CS3244 Homework 3: Labeled Faces in the Wild

EigenFace

A0080485B Yong Yoong Jie John
A0121261Y Cheong Xuan Hao Filbert
A0121520A Quek Chang Rong, Sebastian

Introduction

We tried several different approaches including Gaussian Process (GP), Principal Component Analysis (PCA), Support Vector Machines (SVM), gradient boosting, Convolutional Neural Networks (CNN) and ensembling methods. Furthermore, we analysed the given dataset to look for ways to improve our classifier, and came up with 2 preprocessing/augmenting methods.

Ultimately, our classifier is able to predict the label of a new face image, given it belongs to one of the 7 classes, with an F1 mean score of around 0.95. This places us in the top 10 of the Kaggle competition.

Approaches

GP, PCA, gradient boosting and SVM

Initially, we intended to solve this problem by using Gaussian Process through the feature extraction Gaussian Process Latent Variable Model which Lu and Tang applied to their GaussianFace model [1]. They surpassed the benchmark for image recognition by humans on the Labeled Faces in the Wild unrestricted dataset category - 98.52% vs 97.53%. However, as we encountered difficulties with using the Python GPy library, we resorted to using Principal Component Analysis (PCA), for our dimensionality reduction which has been known to produce reasonable results [2].

We determined the number of components to keep for PCA by using the explained variance method, and chose the value which predicted 98% of the variance. We explored using the well known Xgboost library to classify the dataset which has been used to claim many top placements in Kaggle competitions. However, the results were disappointing as the accuracy was found to hover around 65% as compared to SVMs. We applied the same features used on the Xgboost decision tree model to SVM and found that we had made a substantial improvement to approximately 81%. The hyperparameters used in SVM was the radial-based function kernel which was optimised using the grid-search method. Although our hyperparameter tuning was not optimal, we found that even if we were to use the best hyperparameters on the SVM through bayesian optimisation, the results would not exceed 90% [3]. Hence we decided to experiment with CNNs.

CNNs

Having looked through several research papers [4, 5, 6, 7], we found that majority of the most accurate facial recognition algorithms utilise Artificial Neural Networks, with many using CNNs which are suitable for images. We consulted several online resources to find out more about CNNs [8, 9], including researching on how others have approach a similar problem of digit recognising in another Kaggle competition [10, 11].

We decided to use Keras which works on TensorFlow. By splitting the training data into 2 parts, 1 for training (80%) and the other for validation (20%), we were able to utilise the F1 score as a way to gauge the performance of a specific CNN model on the validation set. We initially built a simple CNN with 1 convolution and 1 max pooling layer, with an eventual fully connected layer. This baseline model performed with an F1 score of around 0.8567. This baseline model uses the ADAM gradient descent algorithm [12] and 40 epochs with a batch size of 50.

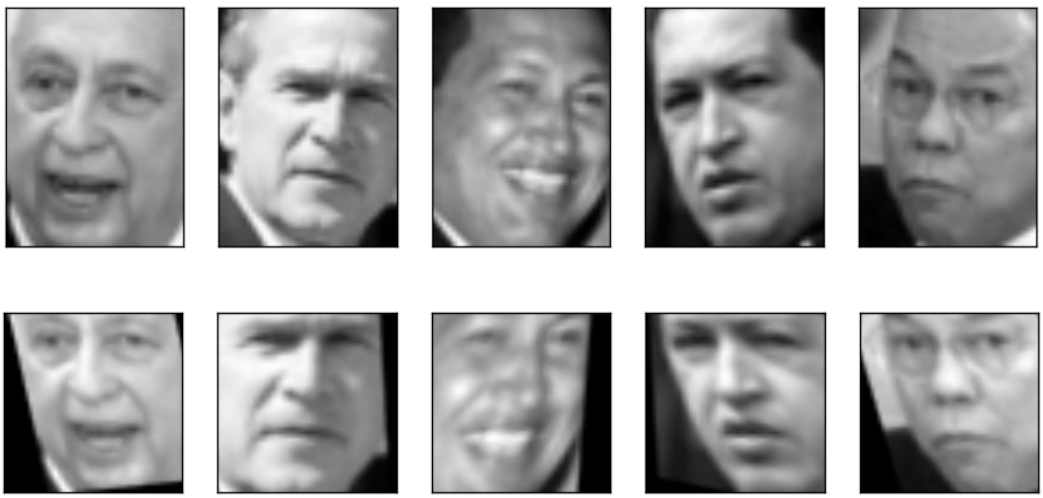
Several other models were also tested. We tweaked the number of layers, number of features in the convolutional layers and the size of these features (i.e the window size). Ultimately, we found that having many layers increased the training time without substantially increasing the score. We hypothesize that with our small dataset (<1000 examples), training a deep CNN is likely to overfit the training examples as the network is likely to find patterns very specific to the training examples.

The eventual model we used had 2 sets of convolution and max pooling layers, each with a different number of features. We included dropout rates of 0.25 for each of these sets to prevent overfitting [13] The model also includes a 256-dimension fully connected layer before the final 7-dimension fully connected layer. Based on the same set of training and validation examples as above, we obtain an F1 score of around 0.90.

Data Analysis and Preprocessing

To attempt to further improve our F1 score, we took a deeper look into the given dataset. By looking at the first 50 images in the training set, we found that there are many pictures where their faces are not aligned, in the sense that their eyes and lips are not always in the same location in the image. We use face landmark estimation [14] to find the face in the image based on a predefined set of landmarks, then using affine transformations, we obtain a more “normalised” set of images. We made use of OpenCV and dlib and adapted a version by OpenFace [15] to preprocess the images.

The following set of images show the transformation, the top row are the raw images given in the dataset and the bottom row are the derived transformed images. Notice that all the faces in the resultant images are approximately centered. Although some images looked to be distorted, the final results from the training on these examples show an improvement in score.



We split the preprocessed images into training and validation sets and trained using the same model parameters as above. We obtained an F1 score of around 0.94. However, this score was based purely on images where a face could be detected. There were several images where no face was detected.

This caveat meant that not only do test images need to be preprocessed, there needs to be a way to handle test images where no face was detected. It is not be feasible to simply use the model based on aligned faces to predict these. As such, we decided to construct an ensemble of 2 CNNs, 1 for the raw images and 1 for the aligned preprocessed images, and take the result with the higher probability score. Images with no face detected will therefore obtain results from the former model.

Data Augmentation

Given the size of the dataset, we also explored the idea of generating more images from the current dataset using affine transformations, similar to that of aligning the images in the previous section. We utilised Kera's in-built image generator library to perform shearing, zooming and/or rotation on the given X_train dataset. The transformations are performed at random and according to the settings we provided (which can be seen in the source code). This added 1000 additional images across all labels. Ultimately, after all the preprocessing, augmentation and training, we managed to obtain an F1 score of 0.95652, as reported by Kaggle. Our training data accuracy score is found to be 0.9962.

Work allocation

We split our 3-man work allocation based on model exploration. One group focused on implementing the convolutional neural networks along with the preprocessing and augmentation of images while the other explored models which could be used for multi-class face recognition.

Statement of Individual Work

We, A0080485B, A0121261Y, A0121520A, certify that we have followed the CS3244 Machine Learning class guidelines for homework assignments. In particular, we expressly vow that we have followed the Facebook rule in discussing with others in doing the assignment and did not take notes (digital or printed) from the discussions.

Included files

1. README.pdf - This readme, detailing the entire report
2. hw3-lfw-svm-xgboost.ipynb - Source code for Xgboost and SVM model
3. hw3-lfw-cnn.ipynb - Source code for the Convolutional Neural Network
4. hw3-lfw-preprocess.ipynb - Source code for the preprocessing/augmentation of image data
5. shape_predictor_68_face_landmarks.dat - Dependency of preprocessing

Requirements

The following are the requirements that do not come in the default Anaconda package:
OpenCV, dlib, Keras, Tensorflow, Xgboost

References:

1. Chaochao Lu, Xiaoou Tang Surpassing Human-Level Face Verification Performance on LFW with GaussianFace *arXiv:1404.3840*, 2014
2. Lindsay, February 26, 2002 http://faculty.iiit.ac.in/~mkrishna/PrincipalComponents.pdf
3. LFW: Results. (2009). Retrieved November 1, 2016, from http://vis-www.cs.umass.edu/lfw/results.html
4. Schroff, F., Kalenichenko, D., & Philbin, J. (2015). FaceNet: A unified embedding for face recognition and clustering. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. doi:10.1109/cvpr.2015.7298682
5. Taigman, Yaniv, Ming Yang, Marc'aurelio Ranzato, and Lior Wolf. "DeepFace: Closing the Gap to Human-Level Performance in Face Verification." *2014 IEEE Conference on Computer Vision and Pattern Recognition* (2014): n. pag. Web.
6. DeepID3: Face Recognition with Very Deep Neural Networks
7. Naive-Deep Face Recognition: Touching the Limit of LFW Benchmark or not?
8. How do Convolutional Neural Networks work? - Brandon Rohrer
9. CS231n Convolutional Neural Networks for Visual Recognition
10. Handwritten Digit Recognition using Convolutional Neural Networks in Python with Keras
11. Digit Recognizer - Kaggle
12. [1412.6980] Adam: A Method for Stochastic Optimization - arXiv.org
13. Dropout: A Simple Way to Prevent Neural Networks from Overfitting
14. Kazemi, Vahid, and Josephine Sullivan. "One Millisecond Face Alignment with an Ensemble of Regression Trees." *2014 IEEE Conference on Computer Vision and Pattern Recognition* (2014): n. pag. Web.
15. OpenFace