Pedestrian detection using principal component analysis

Quantitative Engineering Analysis I Final Project

Alyssa Aranda and Jacob Keller

Olin College of Engineering

I. Abstract

We developed an algorithm to determine whether a person is present in a given image. Our algorithm utilizes principal component analysis (PCA) to represent image data in fewer dimensions, increasing computational efficiency. Pedestrian detection algorithms are important to the development of vehicle safety features and driver assistance technology. Our results suggest that PCA can effectively reduce the size of visual data while preserving enough information to classify pedestrians with reasonable accuracy.

II. Limitations

Our algorithm is trained on a relatively small (400) subset of images from the Caltech pedestrian detection dataset [1]. All images from this dataset were taken during the daytime in an urban environment in the USA. The accuracy of our algorithm would likely be reduced in situations that don't match our training conditions, including images captured during nighttime, poor weather, or captured from different regions.

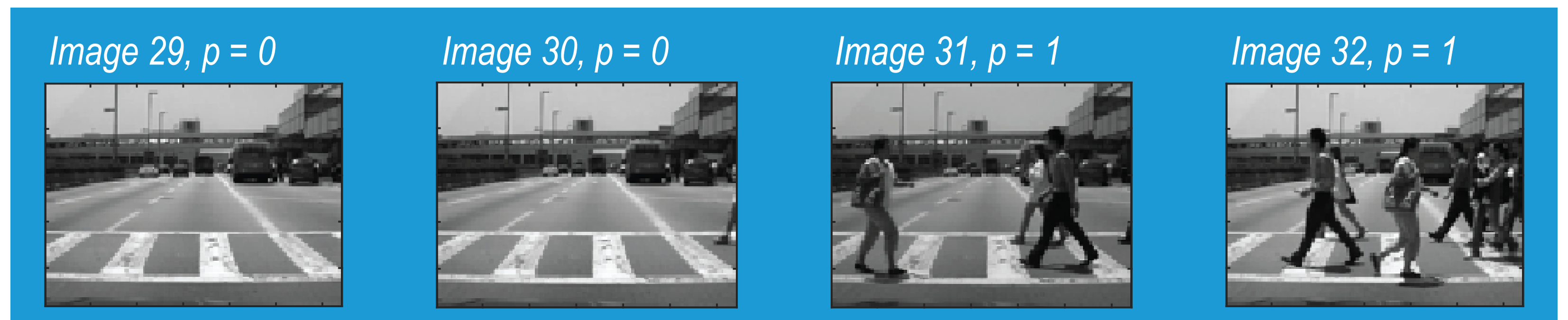

Figure 1: Images 29-32 from the training set. Images 29 and 30 are classified as not containing a pedestrian (p = 0), while images 31 and 32 are classified as containg a pedestrian (p = 1).

III. Prepare data

We simplified the (480 x 640 pixel) images from Caltech's dataset to grayscale and reduced the size by a factor of five to create an (96 x 128 x 598) array containing all image data. We designated the first 400 images as training data and the remaining 198 images as testing data. Four images from the training set are shown in Figure 1.

We used the *reshape()* function to convert the training data from a 3 dimensional array to a 2 dimensional matrix (size 400 x 12288) where each row represents an image and each column represents a specific pixel. We then used the *mean()* function to determine the average image from our training data, displayed in Figure 2. We normalized the training and testing data, then use the *cov()* function, which performs the following operation to find the covariance matrix:

$$covariance_{x,y} = \frac{\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_i)}{(N-1)}$$

Where mu (μ) is the mean training image and N is the number of training images.

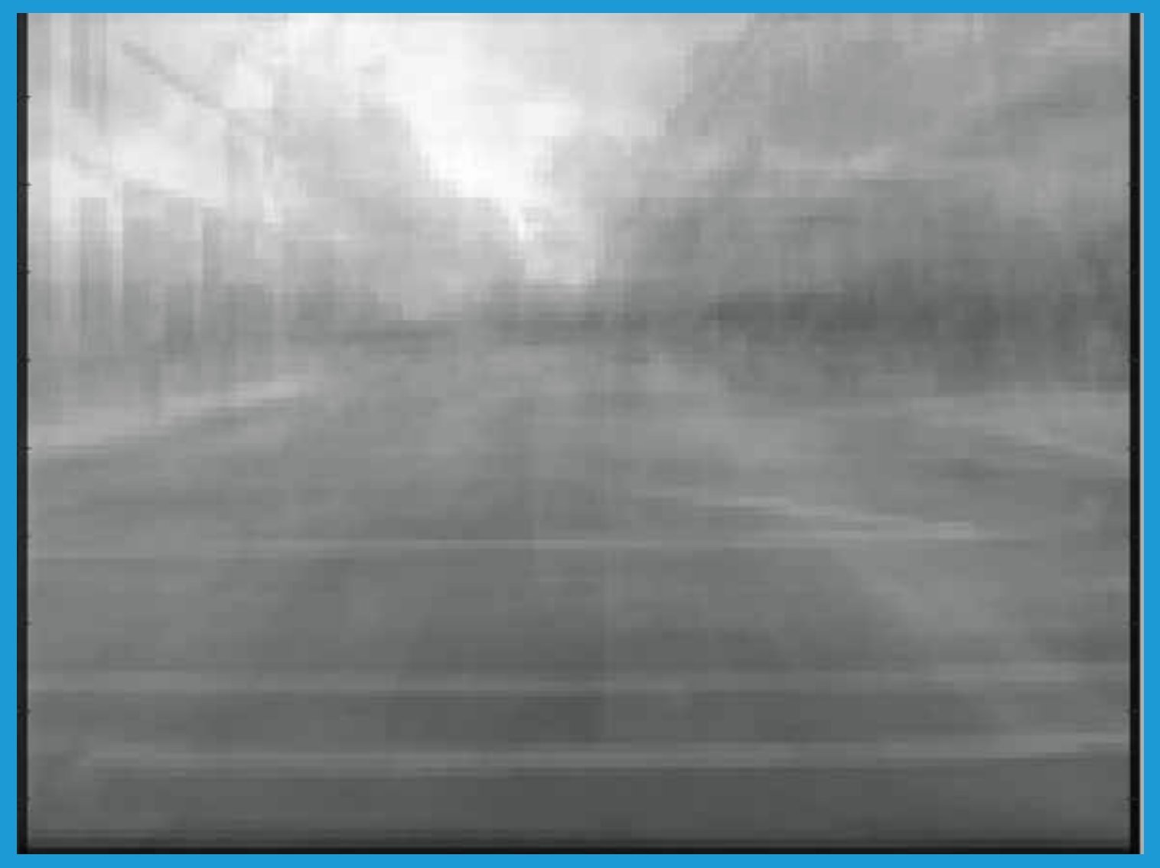

Figure 2: Mean training image. Several common features from images are captured here, including faint crosswalk markings and buildings on the side of the road.

IV. Determine principal components

We apply the function: *[V,D] = eigs(A,k)* where k = number of principal components. This returns a matrix V of the eigenvectors and corresponding diagonal matrix D of the eigenvalues. The eigenvectors in matrix V are the principal components, the first four of which can be observed in Figure 4. The eigenvalues of the first 20 components are plotted in Figure 5.

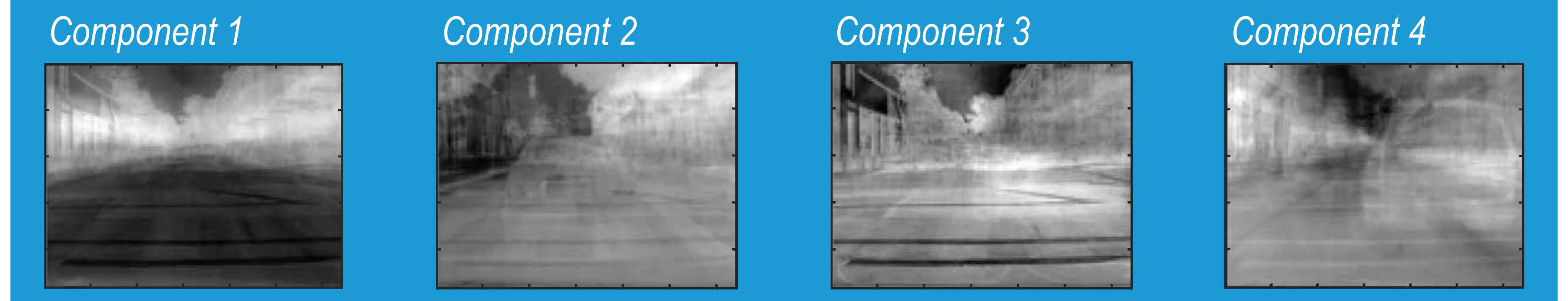

Figure 4: First four (out of 20) principal components displayed as images. Different components emphasize different elements of an image, such as the road or a crosswalk.

V. Project images into eigenspace

To project the images into the eigenspace we simply multiplied the centered data by each principal component. We did this for both the training data and the testing data.

*training_data * V*
*testing_data * V*

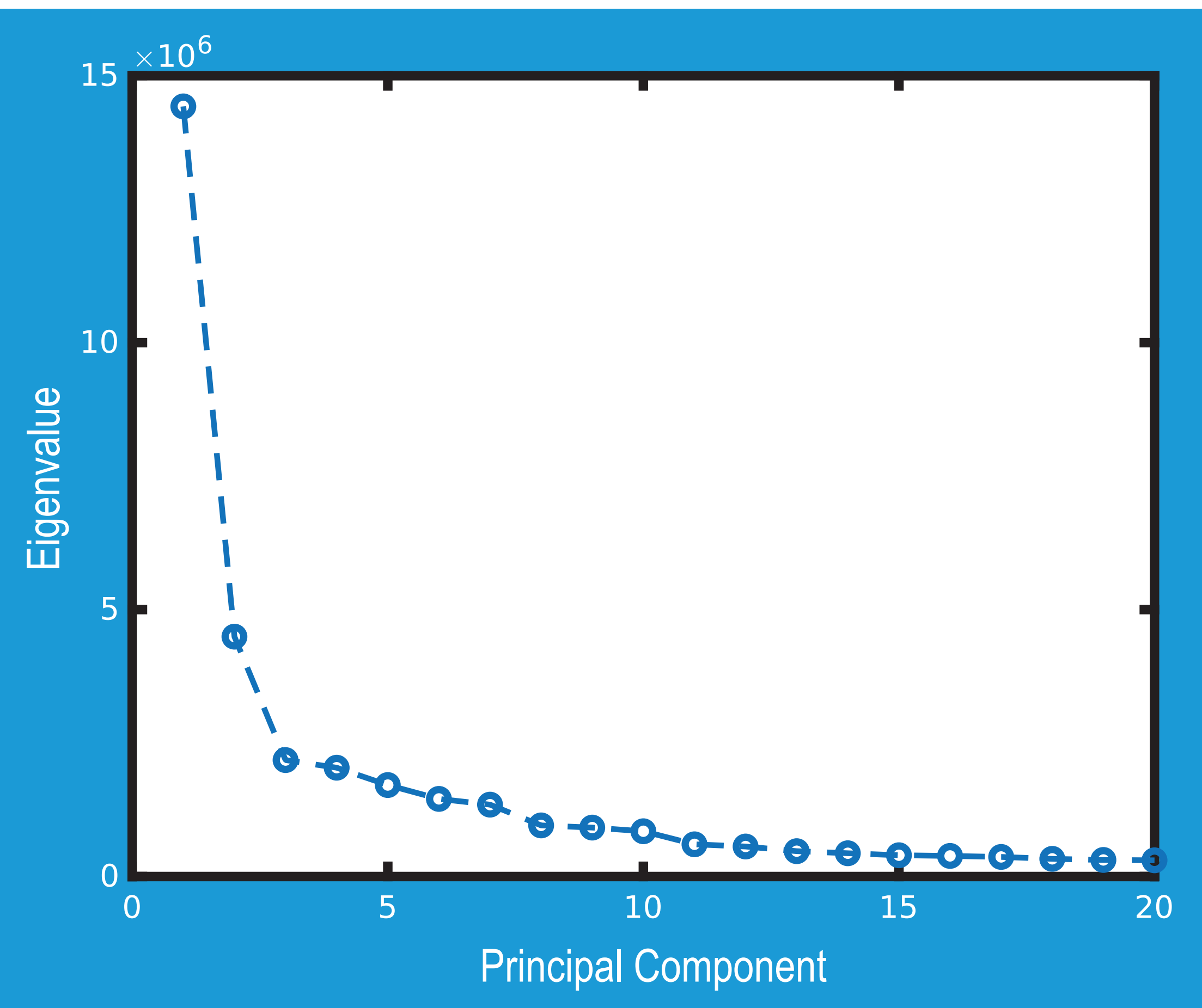

Figure 5: Eigenvalues of first 20 principal components. Graph follows an exponential decay pattern.

VI. Classify test images

For each testing image, we calculate the distance to all training images in the eigenspace. From there we found the closest training image by minimizing the distance using the *min()* function. If the closest training image contains a pedestrian then we predict that the test image does as well. Likewise, if the closest training image does not contain a pedestrian, we predict the testing image does not either.

VII. Calculate acccuracy

To calculate our accuracy we compared our predictions of the testing images with the key to create a vector where the value is 1 if our prediction was correct and 0 if our prediction was wrong. Calculating the mean of this vector gives us an accuracy percentage of around 78%. We found that most errors in our algorithm were from false positives.

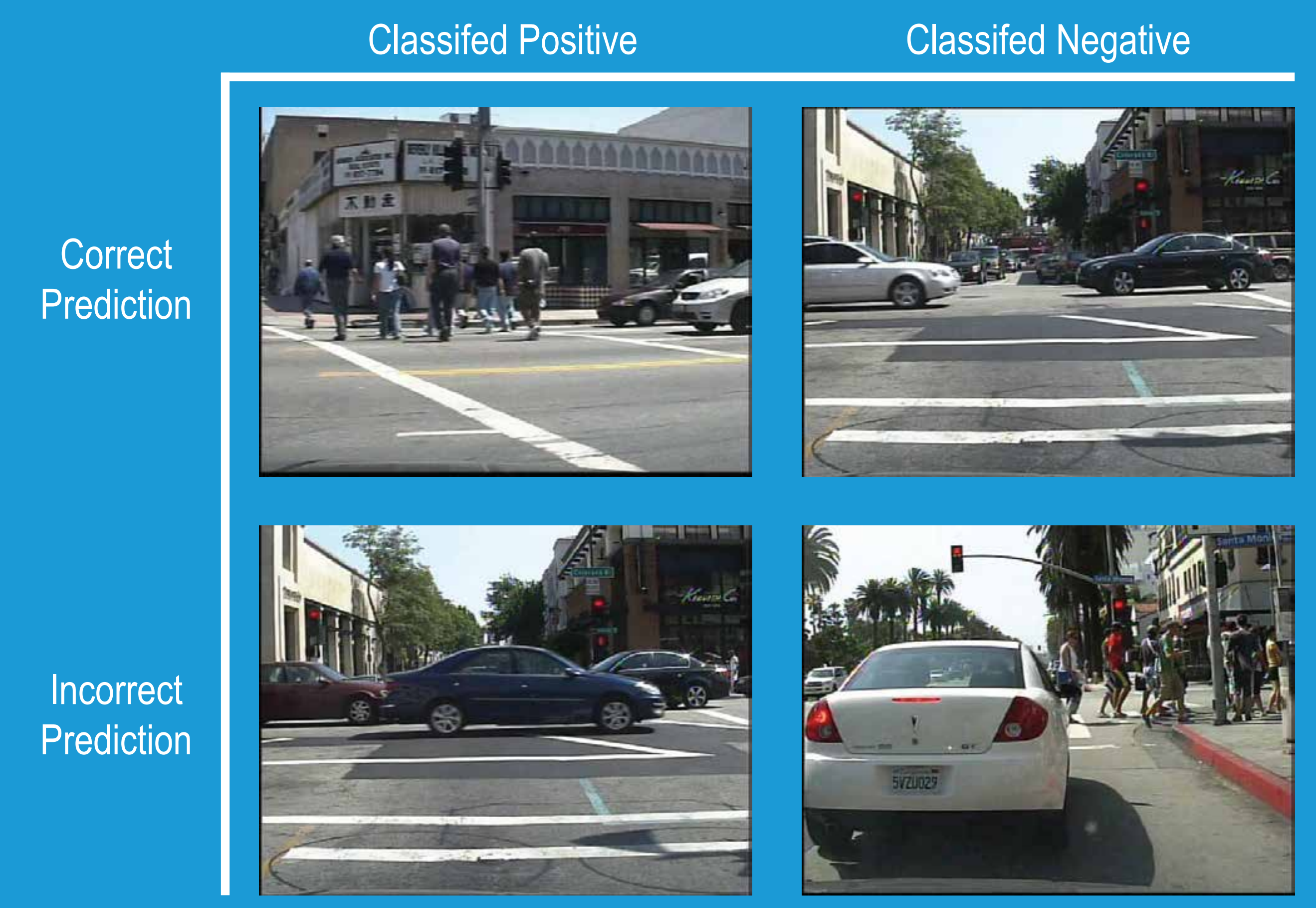

Figure 6: Results for a sample of testing images. Bottom left image is an example of a false positive. Bottom right image is an example of a false negative.

VIII. Conclusion + Future Work

Our objective was to evaluate the effectiveness of PCA for pedestrian detection. Our results demonstrate that our algorithm was reasonably effective in identifying pedestrians, with a maximum accuracy around 78%. From this, we conclude that PCA can reduce the dimensionality of data while maintaining enough information to make an informed prediction. Our results also show that PCA alone is not suitable for pedestrian detection applications where high degrees of accuracy are required. In safety critical use cases, such as vehicle safety measures, an error rate of 22% would likely be unacceptably high.

Existing research demonstrates that PCA can be combined with traditional machine learning models or neural networks to improve the efficiency and effectiveness of these algorithms [4], which is important for real-time computing applications. In the future, we would improve our algorithm's accuracy by implementing a traditional machine learning model.

References

[1] P. Dollar, C. Wojek, B. Schieleand P. Perona, "Caltech Pedestrians". IEEE Conference on Computer Vision and Pattern Recognition, Jun. 20, 2009. doi: 10.1109/CVPR.2009.5206631.

[4] W. Ke, Y. Zhang, P. Wei, Q. Ye, and J. Jiao, "Pedestrian detection via PCA filters based convolutional channel features," in 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 1394–1398, 2015. doi: