Term Project Report

Robert Bland
Michael McCarver

Executive Summary

For our term project, we decided to work on the Face Detection problem. The specifications of this problem were to write an end-to-end program that can recognize a person's face. We decided to limit the scope of our project to only detection and recognition. For a data source we used the recommended MIT-CBCL 10 class image set. We created some custom functions to augment the data set and to diversify our training set. For the 'detector' stage of the problem, we used the dlib library to find the coordinates of a bounding box surrounding the face, and then cropped the image to those coordinates. We also used DLIB to extract features from the images. For the classifier, we implemented a CNN with pytorch and trained it with our augmented training set. Our final model achieved 35% accuracy.

Introduction

The Face Detection problem is to write a program that can detect and recognize a particular person's face. The detection part of this problem was not all that difficult to start because the image data set we were provided only has faces in it. The tricky part is knowing what faces should be thrown out and what should be kept in, as well as what faces should be modified. Recognizing a particular face is the trickiest problem out of the two since it requires some sort of extraction of features. This is mainly because the raw images that were provided have more than just faces in them, so it could be trying to recognize these people based on a bit of shoulder that is given, but the test set has only faces in them and no shoulders or background. Given such information we know what our objects are, which the first objective is to be able to detect a person's face. For this we considered using some of our old code and trying to do Sobel or Harris Corner detector to grab the area where there exists the most key points. We realized, however, that this would be fairly difficult to make it consistent, so we found a good library, dlib, to assist us when it comes to grabbing the bounding box of a face. So, our next main objective was to grab the important facial features of every face that we scanned. We still considered using sobel and harris corner detectors, but dlib also has a great way of extracting features with 68 key points and we thought this would be enough to actually get decent or good results. This comes with it's own problems, however, so that will be discussed later. For our Neural Network, we decided to make this from scratch and created a naive solution first, where it just took in the raw image, as well as a more concrete solution that should perform better. We then created another Neural Network that takes in the cropped faces as well as key points as input to then give us a good result. This also came with more problems which also gets discussed later.

Technical Description

In order to diversify the training data we wrote algorithms to change and modify the original image data. The following is a description of those algorithms:

make_rotations: this function takes as a parameter a list of angles by which to rotate the image. For each angle, the positive and the negative of the angle is applied. Rotations are achieved with scipy.ndimage python module.

make_translations: this function has a hardcoded offset value (we decided on 10 pixels) by which images are transposed. For each image, we create 8 new images representing all combinations of shifting the image by the positive and negative offset in both the x and y direction. The translation algorithm was a custom implementation from a previous homework assignment.

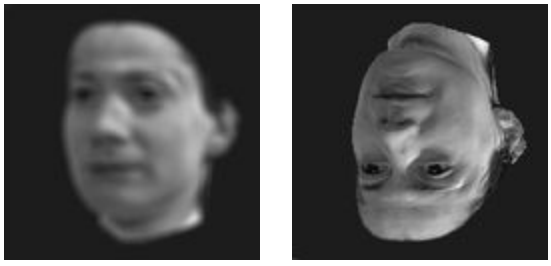
make_blurry: this function convolves an image with a filter of all 1s. As a parameter the function takes a filter size for the kernel, which determines how 'blurry' the image will turn out to be. In order to maintain dimensions, the image is padded before convolving. The convolution code is a custom implementation from a previous homework assignment.

make_mirrored: this function takes as a parameter a list of flip types to apply to the images. Flip types could be x-axis only, y-axis only, or both axis flipping. The cv2.flip() function is used to perform the flip.

Pickle system: in order to ease development we decided to preprocess variations of augmented data and to save them as binary files using python's pickle module instead of processing the data on the fly. For each type of augmentation, we created a data list containing image augmentations and a label list corresponding to the data set. Both lists were packaged in a dict and then pickled, following the convention of the CIFAR-10 dataset. This turned out to be helpful as some of our augmentations took a while to execute.

The following images are samples from the different types of augmentations:

Original Rotated Translated



Blurry Mirrored

For detection we used Dlib to grab the bounding boxes of faces and to detect and extract the important facial features. Architecture wise, this sifts through all of our files in a certain folder with images in them and then attempts to find the face in the image. If no face is found in the image then it excludes this image from future use and extraction, since this no

longer has the ability to give important information. We also store the bounding box information for later use. After dlib grabs those bounding boxes for face and for the exclusion list, we then move onto preprocessing.

When we preprocess the images the first thing we do is we crop the image based on the bounding box of their face and save that to a separate folder. We then load them in as grayscale with good reason. Before I learned that there can be some size restrictions with our approach, so to reduce the overall size and not go over the limit we use one channel instead of 3. Note, that in the naive implementation we use three channels. We chose the width and height to be 128 by 128 and it does not maintain the original images ratio of width and height. This means that some features will be distorted due to this or lose quality, so it is possible that this can lead to worse results since the test images are about 128 by 128 in size.

After loading them in as grayscale we then use dlib's predictor algorithm to determine where the important facial features are. Once these are all located we then construct a new image based on the shapes obtained. This function takes in each part that I sliced, since the points returned from dlib are always in the same order for the same part, and these are reconstructed using a anti-aliasing line function from skimage. This draw function returns the shape when it is done and we keep adding all of these important features to this blank image and we eventually get a nice outline of the faces. One thing we had to do because of errors was create a threshold of some of the outlines, which is why it is not a one-to-one recreation of the original shape, but it is very similar.

For classification, we have two different Neural Nets. The first Neural Net was a naive approach that takes in only the raw image that is resized to all be the same. This means that it takes in more than just the face as a factor including background. The Neural Net itself has 3 convolutional layers with 2 fully connected layers. It uses cross entropy and tries to classify the 10 possible faces that we have. The input for this was d x 64 x 64 x 3. The performance of this architecture speaks for itself since it has no real feature extraction.

The second Neural Network is the improved version that has the same exact structure, but is given different inputs. We kept the same structure because we wanted to make sure that this wasn't a possible factor in why the first Neural Net was doing so poorly. The input given to this one was d x 256 x 128 x 1 and also used cross entropy to give us 10 possible faces to classify. The input was cropped images and important facial features, hence why it is 256 and not 128 for the second part of our input.

Data set

The dataset we chose to use was the MIT-CBCL data set linked to on the Project Options slide of the powerpoint. The dataset contains images of 10 different faces to use for classifying. It comes with a test directory of 2000 images, a training-originals directory with 61 images, and a training-synthetic directory with 3240 images. The training-synthetic images are rendered from 3D head models of the 10 subjects. The training-originals images are high resolution and include frontal, half-profile, and profile views. The test set varies illumination, pose, and background. As mentioned above, we used various algorithms to augment and

extend our training set. The images from before were modified from the synthetic data set. The following images are examples of the same class taken from the training-originals and test directories:


Training-originals Test

Results

To test we used a subset of the tests directory totalling 88 images. In order to speed up development we preprocessed the test images by applying the detection cropping and then extracting features, then drew a new image from just the features. The following image is one such example:

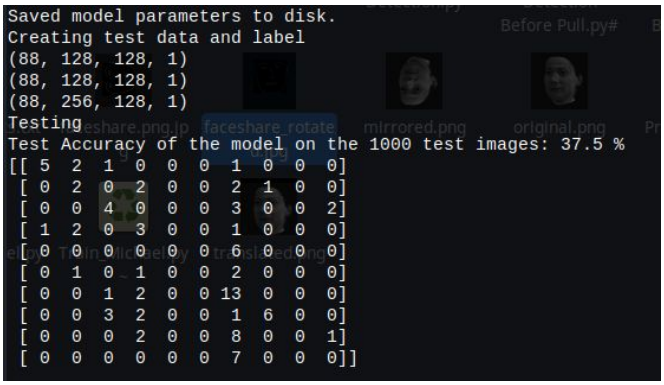
Feature Image

To obtain and measure results we decided to gauge it based on accuracy compared between the naive implementation versus the feature extraction implementation. Our metric was a confusion matrix, and the accuracy was a measure of how many correct faces the best possible training model could recognize, so with this taken into account, we will ignore any bad models we have created for the final comparison, but will also mention our early results of both implementations first. One important thing to note is that we used cross entropy as our loss, so our results might seem poor, but this is an attempt to create something that can recognize multiple faces and not just one.

For the first naive implementation, the results were very poor in the beginning and overfitting was a major problem. Initially, we did not use any standardization and varied all of our parameters and with this we managed to create models that test at around 10% with our testing images. However, this 10% was actually this guessing only one column and in reality we saw that this probably indicated overfitting of one face. So, with these initial results we managed to do some normalization and standardization of data before we pass it in and got a max result of about 20%. This model was not saved, unfortunately, but overall the max percentages of all the models that were created could at most be 20%. On average, however, overfitting was a massive problem in this domain and when it wasn't overfitted it was about 8%-14%. This

showed us that there is plenty of room for improvement and this performed about as well as I expected and I can assume that the 20% might have been due to pure luck.

The improved implementation suffered heavily from overfitting as well. By fine tuning parameters, however, we were able to consistently get results in the range of 30% to 37.5%. The learning rate seems to be the most significant parameter we tweaked, with a larger learning rate tending to perform better. We were keeping the number of epochs run constant across variations, though, so it could be that lower learning rates need more time to explore and find minima. The following is output from a run that produced a 37.5% model.



Confusion Matrix

Conclusions

In conclusion, the facial recognition problem is a very difficult problem. We decided that we would use cross-entropy and try to classify all 10 faces in the data set, but in retrospect we may have seen better performance if we had narrowed our scope to recognizing a single individual. One bottleneck we experienced was the amount of time to generate dataset augmentations. Our pickling approach generated binary files in excess of 1gb size; such models took about 1 hour to generate. Additionally, the bounding box generation and feature extraction took longer than expected. Running these processes on the full 2000 image test set took about 1 hour.

Most of the data augmentations seemed to be useful with the exception of image translation. This could be due to the fact that as part of the detection step we crop the image around the face before we feed the image into the network. This may counter whatever benefits would have been seen from moving the face around the image.

The feature extraction process seems to have contributed to an overall better model than a naive, raw image consumption. This may be because the network is given less data to reason over, but the data is of higher value, allowing more efficient learning.

All in all, the facial recognition problem was a fun challenge and we would enjoy spending more time working on it.