Pattern Recognition & Machine Learning - CSL 2050 Project Report

Souvik Maji[1] Veeraraju Elluru[1] Pranjal Tiwari[1] Ram Suthar[1] Parth Mina[1]

[1]Indian Institute of Technology, Jodhpur
{b22cs089, b22cs080, b22cs095, b22ee081, b22ee074}@iitj.ac.in

Abstract

This report summarizes the workflow of our project. Our project was established to comprehensively understand the applications of traditional Machine Learning techniques to the given dataset - Labelled Faces in the Wild (lfw-people). The main goal of our project was to identify a face image by classifying the above dataset into one of the K classes - a Supervised Classification Task. Features extracted from Convolutional Neural Networks (hereafter, CNN), Local Binary Pattern (hereafter, LBP), and Histogram of oriented gradients(hereafter, HoG) were used for the classification tasks. We utilize methods such as k nearest neighbors (hereafter, kNN), Logistic Regression, Multi-Layer Perceptron (hereafter, MLP), Naive Bayes, Support Vector Machines (hereafter, SVMs), and tree-based Classifiers like Decision Tree, Random Forest, AdaBoost Classifier etc. We further demonstrate and report our thorough analysis to achieve the best performance across all classes of people. The entire code, analysis, demo code, web demo, a video can be found here - GitHub

Keywords: machine learning, deep learning, features, classifiers, nearest neighbors, Logistic Regression, Ensemble Classifiers, Support Vector Machines, Naive Bayes, Gaussian, Classification Report

Contents

1 Acknowledgements

A big thank you and kudos to the creators of the dataset and relevant research - Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments, created by Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. . University of Massachusetts, Amherst, Technical Report 07-49, October, 2007. Specific thanks for the deep-funneled version of the image data: Gary B. Huang, Marwan Mattar, Honglak Lee, and Erik Learned-Miller. Learning to Align from Scratch. Advances in Neural Information Processing Systems (NeurIPS), 2012. Banner photo by John Bakator on Unsplash. We would also like to thank the course instructors and teaching assistants, without whom our project would be incomplete.

2 Introduction

Our goal is to comprehensively understand the applications of traditional Machine Learning techniques to the given dataset - Labeled Faces in the Wild (lfw-people). The main task at hand is the identification of a face image by classifying the above dataset into one of the K classes - a Supervised Classification Task. For these tasks, we need features, and these were extracted from CNN-based, LBP-based, and HoG-based architecture. We then go on to utilize methods such as kNN, Logistic Regression, Multi-Layer Perceptron, Naive Bayes, Support Vector Machines, and 4 tree-based Classifiers for the same. Next, we demonstrate and report our thorough analyses captured in pursuit of achieving the best performance across all classes of people.

3 Approaches Tried

Before we delve into the exact approaches carried out, we first introduce our Machine Learning pipeline for the task at hand -

- Dataset Analysis
- Data Preprocessing
- Feature Extraction
- Classification
- Results Analysis

3.1 Dataset Analysis

3.1.1 Image information:

Each image is available as "lfw/name/name_xxxx.jpg" where "xxxx" is the image number padded to four characters with leading zeroes. For example, the 10th George_W_Bush image can be found as "lfw/George_W_Bush/George_W_Bush_0010.jpg"

3.1.2 Image dimensions:

Each image is a 250x250 jpg, detected and centered using the OpenCV implementation of Viola-Jones face detector. The cropping region returned by the detector was then automatically enlarged by a factor of 2.2 in each dimension to capture more of the head and then scaled to a uniform size.

3.2 Data Preprocessing

Our pre-processing methods are run in every notebook. The pipeline we used is as follows -

- Thorough understanding of our data.
- Running all boiler-plate code, for feature extraction.
- Initializing X and y arrays.
- We then process the images of people if there are at least 70 training samples for that particular class - this leaves us with a **7-way classification problem**.

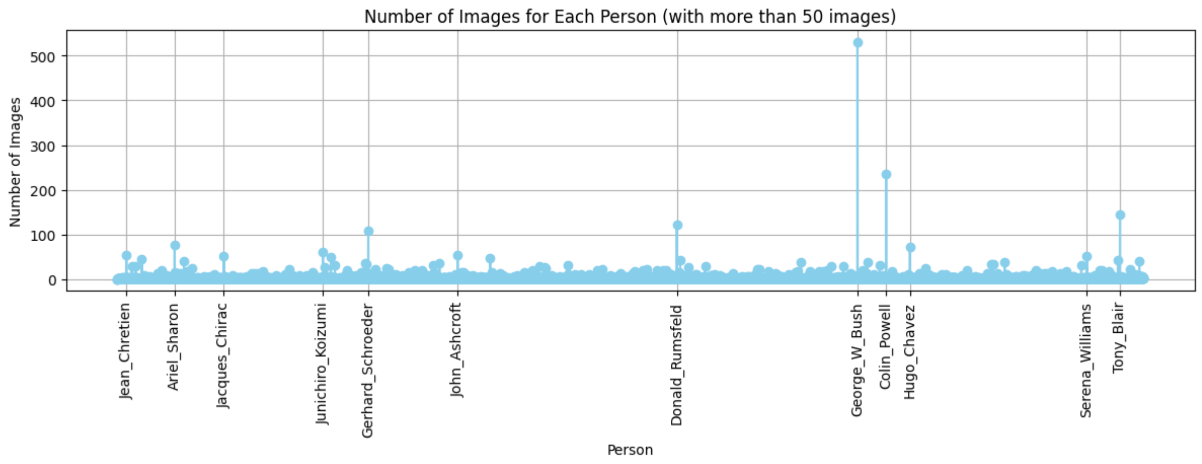

Figure 1: Histogram depicting the number of images for each person, with special tickmarks for classes with more than 50 samples.

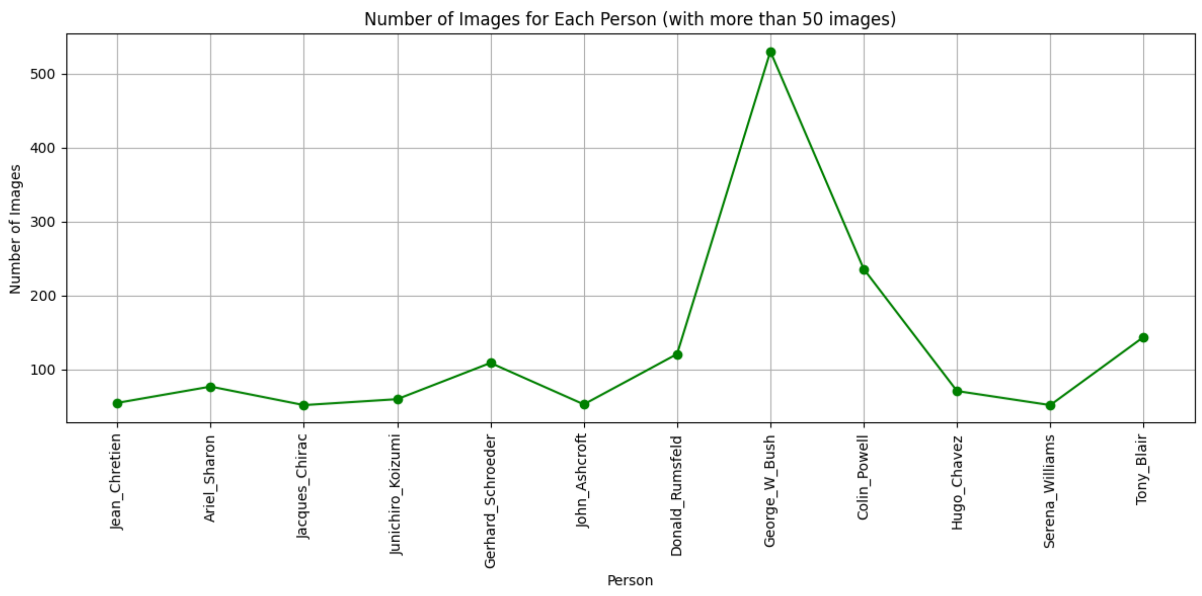

Figure 2: Number of images for classes with greater than 50 samples

- We then read the image using *skimage.io.imread* method.

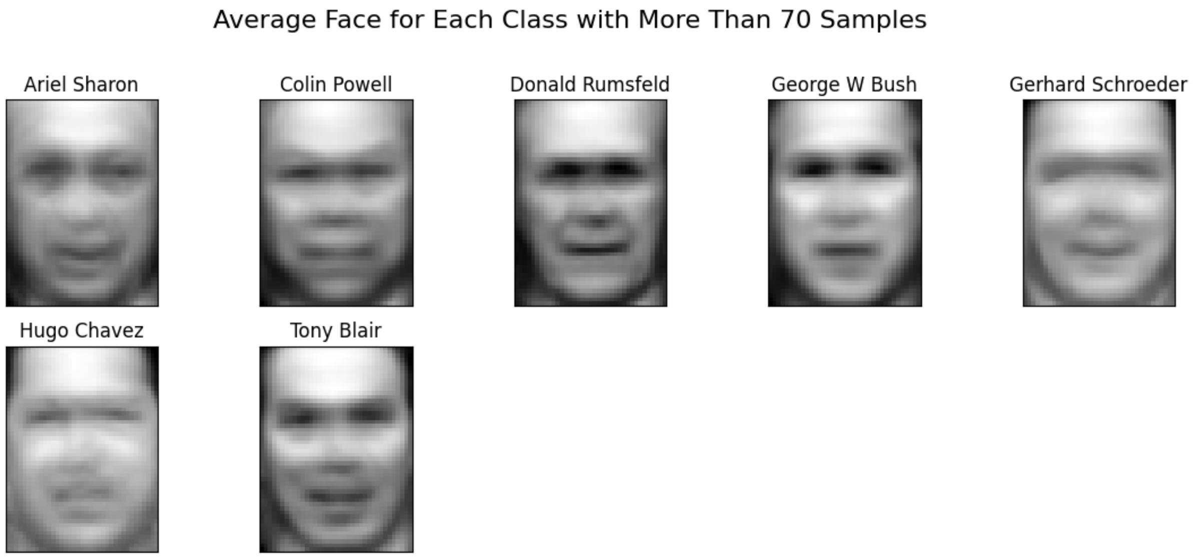

Figure 3: Mean faces for the final 7 classes, with more than 70 samples

- As we are experimenting with all possible feature combinations, one can identify which features are being used based on the name of the notebook. Ex: lbp+CNN+SVM.ipynb notebook uses the concatenated features of LBP and CNN, for the SVM classifier.
- We then perform classification according to the suffix of the .ipynb file name, with suitable hyper-parameter tuning.

3.3 Feature Extraction

For the feature extraction part, we have used the following three feature extraction techniques -

3.3.1 Local Binary Patterns(LBP)

- Brief description:
  - Local Binary Pattern (LBP) is a *texture descriptor* used in image processing and computer vision. It characterizes the texture of an image by **comparing each pixel with** its **neighboring pixels**. Specifically, for each pixel in an image, LBP encodes whether its neighboring pixels have intensities higher or lower than the central pixel, resulting in a binary pattern. These binary patterns are then used to represent the texture of the image, capturing information about local variations in intensity.
  - **LBP is robust to changes** in illumination and is commonly used in tasks such as face recognition, texture classification, and object detection.
- Advantages:
  - **Robustness to Illumination** Changes: LBP is robust to changes in illumination, making it suitable for applications where lighting conditions vary.
  - **Computational Efficiency**: *LBP involves simple pixel comparisons*, making it computationally efficient and suitable for real-time applications.
  - **Texture Discrimination**: LBP captures local texture information effectively, making it suitable for tasks such as texture classification and segmentation.
  - **Compact Representation**: The binary nature of LBP patterns allows for compact representation of texture information, which is beneficial for storage and processing.
- Disadvantages:
  - **Limited Spatial Information**: LBP only considers the local neighborhood of each pixel, which may result in limited spatial information capture. Sensitivity to Noise: LBP may be sensitive to noise, particularly in images with high levels of noise or artifacts.
  - **Parameter Sensitivity**: The performance of LBP can be sensitive to parameter settings, such as the radius of the neighborhood and the number of sampling points, requiring careful tuning.
  - **Lack of Rotation Invariance**: Standard LBP does not inherently provide rotation invariance, meaning that it may not perform well when faced with rotated or transformed textures.

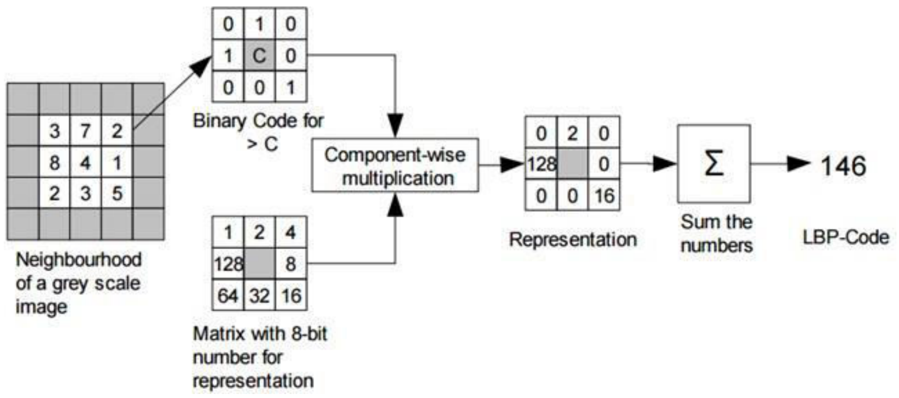

Figure 4: Feature extraction performed by Local Binary Pattern

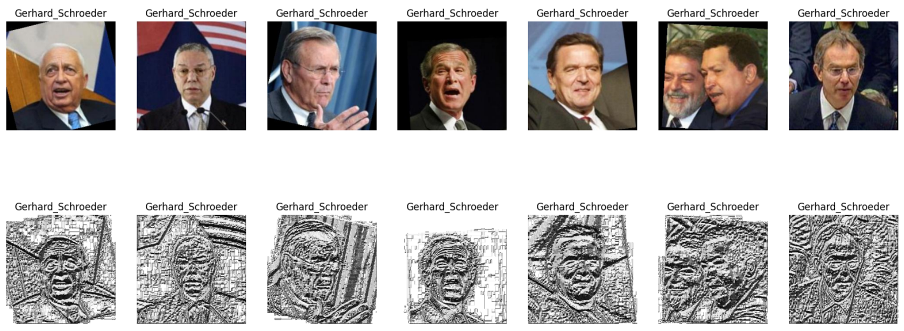

Figure 5: LBP Images for all the 7 classes.

3.3.2 Histogram of Oriented Gradients (HoG)

- Brief description
  - HoG extracts local texture and shape information from image patches by computing gradient orientations and constructing histograms. These histograms are normalized within local blocks, resulting in feature vectors that represent the distribution of gradient orientations.
  - HoG descriptors are commonly used in object detection and recognition tasks for their ability to capture distinctive local features.
- Advantages
  - **Robust to illumination** changes: HoG descriptors are relatively insensitive to variations in lighting conditions, making them suitable for object detection in different environments.
  - **Distinctive local texture** and **shape features**: HoG captures detailed information about the texture and shape of objects, allowing for precise discrimination between different classes.
  - **Partial translation invariance**: HoG features can detect objects even if they are slightly shifted within the image, enhancing robustness to object positioning.
  - **Wide applicability** in object detection and recognition: HoG descriptors have been successfully applied in various tasks such as pedestrian detection, face recognition, and object tracking, showcasing their versatility and effectiveness.
- Disadvantages
  - **Sensitivity to noise**: HoG descriptors may be affected by noise in the image, potentially reducing their effectiveness in noisy environments
  - **Computational complexity**: Computing HoG features involves intensive calculations, particularly for large images or real-time applications, which can be computationally demanding.
  - **Limited global context**: HoG focuses on local texture and shape information, which may limit its ability to capture global context and spatial relationships between objects.
  - **Limited global context**: HoG focuses on local texture and shape information, which may limit its ability to capture global context and spatial relationships between objects.

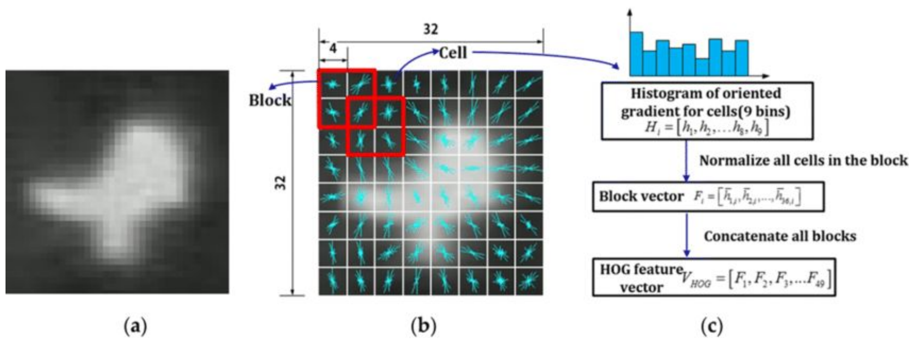
32
4
Cell
Block
32
Histogram of oriented gradient for cells(9 bins)
$H_i = [h_1, h_2, \ldots h_8, h_9]$
Normalize all cells in the block
Block vector $F_i = [\bar{h}_{1,i}, \bar{h}_{2,i}, \ldots, \bar{h}_{36,i}]$
Concatenate all blocks
HOG feature vector $V_{HOG} = [F_1, F_2, F_3, \ldots F_{49}]$
(a)
(b)
(c)

Figure 6: Feature Extraction using Histogram of Oriented Gradients

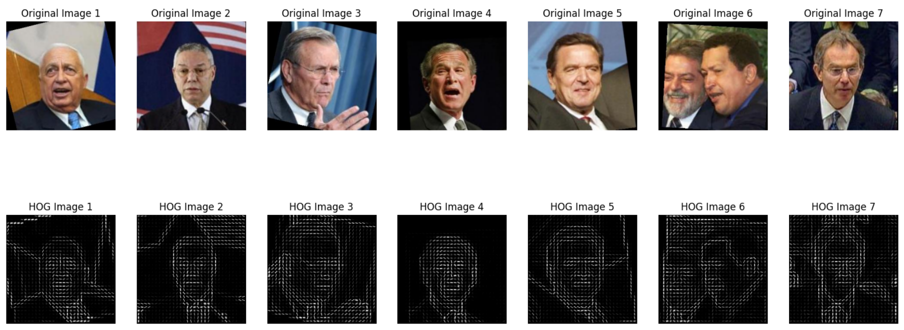

Figure 7: HoG Images for all the 7 classes.

3.3.3 Convolution Neural Network(CNN):

- Brief description
  - Convolutional Neural Networks (CNNs) are deep learning models specifically designed for feature extraction in computer vision tasks. They consist of multiple layers, including convolutional and pooling layers, which learn to extract hierarchical features from input images. Convolutional layers apply learnable filters to the input image, detecting patterns and features at different spatial locations. Pooling layers downsample the feature maps, preserving important information while reducing spatial dimensions.
  - Through the deep architecture of stacked layers, CNNs learn increasingly abstract and high-level features, enabling them to capture complex visual patterns. Trained using backpropagation, CNNs adjust their parameters to minimize a predefined loss function, allowing them to effectively extract features and achieve state-of-the-art performance in tasks such as image classification, object detection, and image segmentation.
- Advantages
  - **Automatic Feature Learning**: CNNs learn features directly from data, eliminating manual feature engineering.
  - **Translation Invariance**: They can detect patterns regardless of their position in an image.
  - **Efficient Parameter Sharing**: CNNs share parameters, reducing the number needed for training.
  - **Local Connectivity**: They focus on local features while preserving spatial information.
  - **State-of-the-Art Performance**: CNNs achieve top results across various computer vision tasks.
- Disadvantages
  - **Complexity**: CNN architectures can be complex and require substantial computational resources for training and inference.

– **Overfitting**: CNNs are prone to overfitting, especially when trained on small datasets or when the model architecture is too complex.
– **Interpretability**: Understanding how CNNs arrive at their decisions can be challenging due to their black-box nature, making them less interpretable compared to simpler models.

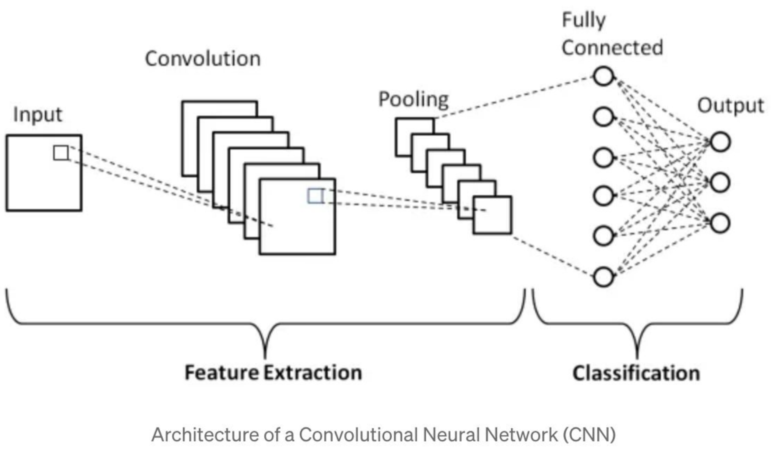

Figure 8: Feature Extraction using a general CNN architecture

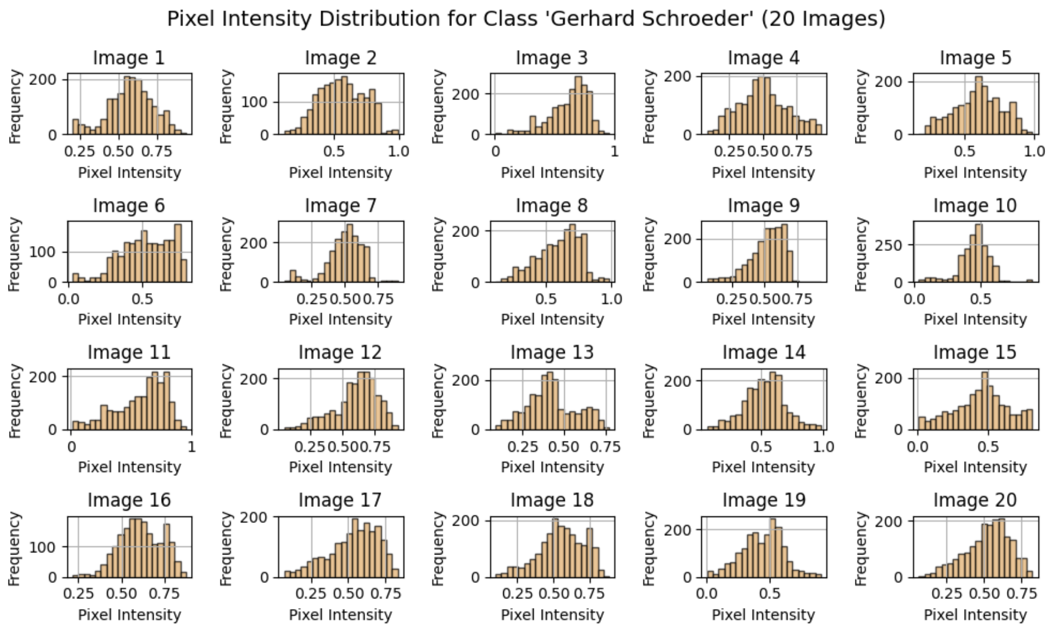

Figure 9: Pixel intensity distribution for class 'Gerhard Schroeder'

We will be implementing all possible permutations and combinations of the above three feature extraction techniques and then compare the results using different classifiers (more on the experimental setting in section 4)

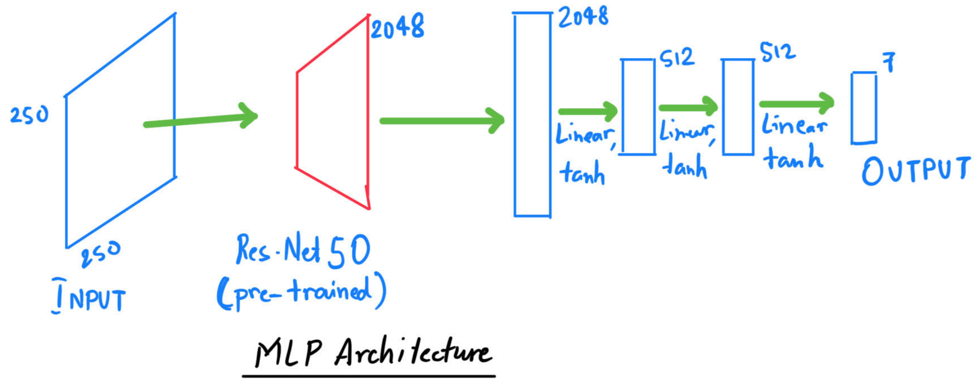

Figure 10: Our MLP architecture

3.4 Classification:

We have utilized a comprehensive set of 9 different classifiers. These are -

- k- nearest neighbors - We used the standard Sci-Kit Learn's nearest neighbors classifier.
- Logistic Regression - We used the standard Sci-Kit Learn's Logistic Regression classifier and also experimented with the lasso version.
- Multilayer Perceptron - Our architecture was built to capture the best performance. We used 2 sets of linear layers with 512 neurons each, and a *tanh* activation as our latent space. Note that we trained the model using the PyTorch framework. (Fig 10 summarizes the architecture)
- Naive Bayes - We used the standard Sci-Kit Learn's Gaussian Naive Bayes classifier.
- Support Vector Machines - We used the standard Sci-Kit Learn's SVC model, with extensive hyperparameter tuning. For this model, we also went one step ahead and tried to implement our own SVC model (based on CNN features only), using the *libsvm* wrapper.
- Ensemble Classifiers - Decision Trees, Random Forest, XgBoost Classifier, and AdaBoost Classifier - each with extensive hyperparameter tuning, to prevent overfitting as much as possible.

4 Experiments and Results

4.1 Experimental Setting

- Every .ipynb notebook in our project starts off by importing dependencies and extracting the dataset. Then we clubbed certain sets of features which is per the name of every notebook. Note that, as a result of the above concatenation, for every classifier, we have 7 feature sets. Now, we either fit the classifiers directly or the fitting was done after performing Principal Component Analysis (PCA)/ Linear Discriminant Analysis (LDA) on the features, to reduce the dimensionality (to capture 95% of the variance). Next, we tuned the hyperparameters, using grid-search or hit-and-trial, if applicable.
- Note that the tree-based classifiers have been used only with single feature sets - as these models overfit easily.
- MLP architecture has already been illustrated (Fig 10).

4.2 Results

In this section, we illustrate the various results obtained after the experimentations and hyperparameter tuning wherever possible.

4.2.1 Table of Accuracies

Every cell in the following table shows the best accuracy on the test data, for a particular classifier (except the tree-based ones) and a particular feature set. The *best* was chosen from the three possibilities - direct fitting and testing, fitting of the model post PCA, or fitting of the model post LDA.

	kNN	LR	MLP	NB[1]	SVM	DT	XgB[2]	AdaB[3]	RF[4]
LBP	0.40	0.51	0.43	0.40	0.46	0.41	0.47	0.46	0.47
HoG	0.83	**0.95**	0.86	0.89	**0.90**	**0.78**	**0.89**	**0.74**	**0.82**
CNN	0.81	0.85	0.83	0.80	0.81	0.60	0.76	0.62	0.73
CNN+LBP	0.79	0.78	0.45	0.77	0.81	-	-	-	-
CNN+HoG	0.84	0.93	**0.89**	0.89	0.86	-	-	-	-
LBP+HoG	0.82	0.87	0.74	0.89	0.78	-	-	-	-
CNN+HoG+LBP	**0.86**	0.90	0.75	**0.90**	0.79	-	-	-	-
Average	0.76	**0.83**	0.71	0.79	0.77	0.60	0.71	0.60	0.67
Top 3 Average	0.84	**0.93**	0.86	0.89	0.86	0.60	0.71	0.60	0.67

Table 1: Best test accuracies for all Classifiers (in boldface).
[1]Naive Bayes, [2]XgBoost Random Forest, [3]AdaBoost Random Forest, [4]Random Forest

	kNN	LR	MLP	NB[1]	SVM	DT	XgB[2]	AdaB[3]	RF[4]
Standard	0.64	**0.85**	**0.83**	0.34	0.79	0.48	**0.76**	**0.62**	0.55
with PCA	0.60	0.41	-	0.46	0.72	0.36	0.67	0.59	0.48
with LDA	**0.81**	0.79	-	**0.80**	**0.81**	**0.60**	0.75	0.61	**0.74**

Table 2: Test accuracies with and without PCA, LDA (for CNN-based features only.).
[1]Naive Bayes, [2]XgBoost Random Forest, [3]AdaBoost Random Forest, [4]Random Forest

- From Table 1, it is clear that CNN and HoG features capture most amount of information required and gives very good results for all the models. Sometimes, concatenating this with LBP features may help. Further, we can notice that LBP alone can not extract features that are very useful, hence when models are fitted with these features alone, they tend to perform poorly.
- Further, we can see that there is not a significant difference in the accuracies of models, when using CNN + HoG or when using all features together, hence we can drop the LBP features to save runtime and memory.
- From Table 2, we can see that using LDA for dimensionality reduction makes a very significant difference in the performance of the models, especially Naive Bayes, Decision Tree, and Random Forest. On the contrary, PCA only deteriorated the performance (except for the case of Naive Bayes Classifier.)
- It is easy to observe that the boldface values in Table 2 are the entries for row labeled *CNN* in Table 1.

4.2.2 Model-wise Analysis

Now let us look at some specific and intriguing results that we have collected.

- kNN classifier - Ranks 4th when we consider the top 3 average.
- Logistic Regression Classifier - This model is the clear winner. Our **Lasso** Logistic Regression model, that is, the standard logistic regression model but with a slightly different set of hyperparameters - *l1* penalty and a *liblinear* solver has provided us the best results, with very minimal overfitting. The simplicity in the model's loss function has indeed helped the model learn the best.
- MLP Classifier - Ranks 3rd when we consider top 3 average. For the MLP architecture used, we found that by creating appropriate data loaders, we were able to train the data faster and simultaneously monitor the epoch vs loss. This helped us to attain testing accuracies up to 89%, with very little overfitting (training accuracy was also around 98%)

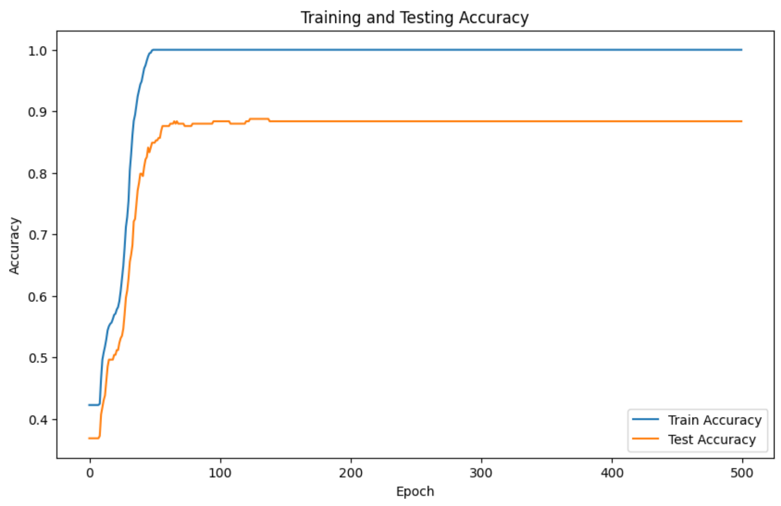

Figure 11: Training, Testing accuracy vs Epoch Graph

- SVM Classifier - Ranks 3rd when we consider top 3 average. For this model, we went one step ahead and tried to implement our own SVC model (based on CNN features only), using the *libsvm* wrapper. However, we noticed that there was not much of a difference in the model's performance.
- Tree-based Classifiers - As it is well-known in Machine Learning theory, tree-based classifiers tend to overfit the data. This was the case here too. Sometimes, we could see differences in training and testing accuracies as high as 40 to 60%, like in the cases of AdaBoost Classifier (based on CNN features) - 100% train vs 62% test, XgBoost Classifier (based on LBP features) - 100% train vs 47% test etc. The interesting part here is that applying Linear Discriminant Analysis reduces the difference in the accuracies (in most of the cases). Hence, reducing the number of features has in this case turned out to be fruitful in improving the model's generalization capabilities.

4.2.3 Run-time analysis

- The extraction of features was fastest for CNN, followed by HoG and LBP (almost the same).
- The run-time for those models which performed classification based on CNN features only, was the least. This was followed by the cases when HoG and LBP features were also concatenated. The models that performed classification based on these features too, took a very long time to execute, owing to the large number of features in HoG (around 70000) and LBP.
- Our Web Demo utilizes the Lasso Logistic Regression model which can classify a given single image in less than $1s$, with very good accuracy.

5 Summary

After extensive experimentation of numerous features, data pre-processing techniques, classifiers, and hyperparameter tuning, we can see that every model has their pros and cons when it comes to the classification of images from the LFW People dataset. Even though it appears that the Lasso Logistic Regression model has the best testing accuracy, we can not ignore the Multi-Layer Perceptron based classifier too. The latter generalizes/learns extremely well, but at the same time, it's a black box and has very less interpretability. Even tree-based methods, with appropriate pruning, boosting, and bagging methodologies can perform better than MLPs.
For the purpose of this project, we will stick to the Lasso Logistic Regression as our go-to model. All the experimentations, analyses, notebooks, references, and contributions have been thoroughly documented.

References

- Lasso Logistic Regression

- Medium Article on Gaussian Naive Bayes
- Medium Article on kNN
- Medium Article on Multi-layer Perceptron
- Medium Article on Ensemble Learning based Classifiers
- All data and metadata were originally found here. Please visit the site for other data versions including original, non-aligned data as well as more information on errata and training/testing model resources.
- Brief Description of dataset - Kaggle

A Contribution of each member

1. Souvik Maji: Implemented the pre-processing of the dataset. Implemented the code for tree-based algorithms, MLP, libSVM, and lasso Logistic Regression. Implemented the demo code.
2. Veeraraju Elluru: Implemented the code for Support Vector machines, MLP, and Tree-based models. Report creation (major). Gave the video voice-over.
3. Pranjal Tiwari: Implemented the code for Logistic Regression models and Tree-based models. Report and project presentation creation (minor).
4. Ram Suthar: Implemented the code for Logistic Regression and k nearest neighbors classifiers. Creation and deployment of the project page and web demo.
5. Parth Mina: Implemented the Naive Bayes classifier. Implemented the pre-processing of the dataset. Created the project presentation.