Machine Learning Engineer Nanodegree

Capstone Project - Augmented Face Recognition

Marc Minnee
March 2020

I. Definition

Project Overview

Image and face recognition is getting a lot of attention in AI and machine learning. Applications can be used for both good and bad purposes.

The good: easy and swift image recognition solves the problem of authentication in ever expanding granular digital authentication workflows. No more password hassle.
The bad: state surveillance threatens human rights when people are constantly being watched and checked upon.

The ugly: predictive policing with image recognition shows too many false positives based on biased (racial) inputs and poor performing models, not recognising subtle differences in facial attributes from human races.

The last one is especially interesting to solve. We should be able to enhance models by reducing biased inputs and account for differences in human race. In order to prevent the ugly becoming the bad…

Problem Statement

The last one is especially interesting to solve. If we'd like to combat the ugly before getting worse, we need to understand why modelling predictive policing is controversial. First of all, used models are ***opaque*** in the sense that authorities don't want to open source these in order to outsmart criminals. This way we cannot objectively determine the performance and evaluate the inputs used to train the model. And if we are to investigate the inputs, we likely find datasets, based on historical data of imprisoned persons which are biased towards Afro-Americans and Latinos.

So, the problem has its origin in ***biased datasets***, training models with more images of a certain kind, also described by Cathy Neil.

We should be able to enhance models by reducing biased inputs and account for differences in human race. In order to prevent the ugly becoming the bad…

So it would be interesting to know if we could model face recognition with face features corresponding to gender and race or skin type labeling, as a means to recognise different gender and races in images in order to augment:

1. Model inputs: is the model fed with a balanced set of images of different gender and races or skin types?
2. Model outputs: are we able to enhance model performance given the (unbiased) features of the input dataset?
3. Performance of face recognition

This model should at least perform better at recognising races in images, as an intermediate feature to be used in unbiased predictive policing and to help audits performed on these models in order to counterbalance state surveillance.

Metrics

As mentioned, the model should perform at recognising races in images, in our model labeled as skin type and gender; this is a typical classification problem. Our benchmark model will be the Gender shades project, where the model evaluation focuses on gender classification to show the need for increased transparency in the performance of any AI products and services that focused on human subjects. Bias in this context is defined as having practical differences in gender classification error rates between groups.

According to what is evaluated in the benchmark model: intersectional subgroups - darker males, darker females, lighter males, lighter females - I intend to use somewhat different classes, labels as black, blond, caucasian, and ‘tanned’ men and women, so I model a 4 ‘skin type’ x 2 gender class labeling.

Metrics are also to be inferred from the dog classification project; instead of dog breed I intend to classify on the skin type and gender classes, with the highest probabilities.

II. Analysis

Data Exploration

As a base input I used a processed CelebA dataset from Kaggle, with 202,599 number of face images of various celebrities. with attribute labels for each image. There are 40 attributes. "1" represents positive while "-1" represents negative, in the file `list_attr_celeba.csv`:

```
5_o_Clock_Shadow, Arched_Eyebrows, Attractive, Bags_Under_Eyes, Bald, Bangs,
Big_Lips, Big_Nose, Black_Hair, Blond_Hair, Blurry, Brown_Hair, Bushy_Eyebrows,
Chubby, Double_Chin, Eyeglasses, Goatee, Gray_Hair, Heavy_Makeup,
High_Cheekbones, Male, Mouth_Slightly_Open, Mustache, Narrow_Eyes, No_Beard,
Oval_Face, Pale_Skin, Pointy_Nose, Receding_Hairline, Rosy_Cheeks, Sideburns,
Smiling, Straight_Hair, Wavy_Hair, Wearing_Earrings, Wearing_Hat,
Wearing_Lipstick, Wearing_Necklace, Wearing_Necktie, Young.
```

However we can extract many facial features, the label we are interested in is race or skin type. We need to infer this label from the other features in our dataset, by labeling a sample of the images by hand and then use a data augmentation or a generative

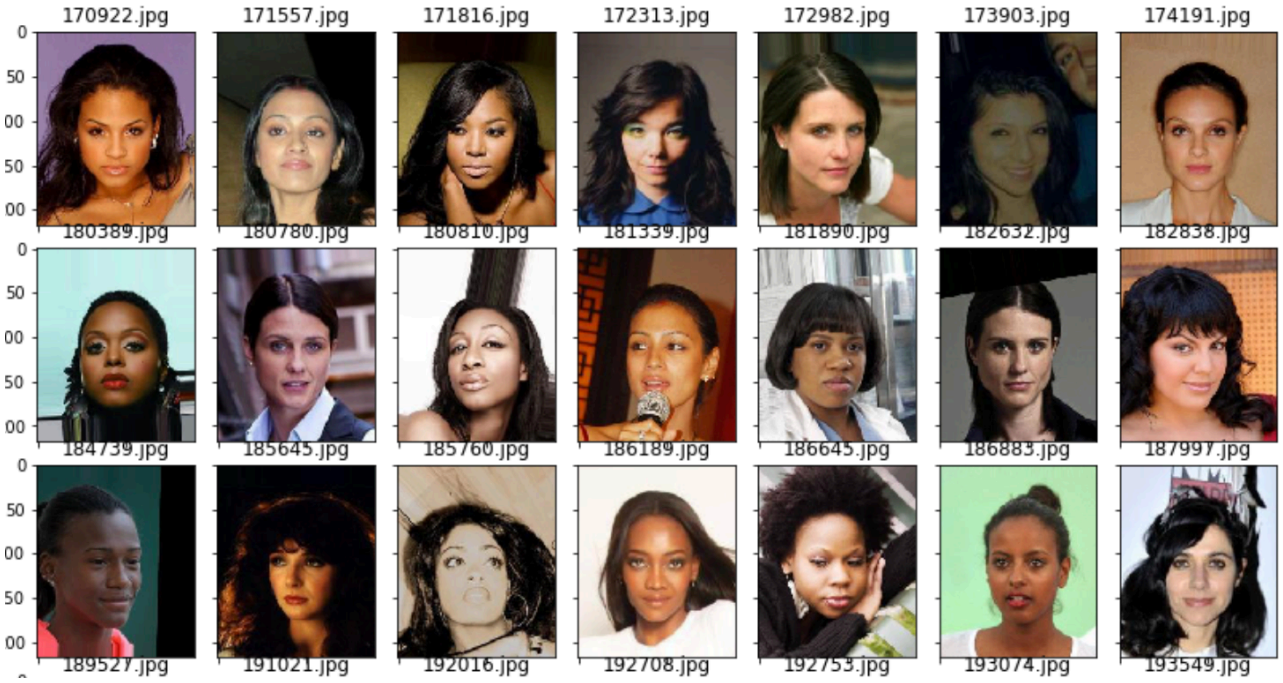
modelling technique, a GAN, to get more training data with skin type labels. It would have been better to use the Gender Shades dataset, which already accounts for different skin types, unfortunately the organization managing this dataset was not responsive to my data acquisition requests. Back to square 1: the CelebA dataset.

Based on the outcome of the Gender shades project, stating it is especially hard to accurately classify black women in face recognition, my aim was to assemble a balanced dataset with 4 x 2 classes (skin type x gender): **black, caucasian, blond and 'tanned'** (everything **not** black, caucasian or blond) **men and women**. Yes, I admit having chosen these classes somewhat arbitrarily or maybe even awkwardly, but the goal here is to generate a balanced dataset, not so much account for segmentations of human race (I'd rather stay away from this sensitive subject).

Exploratory Visualization

Given the feature set, I tried to select *black women* based on some features I thought were useful. Code is from the Filter notebook:

```
images = []
with open(os.path.join('list_attr_celeba.csv'), mode='r') as csv_file:
    csv_reader = csv.DictReader(csv_file, delimiter=',')
    line_count = 0
    i = 0
    for row in csv_reader:
        if line_count == 0:
            print(f'Column names are {", ".join(row)}')
            line_count += 1
        if line_count >= 170400:
            if row["Male"] == '-1' and row["Pointy_Nose"] == '-1' and
row["Bushy_Eyebrows"] == '-1' and row["High_Cheekbones"] == '-1' and
row["Big_Nose"] == '1' and row["Pale_Skin"] == '-1' and row["Black_Hair"] == '1'
and row["Young"] == '1':
                images.append(row["image_id"])
                i += 1
        line_count += 1
    print(f'Processed {line_count} lines and {i} matches.')
```



And here is the visualization. As expected, the results are not quite discriminatory, although I managed to get some samples right.

Algorithms and Techniques

Now we can account for unbiased inputs, a balanced dataset to work with as input for training. I will use the exact same algorithms and techniques from the dog breed classifier, the hard part is in architecting the layers of the CNN. Why CNN? From the Manning book “Deep Learning with PyTorch”, this excerpt explains why we need convolution:

'We also said that, if we ought to recognize patterns corresponding to objects, like an airplane in the sky, we will likely need to look at how nearby pixels are arranged, and we would be less interested at how pixels that are far from each other appear in combination. Essentially, it doesn’t matter if our image of a Spitfire has a tree or cloud or kite in the corner or not.

In order to translate this intuition in mathematical form, we could compute the weighted sum of a pixel with its immediate neighbors, rather than with all other pixels in the image. This would be equivalent to building weight matrices, one per output feature and output pixel location, in which all weights beyond a certain distance from a center pixel are zero. This will still be a weighted sum, i.e. a linear operation.'

Eli Stevens, Luca Antiga. 'Deep Learning with PyTorch MEAP V12'.

To find the right parameters for the CNN is truly something of an art: how many convolution layers, input/output ratio’s, pooling, dropout? I admit googling for some answers, in the end it’s about getting it right by some rule of thumb.

So I used 3 to 4 convolution layers with corresponding max pooling layers. The filters doubled in each layer, concluding with 1 fully connected linear layer outputting 8 nodes, our 8 classes of skin type x gender. For the rest, I followed the dog classification notebook to train en test the model.

Benchmark

The Gender shades scores would be a good benchmark for our test to see if we can predict correct skin type and gender. These are the scores of the commercial AI applications in the Gender shades project:

Gender Classifier	Darker Male	Darker Female	Lighter Male	Lighter Female	Largest Gap
Microsoft	94.0%	79.2%	100%	98.3%	20.8%
FACE++	99.3%	65.5%	99.2%	94.0%	33.8%
IBM	88.0%	65.3%	99.7%	92.9%	34.4%

III. Methodology

Data Preprocessing

I decided to filter the CelebA dataset on the features that are making any sense to my chosen 8 classes and ended up after some probing with these 10 attributes:

```
Big_Lips, Big_Nose, Black_Hair, Blond_Hair, Brown_Hair, Bushy_Eyebrows, Male,
Pale_Skin, Pointy_Nose, Young.
```

Tweaking the features (+1, -1) and feeding this in a 'Filter' notebook generates a reasonable subset to work with per class, however I needed to do quite some data cleansing afterwards, like women labeled as men (transgenders, anyone?), red hair typed as blond, this kind of noise.

Indeed, the real deal of machine learning is not the modern art of training your model, but the medieval job of hand picking the dirt from your input data.

When done I had a nice subset of 50 to 100 samples for each class. Time to get more. First, I tried generating more samples with a GAN, used from the PyTorch tutorial however results were too poor to make use of them, because of the very small subset of samples I had to feed the GAN with. Plan B: I used a very handy and simple data augmentation python library, `imgaug` in order to rapidly get more training data, using a 'Transform' notebook. I ended up with about 10k samples for each of the 8 classes.

Implementation

I followed along with the prepared notebook for the dog breed classification project, so I started out with a simple data transform of image resizing to 64x64 pixels because I already augmented the data in the preprocessing phase:

```
dset = datasets.ImageFolder(root=dataroot,
                            transform=transforms.Compose([
                                transforms.Resize(image_size),
                                transforms.CenterCrop(image_size),
                                transforms.ToTensor(),
                                transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5,
0.5)),
                            ]))

# Create the dataloaders
train_set, test_set, val_set = torch.utils.data.random_split(dset, [48000,
16000, 15733])

loaders = list(map(lambda x: torch.utils.data.DataLoader(x,
batch_size=batch_size,
                                         shuffle=True, num_workers=workers),
[train_set, test_set, val_set]))

print(dset)


Dataset ImageFolder
    Number of datapoints: 79733
    Root location: data/celebsaugmented
```

```
    StandardTransform
Transform: Compose(
               Resize(size=64, interpolation=PIL.Image.BILINEAR)
               CenterCrop(size=(64, 64))
               ToTensor()
               Normalize(mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5))
           )
```

As we have 79733 data samples, I split this set to train/test/validate 60/20/20:

```
train_set, test_set, val_set = torch.utils.data.random_split(dset, [48000,
16000, 15733])

loaders = list(map(lambda x: torch.utils.data.DataLoader(x,
batch_size=batch_size,
                                        shuffle=True, num_workers=workers),
[train_set, test_set, val_set]))
```

For this matter I used quite a simple CNN architecture, borrowed from many tutorials:

```
Net(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (fc1): Linear(in_features=8192, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=8, bias=True)
  (dropout): Dropout(p=0.3, inplace=False)
)
```

So 3 convolution layers and two fully connected layers to output my 8 classes. Up to the training part.

I could have used some cloud provider GPU instance, but I wanted to see how my turbo MacBook was performing on CPU… a pity CUDA is not Mac's friend any longer. So after about 15 minutes per epoch, 20 epochs in total training, the first model performed 40% on the split-off test set, which was as expected and more than the 10% threshold given by the dog breed classification project, but with far less classes.

Let's see how transfer learning performed. I chose to pick the ResNet50 model because of the good references I found on training these kind of images, so again picking from the dog breed classification project:

```
model_transfer = models.resnet50(pretrained=True)

for param in model_transfer.parameters():
    param.requires_grad = False

model_transfer.fc = nn.Linear(2048, 8, bias=True)
fc_parameters = model_transfer.fc.parameters()
```

Training with a smaller set (of about 4k samples in total) reached less accuracy, so that's why I generated 10k samples per class. But, since I used a small base of 50-100 samples per class before augmenting them, I feared overfitting. Because the samples were generated from only a couple of 'parents', maybe this 'incestuous' action had been clouding the machine's judgement. To be sure, I generated a new testset, based on samples that were not derived from the original subsets.

Refinement

Basically, refinement was made by training the CNN in more epochs and shuffling the data again after loading the previously saved parameters. However, I noticed that by doing that a couple of times, so training, saving, loading and training again I risked overfitting, because each time I shuffled the data again, training, test and validation data samples were blurring into each other during the course of action….

So back again to the Filter notebook, this time sampling from the end of the CelebA data set (sample # > 170000) selecting about 50 samples per class and augmenting again until we have some 4k samples per class as a new testset feeding into our model. I trained the model again with no data shuffling and tested it against true unseen test samples.

Furthermore, I tried augmenting the model to have more capacity and added 2 more convolution layers and 1 more connected layer to the model:

```
OtherNet(
  (conv1): Conv2d(3, 6, kernel_size=(3, 3), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(3, 3), stride=(1, 1))
  (conv3): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
  (conv4): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
  (fc1): Linear(in_features=256, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=8, bias=True)
)
```

This time the training went quite fast, because of the reduced feature size of the model.

IV. Results

Model Evaluation and Validation

As stated I had to check if the model generalises well enough, because I feared overfitting. So I build a new testset, based on augmented samples of images from the end of the CelebA dataset.

The first model (Net) delivered a Test Loss: 1.184697 and a Test Accuracy: 56% (8867/15733), but as mentioned with blurred training, test and validation sets. Testing with the unseen test samples it reached Test Loss: 1.591798 and Test Accuracy: 40% (6380/15733). Alas, with an unseen, different augmented sampled testset it reached Test Loss: 1.717787 and Test Accuracy: 29% (200/677).

The second model (OtherNet) reached Test Loss: 0.347686 and Test Accuracy: 87% (13699/15733), again with blurred datasets. Unseen test samples gave a worse result, as expected: Test Loss: 0.891212 and Test Accuracy: 66% (10512/15733), but anyway better than the first model. Finally, on the new testset: Test Loss: 1.544028 and Test Accuracy: 47% (319/677). Also, I accounted for distinct classes on this testset:

```
Accuracy of blackmen : 40 %
Accuracy of blackwomen : 60 %
Accuracy of blondmen : 39 %
Accuracy of blondwomen : 59 %
```

```
Accuracy of caucasianmen : 57 %
Accuracy of caucasianwomen : 41 %
Accuracy of tannedmen : 21 %
Accuracy of tannedwomen : 29 %
```

Apparently, the model performs worse on the last 2 classes. This is expected as these classes are not so distinguishable with respect to the other classes.

Then, transfer learning on Resnet50: Test Loss: 0.645098 and Test Accuracy: 80% (12720/15733). Wow! But then again with the brand new testset: Test Loss: 1.215216 and Test Accuracy: 55% (378/677). As expected: worse, but still better than the previous models, however not so much better..

My fear had become true, this what you get when feeding images that are too much alike after extensive data augmentation. It should have taken more time to build a thoroughly balanced dataset, using more advanced techniques to augment data samples or tuning a GAN to do that.

Justification

In my quest to see if we can build a model recognising skin types and gender, based on an unbiased, balanced dataset, I missed out on a good input dataset like from the Gender shades project. At least we can account for some balancing, but the samples were too much alike after augmentation to really tell if the model generalises well enough.

So, no, I cannot beat the benchmark; the results are in fact not quite so bad, but in any case not to claim we have solved the problem.

V. Conclusion

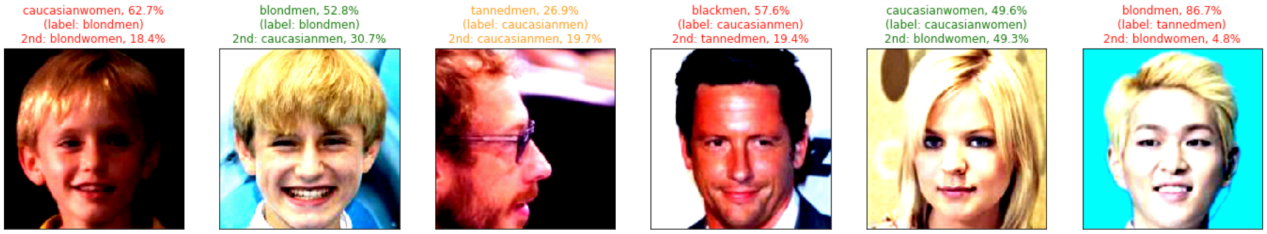
Free-Form Visualization

Given my own flawed expertise in labeling data samples, I wondered how the model performed percentage wise, so I plotted the images with probabilities, for the max and the next best percentage and that gave some interesting insights:

In these random samples taken from the unseen, newly augmented testset we see images with titles in red: clearly no good prediction, in green: good prediction and in orange: in between, which means that the prediction was incorrect, but the second highest probability did match the label. Especially when probabilities are close to each other we could say: either prediction would be relatively correct.

Let's analyse this from left to right:

1. A dark haired boy: I wrongly labeled this image as 'tannedmen' but in fact the model was correct in predicting 'caucasianmen'
2. Dark haired woman, dark image, low contrast: I labeled this as 'tannedwomen', the model predicted 'blackwomen' and 'tannedwomen' respectively, so here again we could argue about the outcome
3. Labeled 'tannedmen', prediction 'caucasianmen` would arguably right as well, besides the fact that 'tannedmen' is the next best probability
4. No special remarks
5. I put quite some 'Asian looking' samples in the 'tannedmen' class, this is why the model's prediction is correct in this case



In the samples above, we could argue again over correct labeling, but there is one aspect which is quite interesting in the last sample of this selection. The image is predicted with high probability (86,7%) as 'blondmen', I labeled it as 'tannedmen' because of the Asian look, but the next best is 'blondwomen'. 'Blond' is certainly a correct feature and I recalled I had some difficulty in labeling this image to the correct gender…

The point here is that although correct labeling is very important, the model is in fact capable of predicting multiple options which are closer to reality, in spite of my own ***biased opinion***. Considering the purpose of my project, this is a very important finding.

Reflection

Starting out with the dog breed classification project, I managed to train a CNN which at first sight did the job on an unbiased, balanced dataset of skin type x gender classes. Unfortunately, I had quite some data cleansing to do in order to get this balance, but in the process I could not help being biased myself in labeling images the correct way. I didn't have the time to traverse 200k samples so I relied on simple data augmentation techniques to build a decent dataset, however prone to overfitting as it turned out.

Trying different CNN architectures and transfer learning I learned to get a grip on model parameters and the impact it has on training time; next time I will use GPUs in orde rto speed things up, but this was also a good test for my new Macbook Pro….

Well, in the end we got some results and the main goal of this project was to produce an unbiased dataset or at least a method to produce one. As we learned from the dog breed classification project, predicting classes of this kind is a real challenge. Even for our 8 classes it turned out to be difficult. Also, it was a pity I could not use the Gender shades dataset, which accounted for better labeling to start with.

I could say that the final model meets my expectations, considering the input data samples and course grained labeling I have done myself. Not to say that this model is fit for production.

Nevertheless, this was a good exercise in finding ways to produce unbiased, balanced datasets, as a means to prevent the ugly becoming the bad…

Improvement

As mentioned, a better input dataset would certainly improve the capability of the model, being trained on well labeled samples with distinguishable features like skin type and other features present in the Gender shades project. Another option would be to take some samples from the CelebA dataset or another dataset with unbiased samples and try to augment them properly with a GAN and then to label these samples accordingly, but that's a lot of work, as I have experienced.

The key to unbiased datasets is to have enough and evenly distributed samples of the different classes, which should be reviewed amongst peers and maybe even beyond them, in order to prevent mixing in your own biased opinion, as I explained before. This would take a project setup preferably decentralised around the world in orde to get lots of samples from different places and different cultures, so more balanced opinions on labeling.

I haven't tried out other algorithms besides CNNs, it could well be possible that other models exist like Logistic regression, KNN or RandomForest for this particular classification problem, when we parametrise the features in a certain direction or perspective.