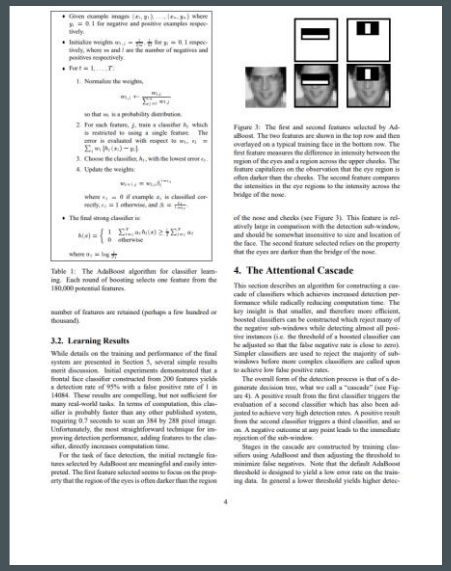
HAARS CASCADE VS DEEP LEARNING MODEL

JUSTIN BISHOP

INTRODUCTION

- The goal of my project will be to blur everyone's faces that is being seen by the camera. I will be comparing two different algorithms against each other. One which is fast but less accurate. The other will be much slower but do a better job.
- Algorithm should automatically recognize the face
- Algorithm should add confidence counter to show the algorithms own assurance on the blur

LITERATURE REVIEW



- Viola, P., and M. Jones. “Rapid Object Detection Using a Boosted Cascade of Simple Features.” Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, 2001, doi:10.1109/cvpr.2001.990517.

Haars Cascade

- Focused highly on Viola and Jones paper titled “Rapid Object Detection Using a Boosted Cascade of Simple Features.”
- In their paper Viola and Jones describe how their algorithm achieved, “a frontal face detection system which achieves detection and false positive rates which are equivalent to the best published results”.

Machine Learning

- Focused on the concept of machine learning
- Used examples from paper “Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks” by K. Zhang, Z. Zhang, Z. Li and Y. Qiao

METHODOLOGY

HAARSCASCADE

- Advantages
  - Pre trained datasets
- Disadvantages
  - Slower than other methods
  - Original
  - Not great at profile of faces

Machine Learning

- Advantages
  - MTCNN model
  - Very easy to implement
  - Can detect more of face
- Disadvantages
  - Much slower run time

IMPLEMENTATION (HAARS CASCADE)

- Uses pre – built XML classifiers through OpenCV [1]
  - Frontal Face
  - Eyes
  - Profile Face
- Gaussian blur
- Time to complete
- Confidence counter in relation to faces detected

```
# Load the required trained XML classifiers
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
profile_cascade = cv2.CascadeClassifier('haarcascade_profileface.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
```

IMPLEMENTATION (MACHINE LEARNING)

- Uses MTCNN model through OpenCV
- Time to complete
- Gaussian blur
- Confidence counter in relation to faces detected

```
1 usage   JustInCase289
def display_blurred_faces(filename):
    # Load image using OpenCV
    image = cv2.imread(image_path)

    # Convert the image from BGR to RGB format (as MTCNN expects RGB)
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # Initialize MTCNN detector
    detector = MTCNN()
```

EXPERIMENTAL SETUP METRICS

Accuracy

How many faces were accurately blurred

Time to complete

How long does it take for the algorithm to run and display back image

Confidence

How confident is the algorithm that it correctly identified and blurred a face

CODE SNIPPETS (HAARS CASCADE

```python
# Detect faces with different minNeighbors
faces_3n = face_cascade.detectMultiScale(gray, scaleFactor: 1.3, minNeighbors: 3)
faces_5n = face_cascade.detectMultiScale(gray, scaleFactor: 1.3, minNeighbors: 5)
faces_7n = face_cascade.detectMultiScale(gray, scaleFactor: 1.3, minNeighbors: 7)

# Detect left-looking profile faces
profiles_left_3n = profile_cascade.detectMultiScale(gray, scaleFactor: 1.3, minNeighbors: 3)
profiles_left_5n = profile_cascade.detectMultiScale(gray, scaleFactor: 1.3, minNeighbors: 5)
profiles_left_7n = profile_cascade.detectMultiScale(gray, scaleFactor: 1.3, minNeighbors: 7)

# Flip the image horizontally
flipped_img = cv2.flip(img, flipCode: 1)
flipped_gray = cv2.cvtColor(flipped_img, cv2.COLOR_BGR2GRAY)

# Detect right-looking profile faces
profiles_right_3n = profile_cascade.detectMultiScale(flipped_gray, scaleFactor: 1.3, minNeigh
profiles_right_5n = profile_cascade.detectMultiScale(flipped_gray, scaleFactor: 1.3, minNeigh
profiles_right_7n = profile_cascade.detectMultiScale(flipped_gray, scaleFactor: 1.3, minNeigh

# Flip the coordinates back
profiles_right_3n = [(img.shape[1] - x - w, y, w, h) for (x, y, w, h) in profiles_right_3n]
profiles_right_5n = [(img.shape[1] - x - w, y, w, h) for (x, y, w, h) in profiles_right_5n]
profiles_right_7n = [(img.shape[1] - x - w, y, w, h) for (x, y, w, h) in profiles_right_7n]

# Combine all detections
all_faces_3n = list(faces_3n) + list(profiles_left_3n) + list(profiles_right_3n)
all_faces_5n = list(faces_5n) + list(profiles_left_5n) + list(profiles_right_5n)
all_faces_7n = list(faces_7n) + list(profiles_left_7n) + list(profiles_right_7n)

```

CODE SNIPPETS (HAARS CASCADE)

```
1 usage   JustInCase289 *
def process_face(x, y, w, h, img, gray, confidence_text):
    # Check if face has already been processed
    if is_face_processed(x, y, w, h):
        return

    # Add face to processed faces
    processed_faces.append((x, y, w, h))

    # Blur the face region
    face_region = img[y:y + h, x:x + w]
    blurred_face = cv2.GaussianBlur(face_region, ksize: (99, 99), sigmaX: 30)
    img[y:y + h, x:x + w] = blurred_face

    # Display confidence on the image
    cv2.putText(img, confidence_text, org: (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, fon

    # Detect and blur eyes within frontal faces
    roi_gray = gray[y:y + h, x:x + w]
    roi_color = img[y:y + h, x:x + w]
    eyes = eye_cascade.detectMultiScale(roi_gray)
    for (ex, ey, ew, eh) in eyes:
        eye_region = roi_color[ey:ey + eh, ex:ex + ew]
        blurred_eye = cv2.GaussianBlur(eye_region, ksize: (49, 49), sigmaX: 30)
        roi_color[ey:ey + eh, ex:ex + ew] = blurred_eye


for x, y, w, h, confidence_text in all_faces:
    process_face(x, y, w, h, img, gray, confidence_text)
```

CODE SNIPPETS (MACHINE LEARNING)

```
1 usage   JustInCase289 *
def display_blurred_faces(filename):
    # Load image using OpenCV
    image = cv2.imread(image_path)

    # Convert the image from BGR to RGB format (as MTCNN expects RGB)
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # Initialize MTCNN detector
    detector = MTCNN()

    # Start the timer
    start_time = time.time()

    # Detect faces in the image
    faces = detector.detect_faces(image_rgb)

    # For each detected face
    for face in faces:
        # Get coordinates and dimensions of the bounding box
        x, y, width, height = face['box']

        # Get the confidence score
        confidence = face['confidence']

        # Extract the face from the image
        face_image = image[y:y + height, x:x + width]
        # Blur the face
        face_image = cv2.GaussianBlur(face_image, ksize: (99, 99), sigmaX: 30)
        # Replace the original face with the blurred version
        image[y:y + height, x:x + width] = face_image
```

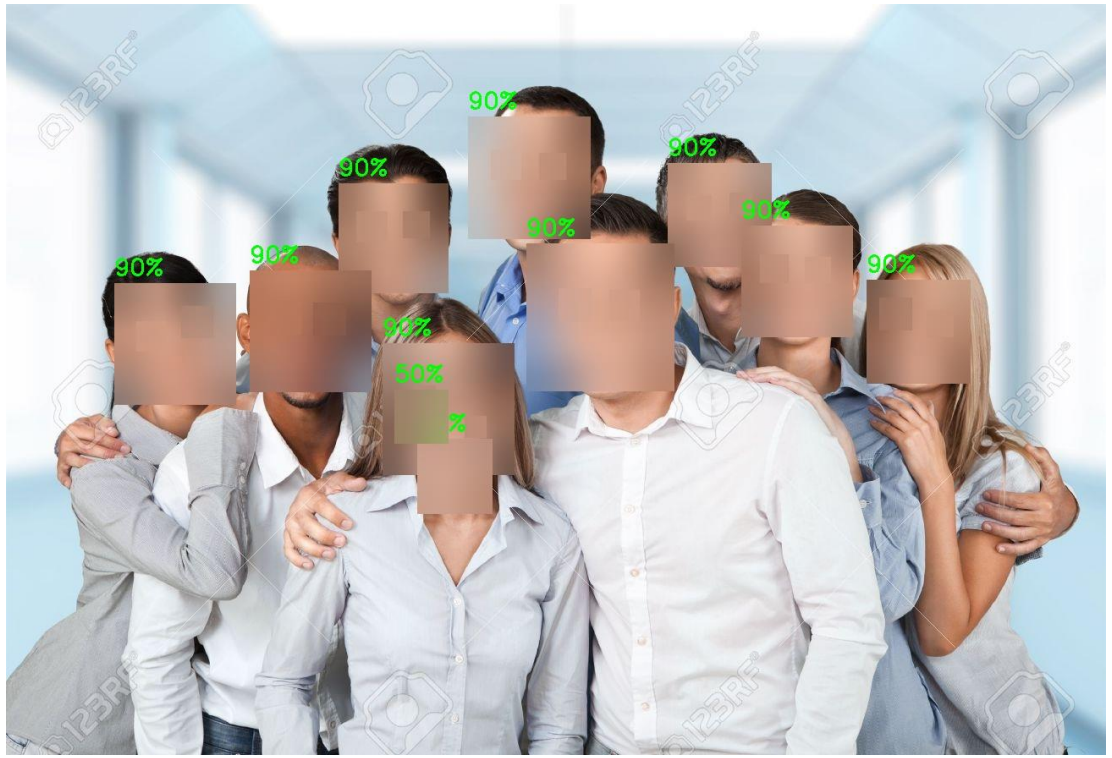
CODE SNIPPETS (MACHINE LEARNING)

```
        # Overlay the confidence score on the image
        confidence_text = f"{confidence:.2f}"
        cv2.putText(image, confidence_text, org: (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, fontScale: 0.5, color: (255, 0, 0), thickness: 1)

    # End the timer
    end_time = time.time()

    # Measure resource usage after processing the image but before displaying it
    cpu_after, mem_after = get_resource_usage()

    # Display the resulting image
    cv2.imshow( winname: 'Blurred Faces', image)
    cv2.waitKey(0)  # Wait until a key is pressed
    cv2.destroyAllWindows()  # Close the window

    # Return the elapsed time
    return end_time - start_time, cpu_after, mem_after


# Single image path variable
image_path = 'C:\\Users\\bisho\\PycharmProjects\\FacialRecog\\MainProd\\TestImages\\StockImageGroup5.jpg'

# Example usage
elapsed_time, cpu_after, mem_after = display_blurred_faces(image_path)

print(f"Time taken to process the image: {elapsed_time:.2f} seconds")
print(f"CPU usage before: {cpu_before}% | After: {cpu_after}%")
print(f"Memory usage before: {mem_before}% | After: {mem_after}%")
```

IINITAL RESULTS

HAARS CASCADE

DEEP LEARNING



IINITAL RESULTS

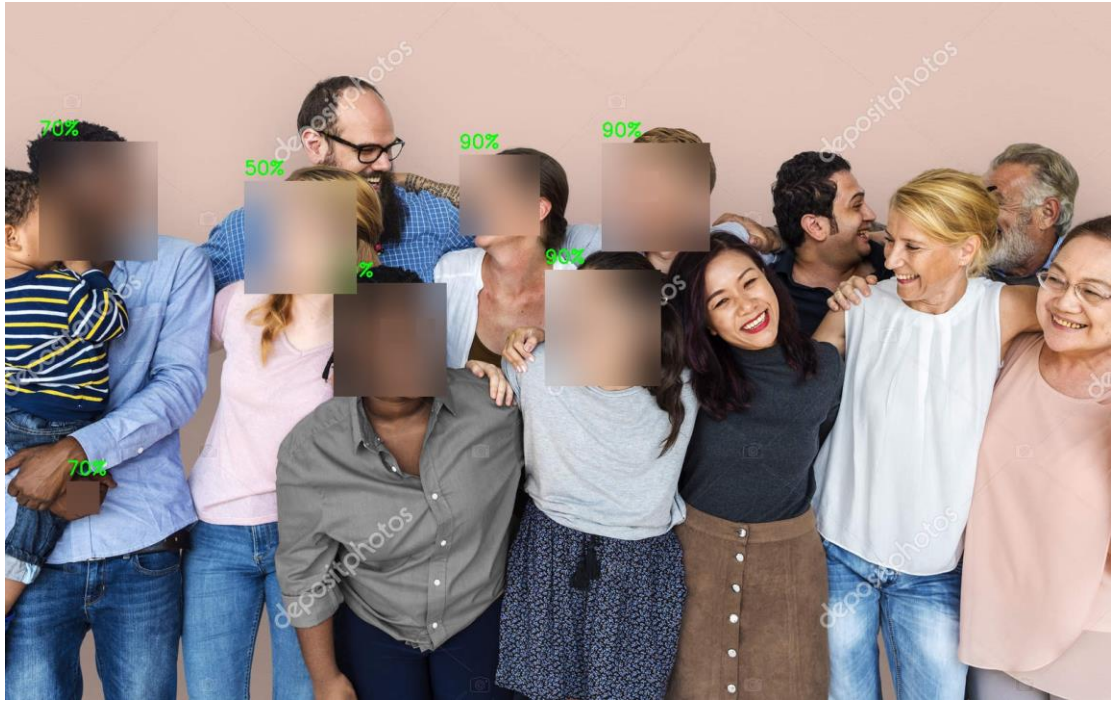
HAARS CASCADE

DEEP LEARNING

IINITAL RESULTS

HAARS CASCADE

DEEP LEARNING



IINITAL RESULTS

HAARS CASCADE

DEEP LEARNING

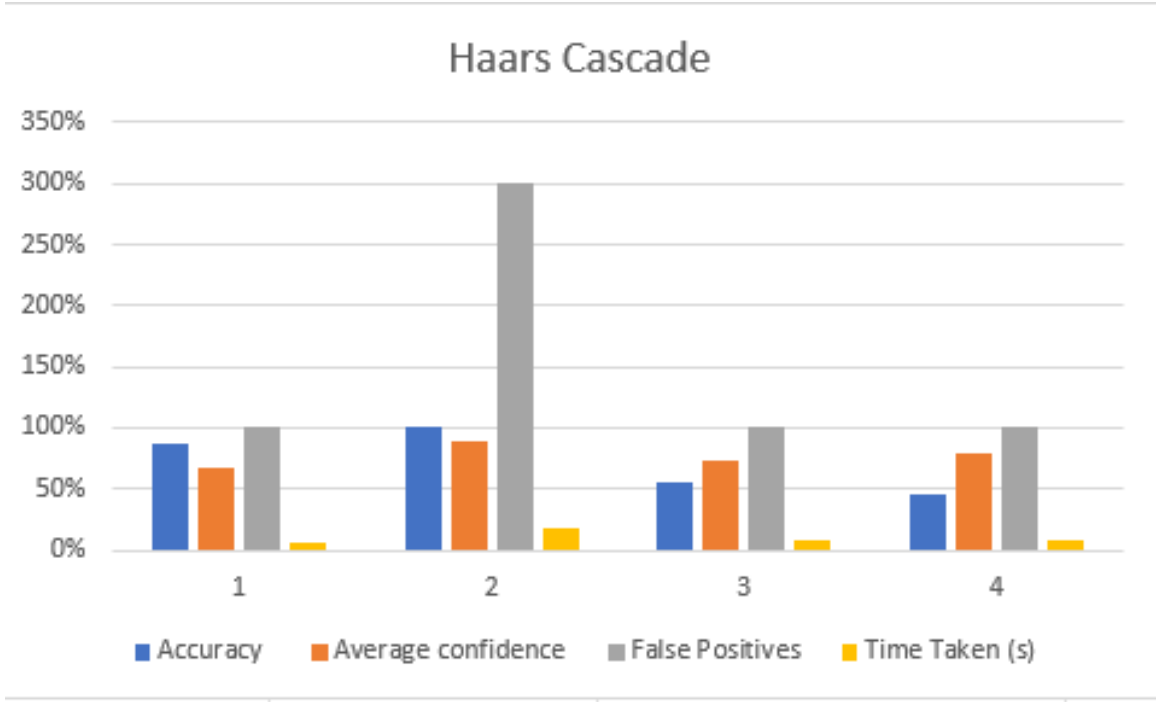
FINAL RESULTS

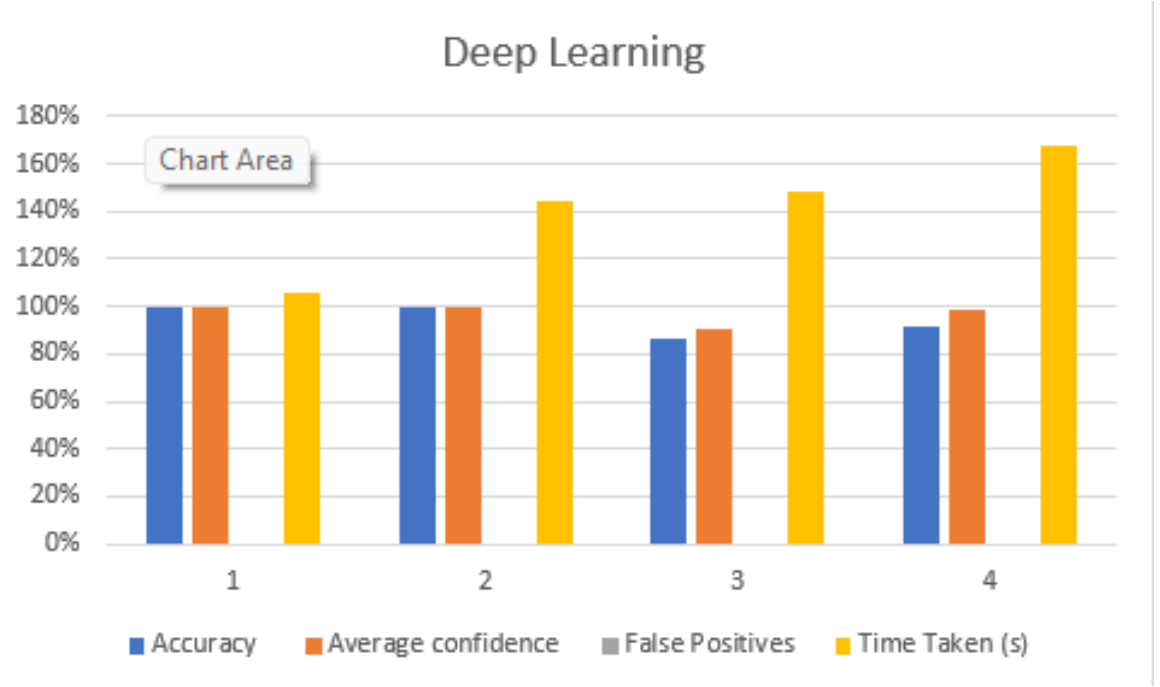
HAAR CASCADE

Group Image #	Accuracy	Average confidence	False Positive	Time Taken (s
1	88%	68%	1	0.0571
2	100%	90%	3	0.1701
3	55%	73%	1	0.0838
4	46%	80%	1	0.0877

DEEP LEARNING

Group Image #	Accuracy	Average confidence	False Positives	Time Taken (s)
1	100%	100%	0	1.0551
2	100%	100%	0	1.4458
3	86%	91%	0	1.4827
4	92%	99%	0	1.6736





FINAL RESULTS

CONCLUSION

Results have shown that the machine learning model beats the Haars Cascade model in every single test minus time to complete the task

Machine learning has much more uses such as when the face is not facing the camera directly

Haars Cascade may be faster for real time tracking due to facial detection being overall quicker

REFERENCES

1. Ali, Alya'a R., and Ban N. Dhannoon. "Real time multi face blurring on uncontrolled environment based on color space algorithm." Iraqi Journal of Science, 2019, pp. 618–1626, https://doi.org/10.24996/ijs.2019.60.7.22.
2. Ali, Alya'a R., and Ban N. Dhannoon. "Real time multi face blurring on uncontrolled environment based on color space algorithm." Iraqi Journal of Science, 2019, pp. 618–1626, https://doi.org/10.24996/ijs.2019.60.7.22.
3. A. Sharifara, M. S. Mohd Rahim and Y. Anisi, "A general review of human face detection including a study of neural networks and Haar feature-based cascade classifier in face detection," 2014 International Symposium on Biometrics and Security Technologies (ISBAST), Kuala Lumpur, Malaysia, 2014, pp. 73-78, doi: 10.1109/ISBAST.2014.7013097.
4. A. Singh; H. Herunde; F. Furtado. "Modified Haar-cascade model for face detection issues". International Journal of Research in Industrial Engineering, 9, 2, 2020, 143-171. doi: 10.22105/riej.2020.226857.1129
5. "Caltech 10K Web Faces." Perona Lab - Caltech 10k Web Faces, www.vision.caltech.edu/datasets/caltech_10k_webfaces/. Accessed 20 Oct. 2023.
6. Devaux, Alexandre. "Face Blurring for Privacy in Street-Level Geoviewers Combining Face ..." ResearchGate, www.researchgate.net/publication/292848049_Face_blurring_for_privacy_in_street- level_geoviewers_combining_face_body_and_skin_detectors. Accessed 18 Sept. 2023.
7. GitHub, github.com/opencv/opencv/tree/master/data/haarcascades. Accessed 6 Oct. 2023.
8. Gosper, Maree, et al. Web-Based Lecture Technologies: Blurring the Boundaries between Face-To ..., files.eric.ed.gov/fulltext/EJ809980.pdf. Accessed 18 Sept. 2023.
9. Howse, Joseph. OpenCV 4 for Secret Agents: Use Opencv 4 in Secret Projects to Classify Cats, Reveal the Unseen, and React to Rogue Drivers. Packt Publishing, 2019.
10. K. Zhang, Z. Zhang, Z. Li and Y. Qiao, "Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks," in IEEE Signal Processing Letters, vol. 23, no. 10, pp. 1499-1503, Oct. 2016, doi: 10.1109/LSP.2016.2603342.
11. Lewis, Michael B. "Face Detection: Mapping Human Performance - Sage Journals." SageJournals, journals.sagepub.com/doi/10.1068/p5007. Accessed 18 Sept. 2023.
12. Meenakshi, M. "Real-time facial recognition system—design, implementation and validation." Journal of Signal Processing Theory and Applications, 10 Nov. 2012, https://doi.org/10.7726/jspta.2013.1001.
13. Opencv. "Opencv/Opencv: Open Source Computer Vision Library." GitHub, github.com/opencv/opencv. Accessed 19 Oct. 2023.
14. R. Padilla, et al. Evaluation of Haar Cascade Classifiers Designed for Face Detection. 2910, Zenodo, Apr. 2012, doi:10.5281/zenodo.1058133.
15. Viola, P., and M. Jones. "Rapid Object Detection Using a Boosted Cascade of Simple Features." Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, 2001, doi:10.1109/cvpr.2001.990517.