defined functions to build the encoder-decoder model.
- Complete the "get_training_sample_probabilities" mentioned in Section 5.4.
- Try to change the smoothing factor and observe what changes it may have on the results.
- Complete the training loop.

7 Notes

- Note that it is better to use Python 2 notebook in this assignment.
- Parts of this assignment are based on the MIT deep learning course.
- The starter code for the assignment can be found in the resources section in Piazza.
- You should deliver a report explaining all your work.
- Cheating will be severely penalized (for both parties). So, it is better to deliver nothing than deliver a copy. Any online resources used must be clearly identified.