Computer Vision Final Project: Adaboost Face Detector

By Erick Guerra, George Huincho, and Michael Mendez

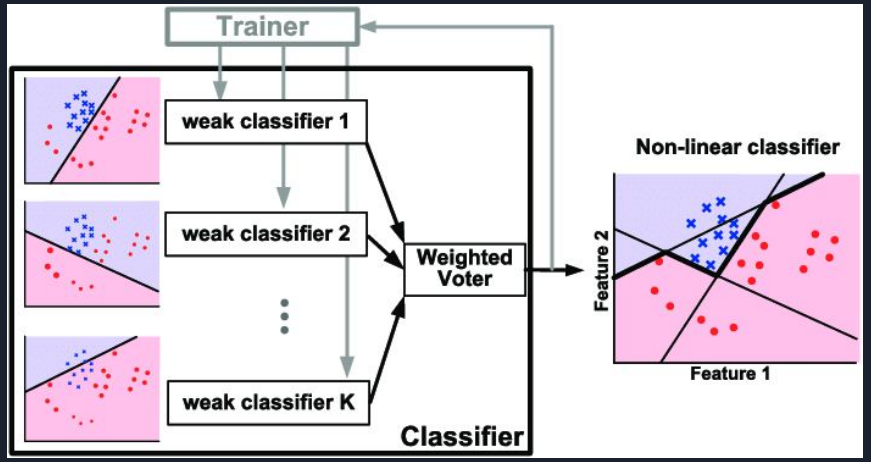
Introduction and Objectives

- We were tasked with implementing a face detector that is trained using Adaboost and rectangle filters.
- Three other concepts were implemented in the detector to improve its accuracy and speed.
  - Skin detection, Bootstrapping, and Classifier Cascades.
- Some challenges we faced were figuring out how to properly train the detector, bootstrapping in a manner that doesn't retrain redundant data, and figuring out a good cascade of classifiers while also being able to apply it to windows properly.

- Important Note: A classifier and a detector are not the same. A classifier looks at an image and identifies it as either being something or not being something. A detector looks at an image and scans multiple windows, applying the classifier, looking for the thing that it is detecting as it identifies the position where it is at.

What is Adaboost?

- Adaboost is short for Adaptive Boosting . In simple terms, it is an algorithm that combines a bunch of weak classifiers with weights to create a single strong boosted classifier.
- In Adaboosting, more weight is put on difficult to classify instances.



How The Data Was Trained

- We scanned the training face and nonface folders then put those images into their appropriate matrices. Integral Images were generated and stored as well.
- We picked 2000 training faces since this amount wasn't too little yet left some faces for bootstrapping in the future.
- We picked 100 training nonfaces for the same reason.
- Training nonfaces were cropped to 100 by 100 to match the size of the faces.

Rounds	Time Elapsed Training
1	31.92 seconds
2	29.01 seconds
3	29.41 seconds
4	29.37 seconds
5	29.54 seconds

Average Elapsed Time Training Data = 29.85 seconds

How The Detector Was Trained With Adaboost

- 1000 weak classifiers were generated to ensure that there were enough and for effectiveness.
- The responses and labels were then processed and fed into the AdaBoost function provided to us to create the boosted classifier.
- Top 50 classifiers were used. We chose this number because it was recommended during lecture.

Results of Boosted Classifier

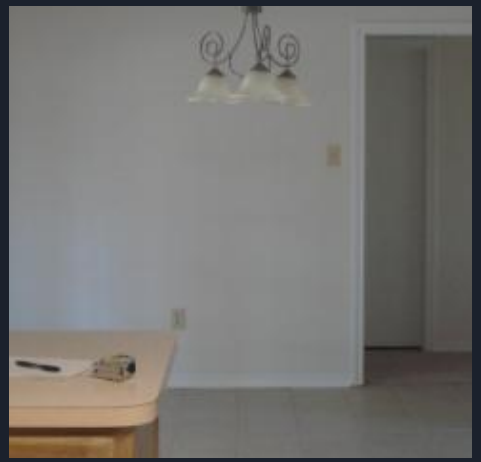
- To test the boosted classifier, it was fed in to the boosted_predict function along with the weak classifiers.
- This function was called on the test cropped faces since those images were 100 by 100 and no window searching was necessary. The function was also called on the test nonfaces which we cropped to 100 by 100 as well.
- This was testing classifying accuracy.
- Results were initially pretty bad.
- A lot of false negatives but not a lot of false positives.

Rounds	Number Correctly Identified	Number of False Positives	Number of False Negatives	Accuracy	Time Elapsed Testing
1	174	1	631	21.59%	1.16 seconds
2	125	13	668	15.51%	1.15 seconds
3	238	13	555	29.53%	1.19 seconds
4	180	1	625	22.33%	1.16 seconds
5	35	3	768	4.34%	1.17 seconds

Average Classifying Accuracy = 18.6%

Results of Boosted Classifier

Example of a False Positive

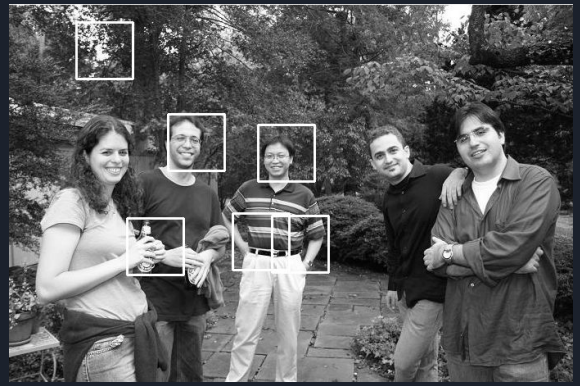


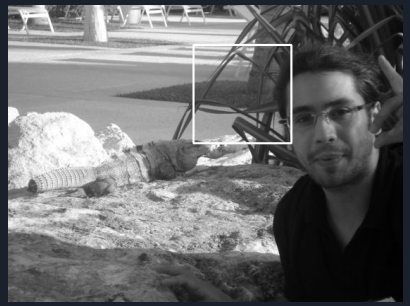
Example of a False Negative

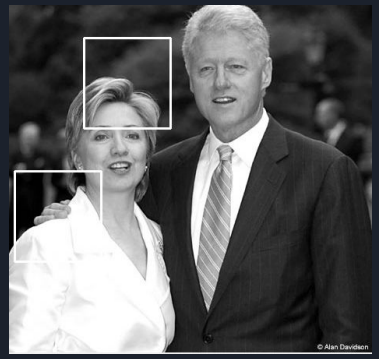
*Possibly because it was not as brightly lit as the other images?

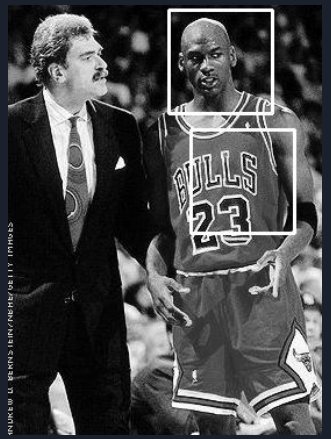
Results of Just Adaboost on Face Detection

- To test the face detector with the boosted classifier, the boosted classifier was fed into the boosted_detector_demo function.
- This function could scan an image through various scales and detect the point where the strong classifier best identified a face.
- This function was tested on the noncropped test faces since it could actually scan through multiple windows and look for the face.
- Results were initially bad with the face detector as well.



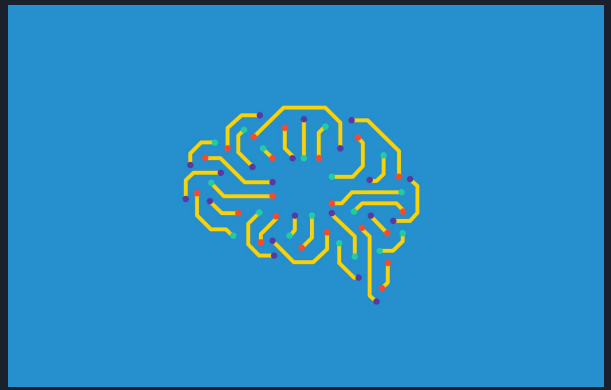






What is Bootstrapping

- Bootstrapping is a process used for improving the quality of the training data.
- In bootstrapping, the program identifies and includes more challenging examples, allowing for gradual construction of a good training set
- Keeps overall memory and time requirements in check.

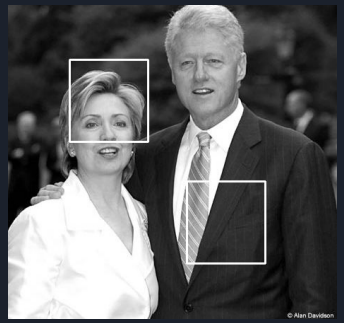


How Bootstrapping Was Implemented

- We had to come up with a way to perform bootstrapping that didn't retrain redundant data.
- After creating the initial boosted classifier, we looked through 200 training faces that weren't used and tested the classifier on those images. If the classifier incorrectly identified them, they replaced already existing training images in the training data set.
- The same process was done for 10 training nonfaces.
- We picked these numbers because they were about 10% of the data, which seemed to be an amount not too big or small.
- After replacing the data with new data, it was processed and a new stronger boosted classifier was created. This was what one round of bootstrapping looked like.
- 2 rounds of bootstrapping were performed.

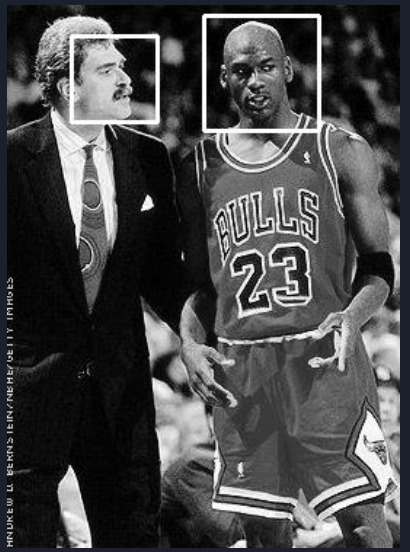
Results of Bootstrapping

- The new bootstrapped strong classifier was tested on the noncropped faces for face detection accuracy and tested on the cropped faces and nonfaces for classifying accuracy.
- Bootstrapping increased the training data elapsed time to 91.98 seconds but significantly improved classifying accuracy. Face detection was more accurate as well but improvements could still be made.



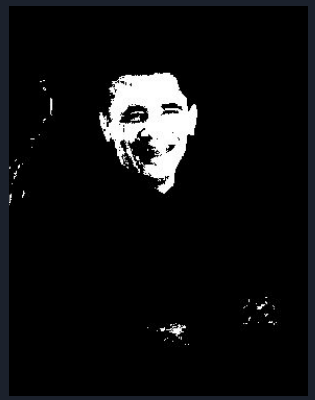
Rounds	Number Correctly Identified	Number of False Positives	Number of False Negatives	Accuracy
1	784	9	12	97.39%
2	789	6	11	97.89%
3	795	5	6	98.64%
4	792	8	6	98.26%
5	775	9	22	96.15%

Average Classifying Accuracy = 97.66%



How Skin Detection Was Implemented

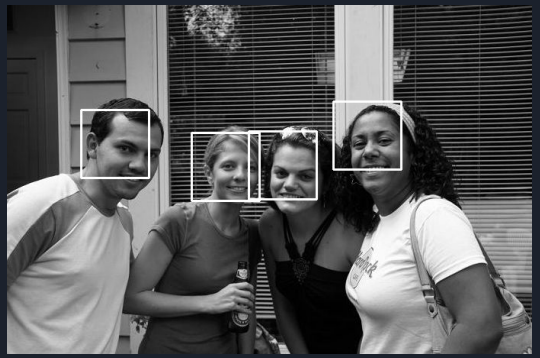
- Skin detection was implemented by loading the negative and positive histograms, passing the image and histograms to the detect_skin function, then setting a threshold on the skin detection to get the skin pixels.
- A threshold of .8 was chosen because it provided decent results.
- The skin pixels were then passed to the boosted_detector_demo function.

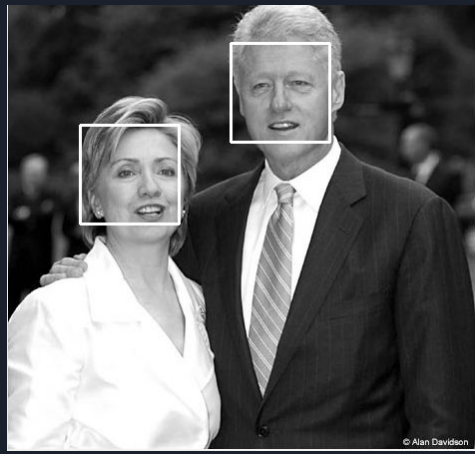


Example of what the skin pixels looked like

Results of Skin Detection

- Implementing skin detection for the face detector greatly improved the detector's ability to detect the faces.

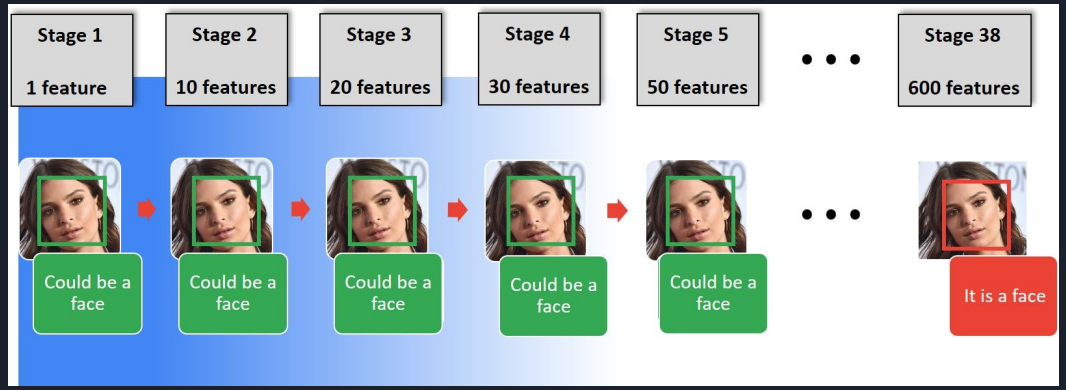




*still has a bit of trouble on the lord of the rings image

What is a Cascade of Classifiers?

- A cascade of classifiers is a sequence of classifiers.
- The first set of classifiers are fast and not extremely accurate, then as you go down the sequence, the classifiers become slower and more accurate.
- Using classifier cascades is essentially a generalization of the filter-and-refine approach



How Cascade Classifiers Were Implemented

- The classifiers that we put together as a cascade were created in the training file using the AdaBoost function.
- We decided to create a cascade made up of 5 strong classifiers that each have a different number of top weak classifiers.
- The first strong classifier is made up of 10 weak classifiers and is the least accurate.
- The following strong classifier in the sequence has 10 more weak classifiers, so the second would be made up of 20, and the third made up of 30, until we reach the last which is made up of 50 top weak classifiers.
- These classifiers were sent into a function we created called cascade_detect that also accepted an image and weak classifiers.
- This cascade_detect function scanned through every 100 by 100 window in the image and retrieved a score for that window which we got using another function we created called cascade_classify.
- The cascade_classify function looks at the window and uses the function eval_weak_classifiers to compare a score against a threshold at each classifier in the cascade.
- The window gets a proper score if it passes all thresholds to the end of the sequence of classifiers, otherwise it is 0.
- The location of the face is identified at the window with the largest score.

Results of Cascade Classifiers

- The cascade_detect function we created that uses a cascade of classifiers is able to detect faces, but unfortunately takes more time to detect than the simple Adaboost.
- We believe this may be because our first strong classifier is actually too weak and doesn't allow us to have a high enough threshold to filter out bad windows fast enough.

Takes 11.04 seconds vs 1.64 seconds

Takes 4.26 seconds vs 1.7 seconds

Conclusion

- Our group managed to train a face detector with Adaboost, while also implementing bootstrapping, skin detection, and classifier cascades.
- 2 rounds of bootstrapping significantly increased classifier and detector accuracy.
- Skin detection greatly improved the face detector's accuracy as well.
- Ideally a proper implementation of a cascade of classifiers should filter and refine which should shorten the time it takes to detect a face.
- Some future improvements that could be made would be going for a different approach with skin detection for a better output in images like the lord of the rings image. Something more complex and intricate could produce better results. Also messing around with the number of classifiers in the cascade of classifiers or testing different numbers of weak classifiers per strong classifier would likely fix our issue with our classifier cascades implementation.

Questions?